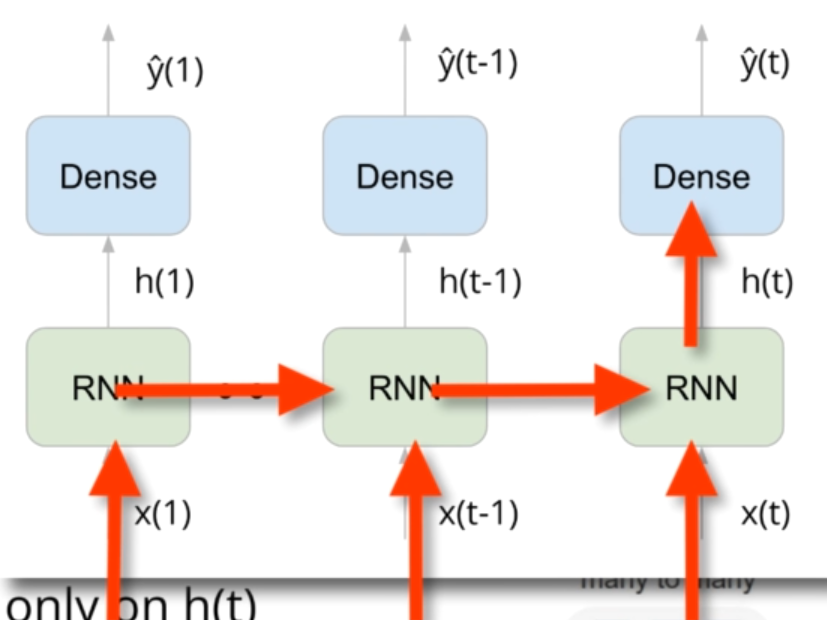
# Beginner Corner

## From RNN to Transformers

As we know from RNN that the current output depends on all the previous inputs values not the next ones. Like the y(t) depends on x(1) upto x(t) as shown below.

But take the language translation task, the current output depends not only on previous inputs but also on next ones. Like in Spanish, the question not end only with ? but start with inverted ? also.

So we need the next values as well which is not the case with RNN like LSTM



### Seq2Seq

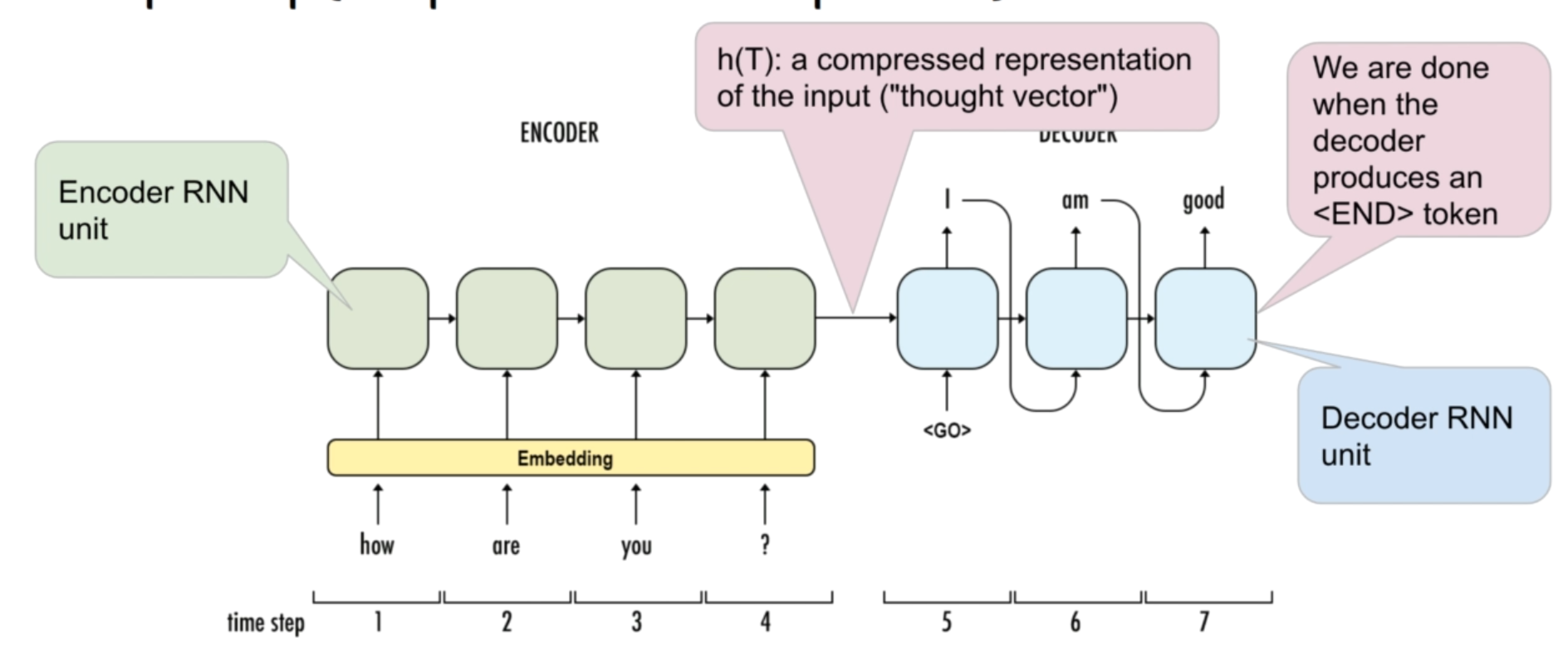
The initial solution to this problem is to used Seq2Seq models that has 2 RNNS. 1 for Encoder and 1 for Decoder as shown below.

The encoder job is to take all the input sequences and have no dense layer to produce the output. Then it produce a big vector h(T) of hidden states. This then feed into decoder.

The first input to decoder is GO (represent start) and the thought vector. The output of the current time stamp is input to the next sequence.

**Problems**

* One problem is that it was not good for long sentences. As the h(T) vector has same length for short and long sentences. Long sentences contain more info, so it can’t solve this problem
* Also RNN not good at remembering dependencies in long terms.

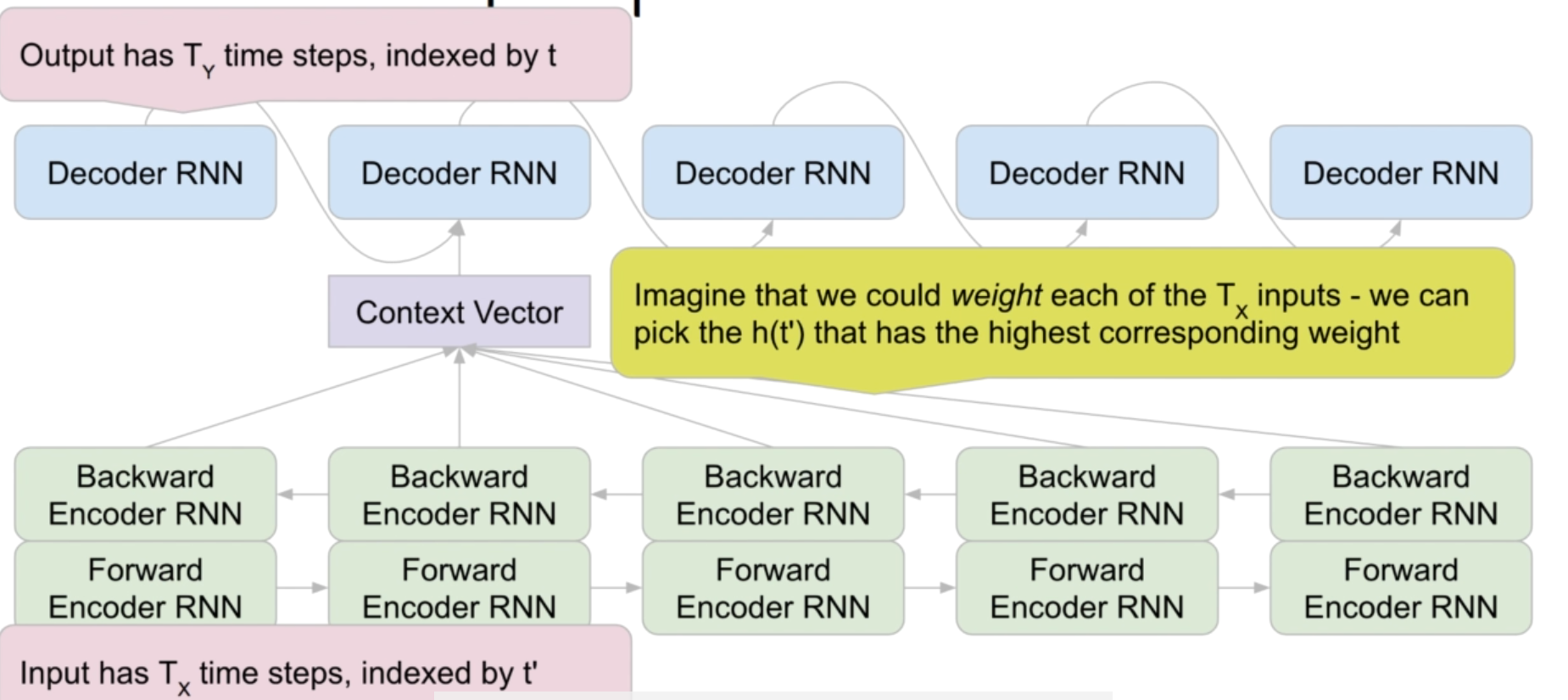


**Solution (Attention Seq2Seq)**

The solution is to have Attention in seq2seq models.

* Attention: For each output token, we want to know which input token(s) to pay attention to.

As you can see in Decoder, we use 2 RNNs (Forward and Backward). Now each input have context from previous and next sequences. The decoder part mostly remain the same instead now we feed the context vector at each time stamp unlikely the simple seq2seq models. The Tx and Ty has different lengths now.



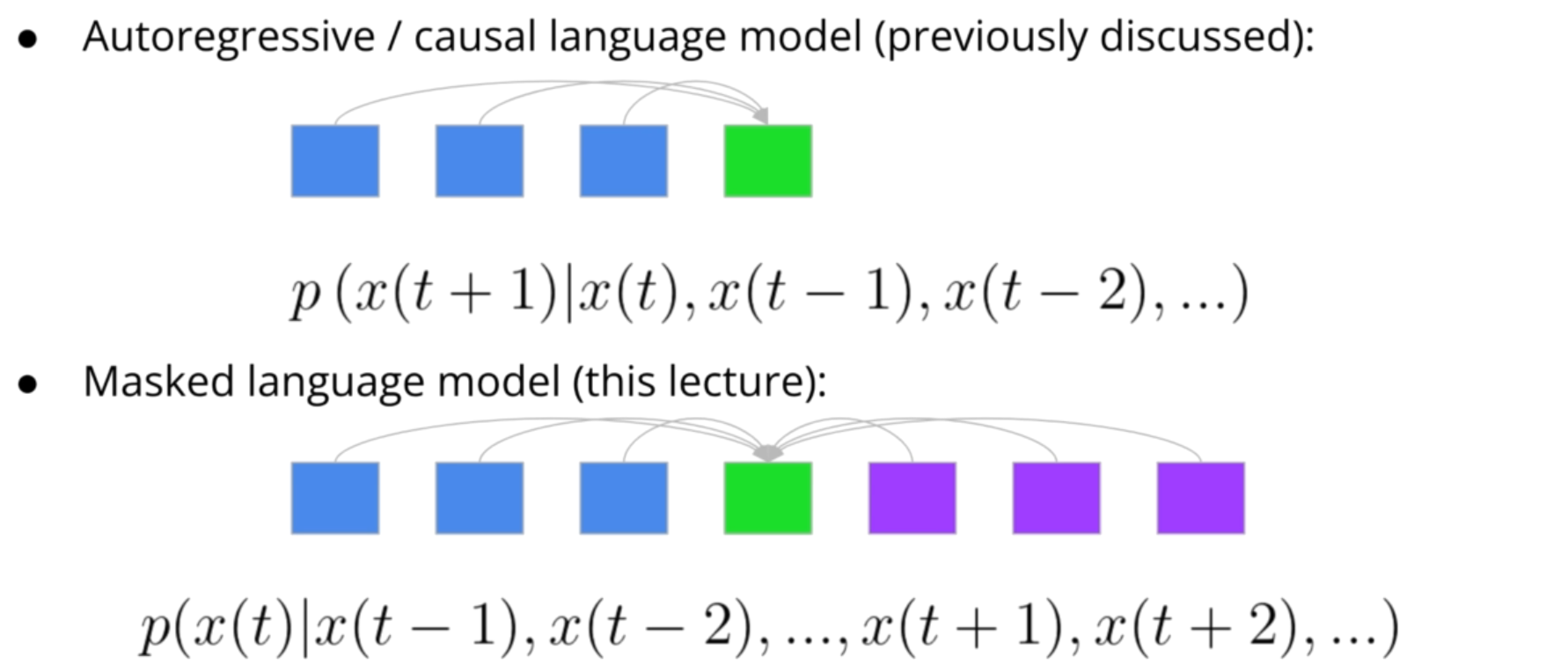
Later in 2017, Paper come out “**Attention is all you need”** which says to keep only Attention and get rid of RNN because of

* RNN are slow, output must be computed sequentially which cannot be parallalized.
* Vanishing gradient problem which was solved by LSTM and GRU but upto some point only.

## Autoregressive and Masked Language Model

As you can see that Autoregressive model only generate next word based on all previous words as shown. However the Masked model generate next word based on previous and next words in corpus as shown below

* GPT-2 is example of Autoregressive models
* BERT is example of Masked models
* Article spinning is based on masked model that is used in SEO



# Fine Tuning

## Text Processing and Tokenization

We will follow the following steps when dealing with any NLP task.

* Tokenize (Simply split strings on punctuation, commas, etc)
* Convert tokens into integers
* As NN not work with different sequence length, next we will **pad/truncate** the sequences

Earlier, we do the word level tokenization. For example, in a sentence **“I like cats. Do you like cats?”.** The word cats. and cats? are 2 distinct tokens. As this is wrong.

Sometime we then do character level tokenization.

### Sub-word Tokenization

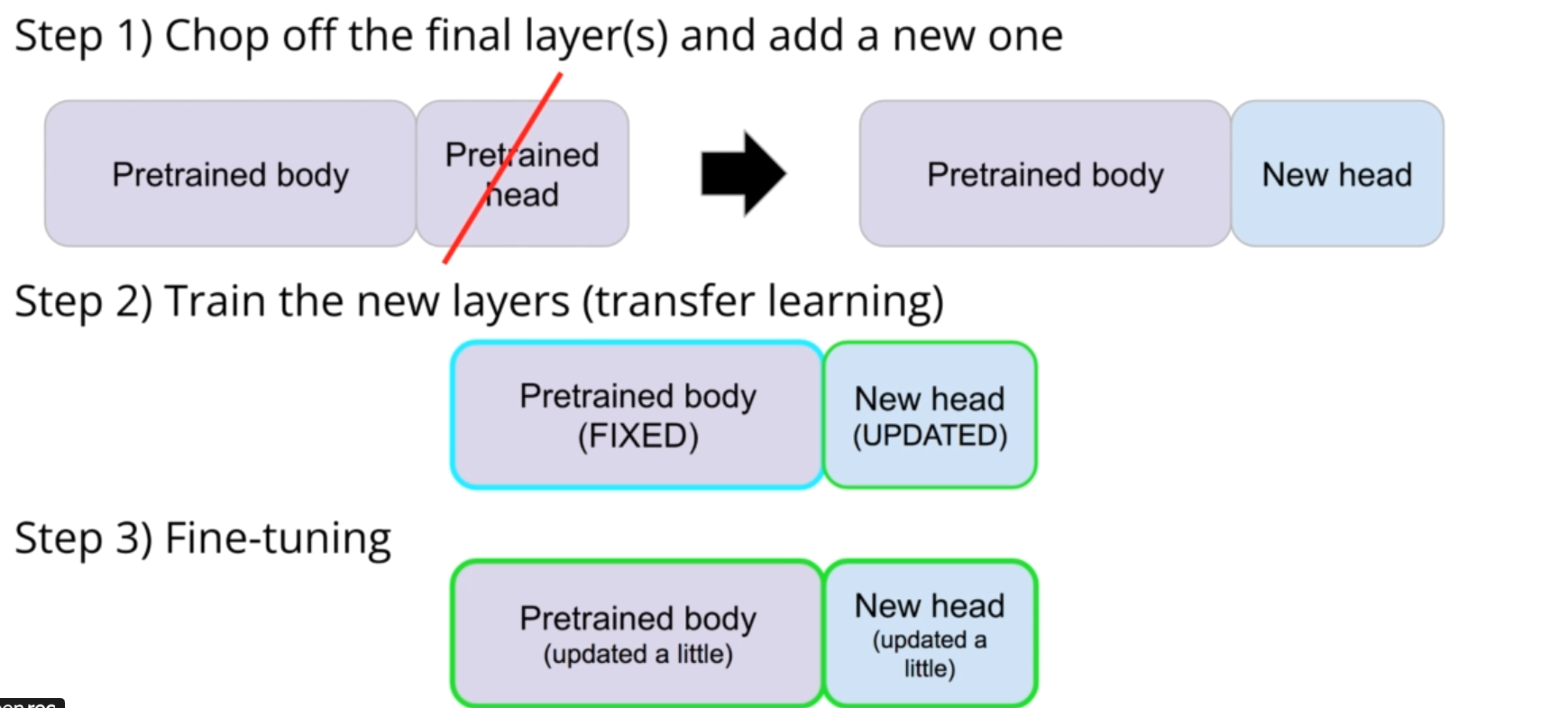
But in transformers, we do subword tokenization. In this, we sometime split one word into multiple tokens. For example **“run, running, runs”,** we split on run as they all have the same meaning.

* Remember, each model use different tokenization. So use the related tokenization for that model.

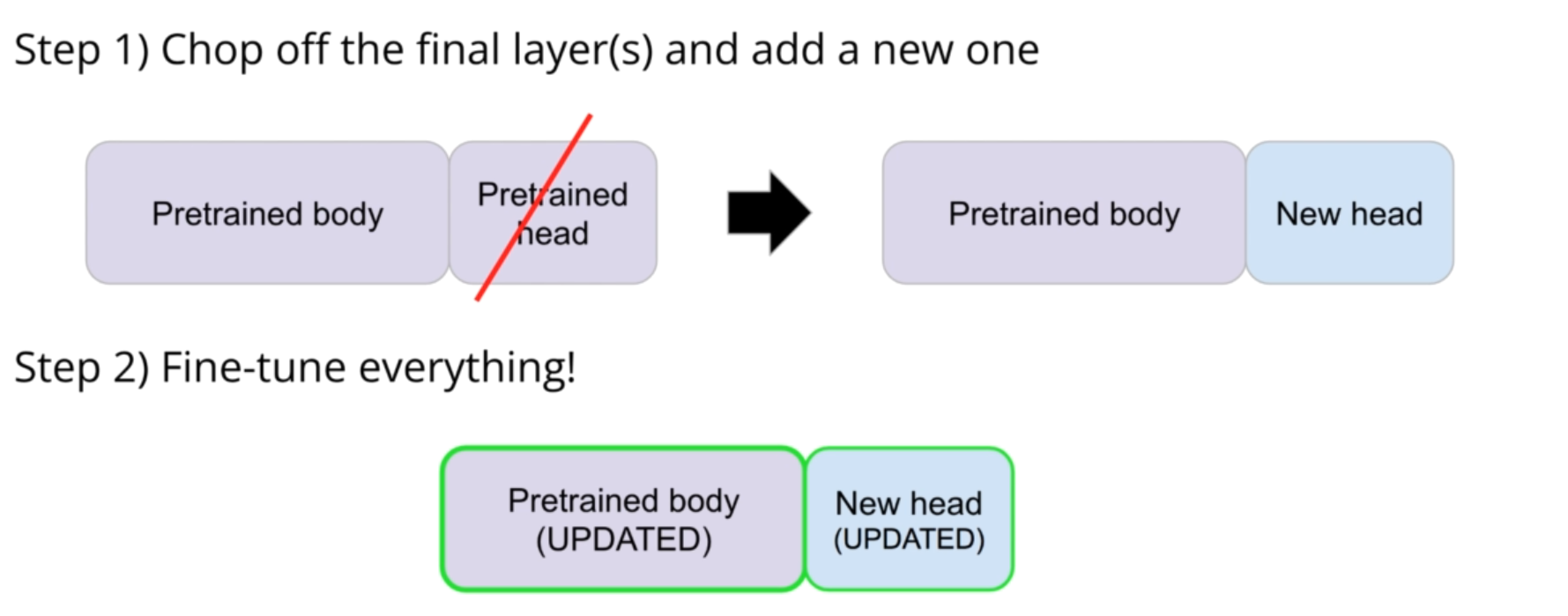
## Transfer Learning and Fine-tuning

These 2 are two different tasks. Let’s discuss them

* Everytime we take some pre-trained weights from other models and add our own final layer and train the final layer, this is called transfer learning.
* In fine-tuning, we update the weights of all the model include new layers and old layers a little bit as shown in the figure.



In transformers, we do the fine-tuning in the way shown below.



So we can say that we can do the following.

* Transfer learning without fine-tuning.
* Fine-tuning without transfer learning.
* Both at the same time.

In NLP, Experiment is the key.