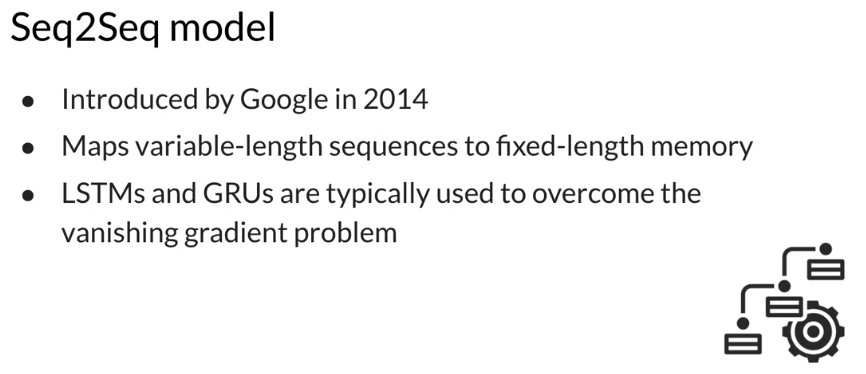
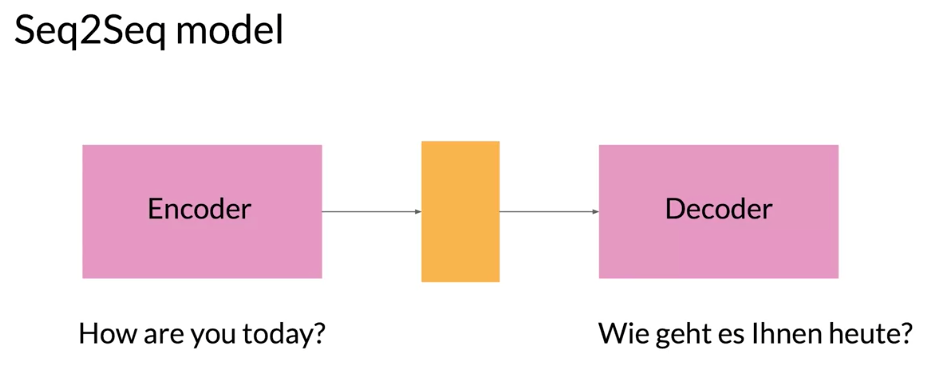
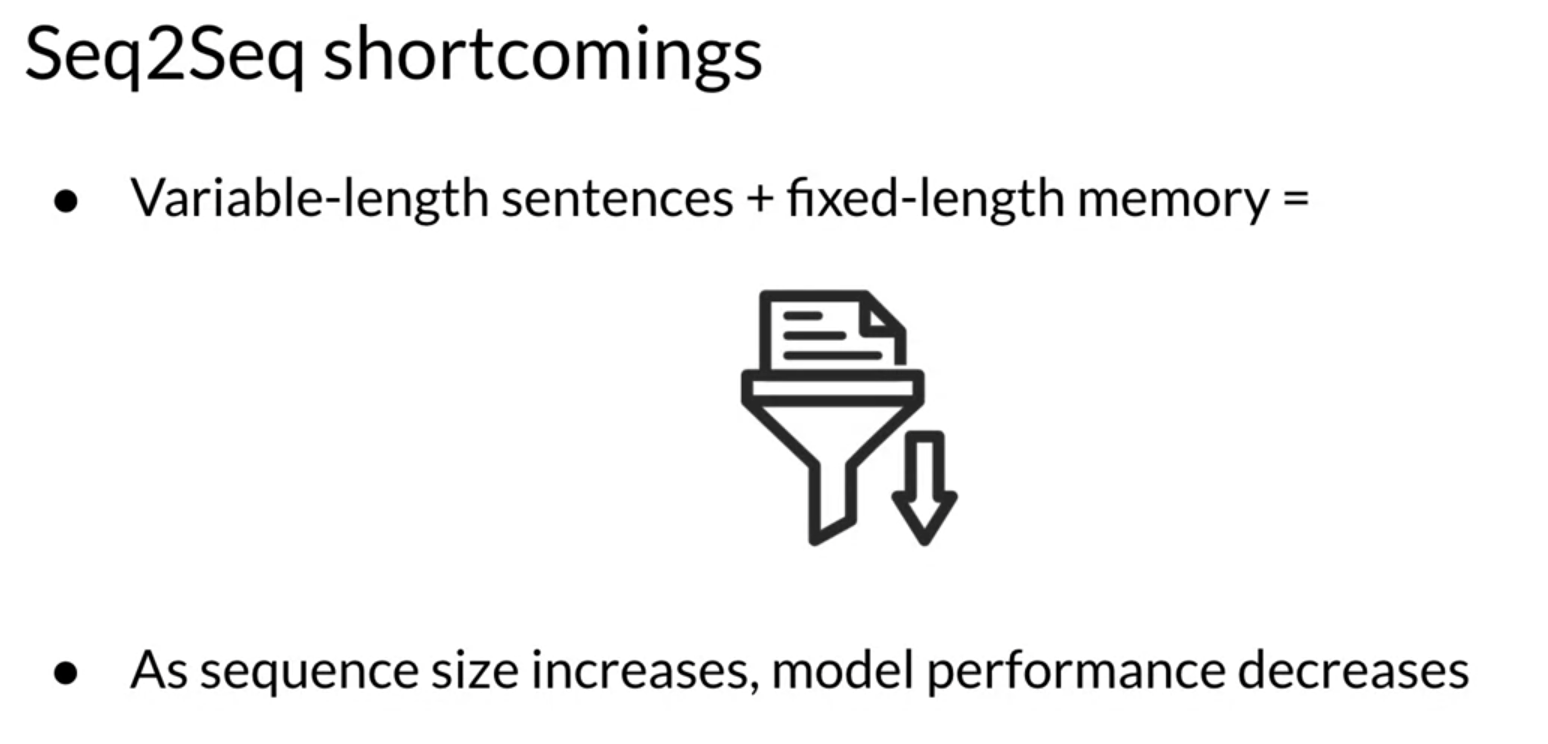
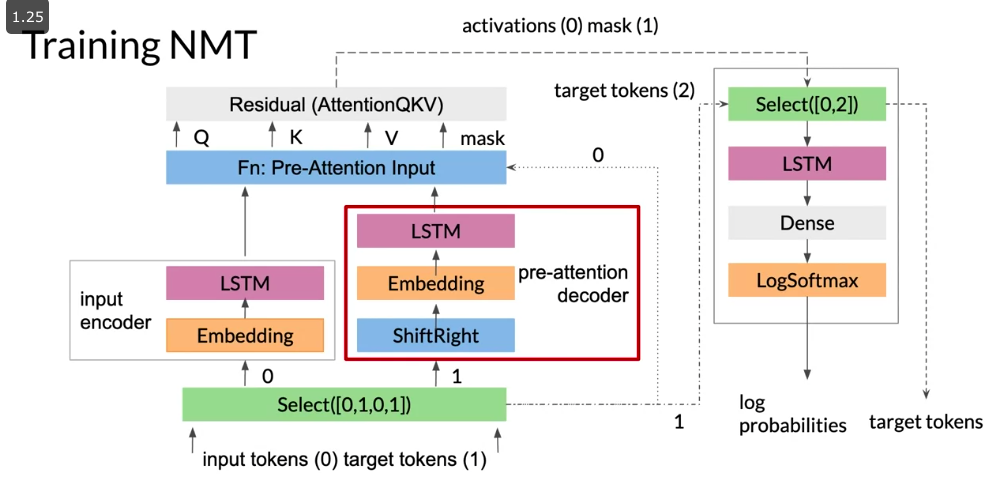
[Natural Language Processing with Attention Models](https://www.coursera.org/learn/attention-models-in-nlp/home/welcome)

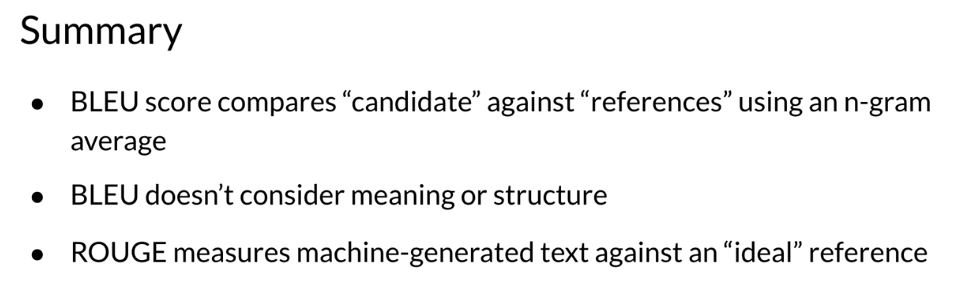
**Week 1 - Neural Machine Translation**

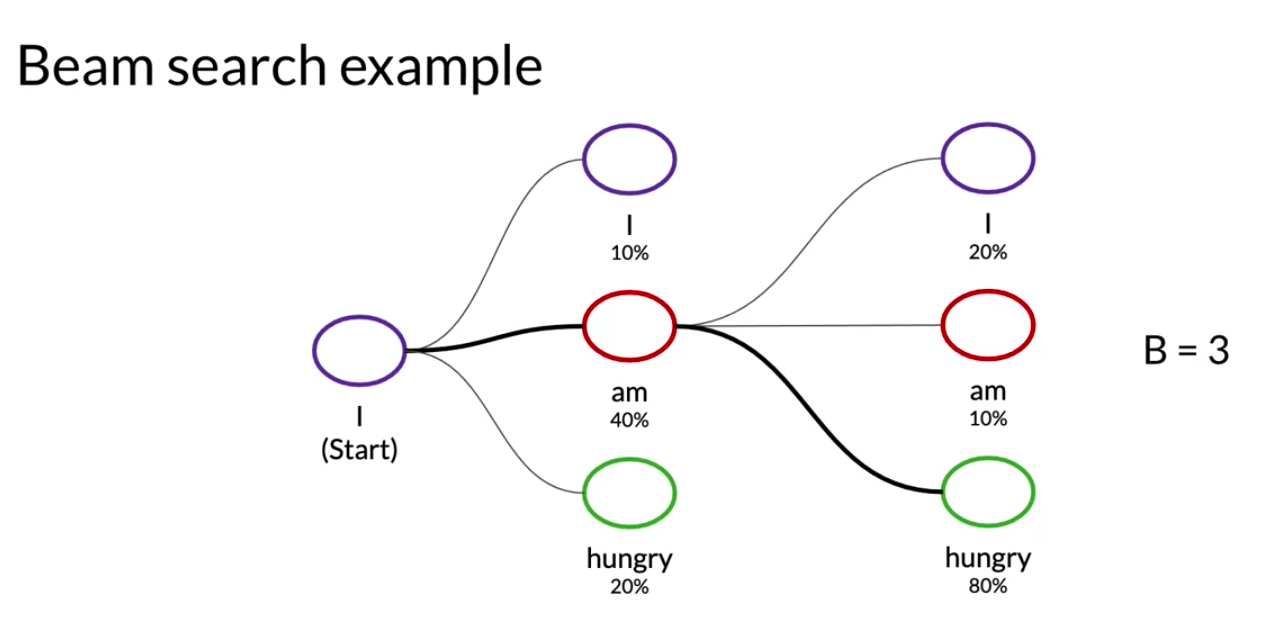
****

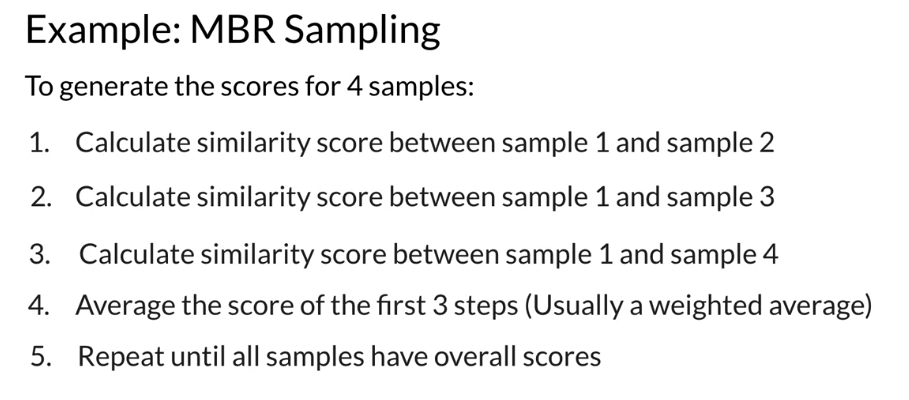
****



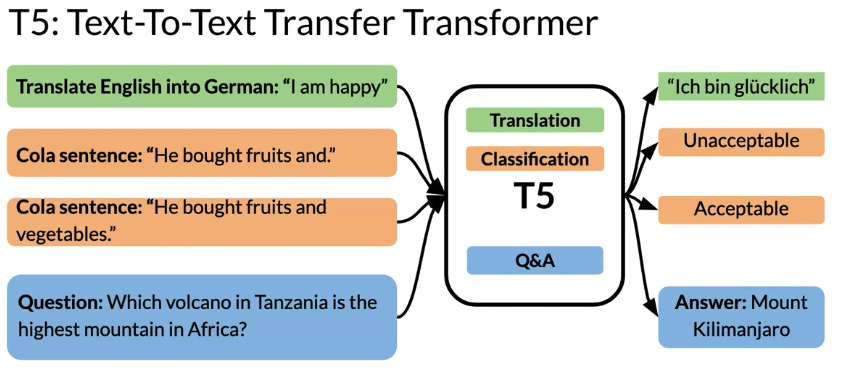


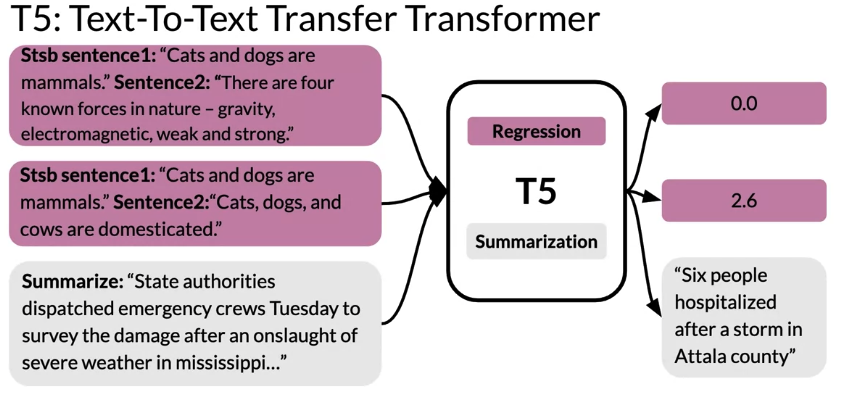


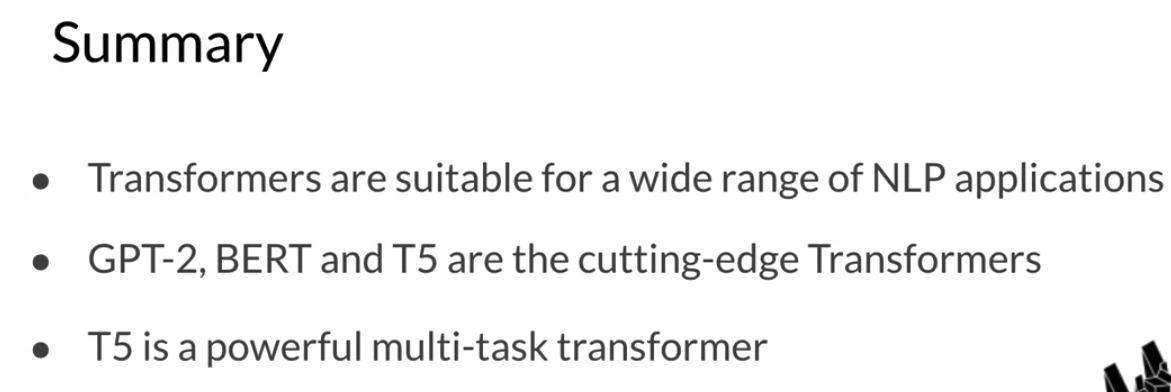


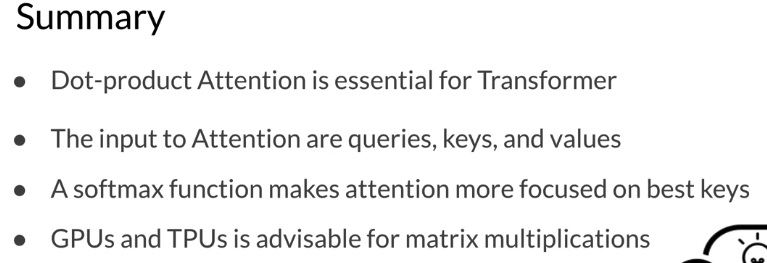


**Week 2 - Text Summarization**

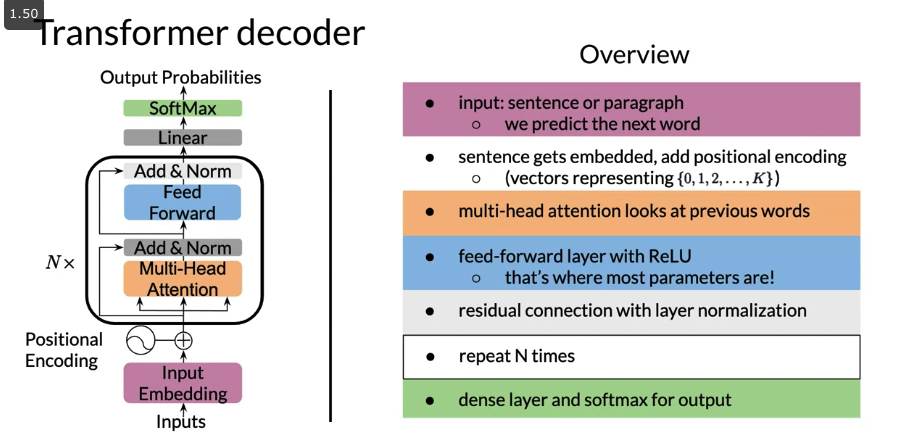


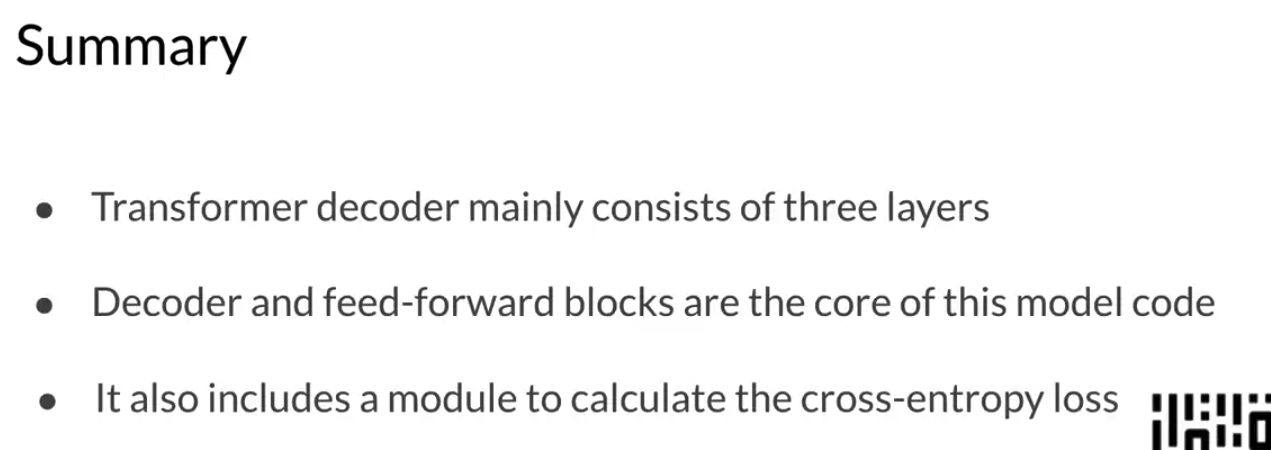








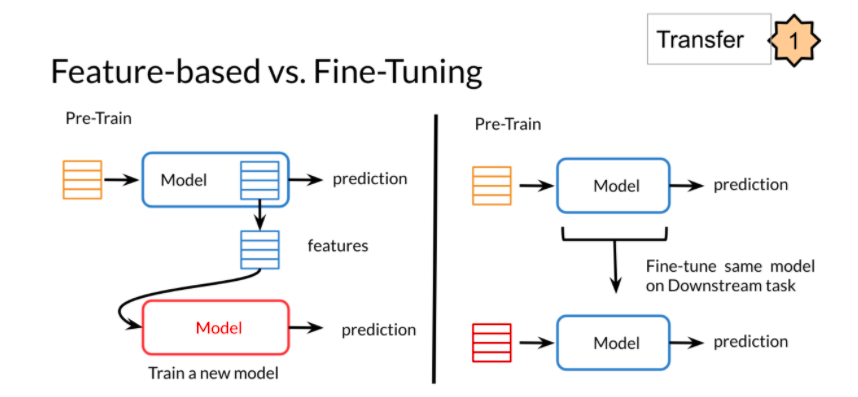


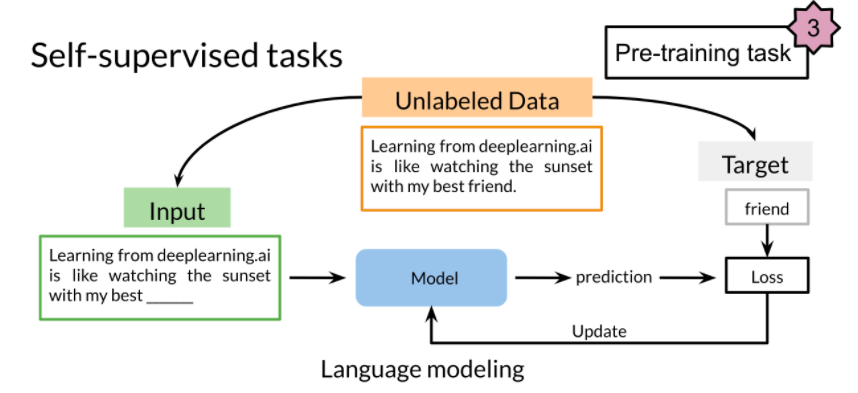


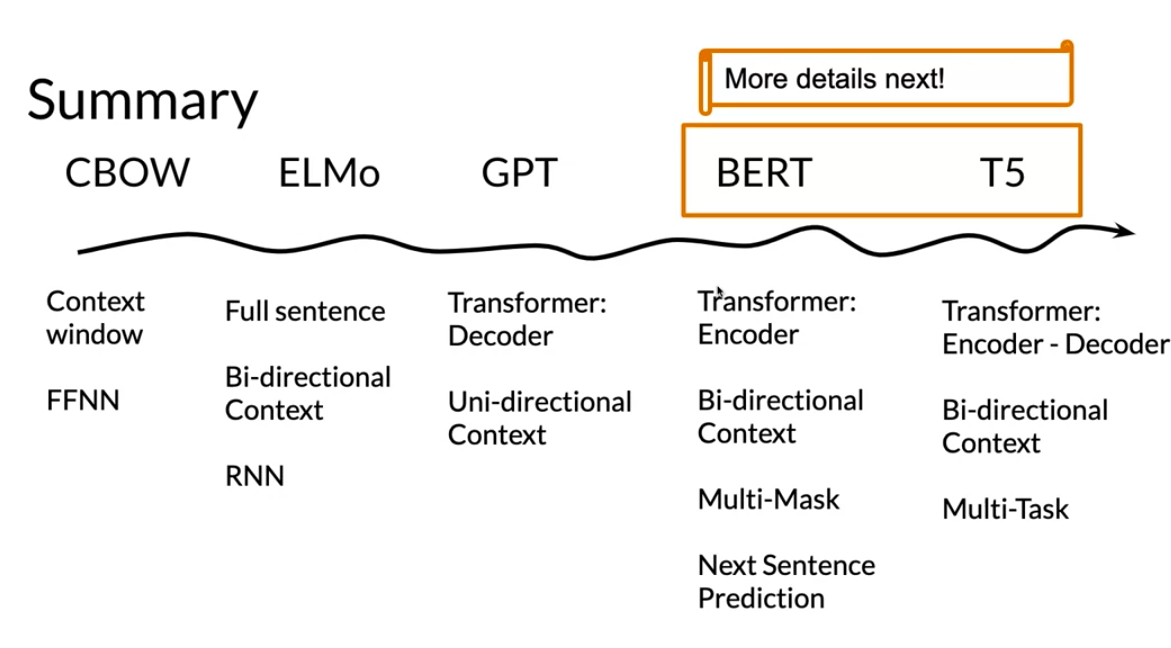
**Week 3 - Text Summarization**

There are three main advantages to transfer learning:

* Reduce training time
* Improve predictions
* Allows you to use smaller datasets

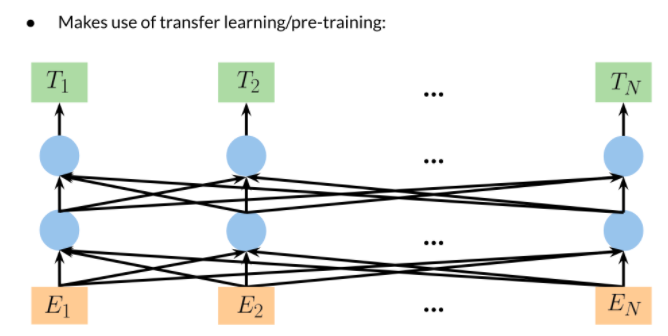






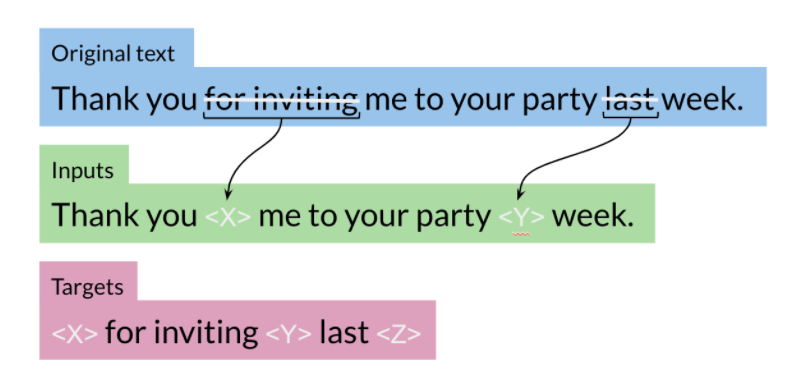
BERT

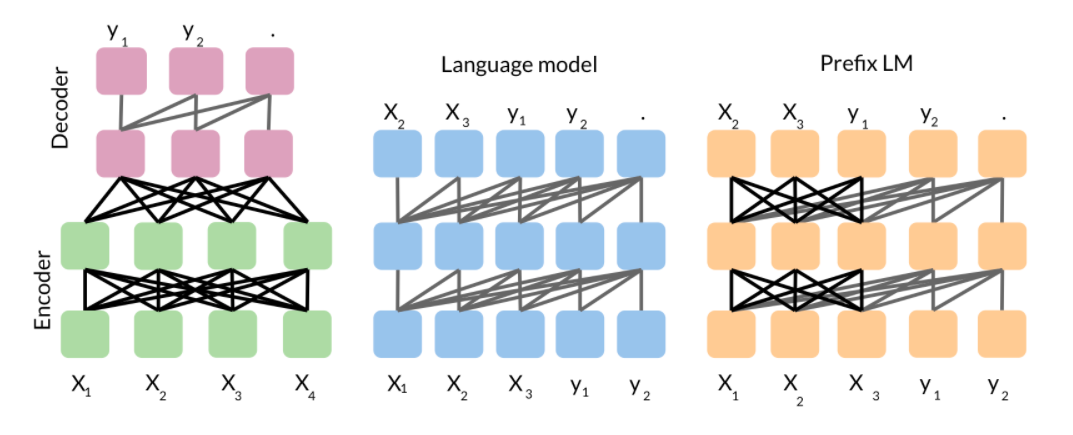
* Choose 15% of the tokens at random: mask them 80% of the time, replace them with a random token 10% of the time, or keep as is 10% of the time.
* There could be multiple masked spans in a sentence
* Next sentence prediction is also used when pre-training.





Transformer T5

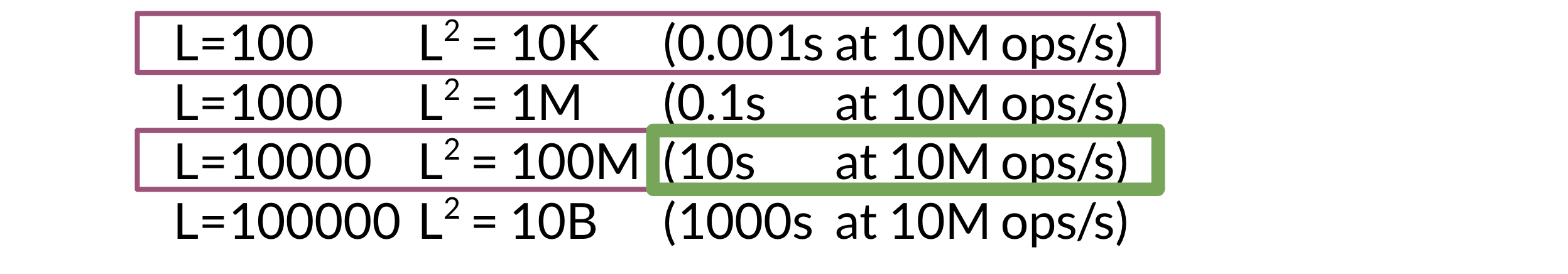




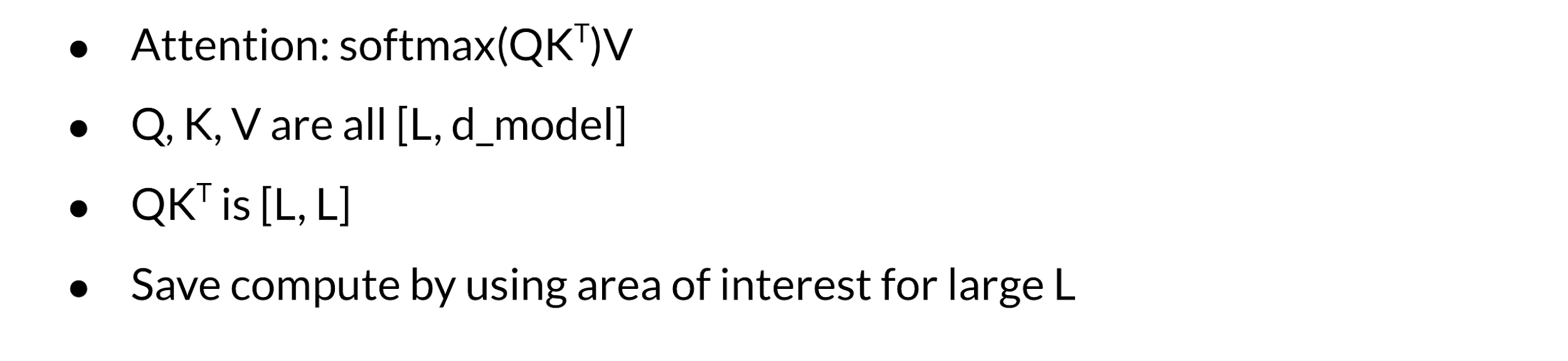
**Week 4 - Chatbot**

**Transformer Complexity**

One of the biggest issues with the transformers is that it takes time and a lot of memory when training. Concretely here are the numbers. If you have a sequence of length L, then:



So if you have N layers, that means your model will take N times more time to complete. As L gets larger, the time quickly increases.



When you are handling long sequences, you usually don't need to consider all L positions. You can just focus on an area of interest instead. For example, when translating a long text from one language to another, you don't need to consider every word at once. You can instead focus on a single word being translated, and those immediately around it, by using attention.

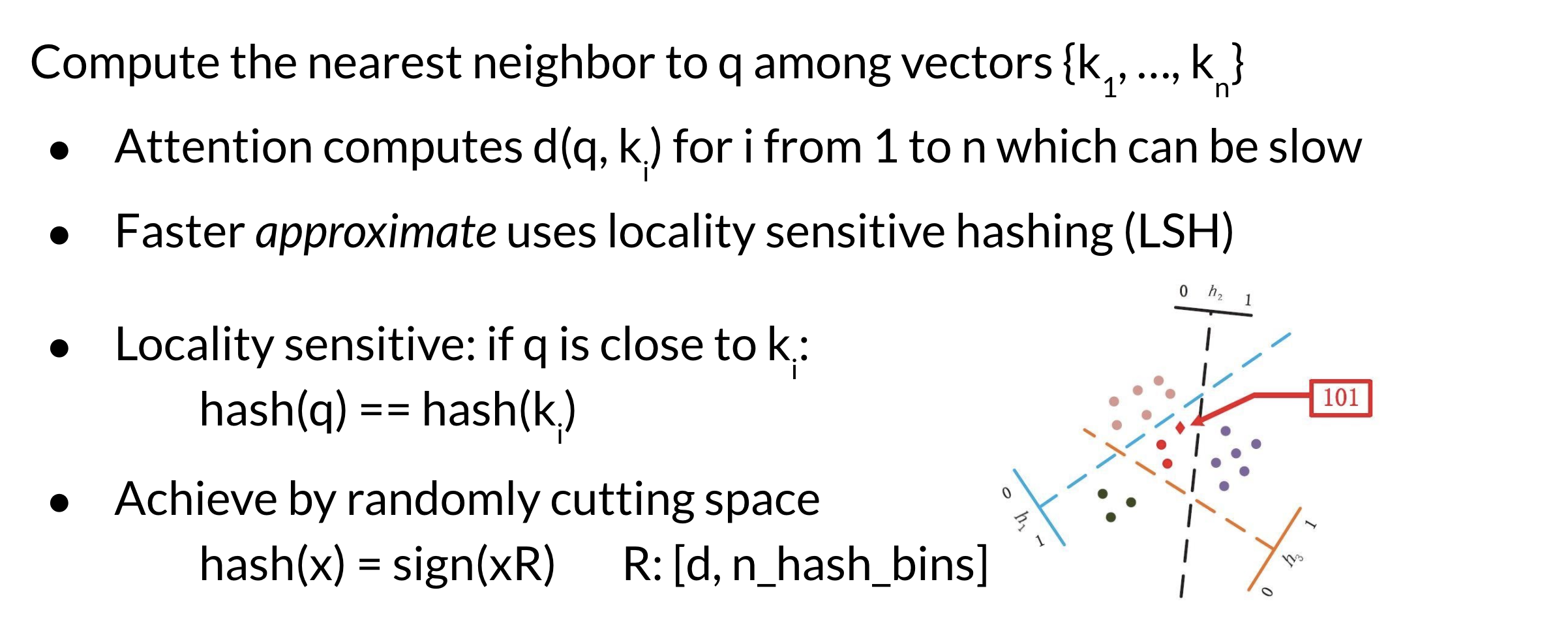
To overcome the memory requirements you can recompute the activations. As long as you do it efficiently, you will be able to save a good amount of time and memory. You will learn this week how to do it. Instead of storing N layers, you will be able to recompute them when doing the back-propagation. That combined with local attention, will give you a much faster model that works at the same level as the transformer you learned about last week.

**LSH Attention**

In Course 1, I explained how locality sensitive hashing (LSH) works. You learned about:

* KNN
* Hash Tables and Hash Functions
* Locality Sensitive Hashing
* Multiple Planes

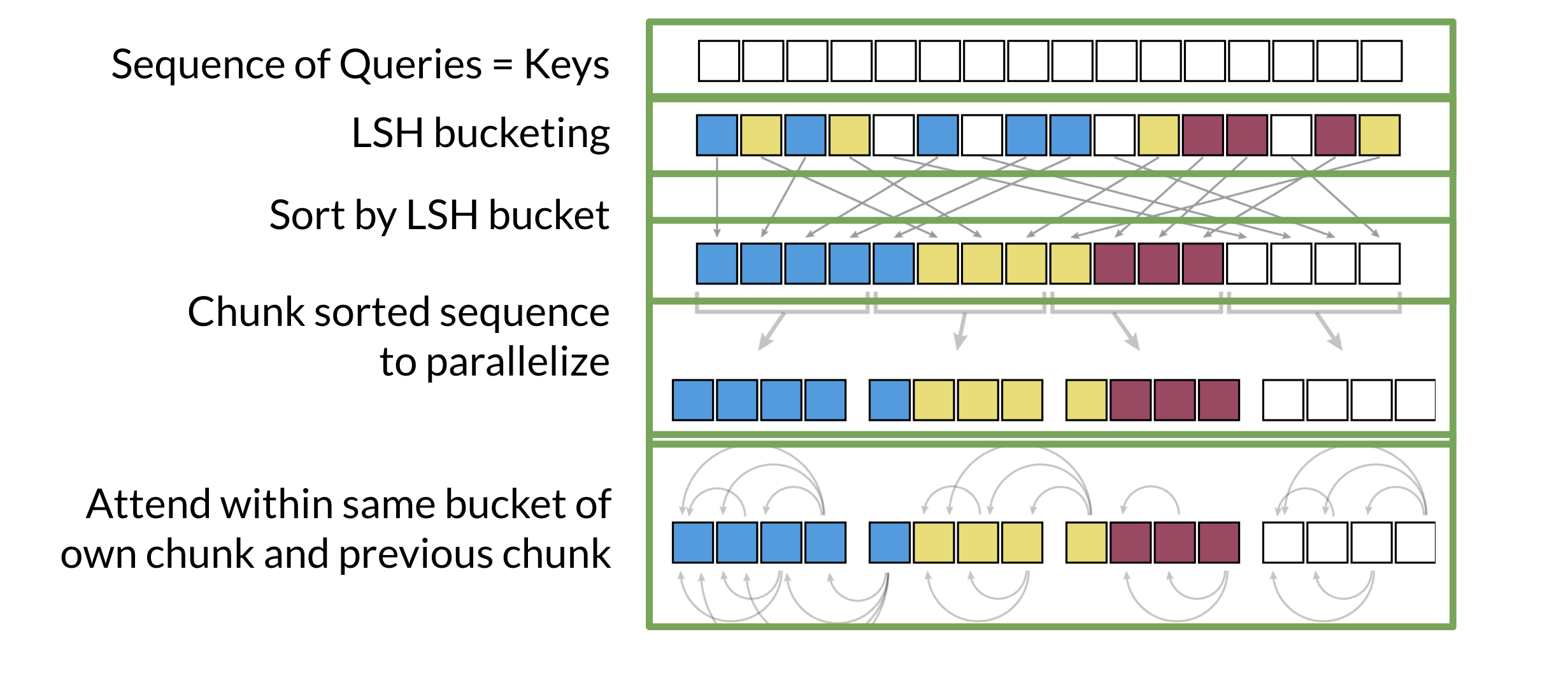
Here are the steps you follow to compute LSH given some vectors. The vectors could correspond to the transformed word embedding that your transformer outputs. Attention is used to try which query (q) and key (k) are the most similar.



To do so, you hash q and the keys. This will put similar vectors in the same bucket that you can use. The drawing above shows the lines that separate the buckets. Those could be seen as the planes.



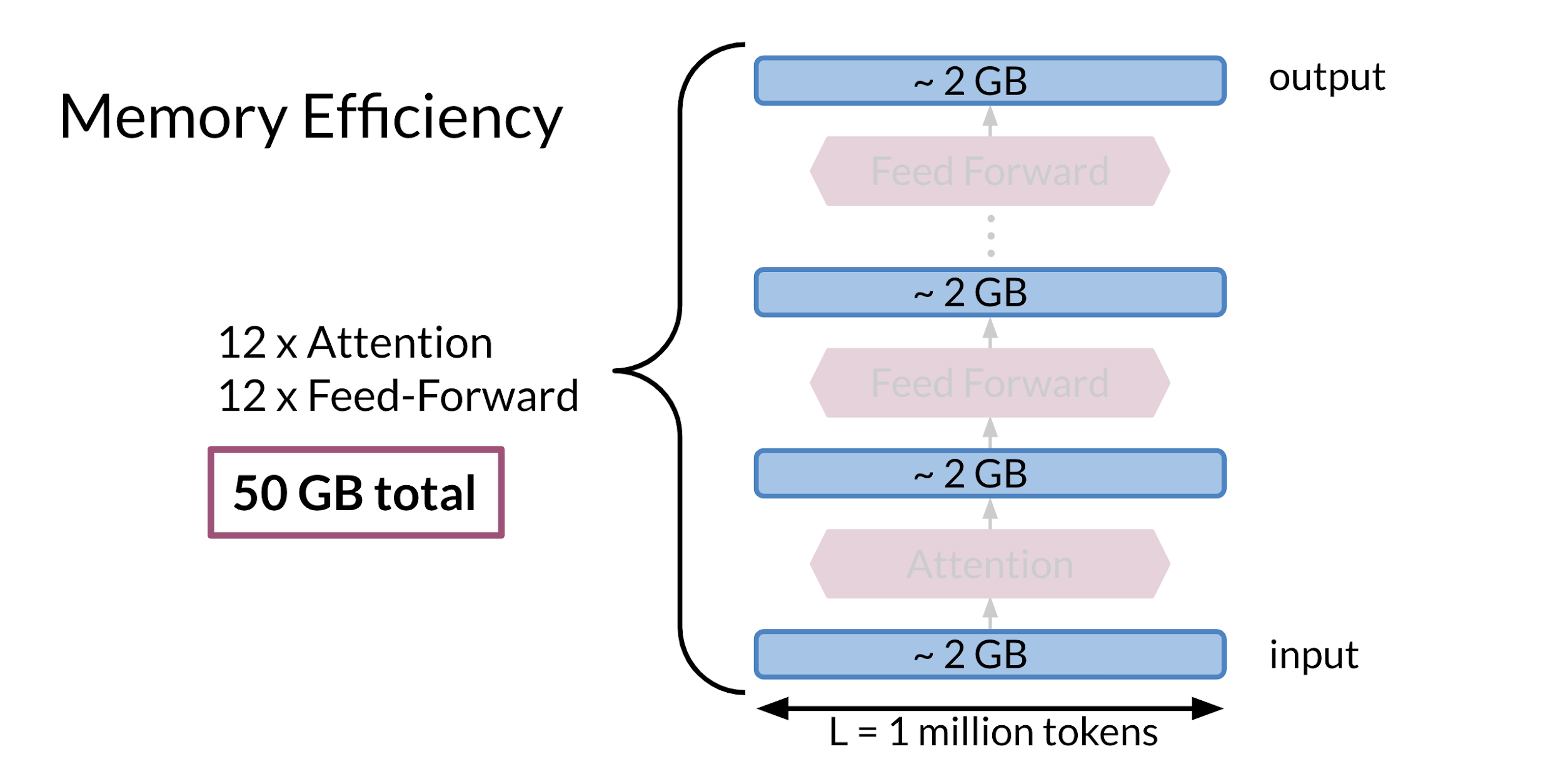
Once you hash Q and K you will then compute standard attention on the bins that you have created. You will repeat the same process several times to increase the probability of having the same key in the same bin as the query.



Given the sequence of queries and keys, you hash them into buckets. Check out Course 1 Week 4 for a review of the hashing. You will then sort them by bucket. You split the buckets into chunks (this is a technical detail for parallel computing purposes). You then compute the attention within the same bucket of the chunk you are looking at and the previous chunk. Why do you need to look at the previous chunk?

**Motivation for Reversible Layers: Memory!**

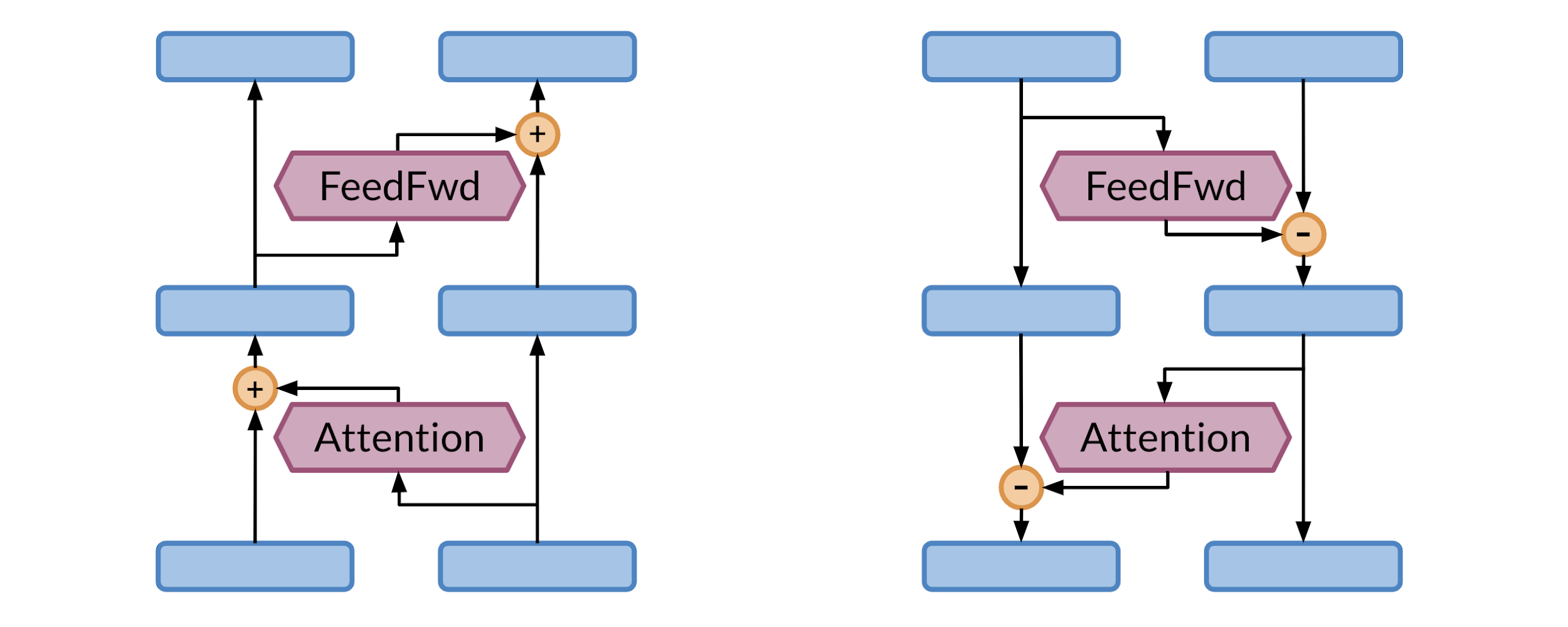
Every time you run a forward propagation, you need to compute the back propagation to update the weights. The biggest issue with doing this is that you have to store the weights to be able to compute the back-prop. With these very large models, that could be a lot of memory.



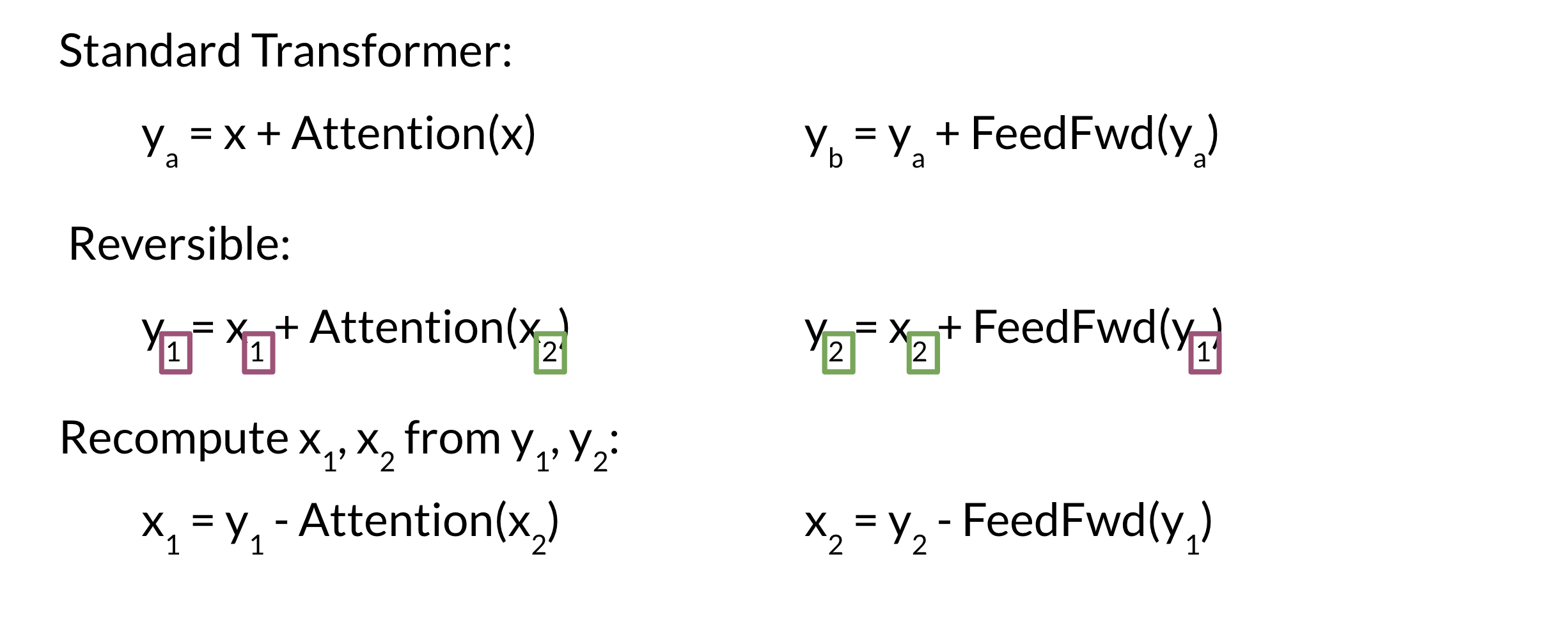
For example in the model above it requires 2GB to compute the Attention and 2GB for the feed forward. You have 12 layers for attention and 12 layers for the feedforward. That is equal to 12 \* 2 + 12\*2 + 2 (for the input) = 50 GB. That is a lot of memory. In the next video you will learn how to solve such problems.

**Reversible Residual Layers**

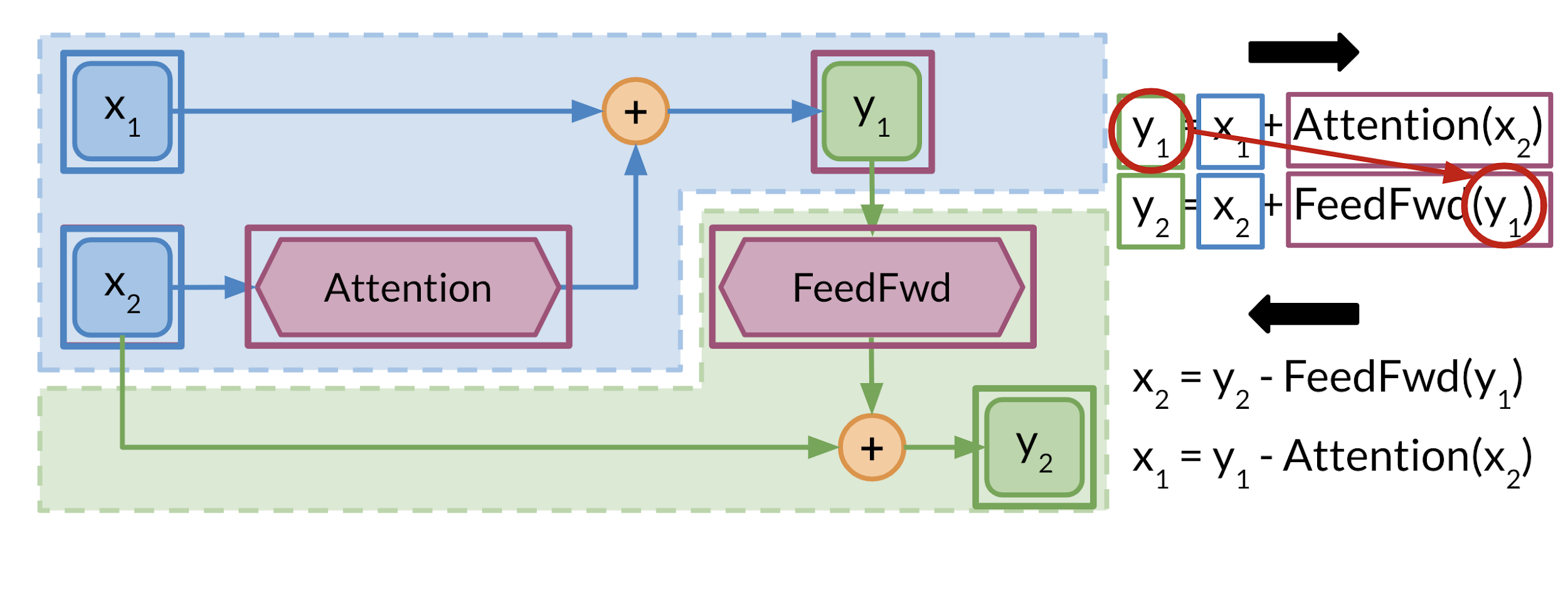
Reversible residual layers allow you to reconstruct the forward layer from the end of the network. Usually you have two similar branches in the network that you use to compute the network.



In the left picture, you have the forward propagation. One side of the network is used as input and the other is used for the attention. On the left side, the same thing is happening but in the opposite direction.



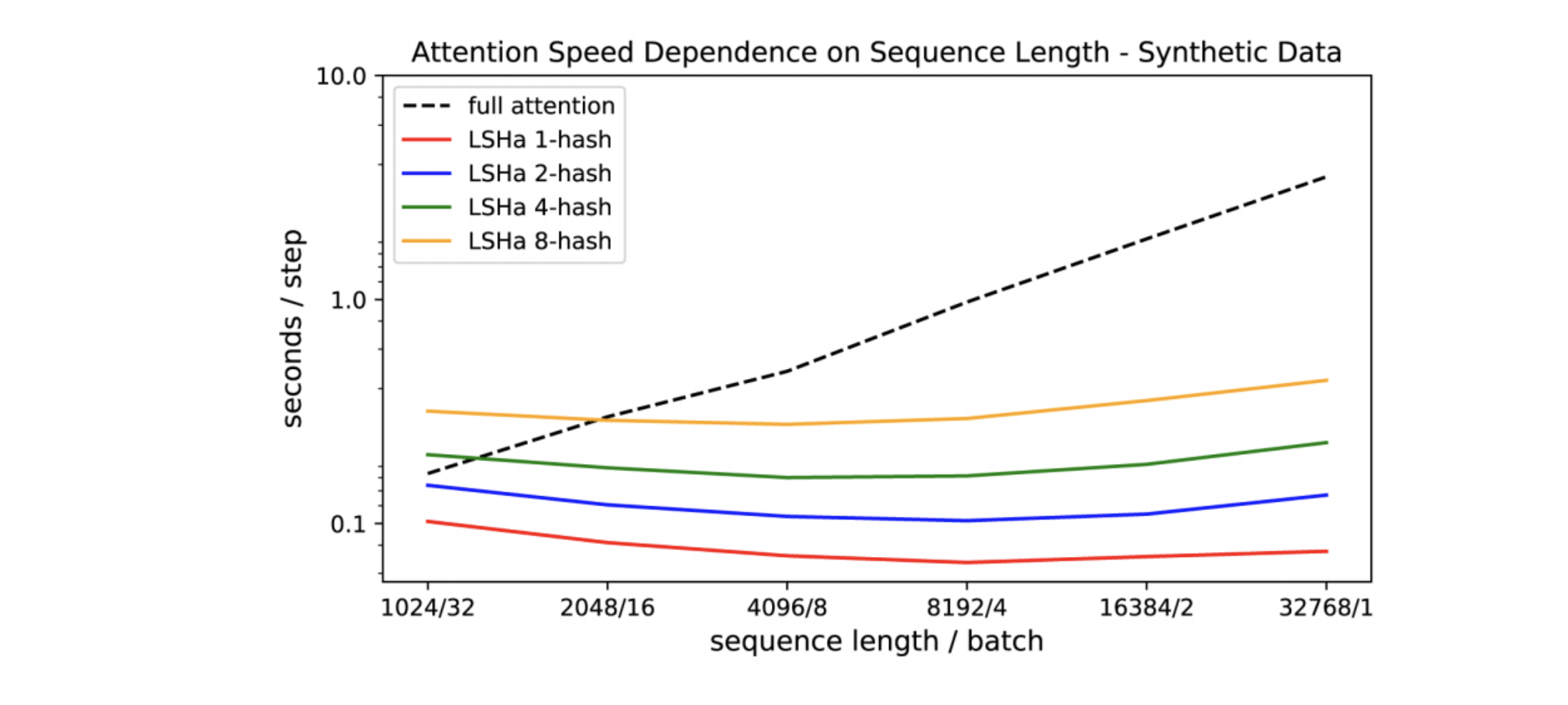
Take a few minutes and try to understand the equations above. You basically make use of the two branches of the network. When coming back for the back propagation, you only need the y's to compute x\_2 and then you can use x\_2 along with y\_1 to compute x\_1. Pretty neat! Now you don't have to store the weights, because you can just compute them from scratch. This image shows you a visualization of what is happening.



# Reformer

The reformer allows you to fit up to 1 million tokens on a single 16 gigabyte GPU. It is designed to handle context windows of up to 1 million words. It combines two techniques to solve the problems of attention and memory allocation which are bottlenecks for the transformer networks.

Reformer uses locality sensitive hashing, which you saw earlier in this specialization, to reduce the complexity of attending over long sequences. It also uses reversible residual layers to more efficiently use the memory available. In the picture below you can see how the reformer performs when compared to a normal full-attention model.

****