That is an excellent, deeper question. To truly understand the Transformer, we must trace a sentence's journey layer by layer.
Let's use the sentence: **"The animal did not cross the street because it was too tired."**
The goal of the Transformer's **Encoder** is to produce a vector representation for each word that incorporates its meaning *and* its relationship with every other word in the sentence (crucially, figuring out what "**it**" refers to).

🎨 Step 1: Input Preparation (The Embedding Layer)

The input sentence is converted into a sequence of vectors.

1. **Tokenization:** The sentence is broken down into tokens (e.g., words and punctuation): [The, animal, did, not, cross, the, street, because, it, was, too, tired, .]
2. **Input Embedding:** Each token is mapped to a vector (e.g., $d_{\text{model}}=512$ dimensions). These vectors capture the basic, context-free meaning of the word (e.g., the vector for "street" is similar to "road").
3. **Positional Encoding:** Since the core attention mechanism has no memory of order, a unique **Positional Encoding** vector is added to the word embedding vector.

Result: We now have a matrix where each row is a vector of size d_{model}, representing the word **plus its position** in the sentence. This matrix is the input to the first Encoder layer.

🏗️ Step 2: The Encoder Layer (Processing Context)

The encoder is a stack of identical layers. We will trace the input through a single layer, which is then repeated for the remaining layers (typically 6 in the original paper).

2.1. Sub-Layer 1: Multi-Head Self-Attention (The Core)

This is the most crucial step, where the model figures out word relationships.

- **Query, Key, Value Projections:** The input vector matrix is multiplied by three different learned weight matrices (W^Q, W^K, W^V) to create three new vector matrices: **Query (Q), Key (K),** and **Value (V)** for the entire sentence.
    - Q (The question): What am I looking for? (e.g., for the word "it")
    - K (The index): What information do I contain? (e.g., for "animal" and "street")
    - V (The content): What information should I contribute?
- **Attention Calculation (The Math):**
    - The **Attention Score** is computed by taking the dot product of the Query vector for the current word (e.g., "**it**") with the Key vectors of *all* other words. This score measures the **relevance** of every word to "**it**."
    - The scores are normalized using **Softmax** to create **Attention Weights** (probabilities that sum to 1).
        - *Example:* The word "**it**" might receive a high attention weight from "**animal**"

(e.g., 0.65), a moderate weight from "**street**" (e.g., 0.30), and low weights from everything else.
    - The Attention Weights are then multiplied by the Value vector of each word and summed up.

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

- **Multi-Head:** This process is performed *simultaneously* by multiple "heads" (e.g., 8 heads). Each head learns to focus on different types of relationships (e.g., one head focuses on grammatical dependencies, another on semantic relationships). The results from all heads are concatenated and linearly projected back into a single d_{model} vector.
- **Add & Norm:** The output of the Attention is added to the original input (a **Residual Connection**) and then passed through **Layer Normalization**.

Result: The vector for "**it**" is now a highly rich, **contextualized** vector that primarily contains information mixed from "**animal**" and "**street**," reflecting the ambiguity, which the model attempts to resolve in later layers.

2.2. Sub-Layer 2: Position-wise Feed-Forward Network (FFN)

- The output from Sub-Layer 1 is passed through a simple two-layer feed-forward network. **Crucially, this is applied identically and independently to the vector of *each word* in the sequence.**
- It serves to introduce non-linearity and further process the contextual information derived in the self-attention step.
- **Add & Norm:** This output is again added to the input of this sub-layer (Residual Connection) and normalized.

Result: The sentence matrix has now been processed through a full Encoder layer, where the meaning of every word has been updated with information from the entire sequence.

🧱 Step 3: Stacking the Encoders

The output of the first Encoder layer (the new, contextualized sentence matrix) becomes the input for the second Encoder layer, and so on (typically 6 times). With each layer, the contextualization becomes deeper and more abstract.

💡 Step 4: The Decoder Output (Generation)

The final output of the Encoder stack is passed to the Decoder. The Decoder generates the output sequence (e.g., a translation) one word at a time.

1. **Decoder Masked Self-Attention:** It performs Self-Attention on the words it has *already* generated, but is **masked** to prevent it from "cheating" and looking at future words in the sequence it's trying to predict.

2. **Encoder-Decoder Attention (Cross-Attention):** This is where the decoder looks at the **Encoder's final output**. The Query (Q) comes from the decoder's previous output, and the Key (K) and Value (V) come from the encoder's output. This allows the decoder to find the **most relevant parts of the source sentence** for generating the next word.
3. **Decoder FFN & Final Layer:** The result passes through an FFN and a final linear/Softmax layer, which outputs a probability distribution over the entire vocabulary (e.g., the word "tired" is the most likely next word). The highest probability word is selected and becomes the new input for the next time step.

Would you like me to focus on the **Multi-Head Attention** in more detail, perhaps with a visual representation of the attention weights?