

The background is a solid yellow color. It features several geometric shapes: a large cyan circle on the right side, and four black-outlined triangles of various sizes and orientations scattered around the circle and text. The text is in a bold, sans-serif font.

GRUPO DE ESTUDIO

NLP CON TRANSFORMERS

-CAPÍTULO 1

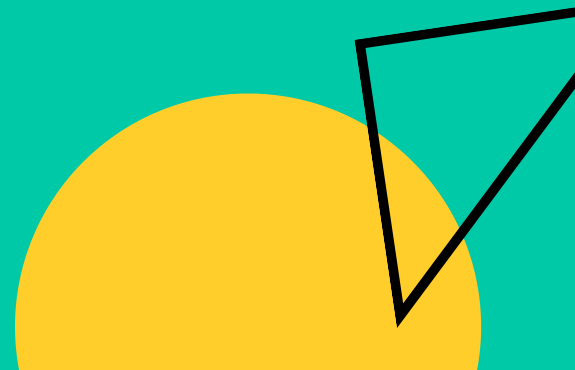
SÁBADO 9/4

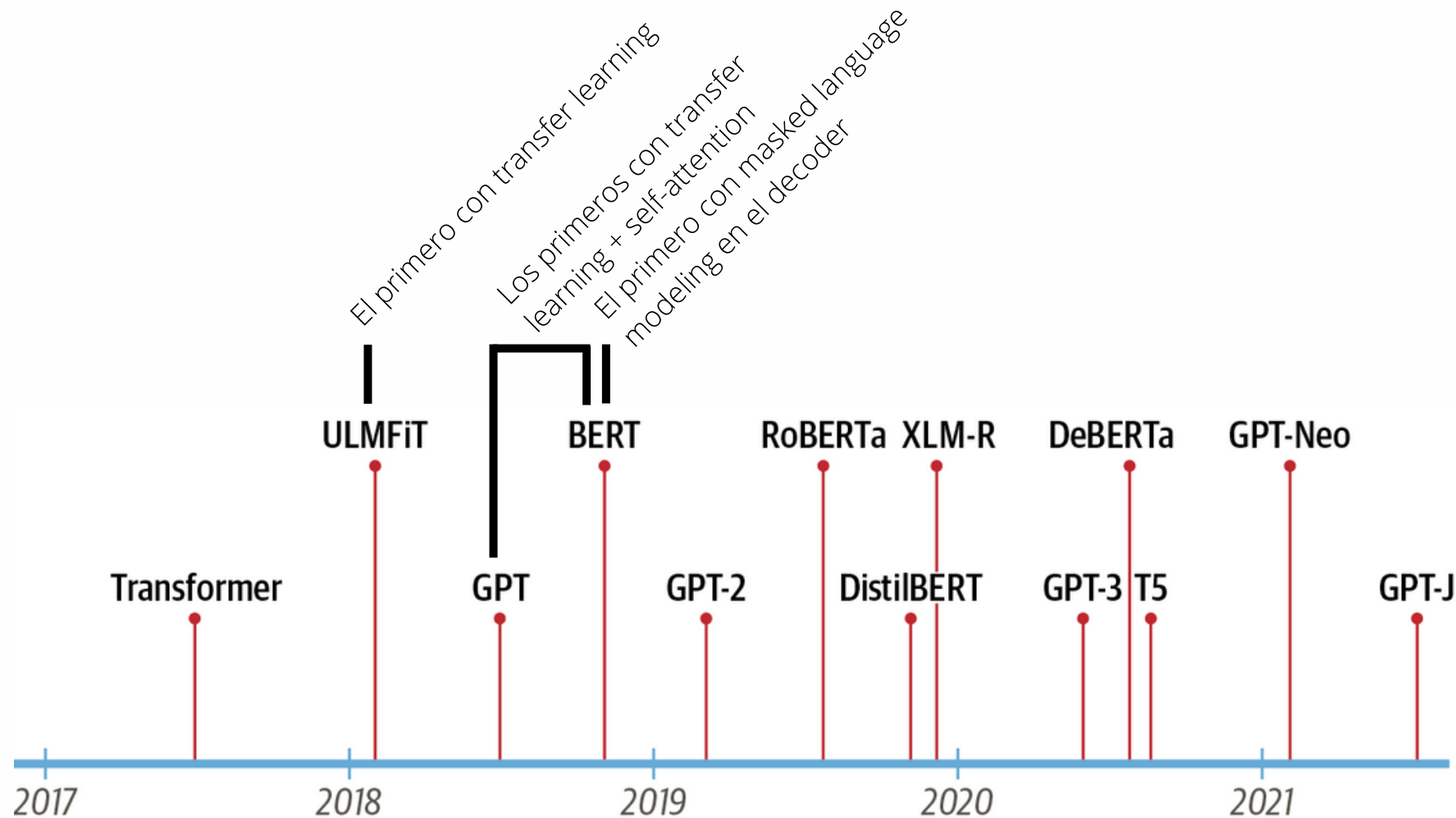


'HELLO TRANSFORMERS'

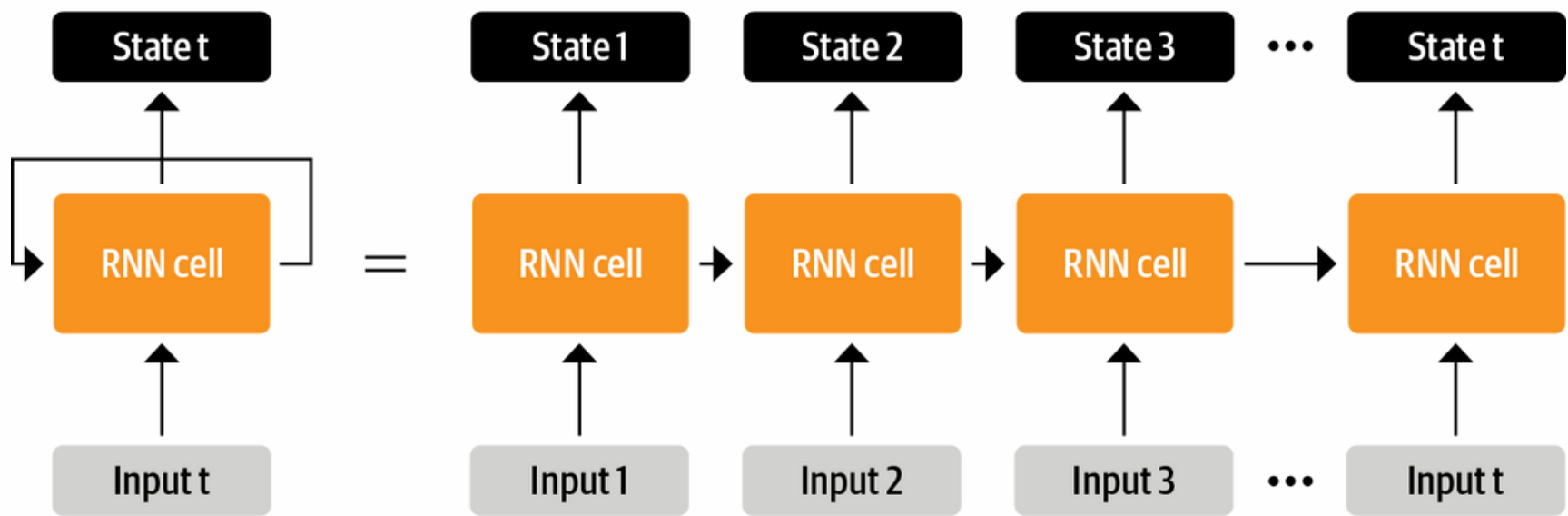
> Transformers es un **tipo de arquitectura de redes neuronales** que rompió el paradigma existente hasta 2017, las redes neuronales recurrentes.

> Seguramente lo utilizaste sin saberlo: **Google utiliza BERT** en su motor de búsqueda, **Github Copilot utiliza TabNine**, entre otros.





Timeline de Transformers: la historia de los modelos de lenguaje secuenciales



Arquitectura de una RNN (Recurrent Neuronal Network)



¿QUÉ CAMBIÓ TRANSFORMERS?

01

ENCODER-DECODER
FRAMEWORK

02

MECANISMOS DE
ATENCIÓN

03

ENSEÑANZA
X TRANSFERENCIA

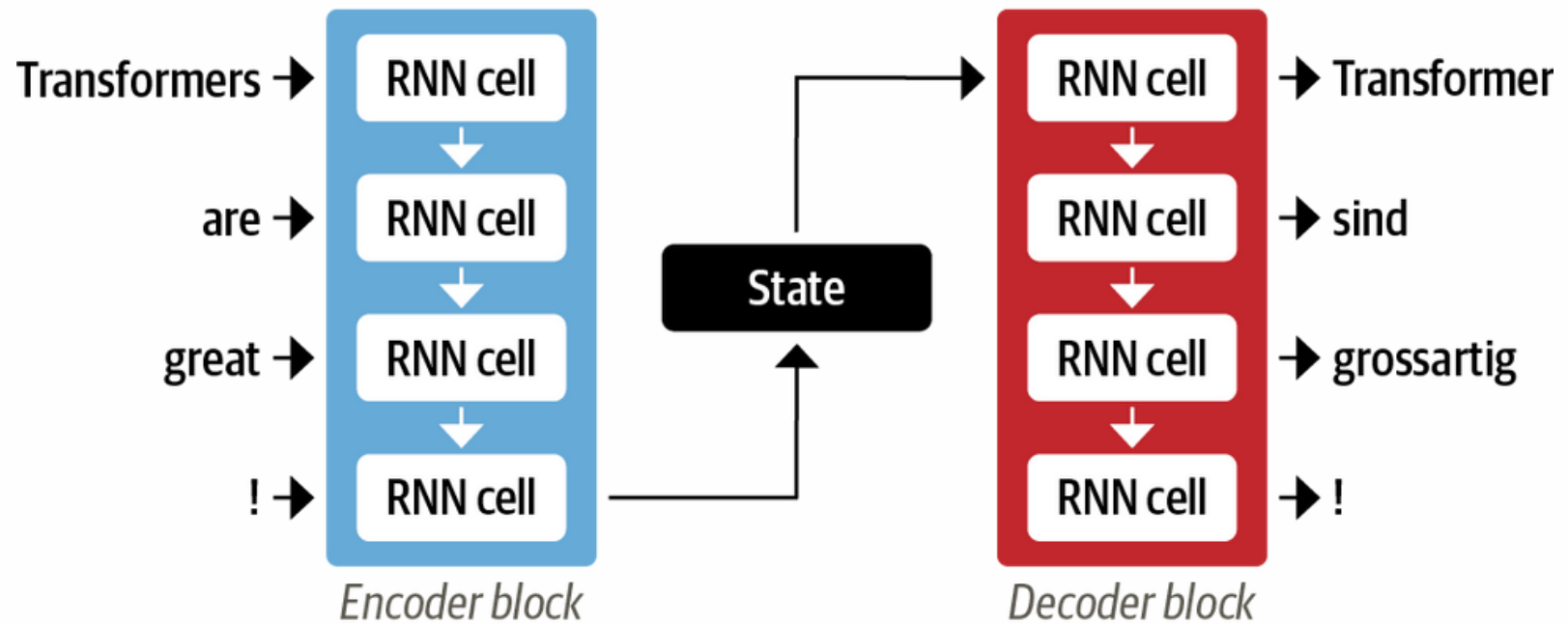


01

ENCODER- DECORDER

FRAMEWORK

- > La arquitectura Transformers funciona con una serie de capas de **codificación y decodificación**.
- > Esa serie de procesos la hacen más **performante** que los modelos anteriores, ya que tiene una ventana de **regresión infinita** y los procesos se realizan **en paralelo**.
- > En esto tienen que ver mucho los **mecanismos de atención** que vamos a repasar más adelante.



Arquitectura encoder-decoder de una RNN

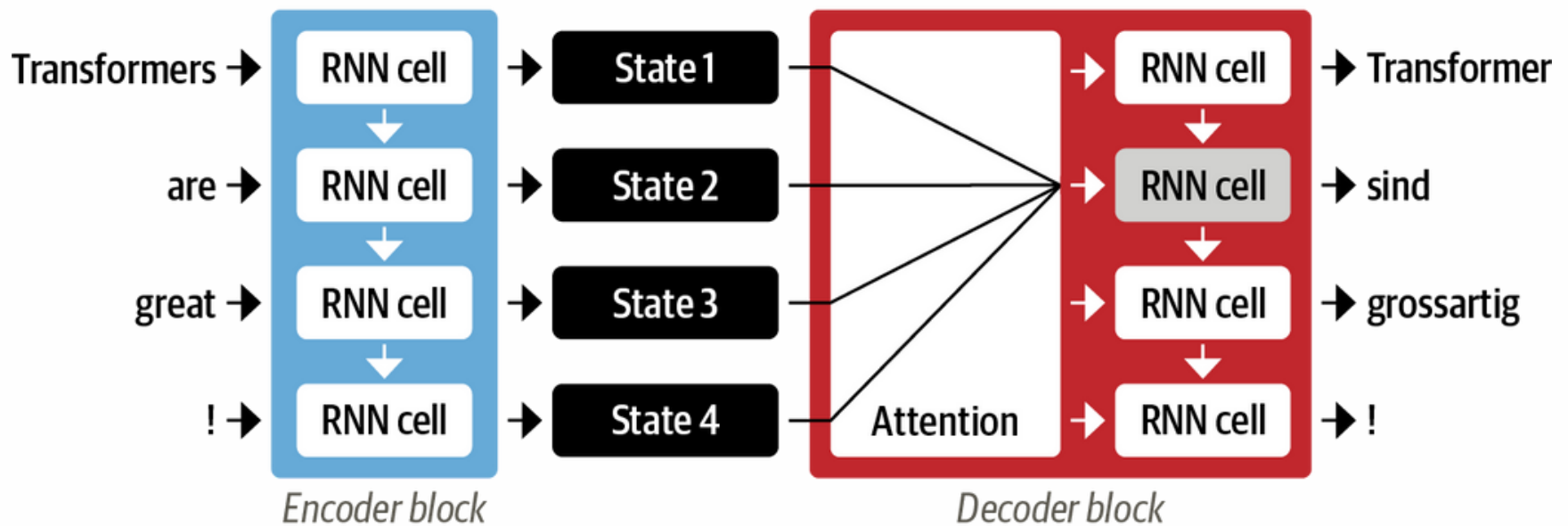
ENCODER- DECORDER

FRAMEWORK

> El **encoder** toma las palabras de un texto y las **vectoriza**, es decir las transforma en una secuencia de números del 0 al 1.

> Esta representación numérica guarda información sobre el sentido de la palabra en la oración gracias a los **mecanismos de atención** que vamos a ver más adelante y es utilizada como input para el **decoder** hasta que se complete la secuencia con un **token de final de secuencia**.

01



Arquitectura encoder-decoder: RNN cells + una capa de mecanismos de atención. La arquitectura transformer reemplazó la capa de RNN cells del encoder y del decoder por una capa de self-attention.

MECANISMOS DE ATENCIÓN

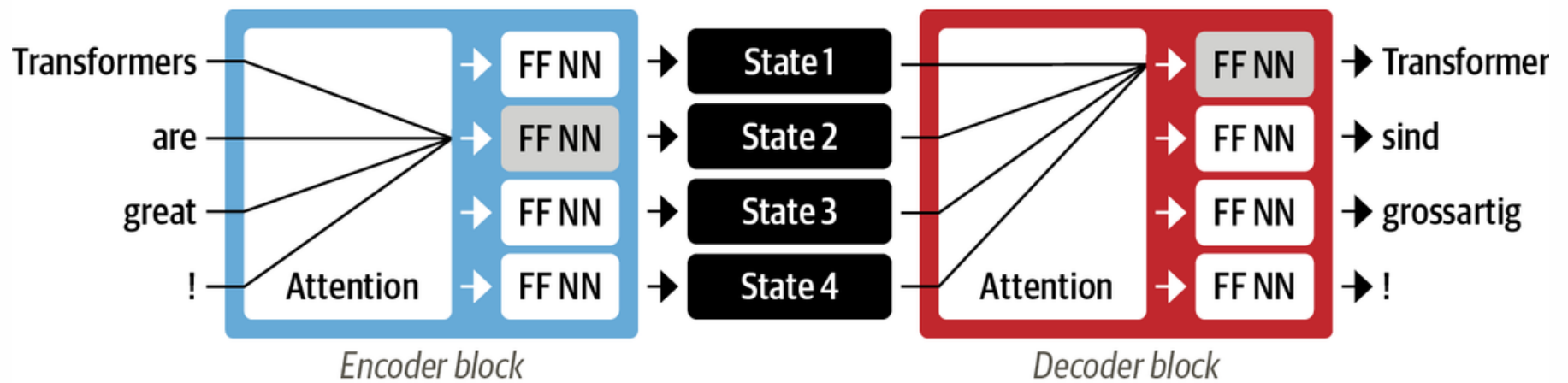
> Los **mecanismos de atención** son lo que hace la diferencia para la estructura Transformer.

> En vez de generarse un **hidden state** x secuencia, se generan tantos hidden states como step (cantidad de palabras) haya.

> Pero generar todos esos states crea un input muy pesado para el decoder, por eso el decoder asigna un peso a cada state, para decidir cuál priorizar (a cuál prestarle atención) en base a los states anteriores.

> En vez de generarse un **hidden state** x secuencia, se generan tantos hidden states como step (cantidad de palabras) haya.

02



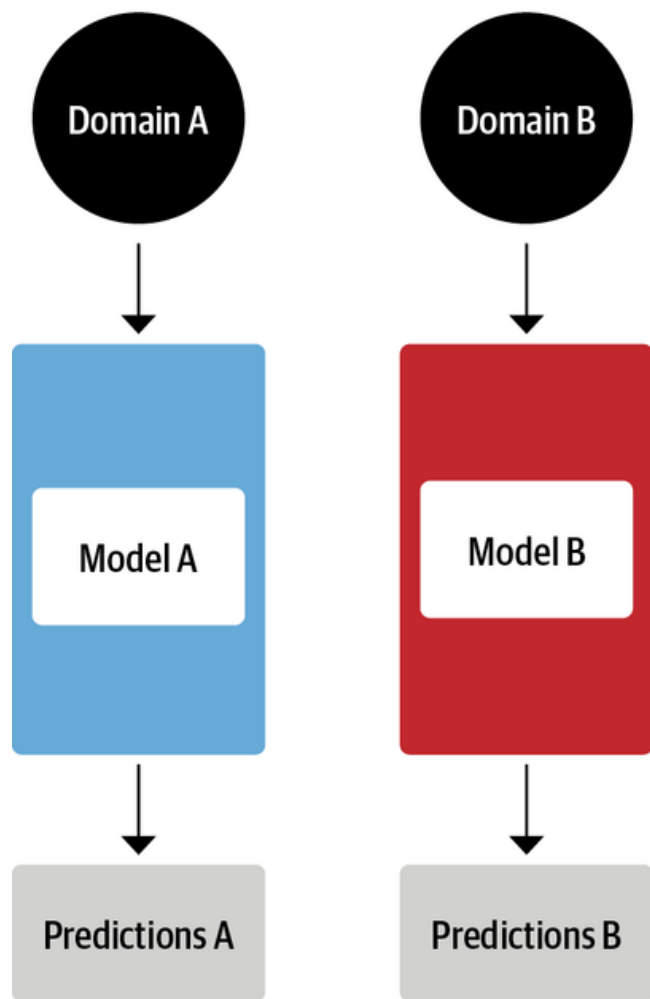
Arquitectura decoder-transformer

TRANSFER LEARNING

> Se considera que un modelo de lenguaje tiene un cuerpo (body) y una cabeza (head). Esta última representa la parte de red que se encarga de tareas más específicas.

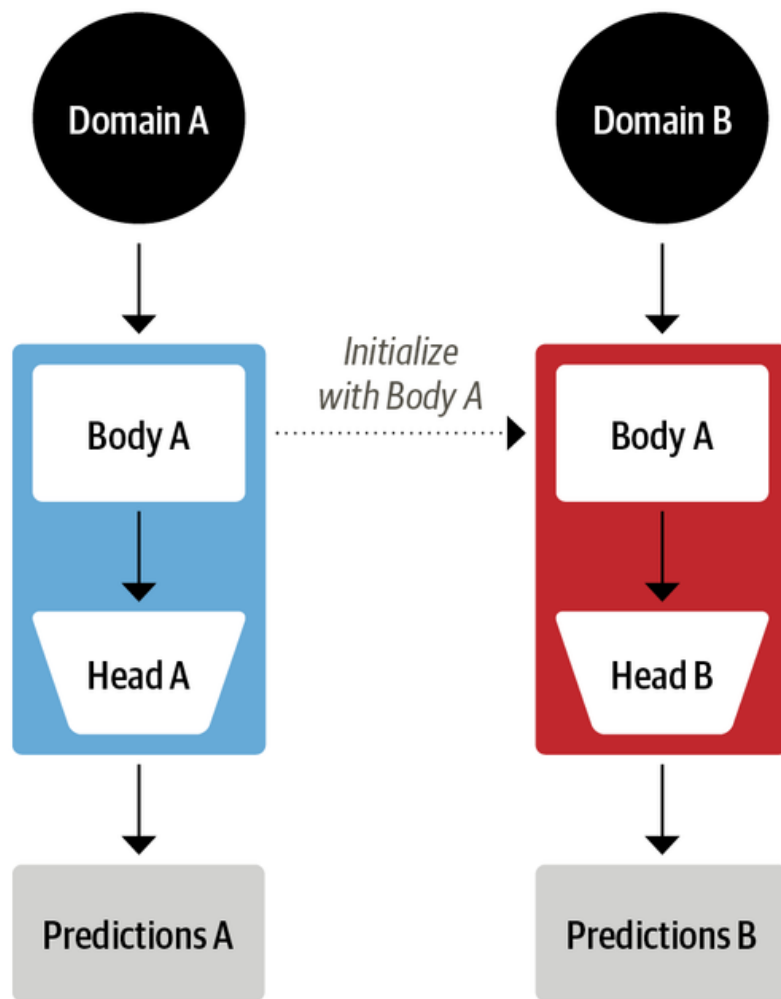
03

*Training and evaluation on
the same task/domain*



Supervised learning

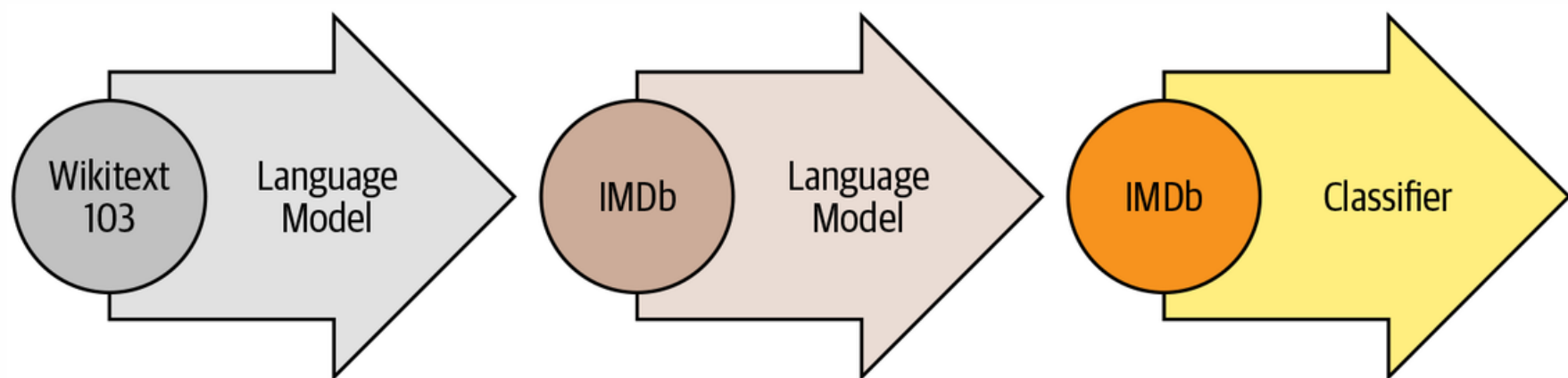
*Extract knowledge from source task,
and apply to different target task*



Transfer learning

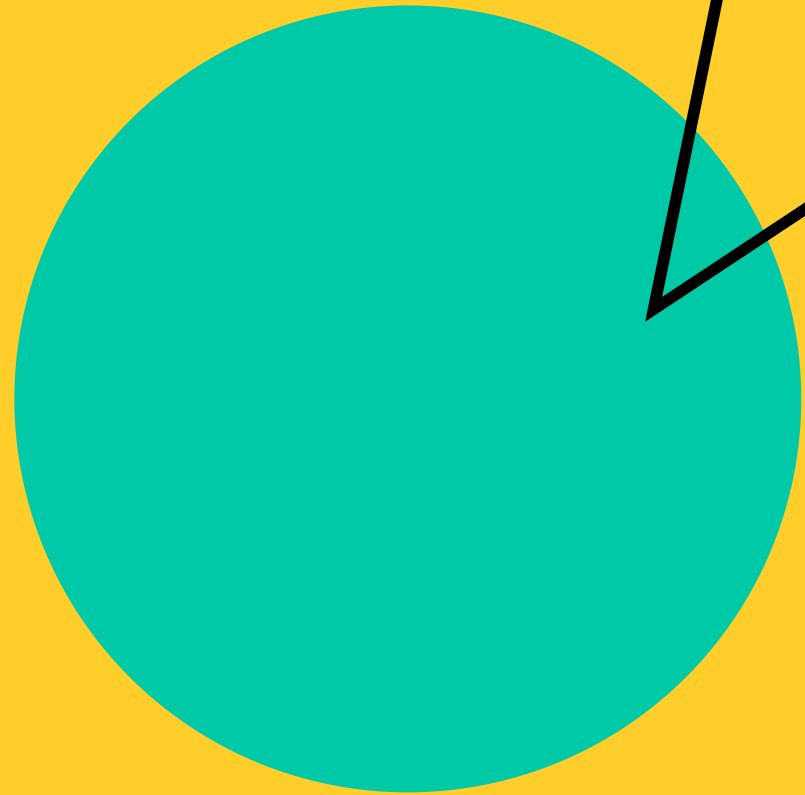
TRANSFER LEARNING

- **Pretraining:** predecir la siguiente palabra en base a las palabras anteriores -> **language modeling**
- **Domain adaptation:** una vez que el modelo está pre-entrenado con un corpus extenso, el paso siguiente es adaptarlo a un dominio específico **equilibrando los pesos** del modelo
- **Fine-tuning:** en este paso, se agrega una **capa de clasificación**, por ejemplo, de análisis de sentimientos en las reviews de una lista de películas. El fine-tuning es mucho más performante que entrenar un modelo como classifier de cero.



Ejemplo de los tres pasos de una red ULMFiT: pretraining, domain adaptation y fine-tuning

**¿UNA IDEA
GENERAL?**





TRANSFORMERS ES...


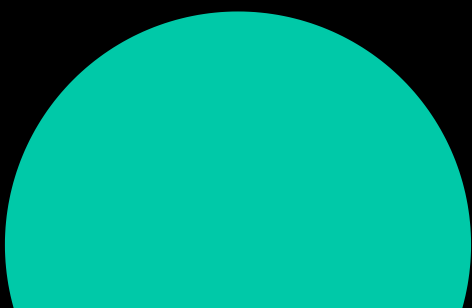
+ESCALABLE

> Poder utilizar modelos pre-entrenados hace más simple la construcción de modelos de tareas más específicas

+PERFORMANTE

> La arquitectura Transformer es más eficiente porque no es ni secuencial, ni recurrente.

> Un framework de transfer learning también hace más eficiente al modelo.





“

Con el lanzamiento de **Hugging Face Transformers**, una API que unifica más de 50 arquitecturas y tres frameworks interoperables (PyTorch, Tensorflow, JAX) se avanzó en la investigación de estos modelos, haciendo más simple integrarlos en muchas aplicaciones de la actualidad



**-NLP WITH
TRANSFORMERS**

HUGGING FACE TRANSFORMERS

Pasos para generar un modelo

- > Implementarla arquitectura del modelo (en PyTorch o Tensorflow)
- > Cargar los pesos del pre-entrenamiento (si se puede) de un servidor
- > Pre-procesar los inputs, pasarlos por el modelo y aplicar una tarea específica luego del proceso
- > Implementar data loaders y definir las loss functions y optimizadores para entrenar el modelo



CASOS – PIPELINES

- > Clasificación de textos (sentiment analysis)
- > Name entity recognition (NER)
- > Respuesta de preguntas
- > Resumen de textos
- > Traducciones
- > Generación de textos

ECOSISTEMA HUGGING FACE

- > Compuesto por las librerías + Hub
- > Las **librerías** aportan el código
- > El **Hub** aporta los pesos pre-entrenados, datasets, métricas de evaluación, entre otros. Está compuesto por más de 10.000 modelos de lenguaje.