Deep Learning

Prof. Dr. Vladimir Costa de Alencar
www.valencar.com
LANA / UEPB

LANA
DATA ANALYSIS LABORATORY

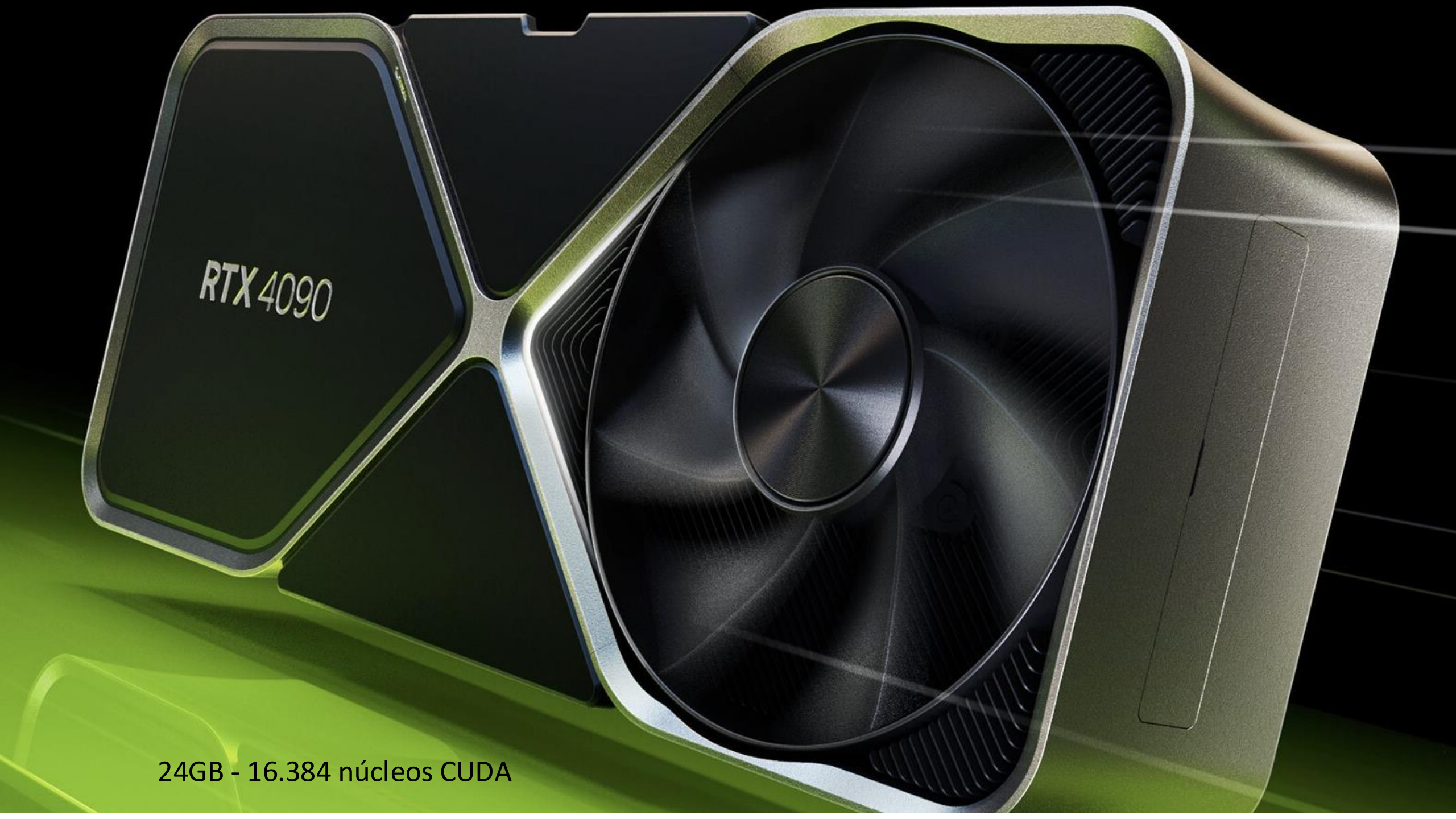
Dados...

Flight Simulator

Game

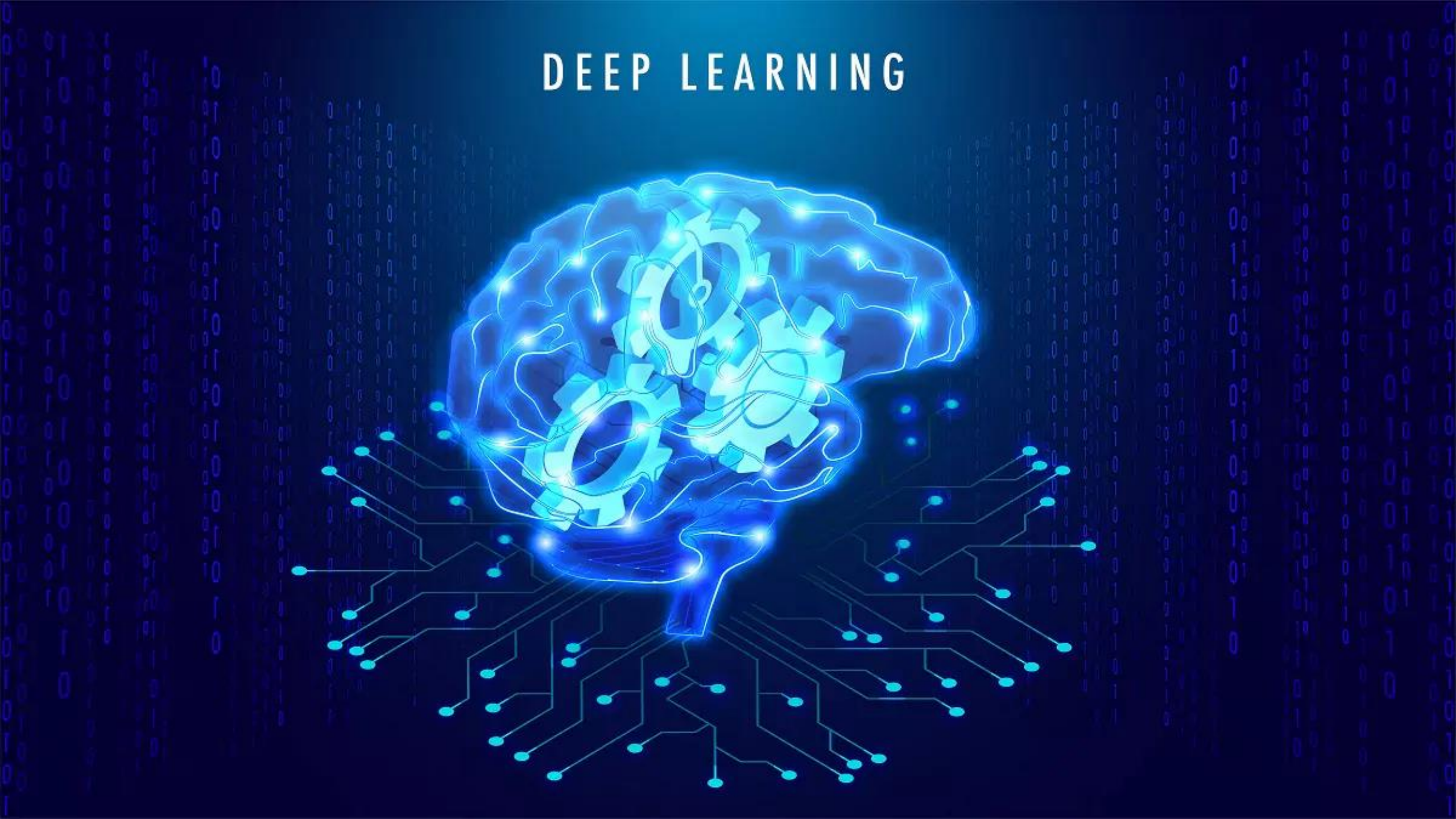
GPU NVidia

2.944 CUDA Cores



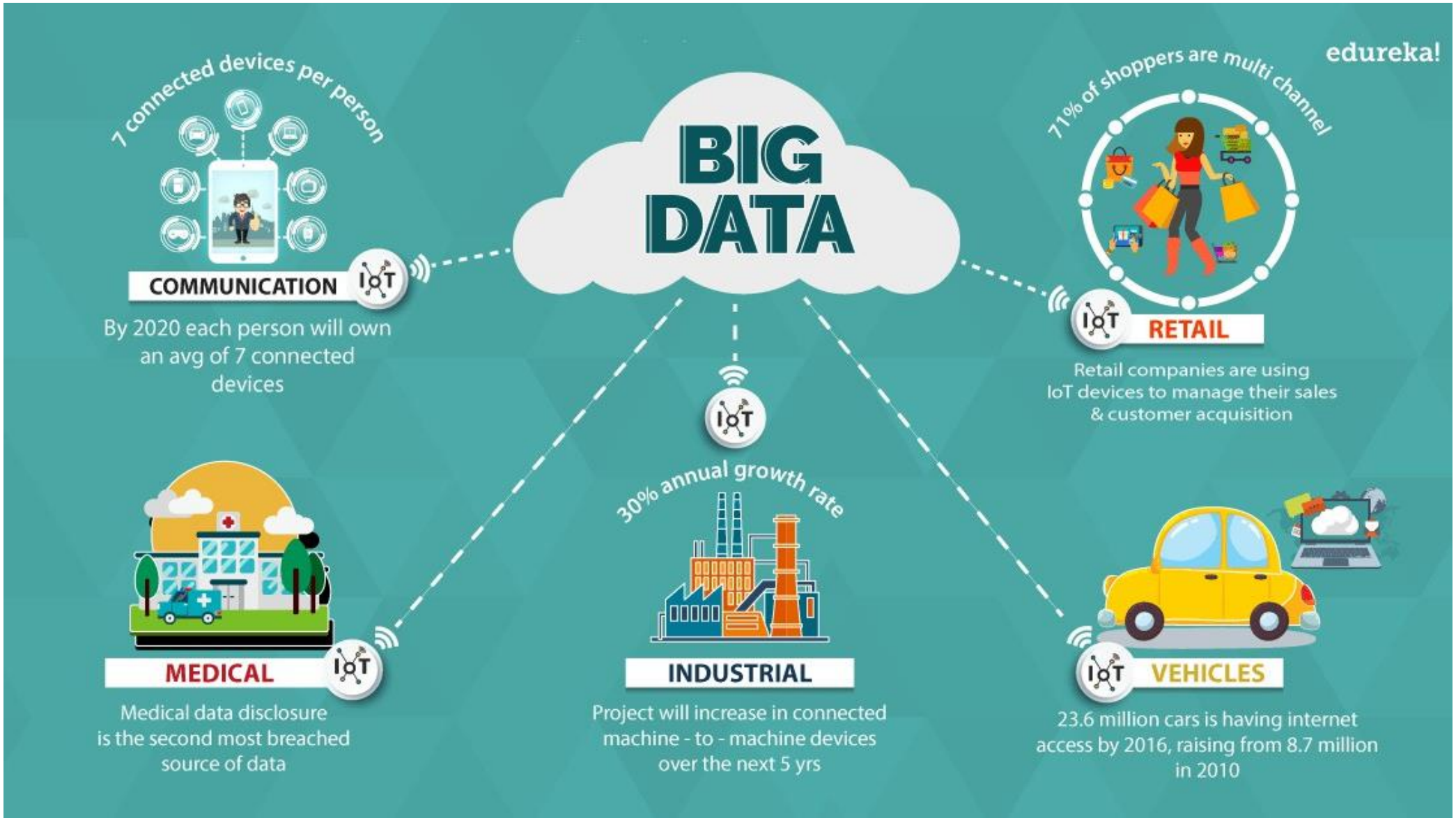
24GB - 16.384 núcleos CUDA

Big Data


DEEP LEARNING

Big Data + Redes Neurais + GPU = Deep Learning

O Big Data vem chamando atenção pela acelerada escala em que volumes cada vez maiores de dados são criados pela sociedade.

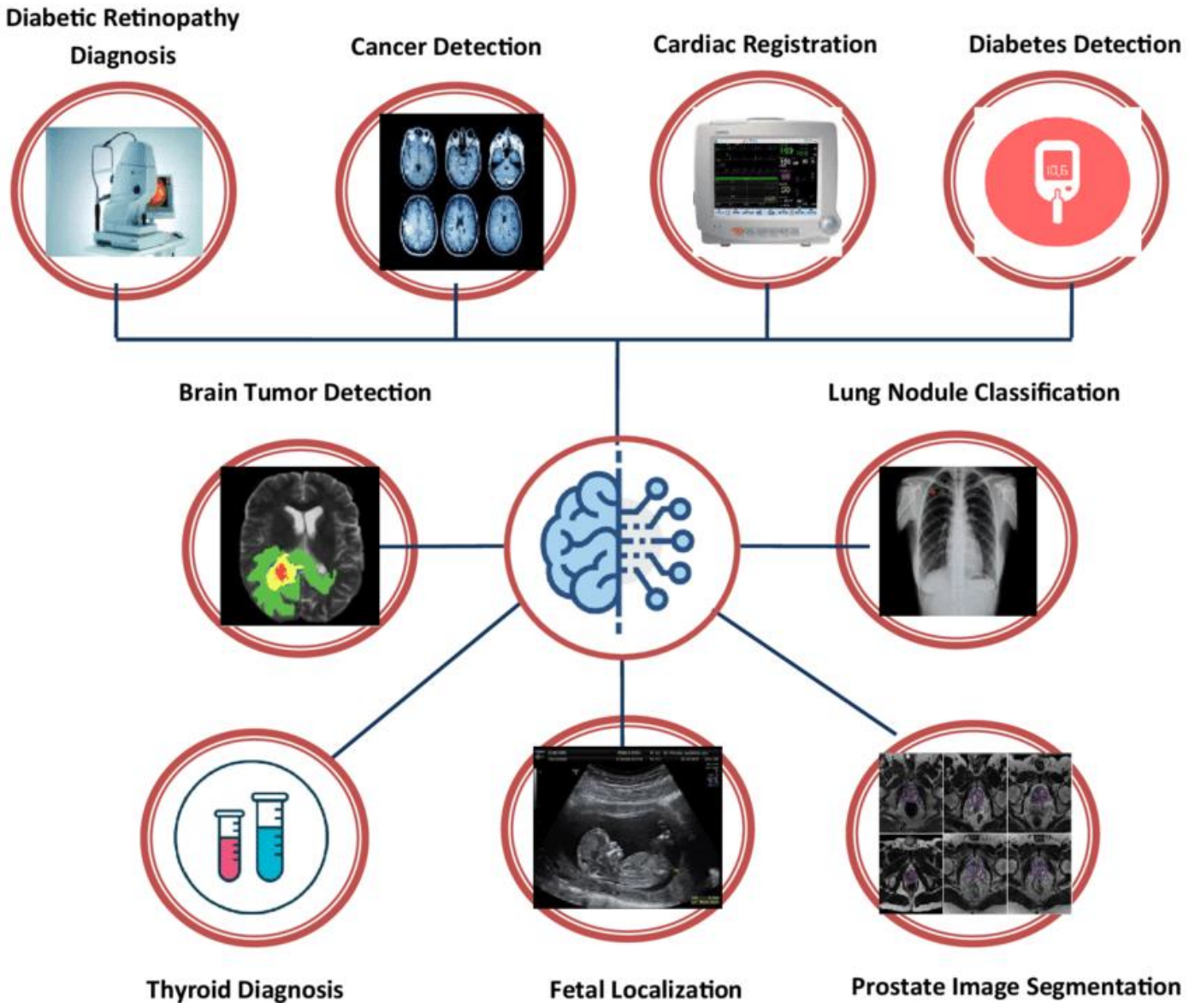


Applications of Machine Learning

- Image & Speech Recognition
- Medical Diagnosis
- Statistical Arbitrage
- Learning Associations

- Classification
- Prediction
- Extraction
- Regression

Applications of Deep Learning and Big IoT personalized Healthcare Services


Diabetic Retinopathy Diagnosis
Cancer Detection
Cardiac Registration
Diabetes Detection
Brain Tumor Detection
Lung Nodule Classification
Thyroid Diagnosis
Fetal Localization
Prostate Image Segmentation

5 Most Common Types of Deep Learning Vision Models

Classification

Identifies the concept of whole image and categorizes
into different classes. (unit of interpretation: image)

Segmentation

Recognizes an object, its shape and location within an image. (unit of interpretation: pixel)

Object Detection

Distinguishes the class of each object and detect its location (unit of interpretation: object)

Optical Character Recognition (OCR)

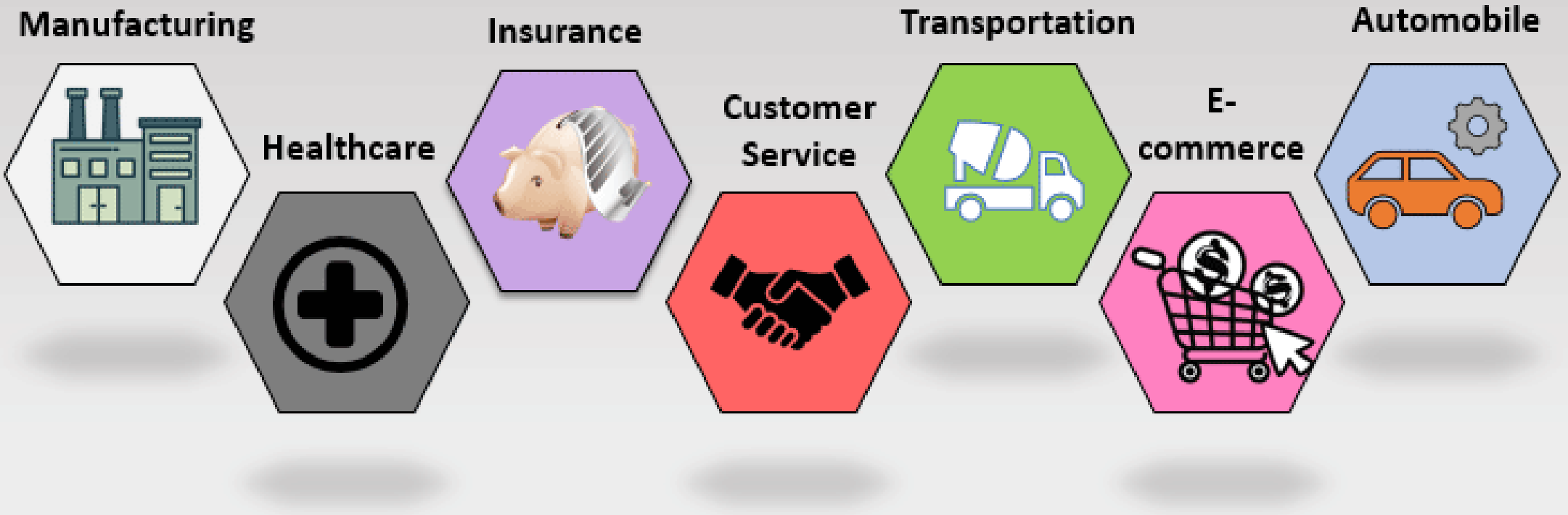
Detects texts in the images and recognizes each character (unit of interpretation: character)

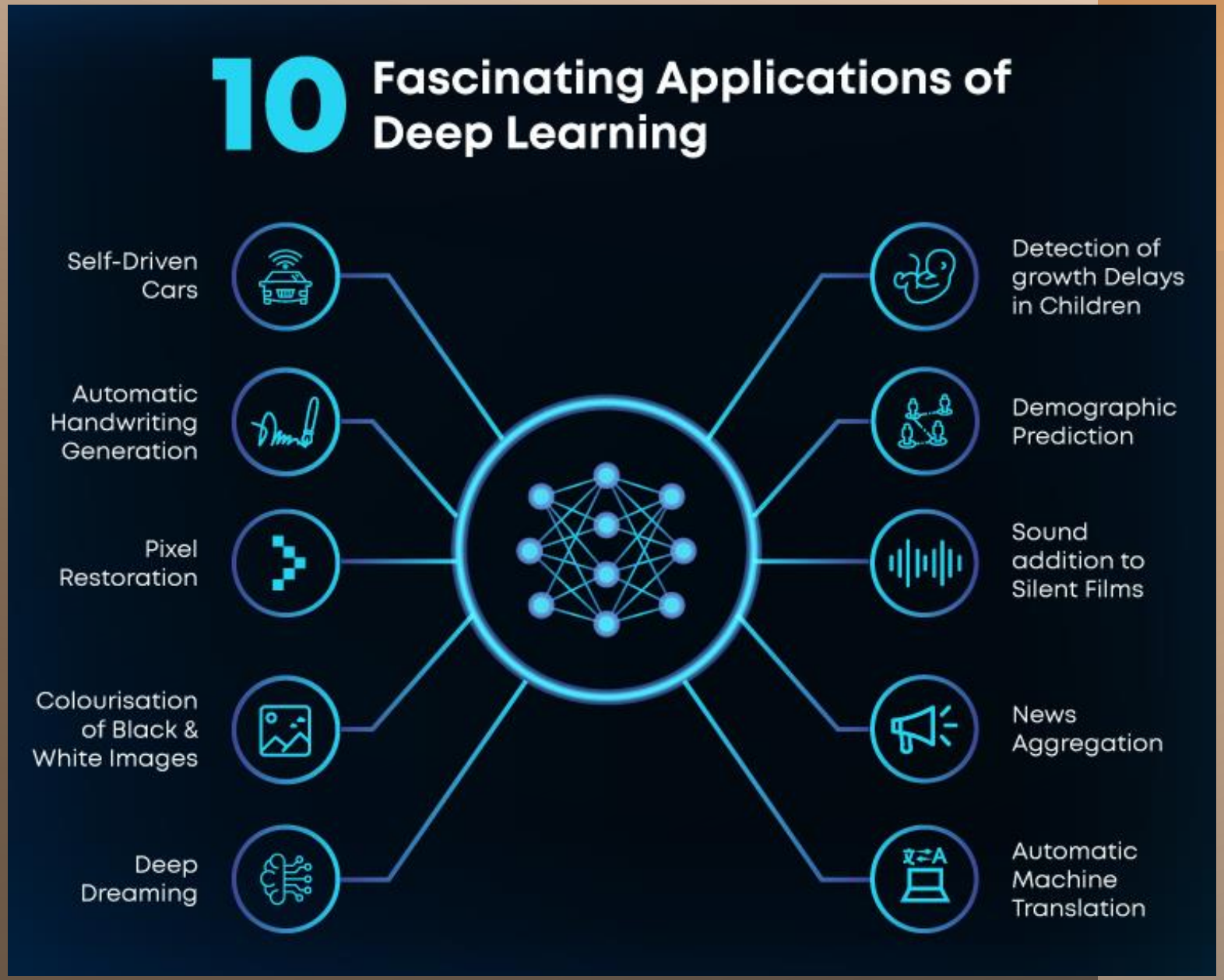
Anomaly Detection

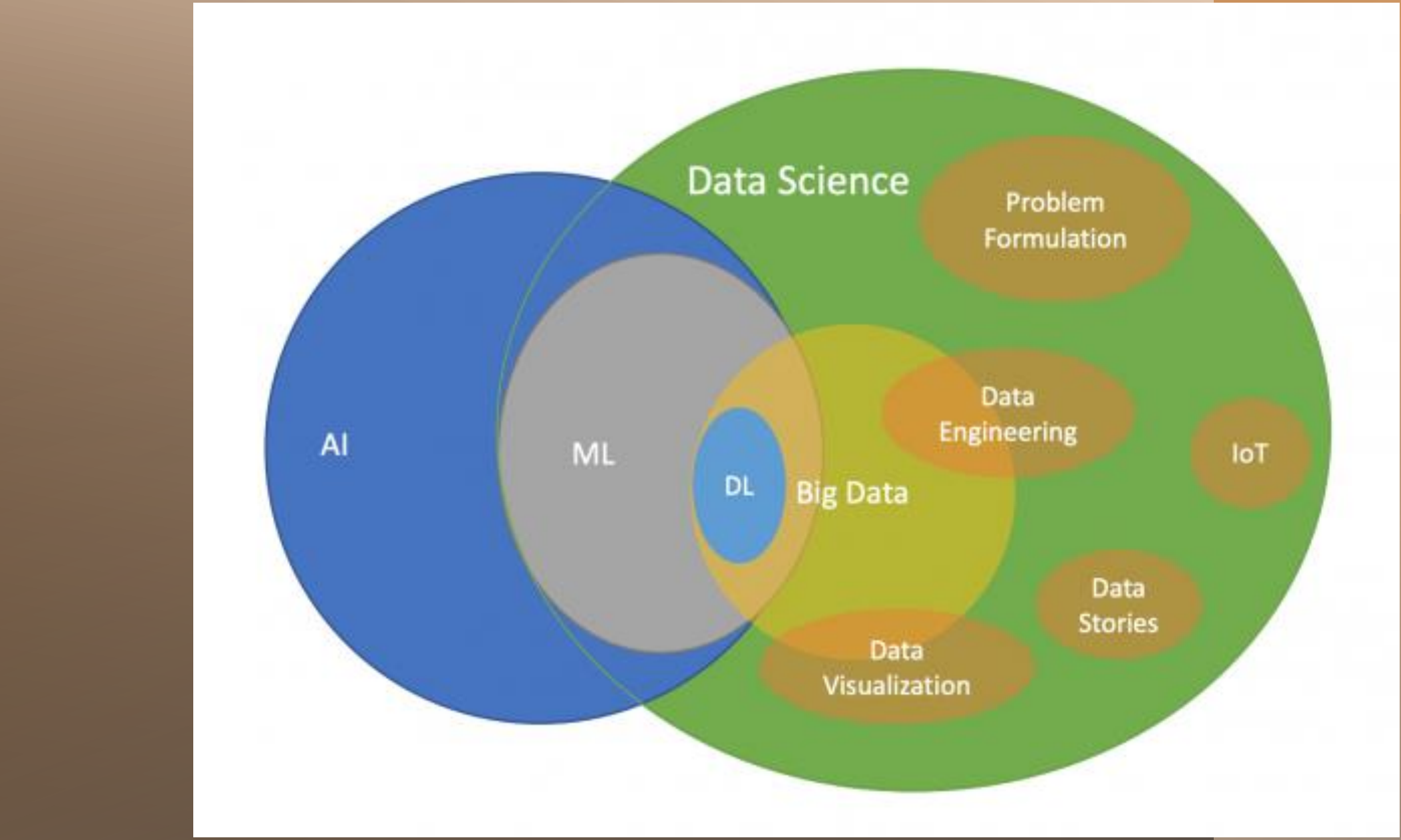
Identifies outliers and captures rare items or observations which differs significantly from majority of the data

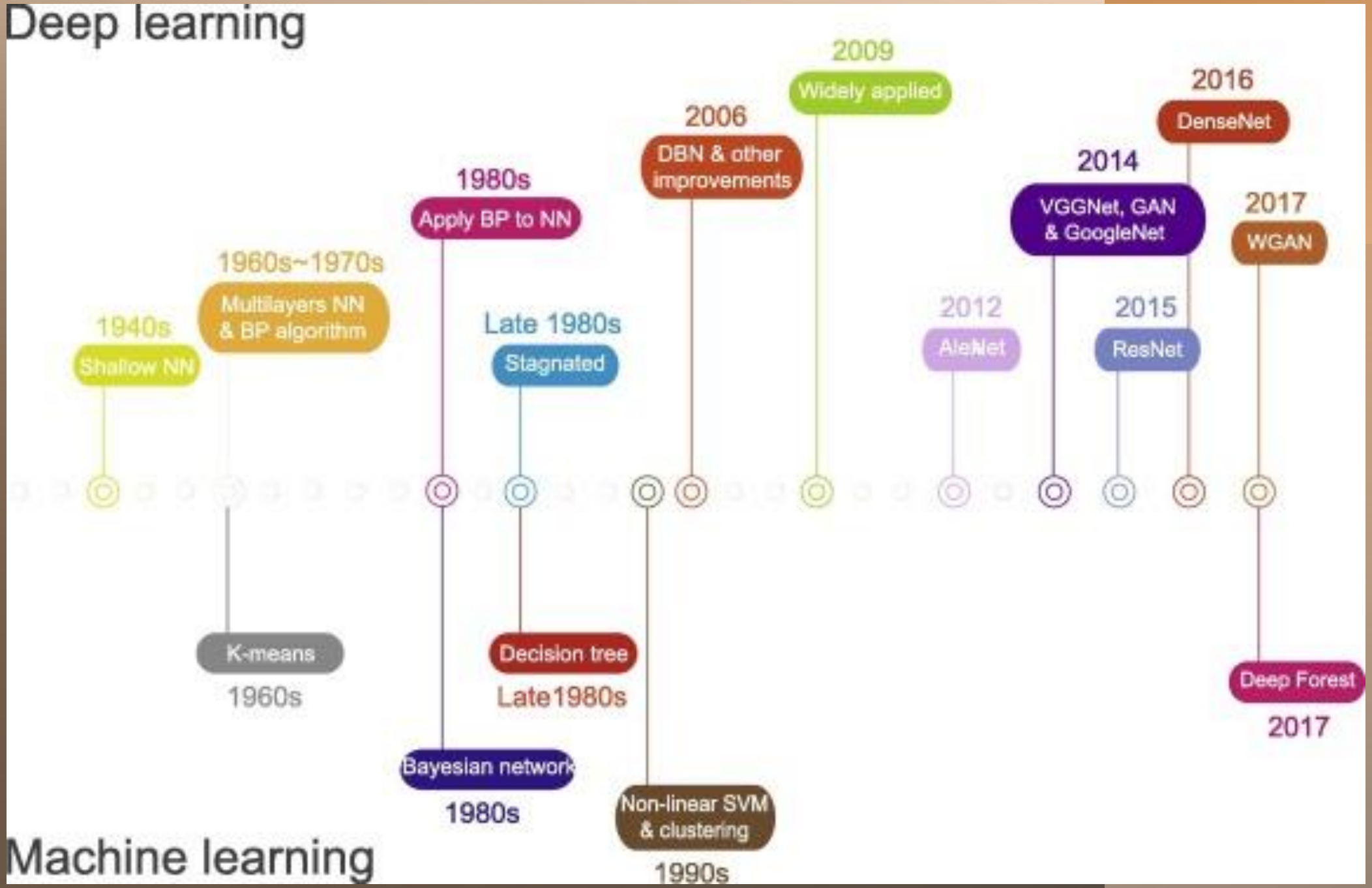
NEUROCLE

Application of Machine Learning




10 Fascinating Applications of Deep Learning
Self-Driven Cars
Automatic Handwriting Generation
Pixel Restoration
Colourisation of Black & White Images
Deep Dreaming
Detection of growth Delays in Children
Demographic Prediction
Sound addition to Silent Films
News Aggregation
Automatic Machine Translation


Data Science
Problem Formulation
AI
ML
DL
Big Data
Data Engineering
IoT
Data Stories
Data Visualization


Deep learning
1940s
Shallow NN
1960s~1970s
Multilayers NN & BP algorithm
K-means
1960s
1980s
Apply BP to NN
Bayesian network
1980s
Late 1980s
Stagnated
Decision tree
Late1980s
2006
DBN & other improvements
Non-linear SVM & clustering
1990s
2009
Widely applied
2012
AleNet
2014
VGGNet, GAN & GoogleNet
2015
ResNet
2016
DenseNet
2017
WGAN
Deep Forest
2017
Machine learning

ARTIFICIAL INTELLIGENCE

IS NOT NEW

ARTIFICIAL INTELLIGENCE

Any technique which enables computers to mimic human behavior

MACHINE LEARNING

AI techniques that give computers the ability to learn without being explicitly programmed to do so

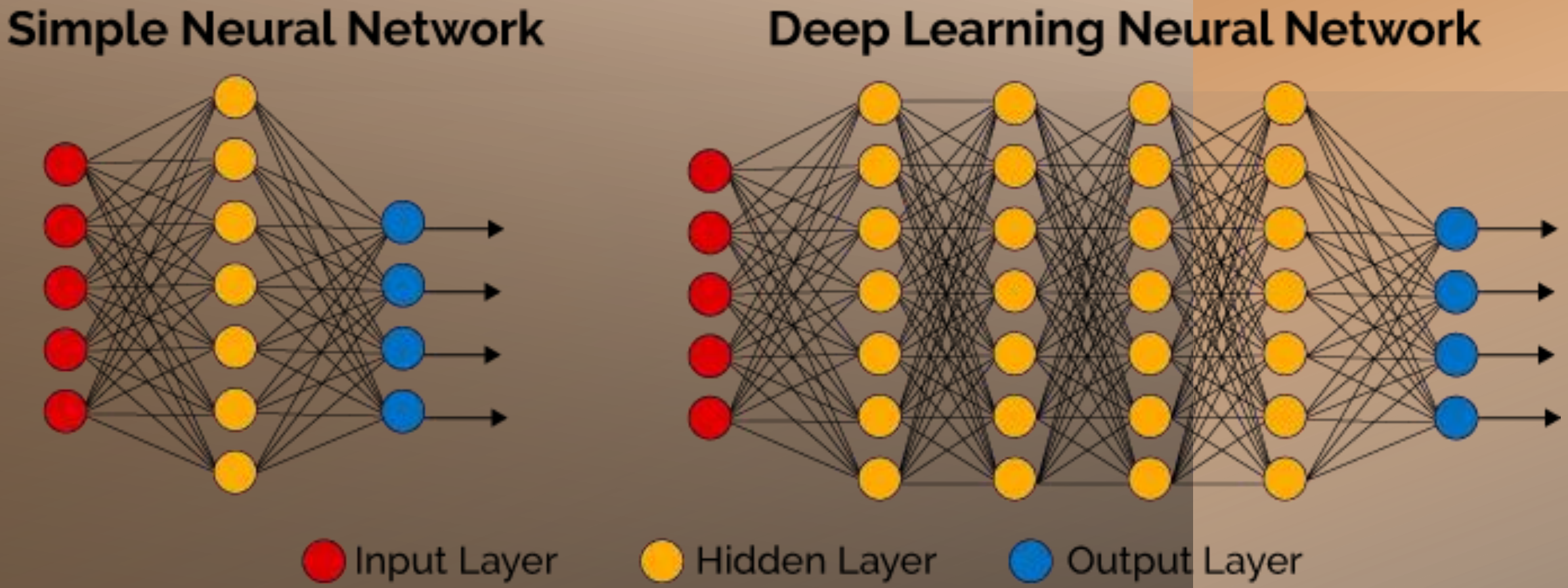
DEEP LEARNING

A subset of ML which make the computation of multi-layer neural networks feasible

1950's 1960's 1970's 1980's 1990's 2000's 2010s

ORACLE

Introdução – O que é Deep Learning



Deep Learning

- Aprendizagem Profunda ou Deep Learning, é uma sub-área da Aprendizagem de Máquina, que emprega algoritmos para processar dados e imitar o processamento feito pelo cérebro humano.

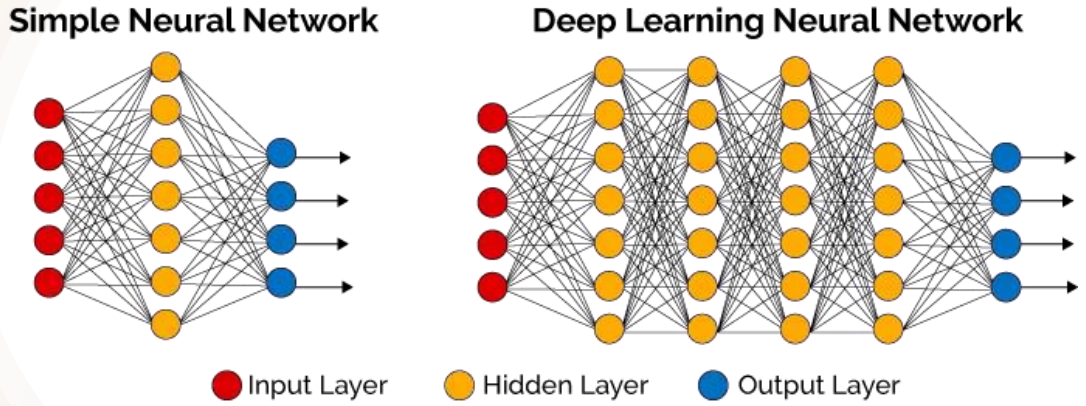
- Deep Learning usa camadas de neurônios matemáticos para processar dados, compreender a fala humana e reconhecer objetos visualmente.

- A informação é passada através de cada camada, com a saída da camada anterior fornecendo entrada para a próxima camada.

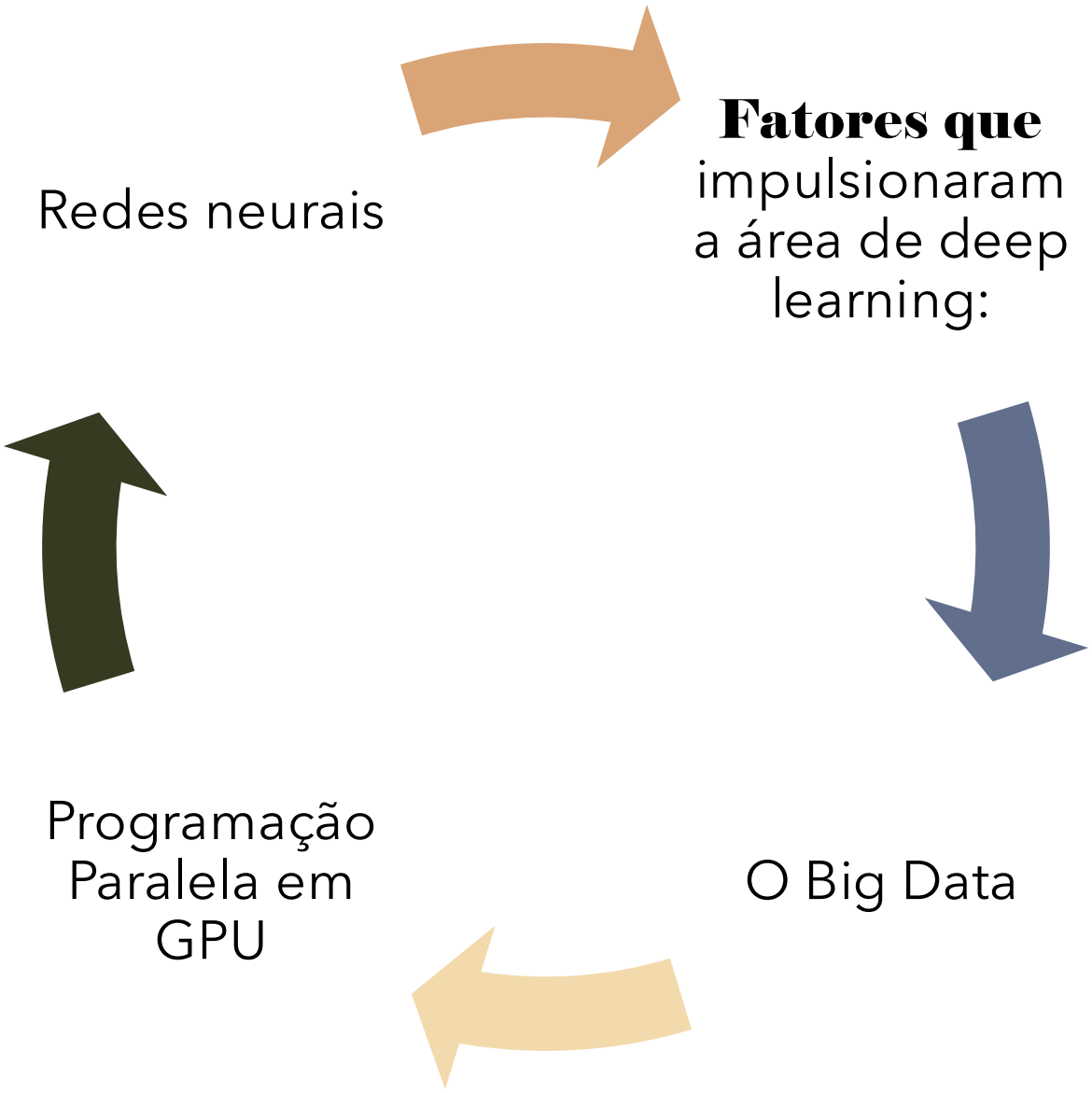
- A primeira camada em uma rede é chamada de camada de entrada, enquanto a última é chamada de camada de saída.

- Todas as camadas entre as duas são referidas como camadas ocultas Cada camada é tipicamente um algoritmo simples e uniforme contendo um tipo de função de ativação

-



Deep Learning - Origem



Programação Paralela em GPU

- A computação acelerada por placas de vídeo é o uso de uma unidade de processamento gráfico (GPU, Graphics Processing Unit) juntamente com uma CPU para acelerar aplicativos de aprendizado profundo, análise e engenharia.

- Pioneiros em 2007 pela NVIDIA, os aceleradores de placas de vídeo agora potencializam data centers de eficiência energética em laboratórios governamentais, universidades, corporações e empresas de médio e grande portes em todo o mundo.

- Elas desempenham um papel fundamental na aceleração de aplicativos em plataformas que variam de inteligência artificial até carros autônomos, drones e robôs

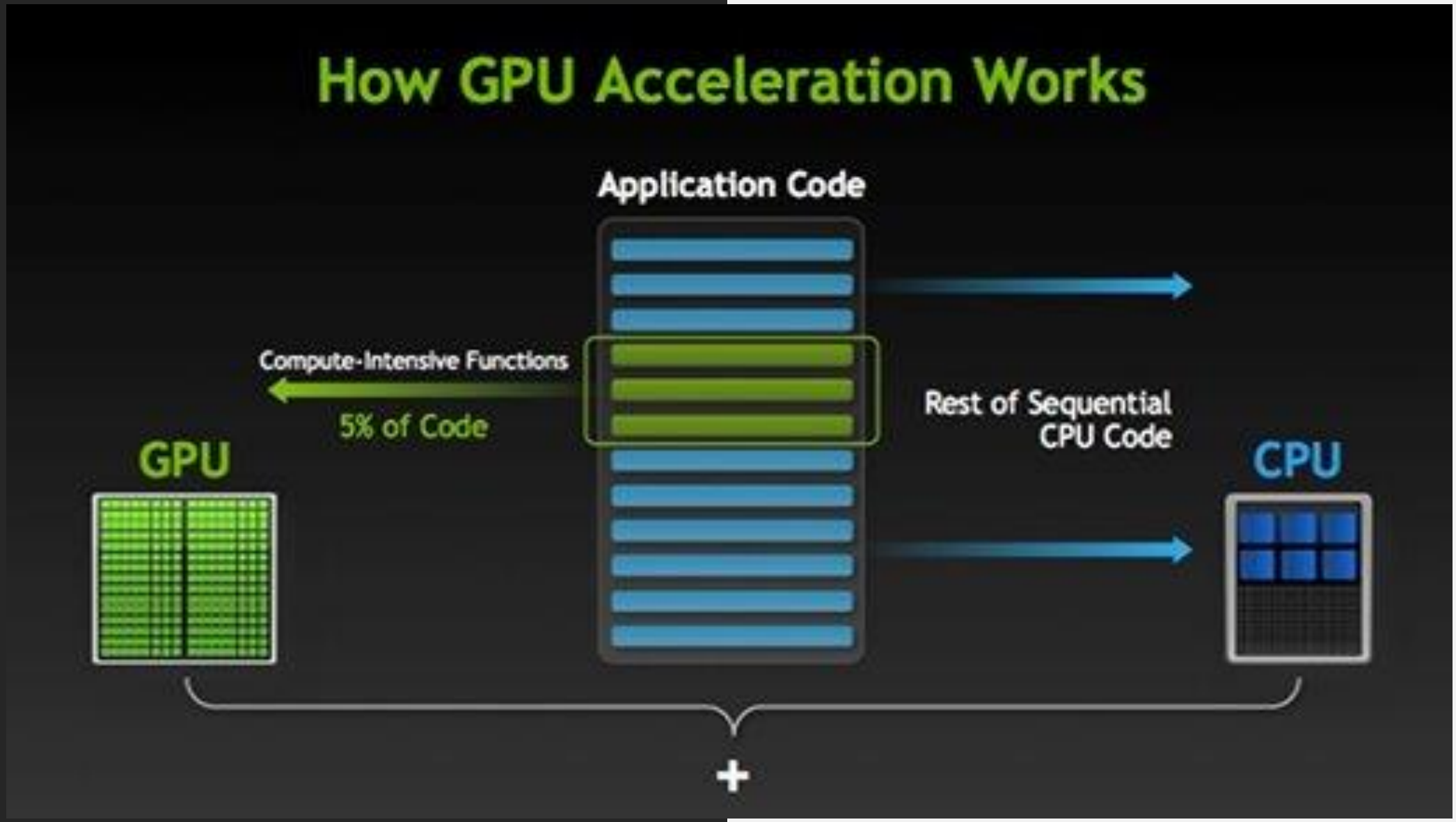
Programação Paralela em GPU

- A computação acelerada por placas de vídeo libera porções do aplicativo com uso intenso de computação para a placa de vídeo (GPU), enquanto o restante do código ainda é executado na CPU.

- Da perspectiva do usuário,

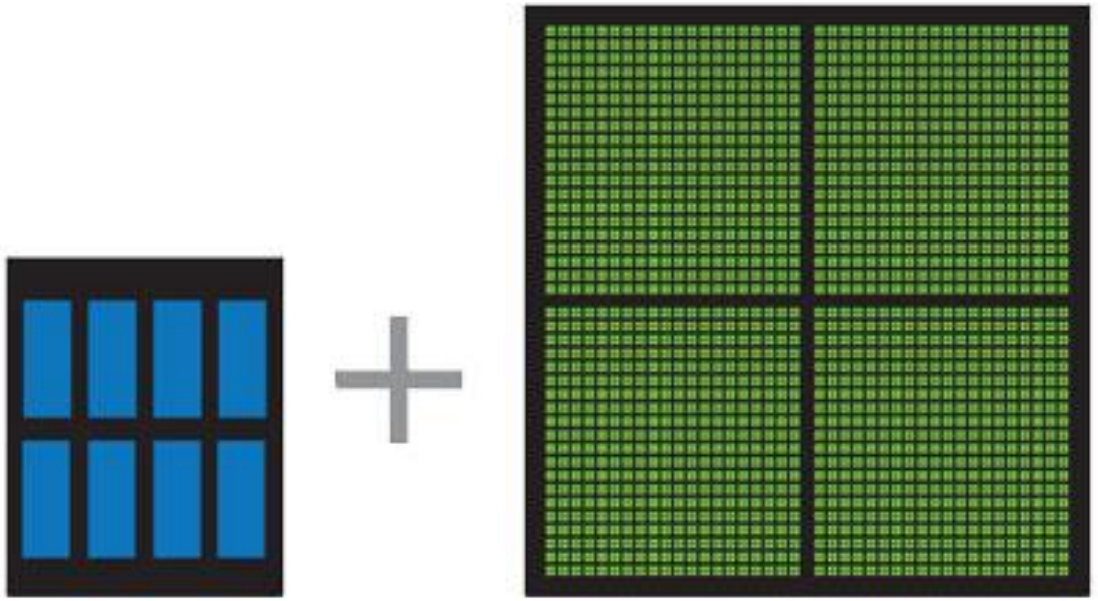
os aplicativos simplesmente

são executados muito mais rápido

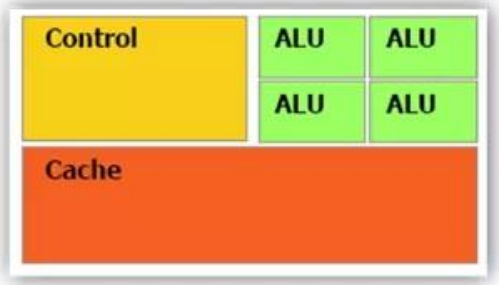

How GPU Acceleration Works
Application Code
Compute-Intensive Functions
5% of Code
Rest of Sequential
CPU Code
GPU
CPU
+

Programação Paralela em GPU

- Uma CPU tem alguns núcleos otimizados para o processamento serial sequencial, enquanto uma placa de vídeo - **GPU** tem uma arquitetura paralela gigantesca que consiste em **milhares de núcleos menores** e mais eficientes criados para lidar com múltiplas tarefas simultaneamente

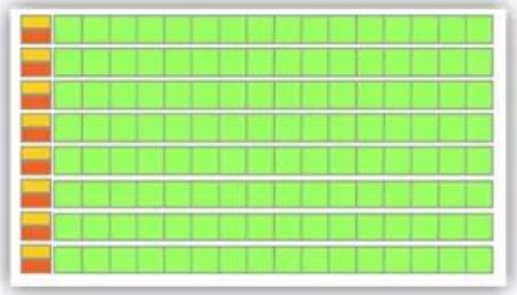


CPU



* Low compute density
* Complex control logic
* Large caches (L1$/L2$, etc.)
* Optimized for serial operations
  - Fewer execution units (ALUs)
  - Higher clock speeds
* Shallow pipelines (<30 stages)
* Low Latency Tolerance
* Newer CPUs have more parallelism

GPU

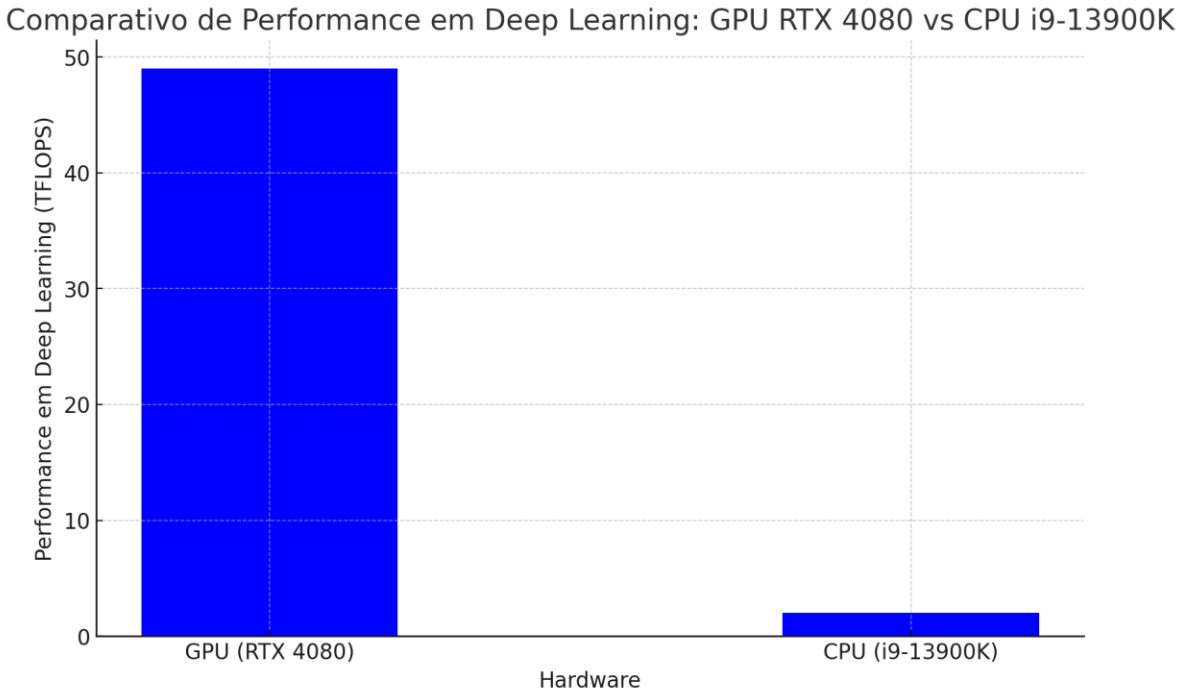


* High compute density
* High Computations per Memory Access
* Built for parallel operations
  - Many parallel execution units (ALUs)
  - Graphics is the best known case of parallelism
* Deep pipelines (hundreds of stages)
* High Throughput
* High Latency Tolerance
* Newer GPUs:
  - Better flow control logic (becoming more CPU-like)
  - Scatter/Gather Memory Access
  - Don't have one-way pipelines anymore

Programação Paralela em GPU

Ao rodar na GPU RTX 4080**,** o treinamento da rede aproximadamente 25 **vezes** mais rápido em comparação com a CPU.

Se levarmos isso a utilização real, em que uma rede dessas pode treinar por semanas, utilizando apenas CPU seria algo impossível de se obter resultados



Deep Learning = Big Data + Redes Neurais + GPU

Big Data

facebook 350 millions images uploaded per day

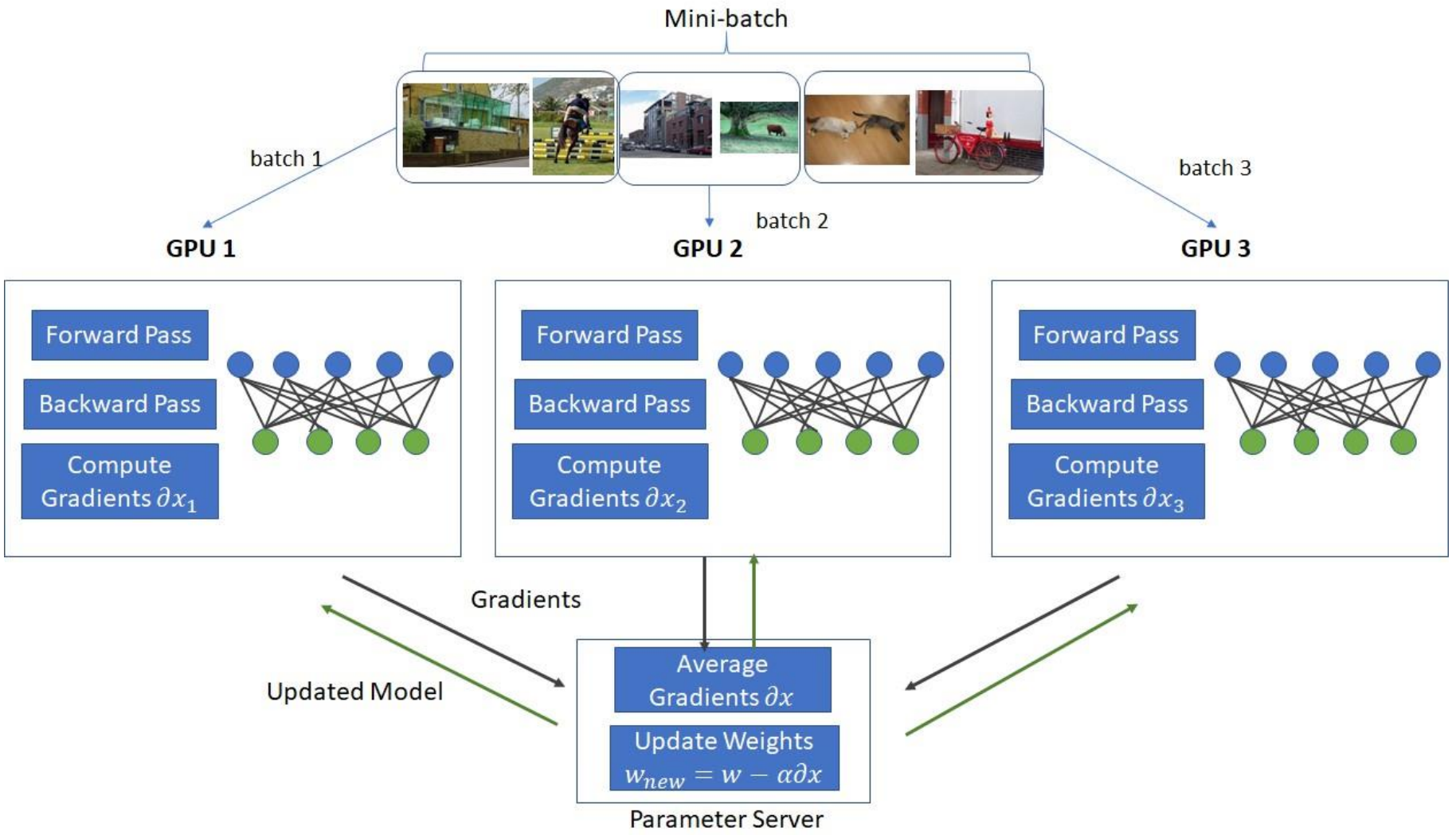
Walmart 2.5 Petabytes of customer data hourly

YouTube 300 hours of video uploaded every minute

Better Algorithms

GPU Acceleration

Deep Learning = Big Data + Redes Neurais + GPU

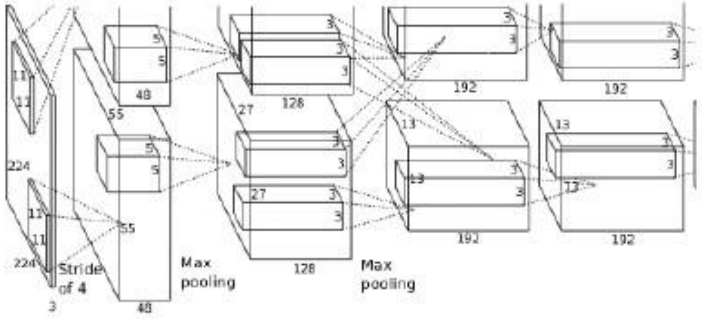


Deep Learning = Big Data + Redes Neurais + GPU

The Deep Learning "Computer Vision Recipe"

Big Data: ImageNet

+



Deep Convolutional Neural Network

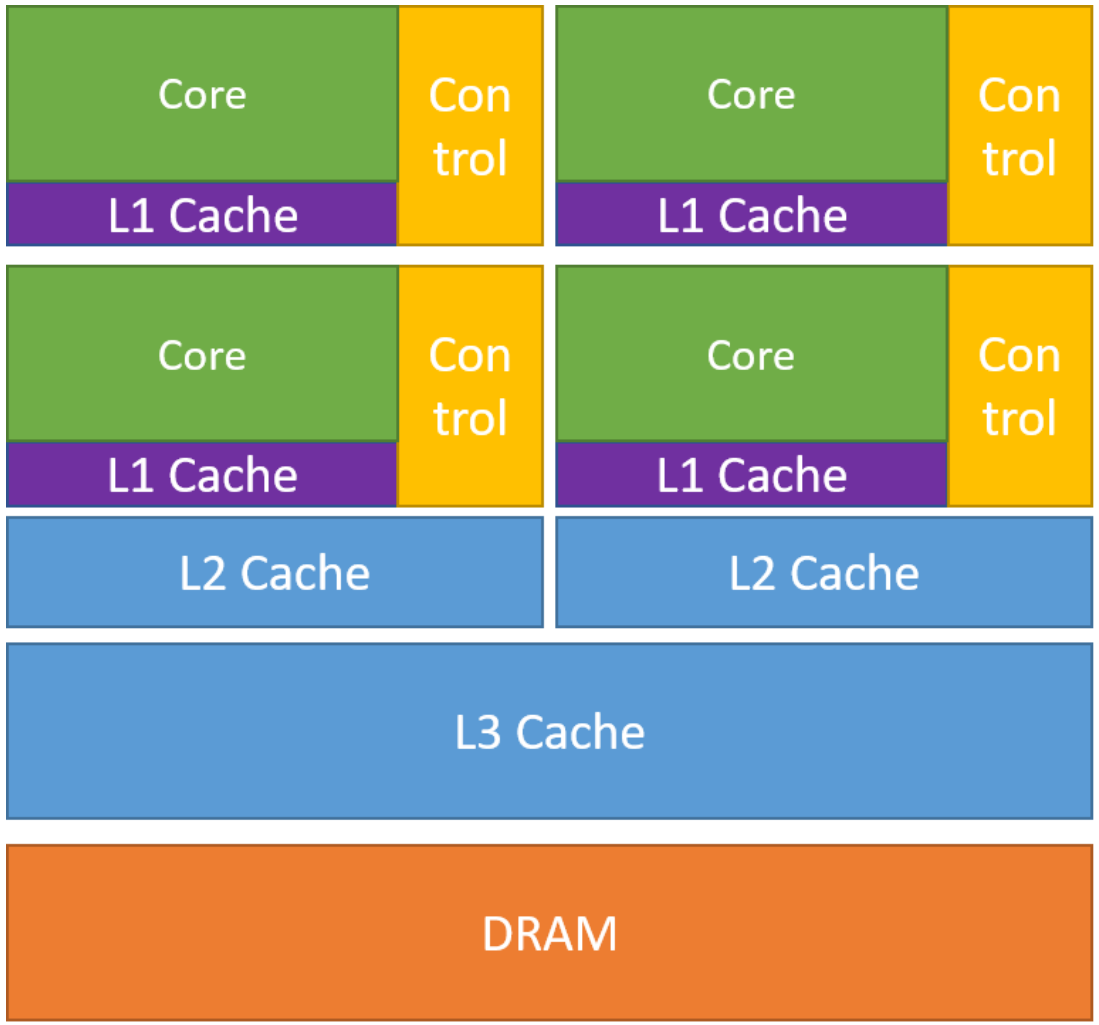
+

Backprop on GPU

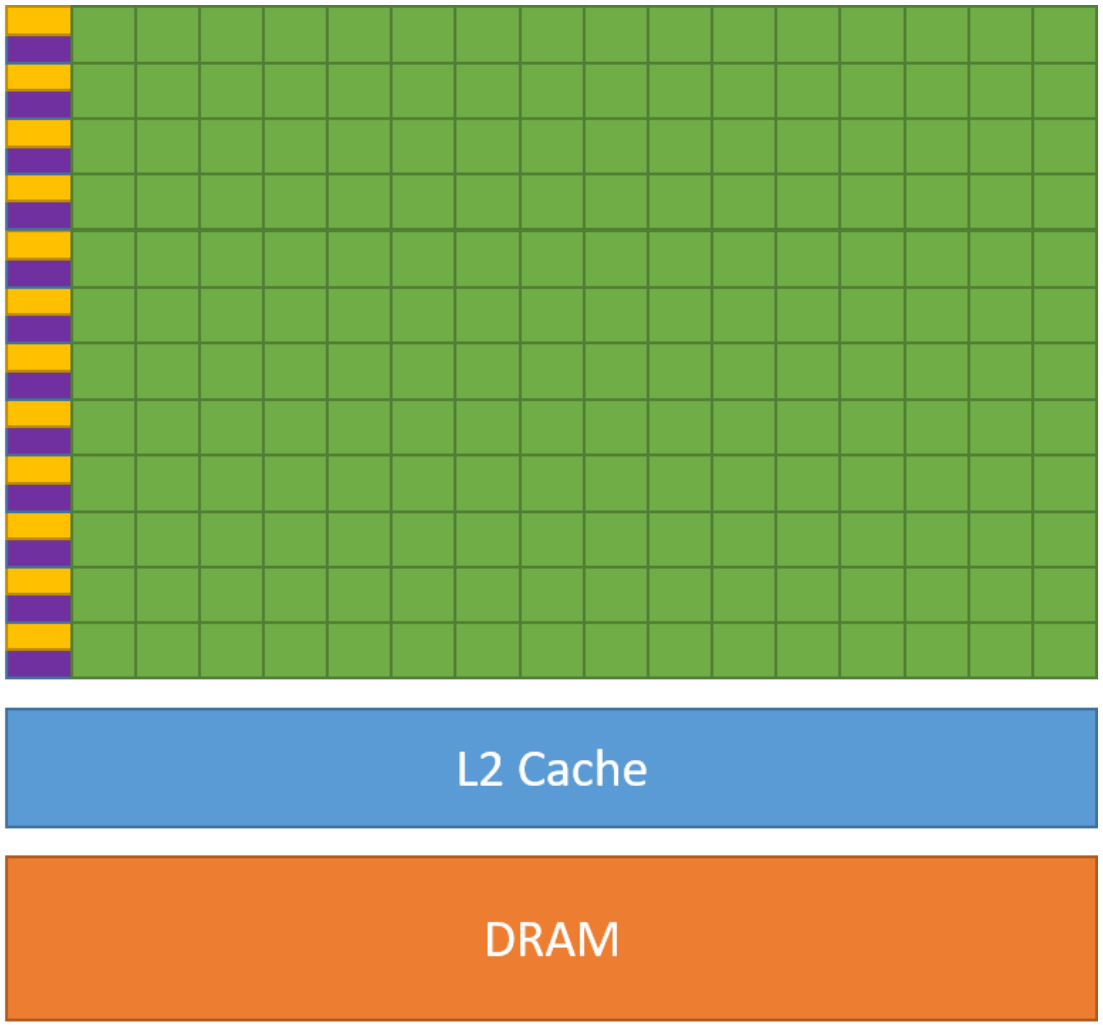
=

Learned Weights

Arquitetura CPU x GPU



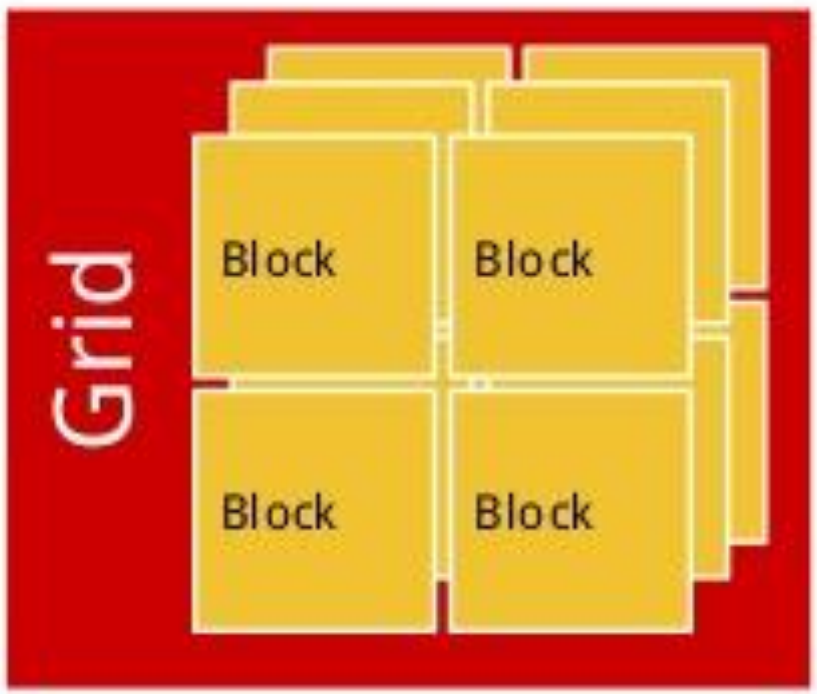
CPU

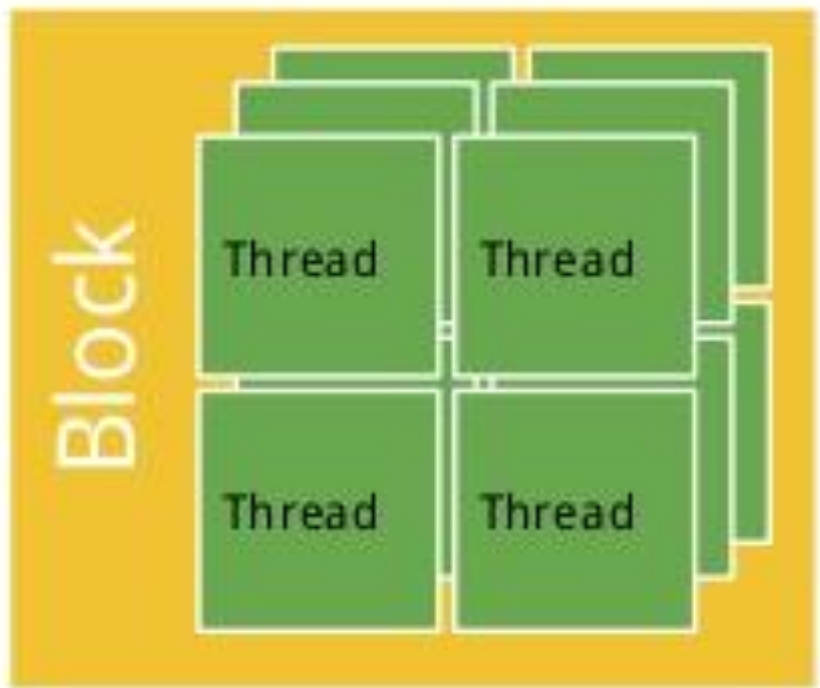


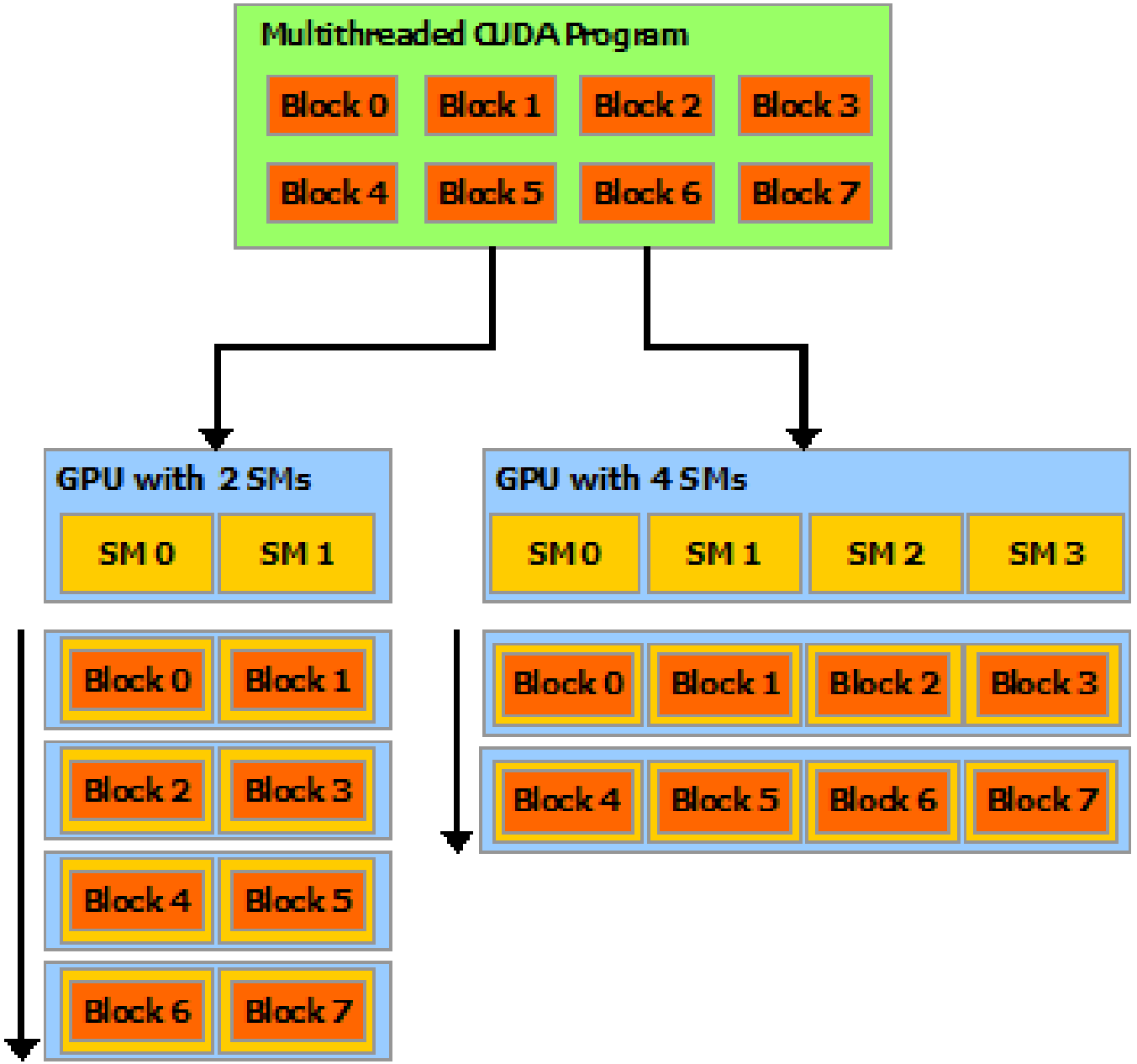
GPU

CUDA Thread Organization

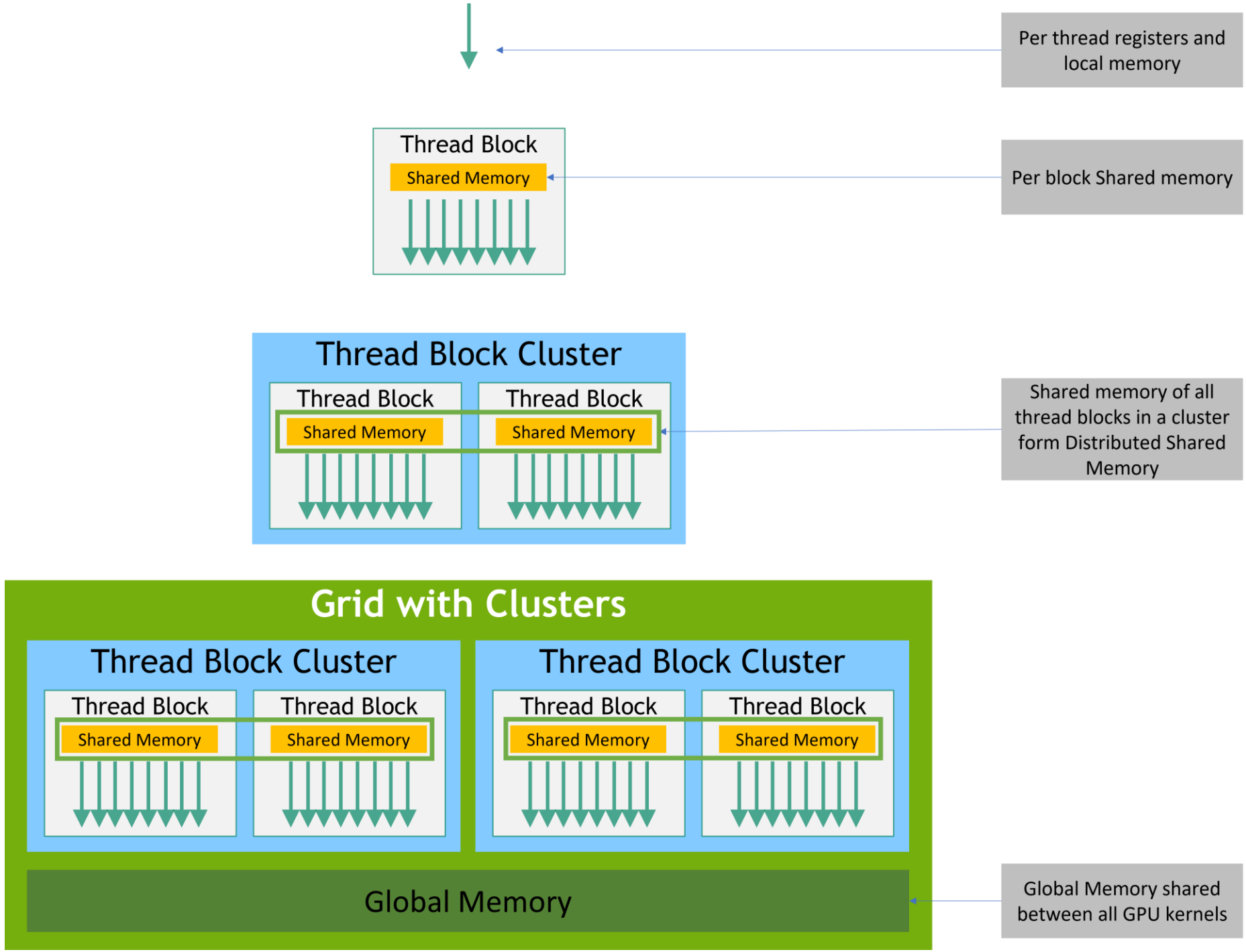
- **Grid** = [Vector~3D Matrix] of Blocks
  - **Block** = [Vector~3D Matrix] of Threads
    - **Thread** = One that computes



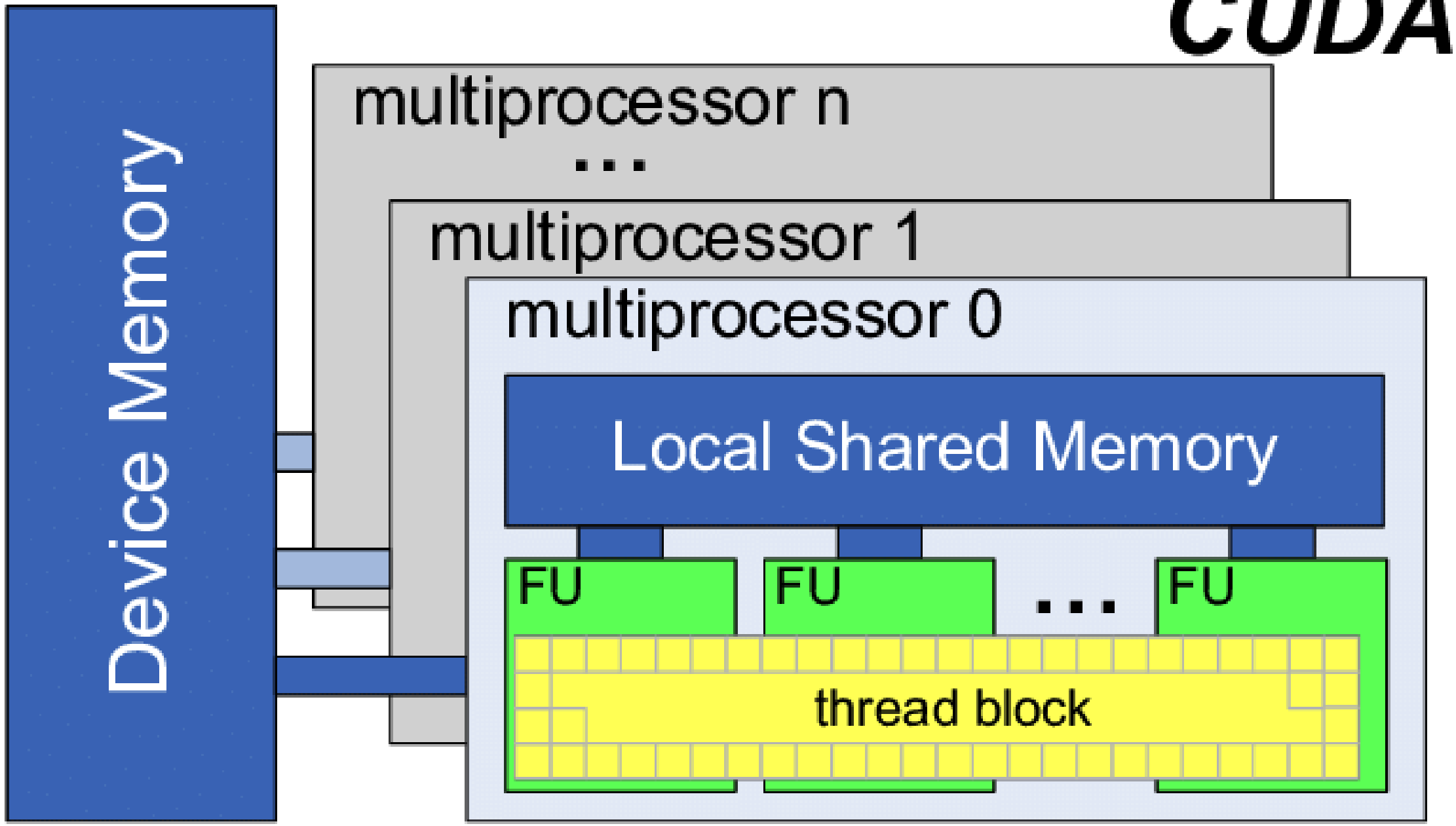


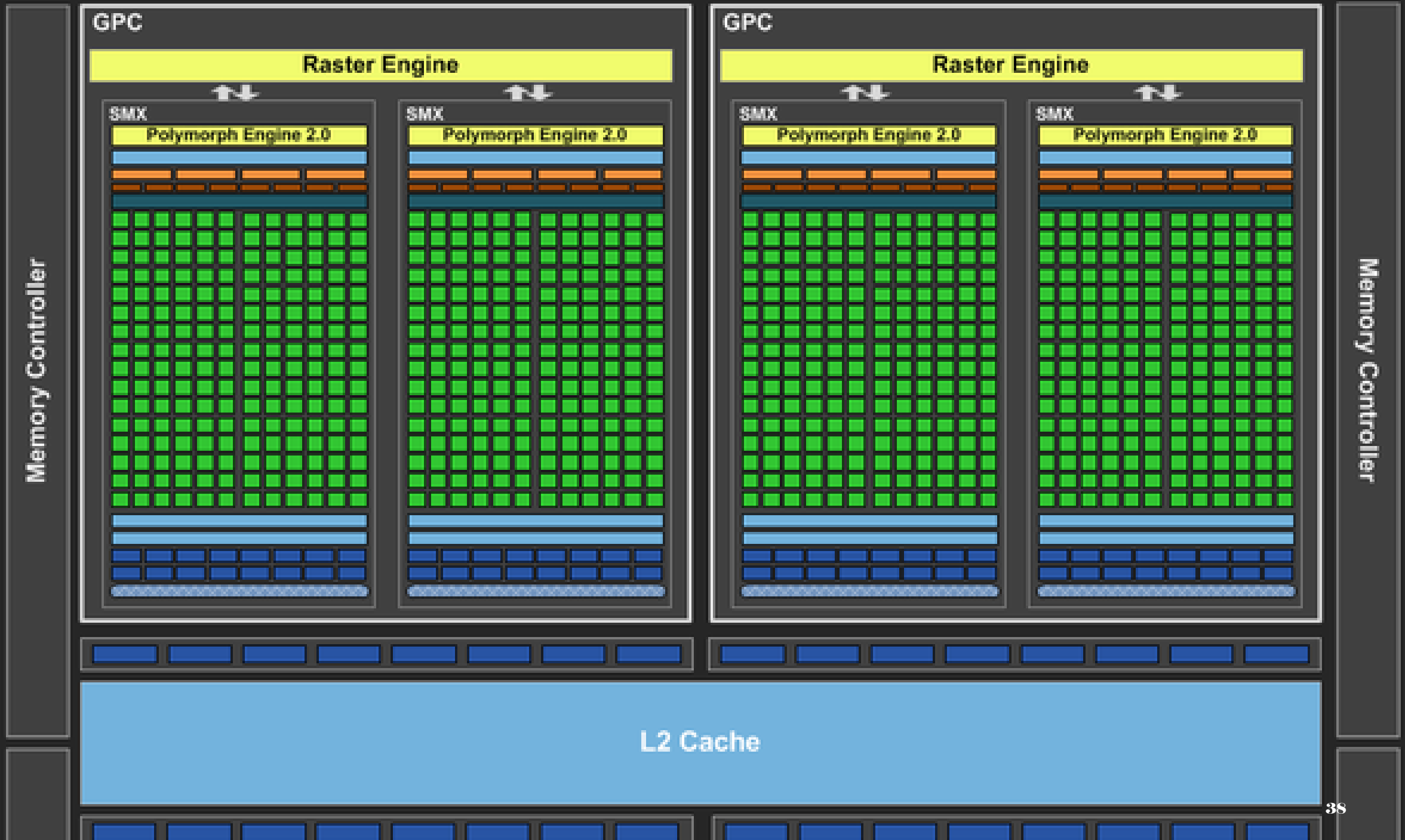


SM – Shared Memory

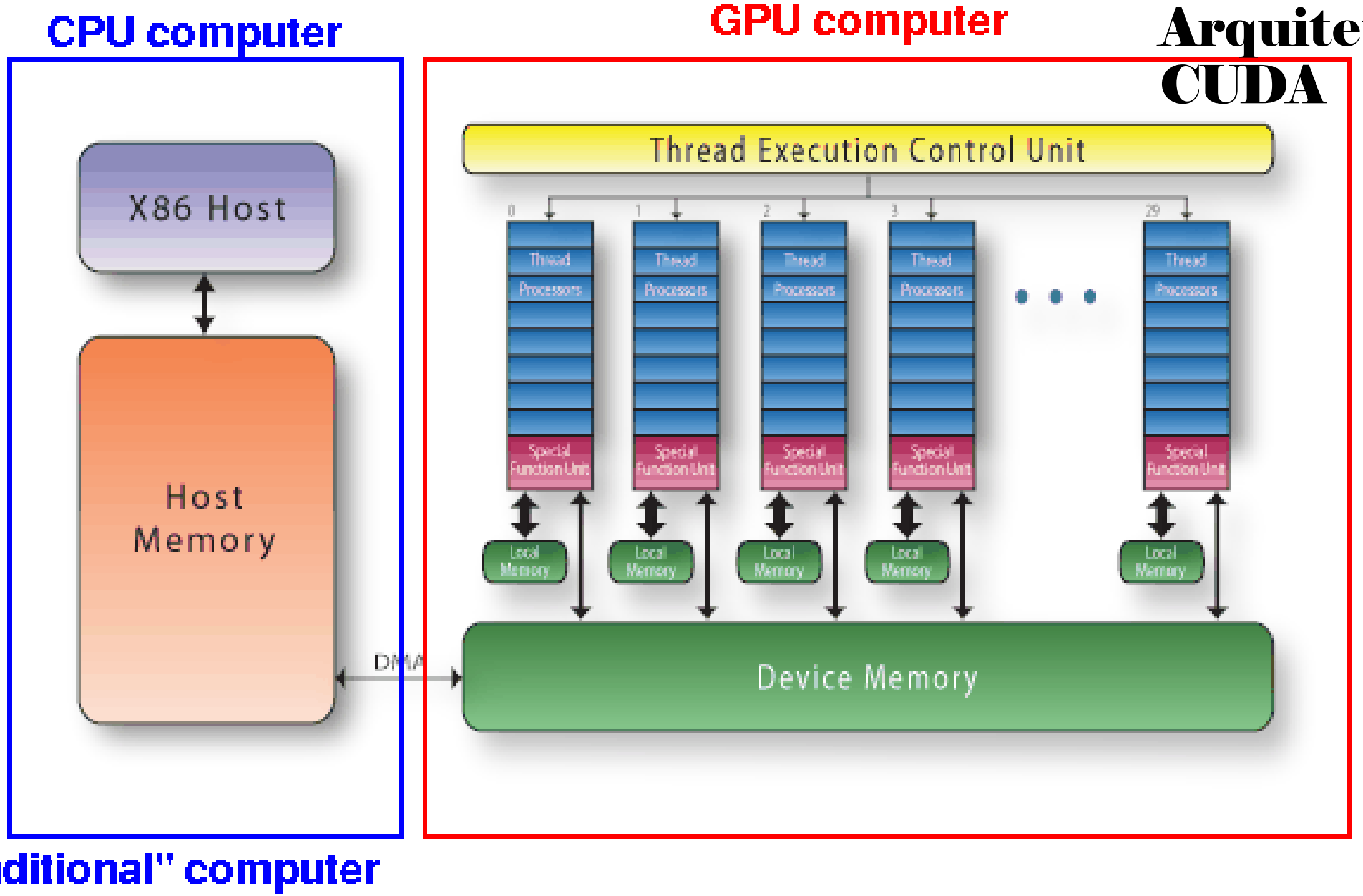

Per thread registers and local memory
Thread Block
Shared Memory
Per block Shared memory
Thread Block Cluster
Thread Block
Shared Memory
Thread Block
Shared Memory
Shared memory of all thread blocks in a cluster form Distributed Shared Memory
Grid with Clusters
Thread Block Cluster
Thread Block
Shared Memory
Thread Block
Shared Memory
Thread Block Cluster
Thread Block
Shared Memory
Thread Block
Shared Memory
Global Memory
Global Memory shared between all GPU kernels

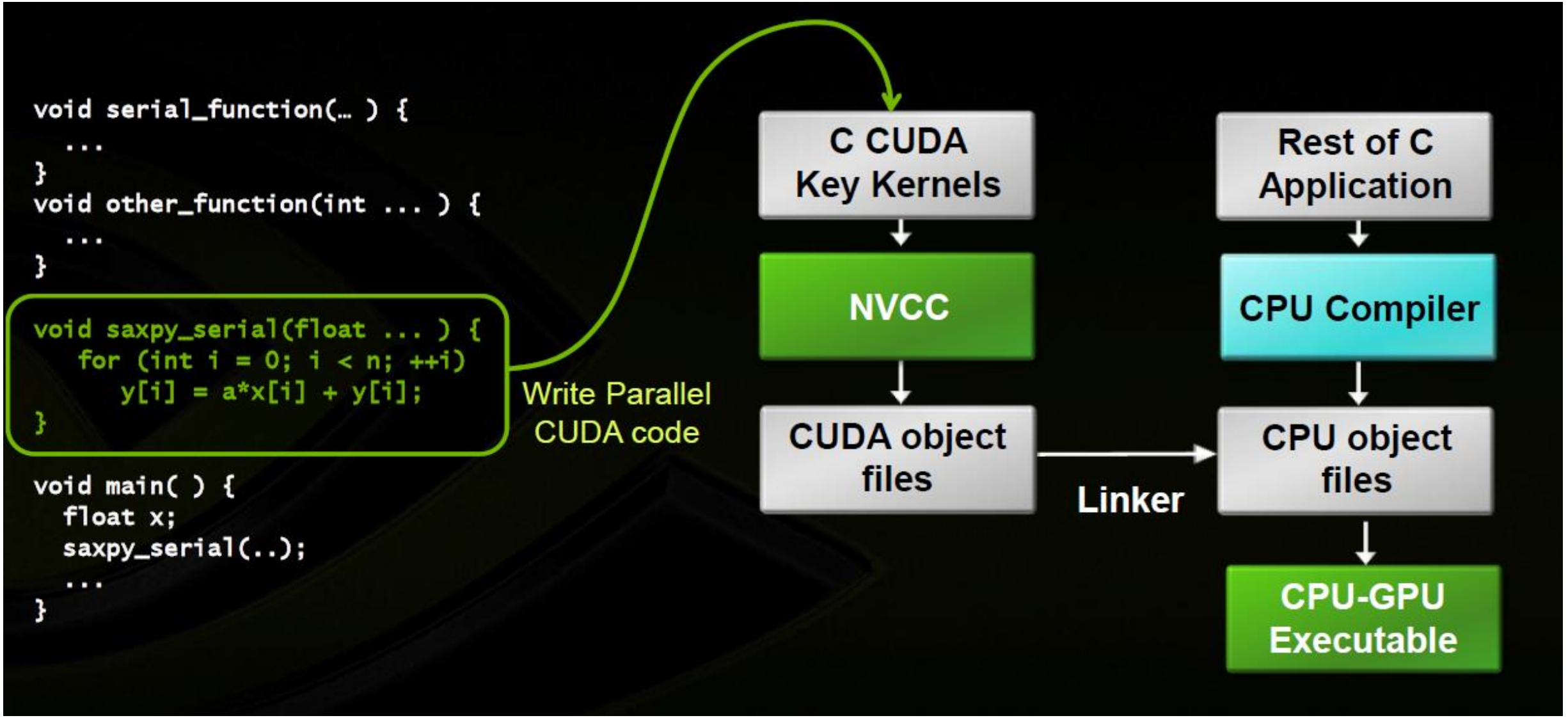
CUDA


Device Memory
multiprocessor n
...
multiprocessor 1
multiprocessor 0
Local Shared Memory
FU
FU
...
FU
thread block


GPC
Raster Engine
SMX
Polymorph Engine 2.0
SMX
Polymorph Engine 2.0
GPC
Raster Engine
SMX
Polymorph Engine 2.0
SMX
Polymorph Engine 2.0
Memory Controller
Memory Controller
L2 Cache

Arquitetura CUDA




void serial_function(…) {
...
}
void other_function(int ...) {
...
}
void saxpy_serial(float ...) {
for (int i = 0; i < n; ++i)
y[i] = a*x[i] + y[i];
}
void main( ) {
float x;
saxpy_serial(..);
...
}
Write Parallel CUDA code
C CUDA Key Kernels
NVCC
CUDA object files
Rest of C Application
CPU Compiler
CPU object files
Linker
CPU-GPU Executable

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```

```
__global__ void add(int *a, int *b, int *c, int n)
{
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < n)
        c[i] = a[i] + b[i];
}

int main() {
    int n = 10;
    int *a, *b, *c;
    int *d_a, *d_b, *d_c;
    int size = n * sizeof(int);

    a = (int *)malloc(size);
    b = (int *)malloc(size);
    c = (int *)malloc(size);

    for (int i = 0; i < n; i++) {
        a[i] = i;
        b[i] = i * 2;
    }
```

```
cudaMalloc((void **)&d_a, size);
cudaMalloc((void **)&d_b, size);
cudaMalloc((void **)&d_c, size);

cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_b, b, size, cudaMemcpyHostToDevice);

add<<<1, n>>>(d_a, d_b, d_c, n);

cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost);

for (int i = 0; i < n; i++)
    printf("%d + %d = %d\n", a[i], b[i], c[i]);

cudaFree(d_a);
cudaFree(d_b);
cudaFree(d_c);
free(a);
```

Na plataforma CUDA, no Paralelismo, usamos os seguintes conceitos:

__global__ - Esta palavra-chave é um qualificador usado para dizer ao compilador CUDA que a função deve ser compilada para a GPU.
Para o CUDA C/C ++, o compilador nvcc irá lidar com a compilação deste código.

blockIdx.x - Esta é uma variável usada dentro de um kernel de GPU para determinar a ID do bloco que está atualmente executando o código. Uma vez que haverá muitos blocos em paralelo, precisamos desta ID para ajudar a determinar qual parte dos dados um bloco particular funcionará.

threadIdx.x - Esta é uma variável usada dentro de um kernel de GPU para determinar o ID da thread que está atualmente executando o código no bloco ativo.

blockDim.x - Esta é uma variável que retorna um valor que indica o número de threads que há por bloco. Lembre-se de que todos os blocos agendados para executar na GPU são idênticos, exceto para o valor de blockIdx.x.

myKernel <<< numero_de_blocos, threads_por_bloco>>> (...) - Esta é a sintaxe usada para iniciar um kernel na GPU. Dentro de "<<< >>>", estabelecemos dois valores. O primeiro é o número total de blocos que queremos executar na GPU, e o segundo é o número de threads que há por bloco.

```
void saxpy(int n, float a,
           float *x, float *y)
{
  for (int i = 0; i < n; ++i)
    y[i] = a*x[i] + y[i];
}

int N = 1<<20;



// Perform SAXPY on 1M elements
saxpy(N, 2.0, x, y);
```

```
__global__
void saxpy(int n, float a,
           float *x, float *y)
{
  int i = blockIdx.x*blockDim.x + threadIdx.x;
  if (i < n) y[i] = a*x[i] + y[i];
}

int N = 1<<20;
cudaMemcpy(x, d_x, N, cudaMemcpyHostToDevice);
cudaMemcpy(y, d_y, N, cudaMemcpyHostToDevice);

// Perform SAXPY on 1M elements
saxpy<<<4096,256>>>(N, 2.0, x, y);

cudaMemcpy(d_y, y, N, cudaMemcpyDeviceToHost);
```

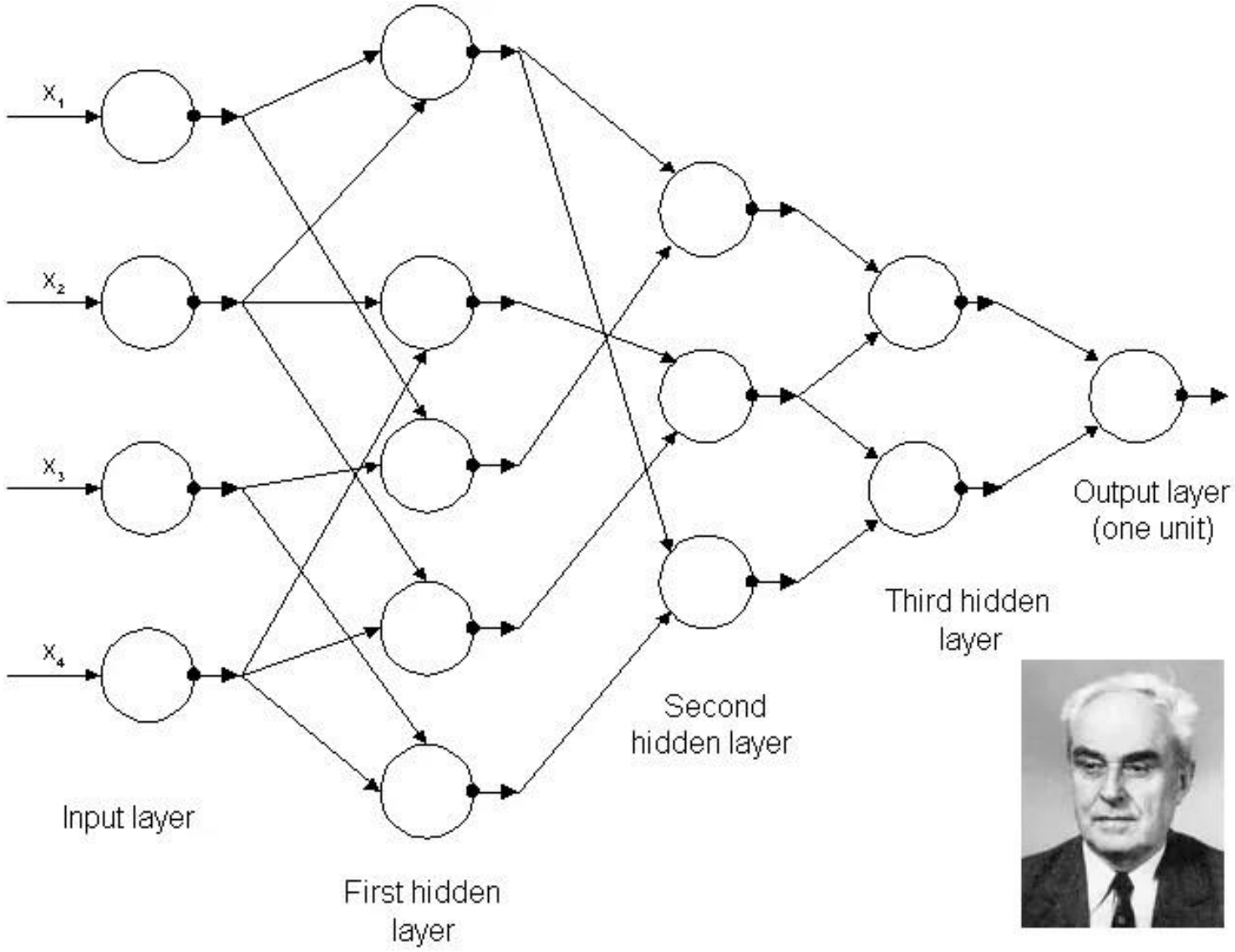
Deep Learning

- A aprendizagem profunda é responsável por avanços recentes em visão computacional, reconhecimento de fala, processamento de linguagem natural, reconhecimento de áudio, carros autônomos, etc.

- O aprendizado profundo é baseado no conceito de redes neurais artificiais, ou sistemas computacionais que imitam a maneira como o cérebro humano funciona.

Deep Learning

Arquitetura da primeira rede profunda conhecida treinada por Alexey Grigorevich Ivakhnenko em **1965**.



Deep Learning

1

Atualmente, o processamento de Big Data e a evolução da Inteligência Artificial são ambos dependentes da Aprendizagem Profunda.

2

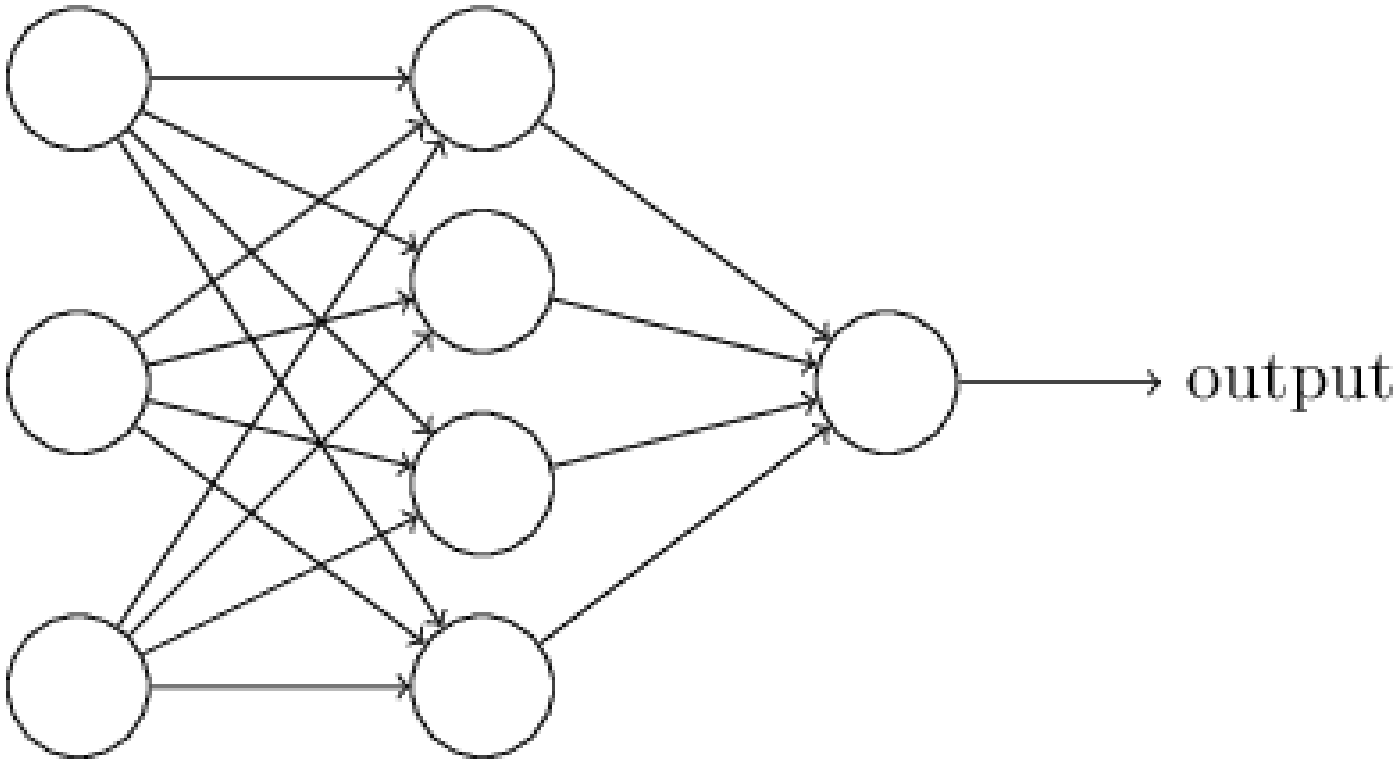
Com Deep Learning podemos construir sistemas inteligentes e estamos nos aproximando da criação de uma IA totalmente autônoma.

3

Isso vai gerar impacto em todas os segmentos da sociedade e aqueles que souberem trabalhar com a tecnologia, serão os líderes desse novo mundo que se apresenta diante de nós.

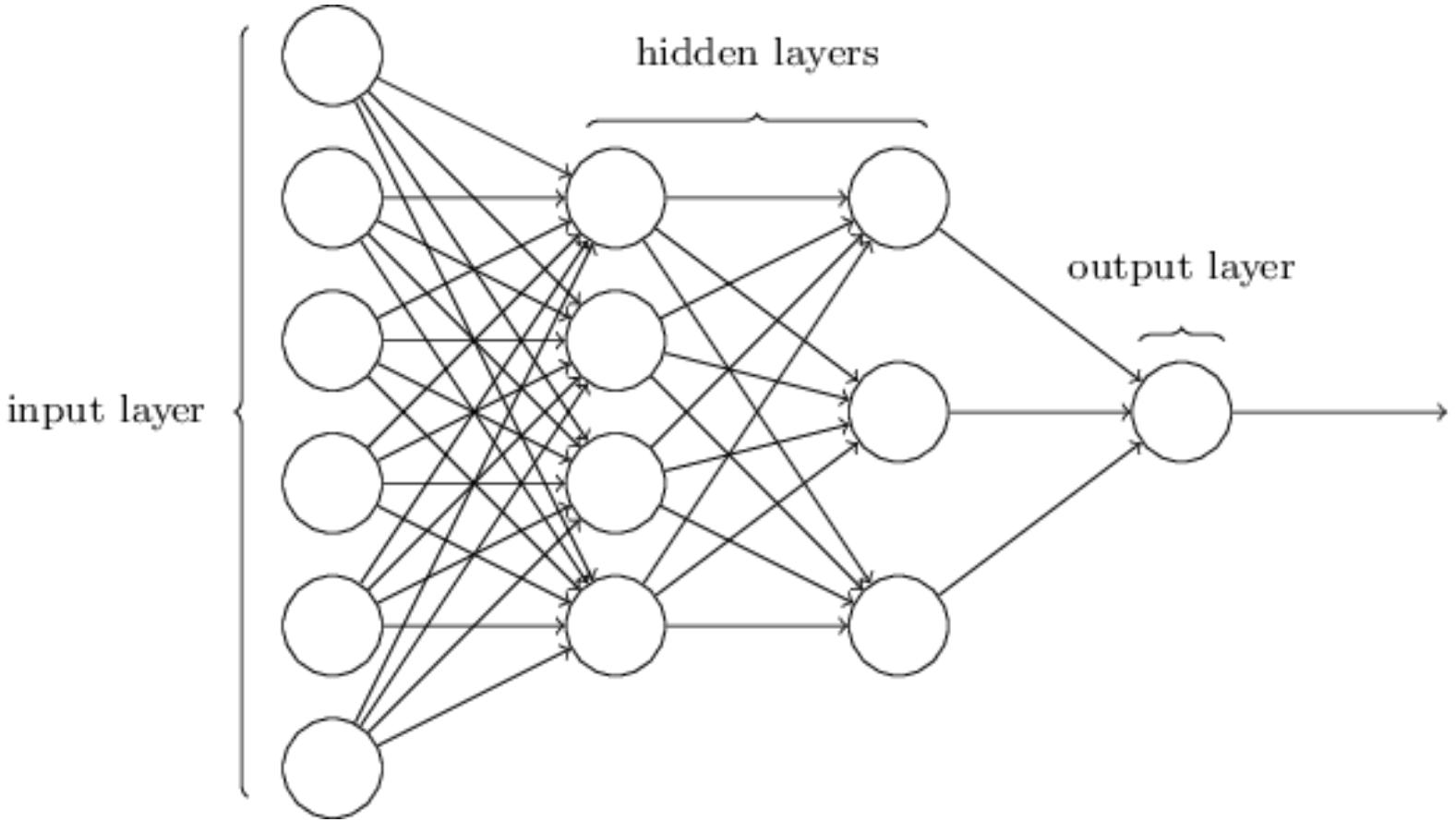
A Arquitetura das Redes Neurais

Suponha a seguinte rede:



A Arquitetura das Redes Neurais

A seguinte rede de quatro camadas tem duas camadas ocultas:



A Arquitetura das Redes Neurais

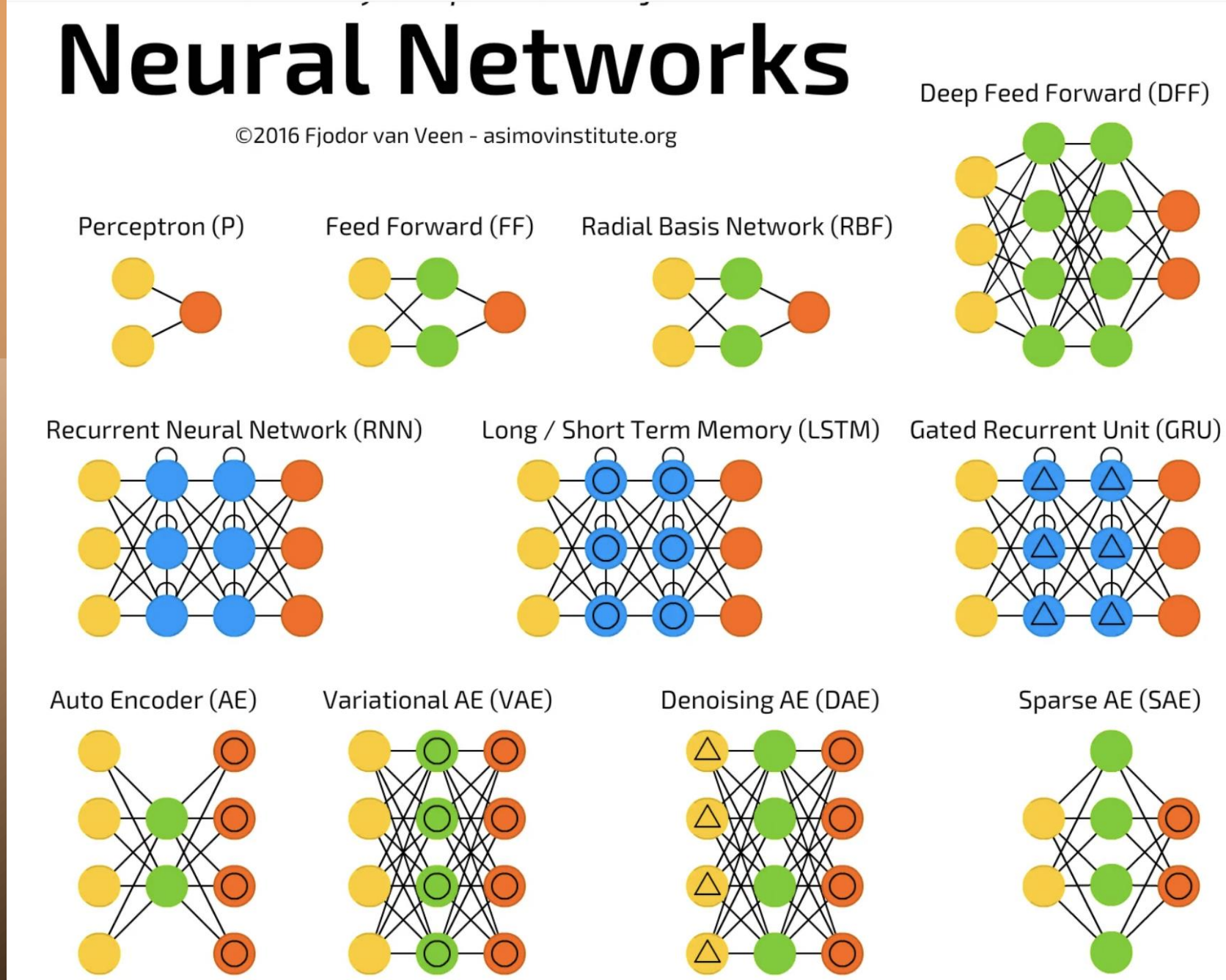
Geralmente, as arquiteturas de redes neurais podem ser colocadas em 3 categorias específicas:

1 – Redes Neurais Feed-Forward

2 – Redes Recorrentes

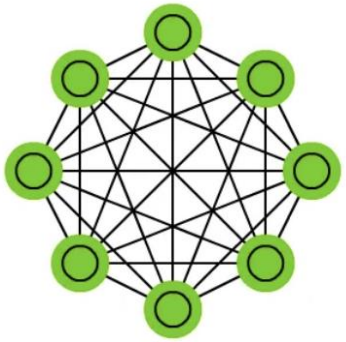
3 – Redes Conectadas Simetricamente

Arquiteturas de redes neurais:

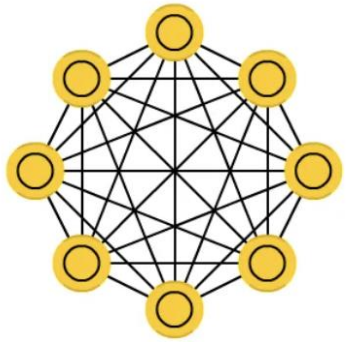


Arquiteturas de redes neurais:

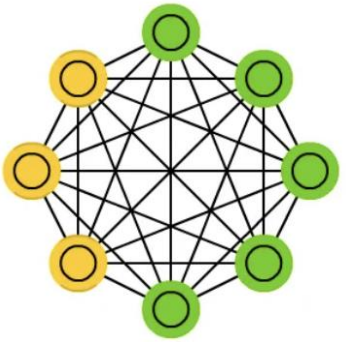
Markov Chain (MC)



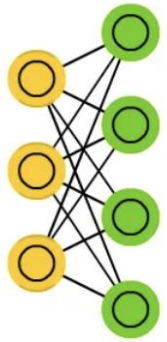
Hopfield Network (HN)



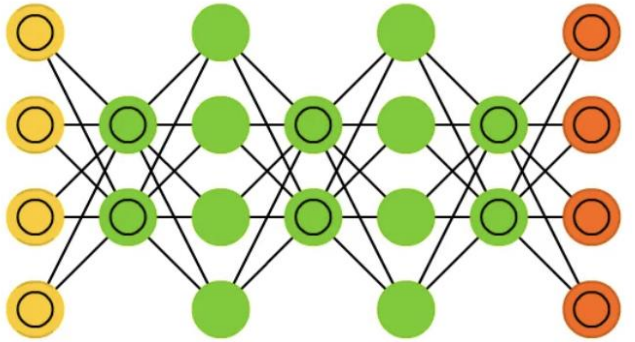
Boltzmann Machine (BM)



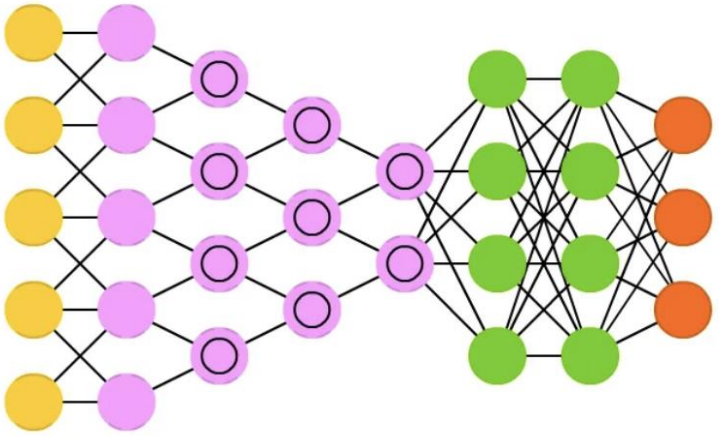
Restricted BM (RBM)



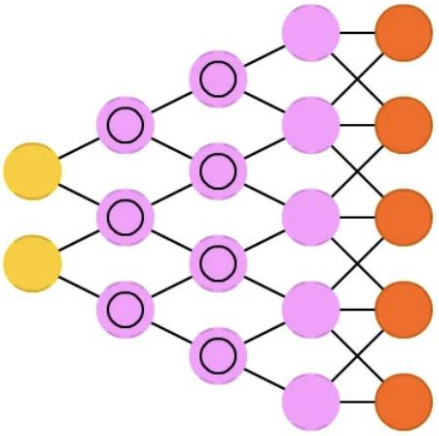
Deep Belief Network (DBN)



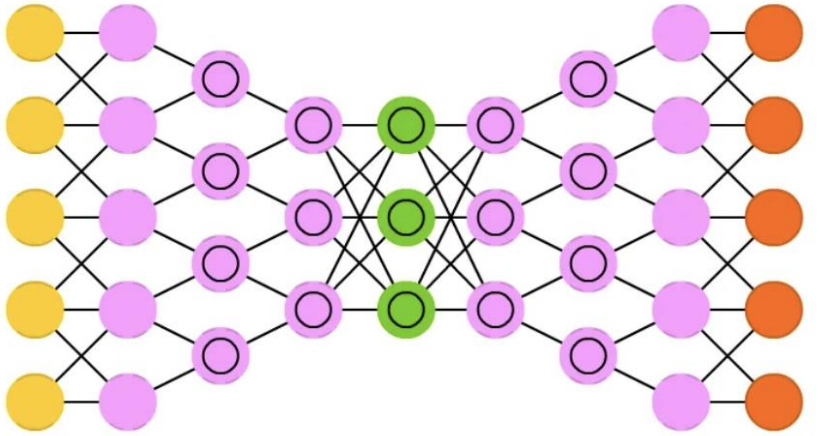
Deep Convolutional Network (DCN)



Deconvolutional Network (DN)

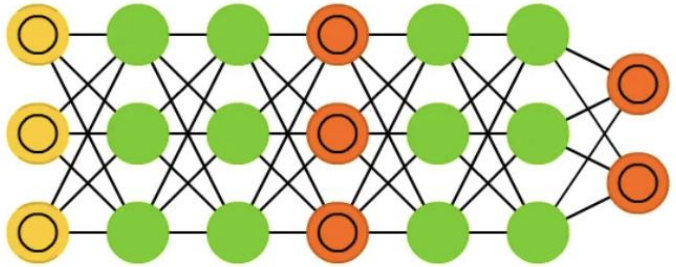


Deep Convolutional Inverse Graphics Network (DCIGN)

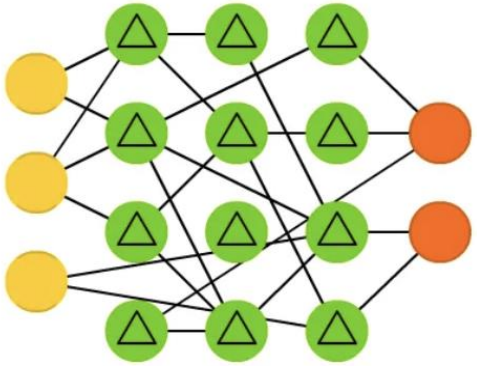


Arquiteturas de redes neurais:

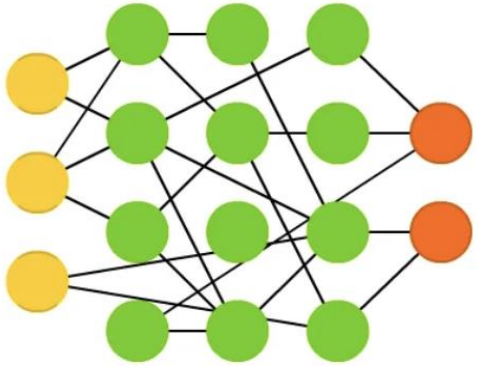
Generative Adversarial Network (GAN)



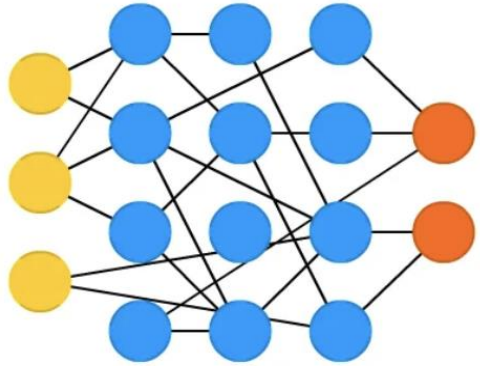
Liquid State Machine (LSM)



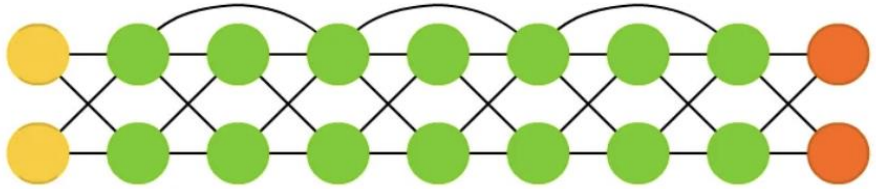
Extreme Learning Machine (ELM)



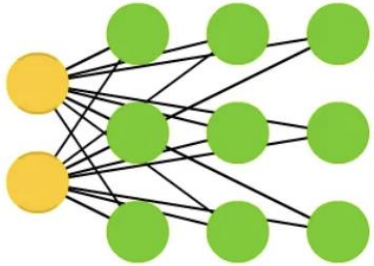
Echo State Network (ESN)



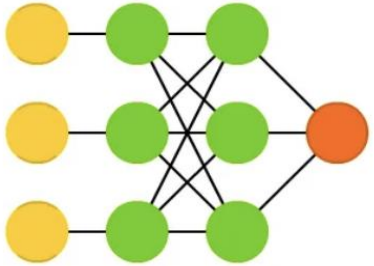
Deep Residual Network (DRN)



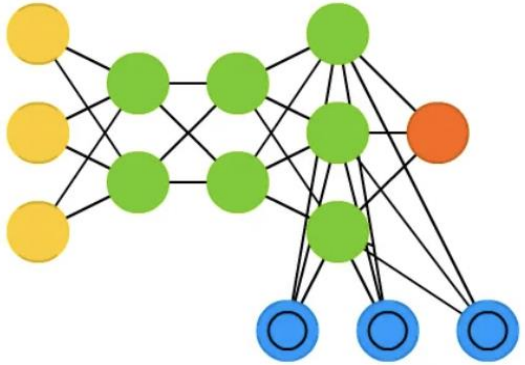
Kohonen Network (KN)



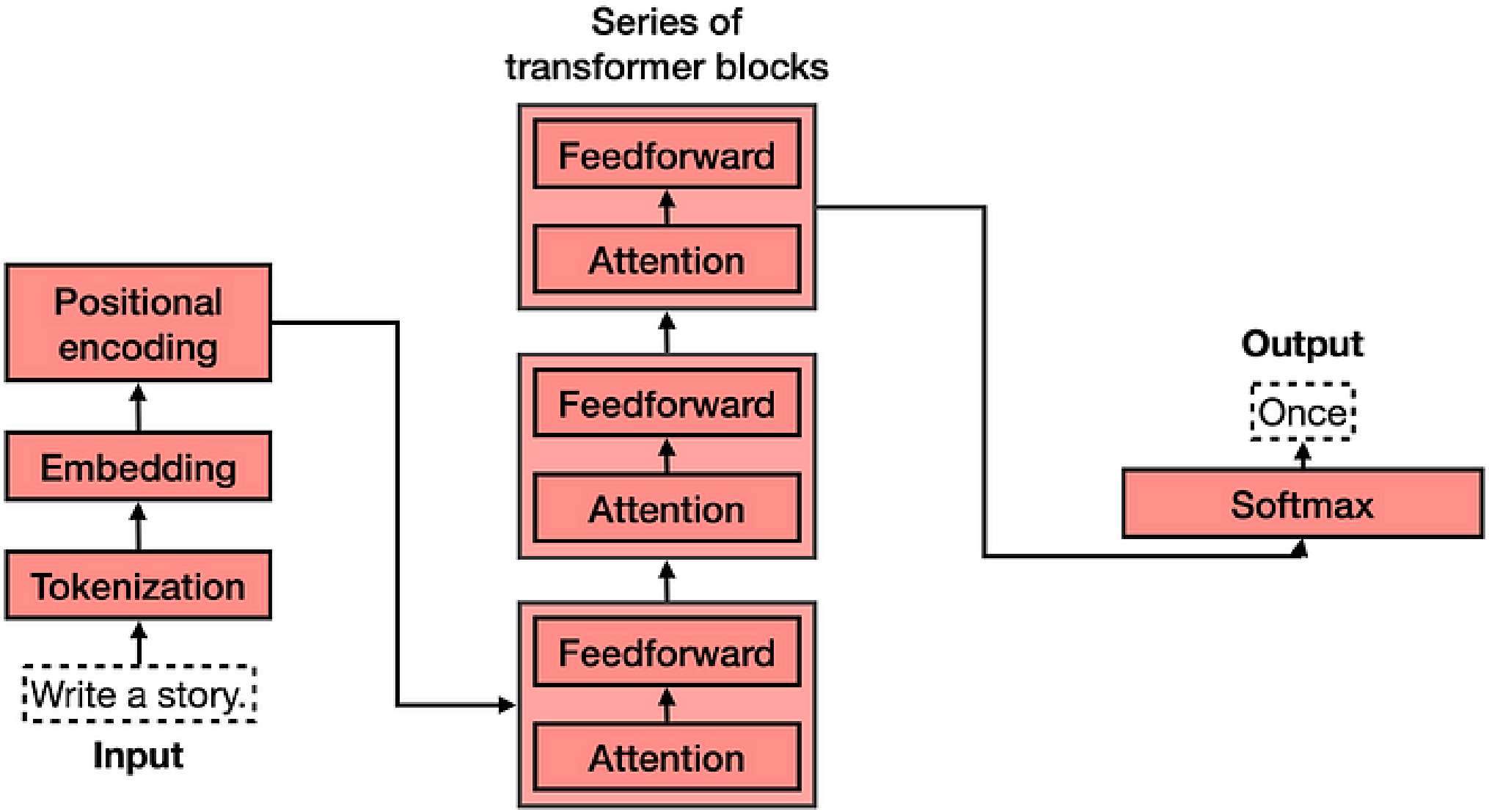
Support Vector Machine (SVM)



Neural Turing Machine (NTM)



Arquiteturas de redes neurais:



Arquiteturas de redes neurais:

Redes Multilayer Perceptron

Redes Neurais Convolucionais

Redes Neurais Recorrentes

Long Short-Term Memory (LSTM)

Redes de Hopfield

Máquinas de Boltzmann

Deep Belief Network

Deep Auto-Encoders

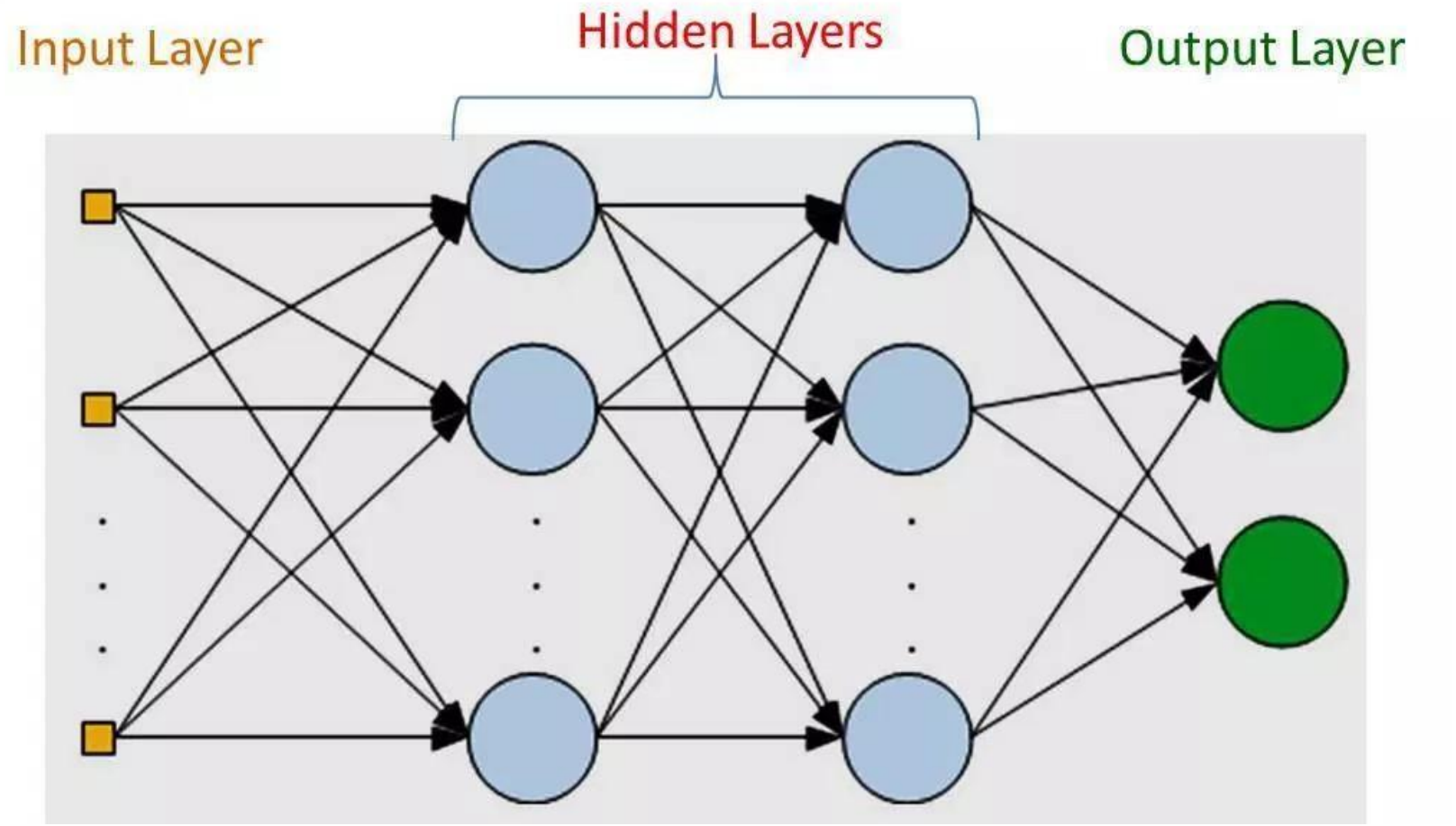
Generative Adversarial Network

Deep Neural Network Capsules (2017)

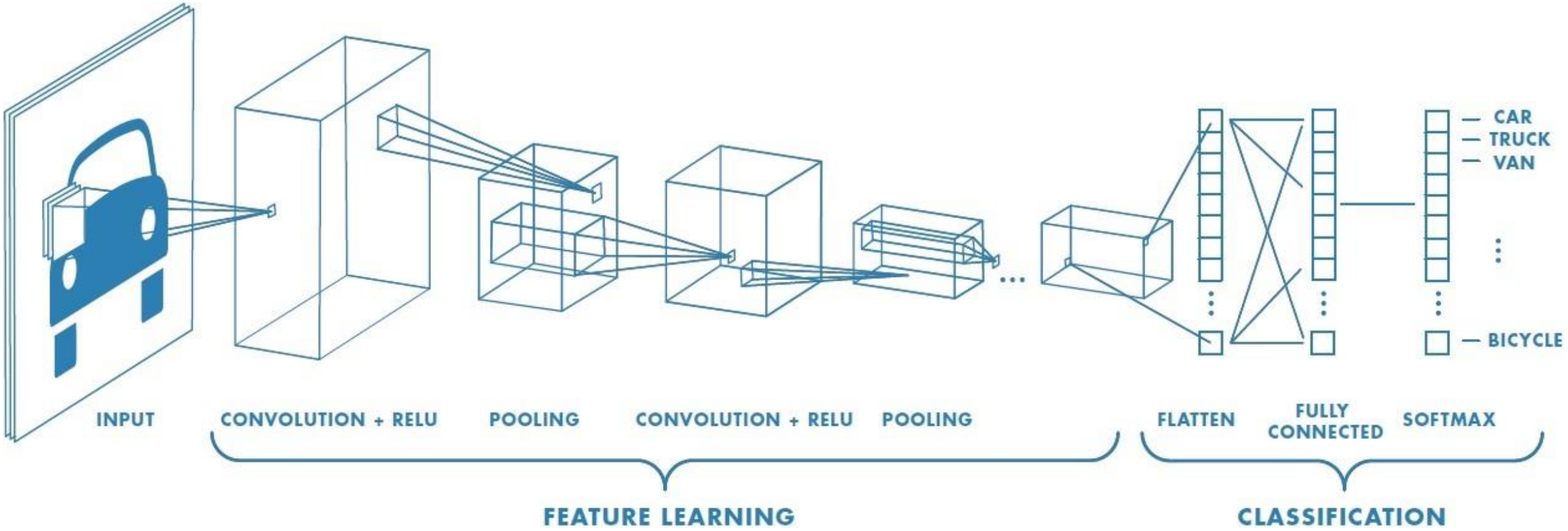
Transformers (ChatGPT, BERT, llama)

Visual Transformers - ViT

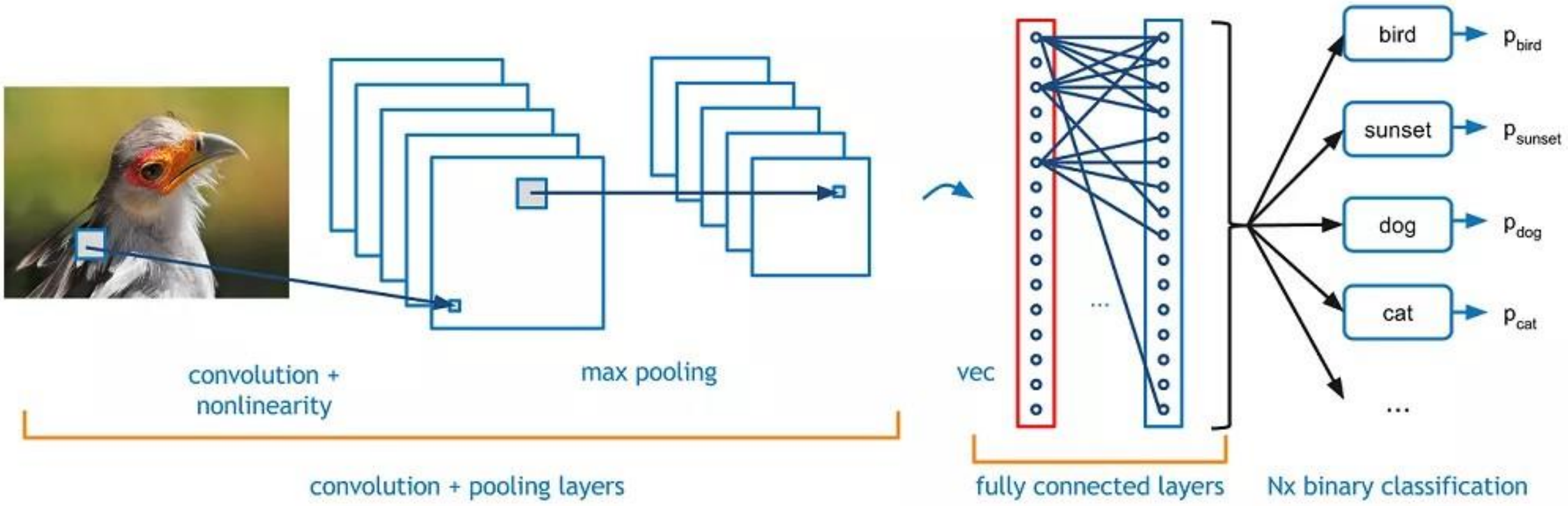
1- Redes Multilayer Perceptrons (MLP)



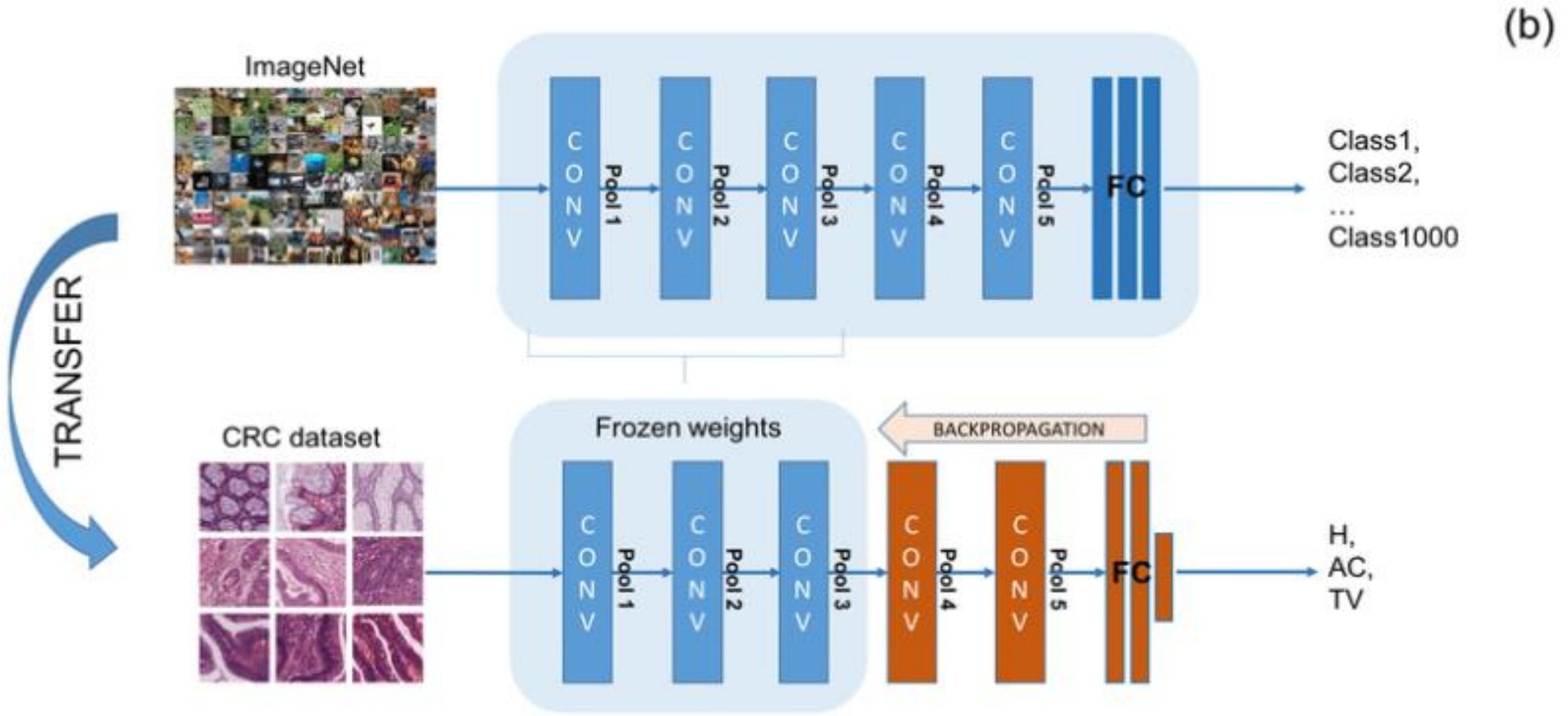
2- Redes Neurais Convolucionais (CNN)



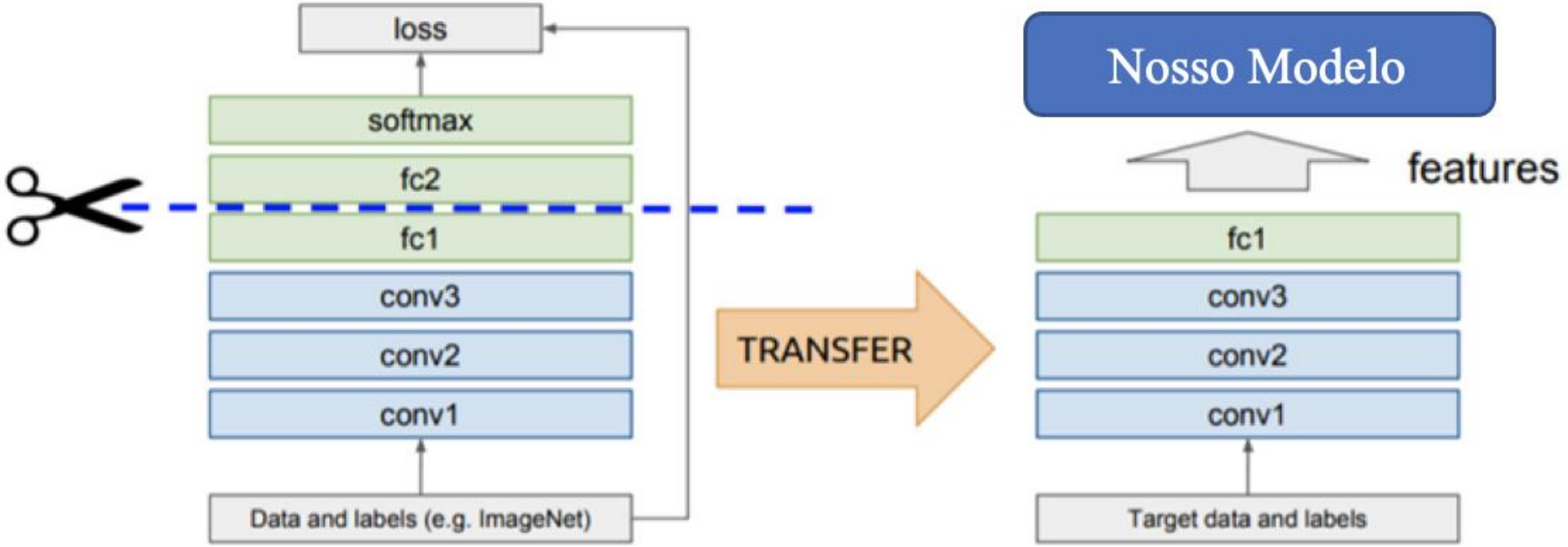
2- Redes Neurais Convolucionais (CNN)



2- CNN – Transfer Learning



2- CNN – Transfer Learning

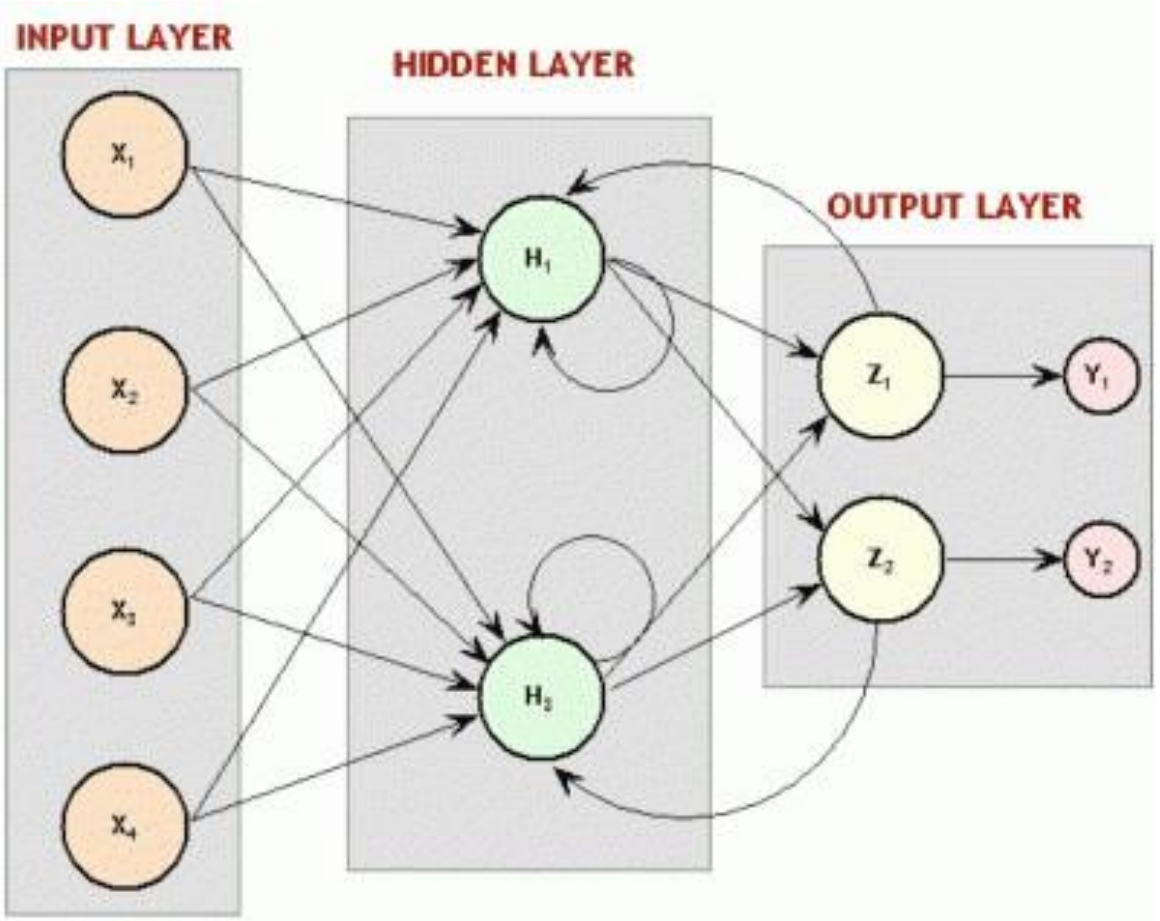


3- Redes Neurais Recorrentes (RNN)

As redes recorrentes são um poderoso conjunto de algoritmos de redes neurais artificiais especialmente úteis para o processamento de dados sequenciais, como som, dados de séries temporais ou **linguagem natural**.

Uma versão de redes recorrentes foi usada pelo DeepMind no projeto de videogames com agentes autônomos.

3- Redes Neurais Recorrentes (RNN)

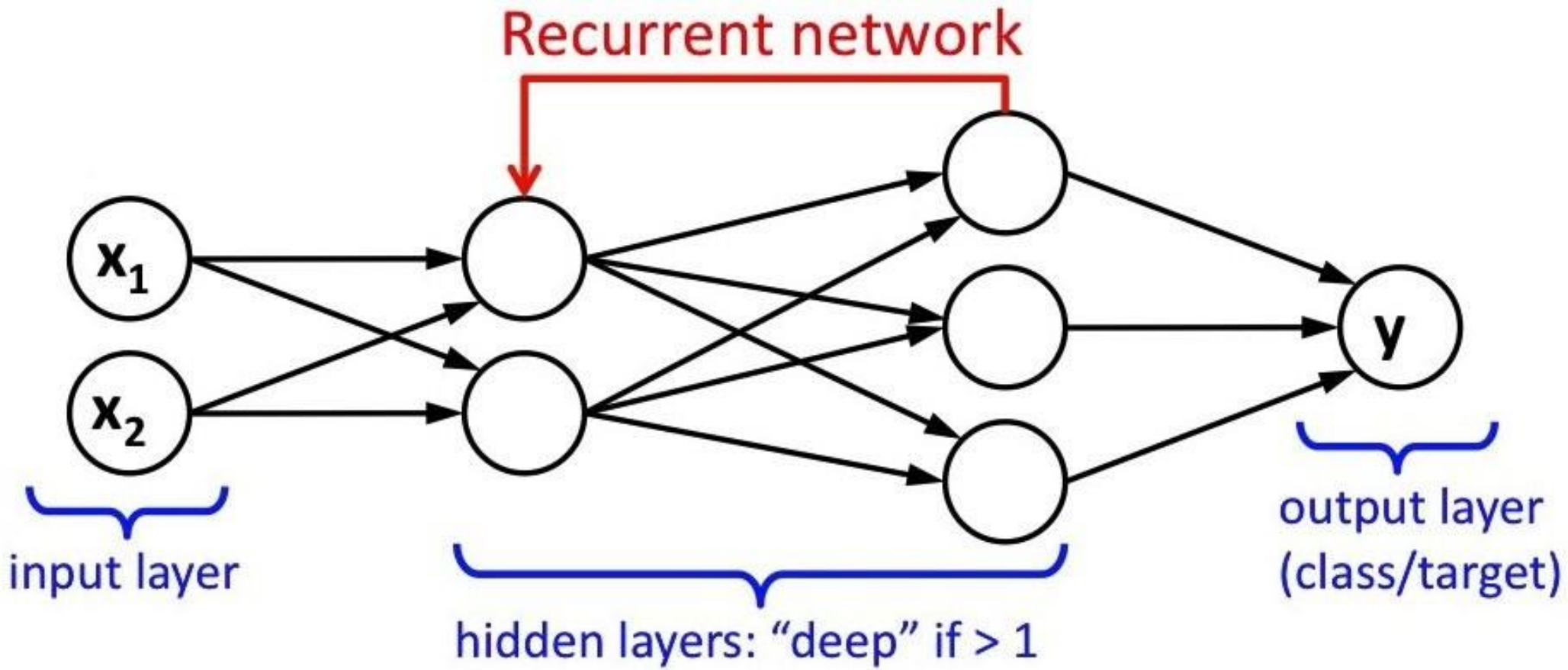


4- Long Short-Term Memory (LSTM)

Os LSTMs ajudam a preservar o erro que pode ser copiado por tempo e camadas. Ao manter um erro mais constante, eles permitem que as redes recorrentes continuem aprendendo durante vários passos de tempo (mais de 1000), abrindo assim um canal para vincular causas e efeitos remotamente.

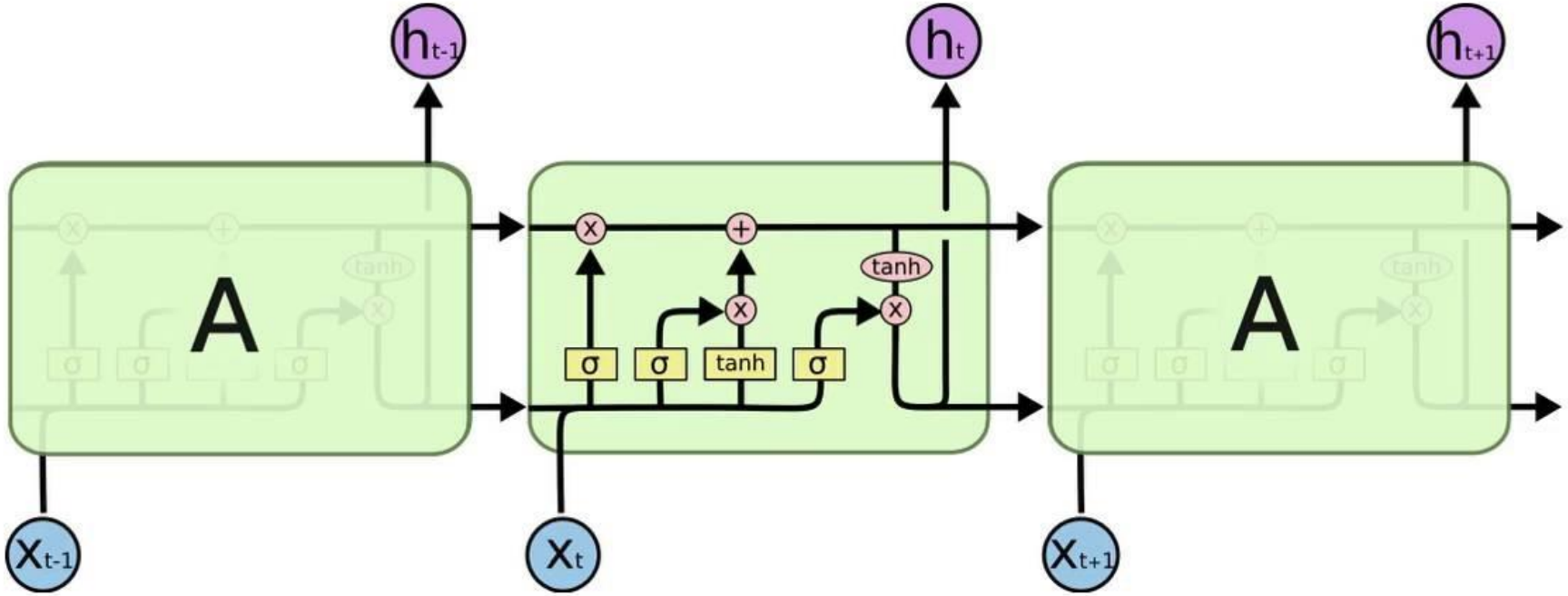
Este é um dos desafios centrais para a aprendizagem de máquina e a IA, uma vez que os algoritmos são frequentemente confrontados por ambientes onde os sinais de recompensa são escassos e atrasados.

4- Long Short-Term Memory (LSTM)

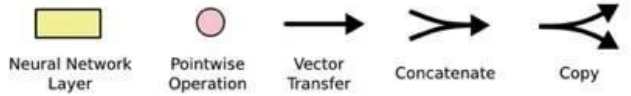


4- Long Short-Term Memory (LSTM)

Long-Short Term Memory module: LSTM

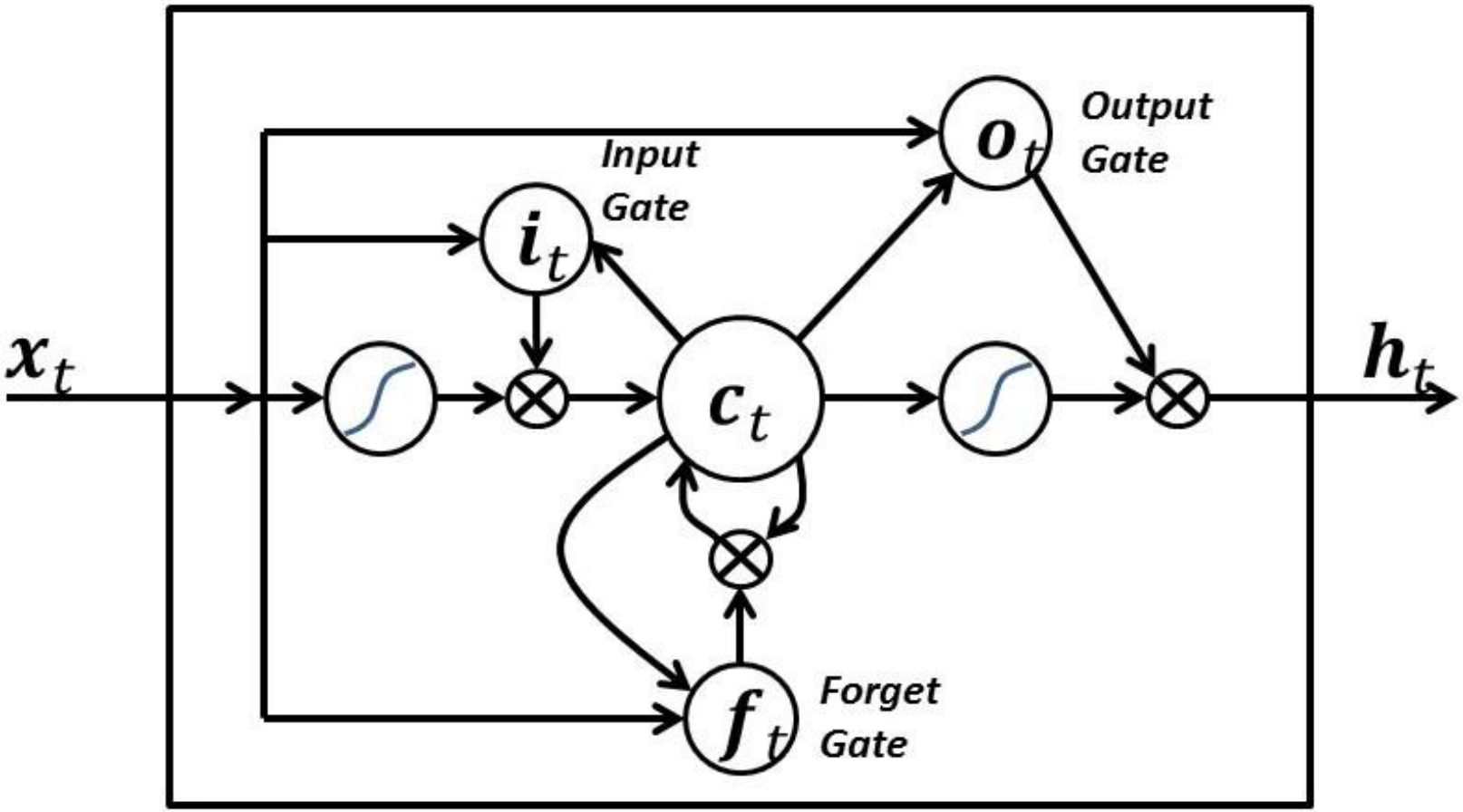


long-short term memory modules used in an RNN

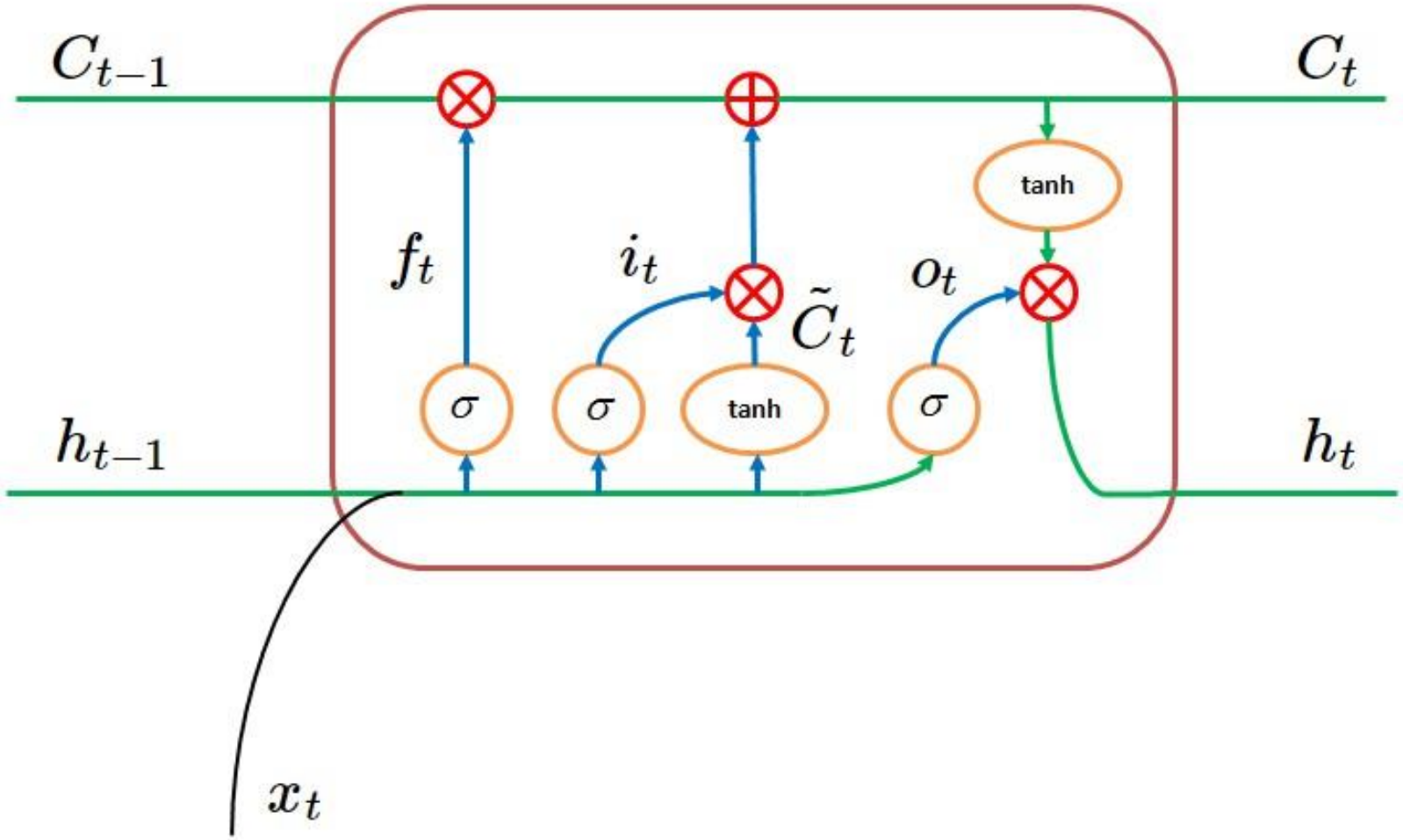


http://colah.github.io/posts/2015-08-Understanding-LSTMs/

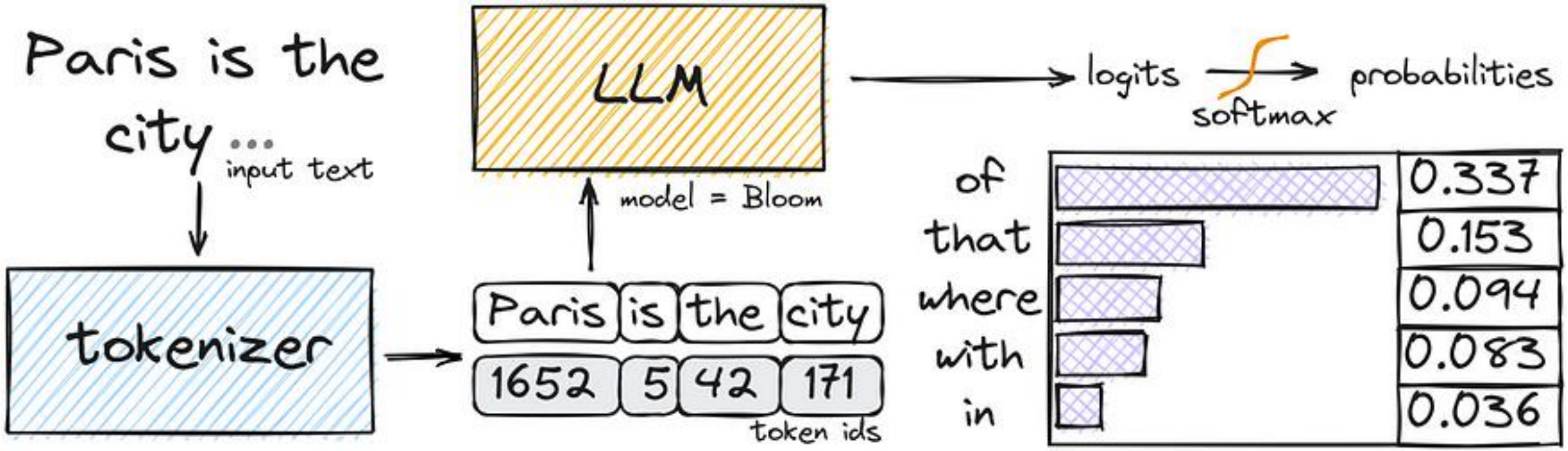
4- Long Short-Term Memory (LSTM)



4- Long Short-Term Memory (LSTM)

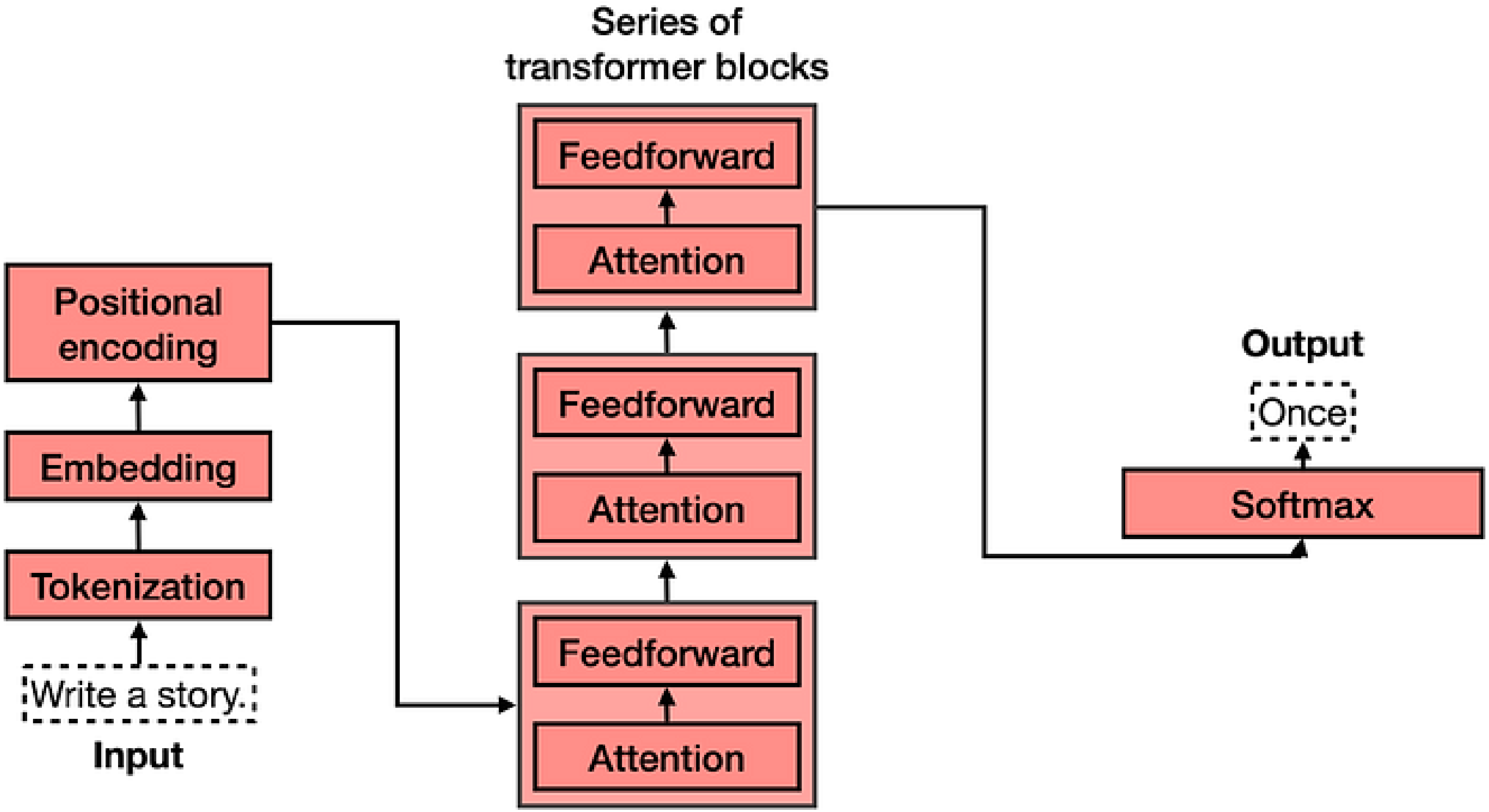


Grandes Modelos de Linguagem – LLM - Transformers

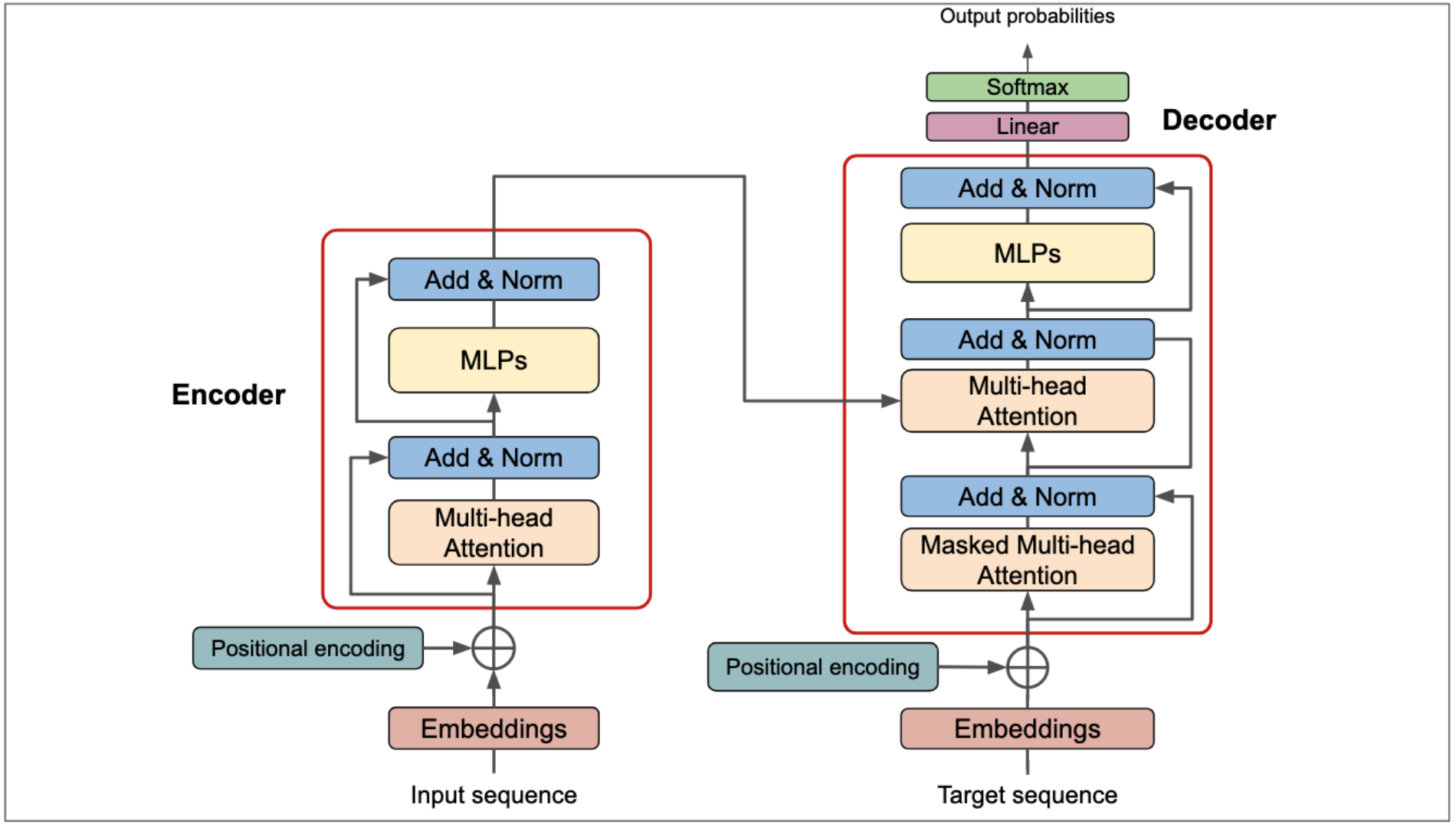


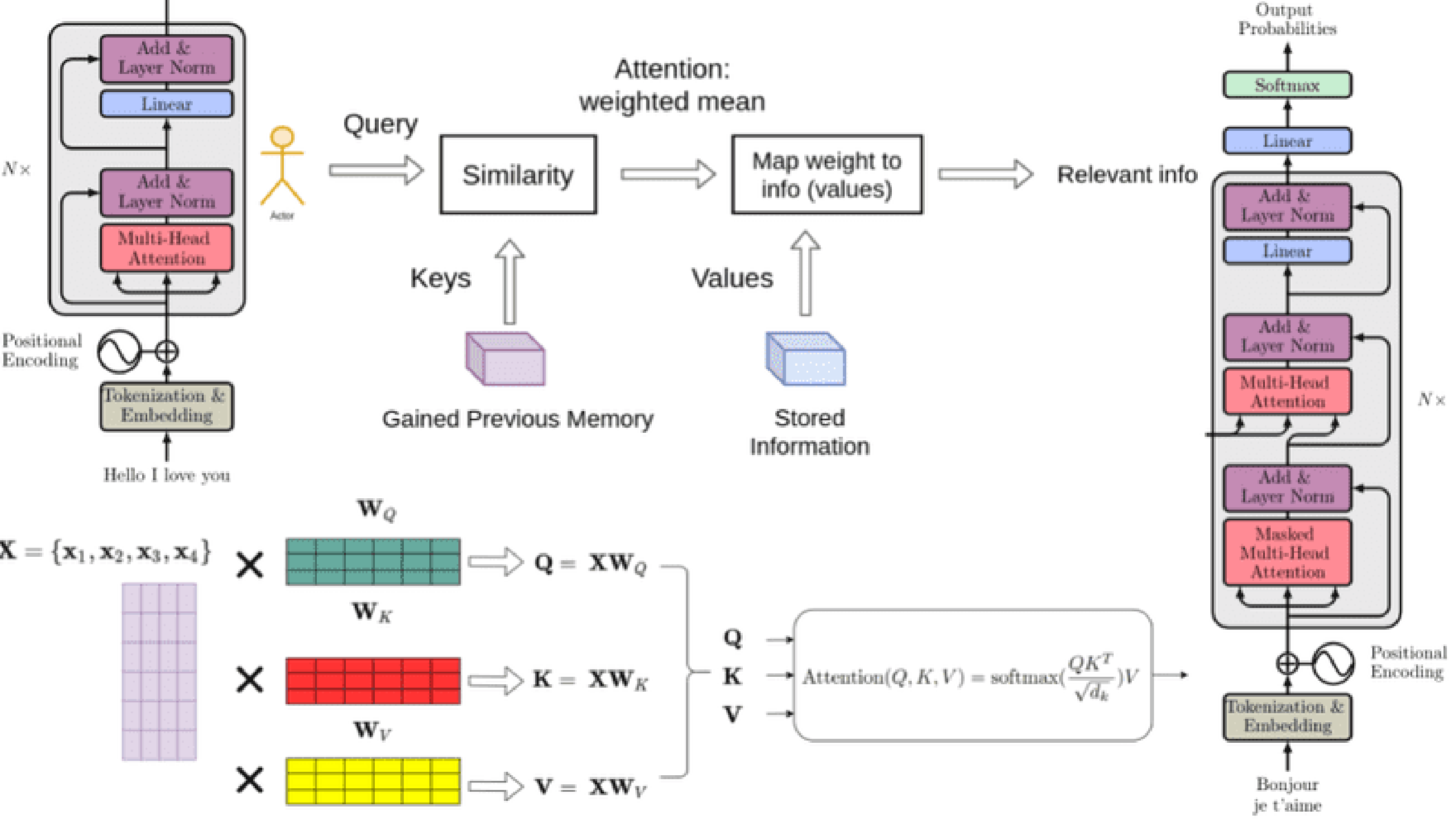
Embedding

Arquiteturas dos Transformers

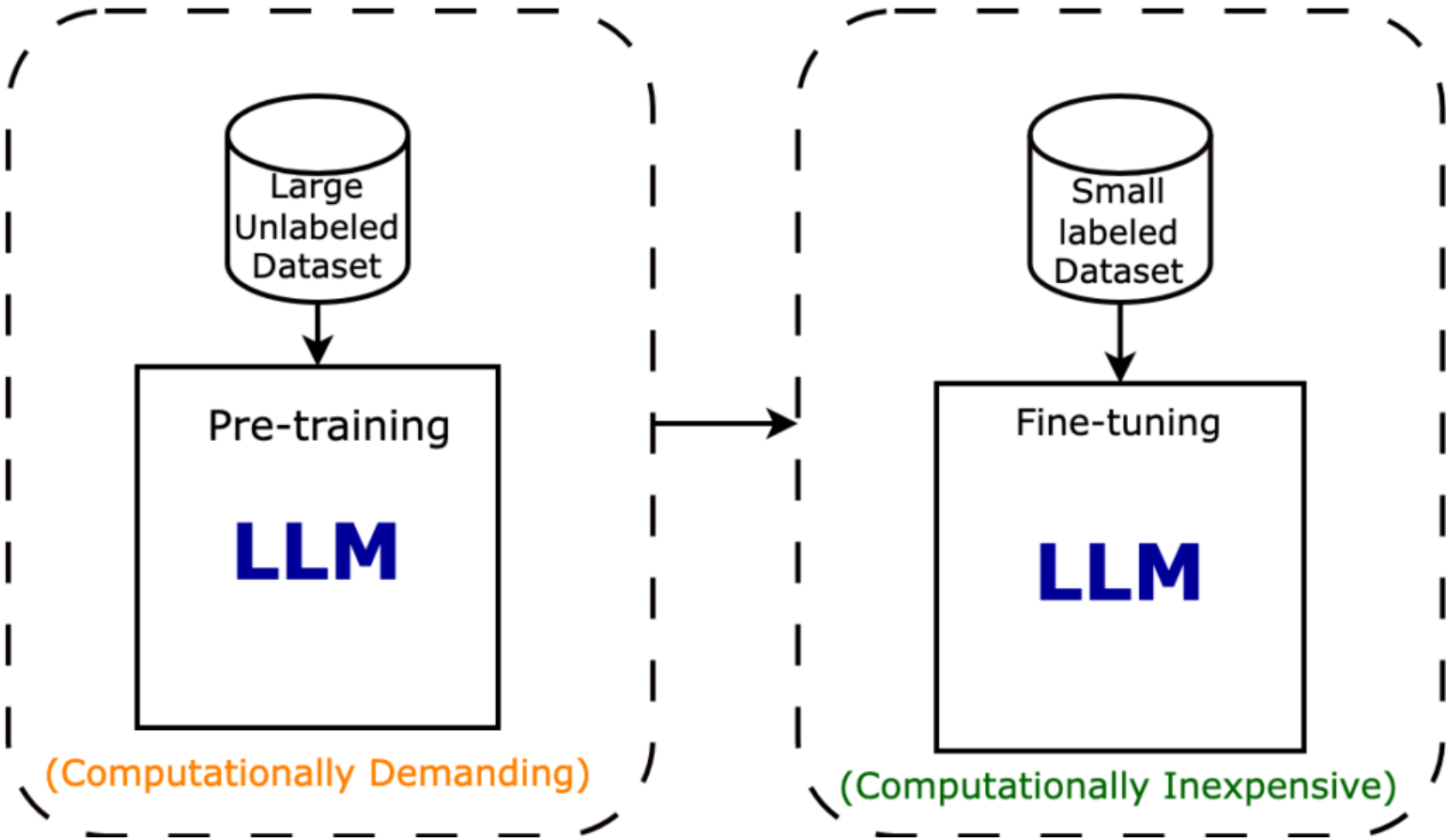


Transformers

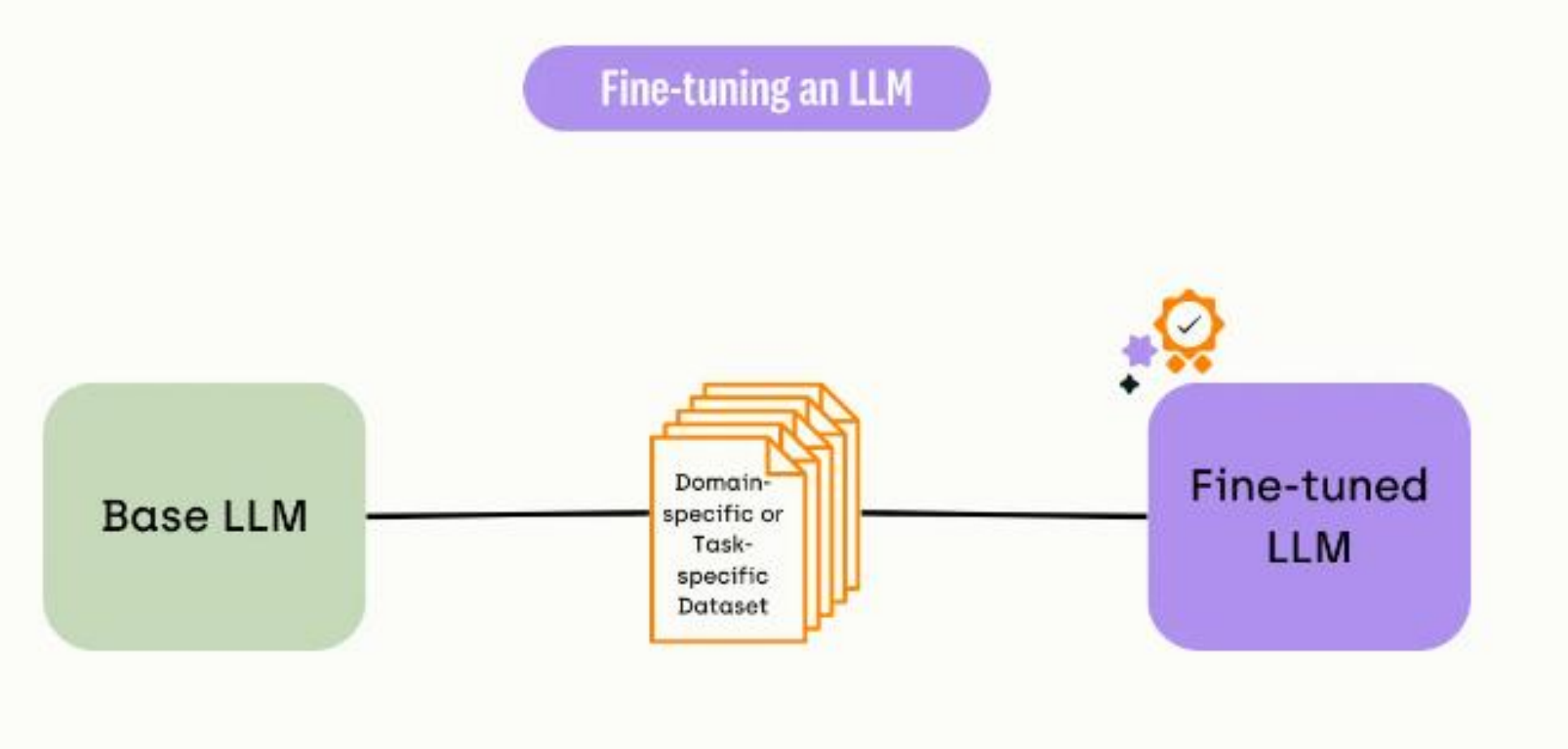



Add & Layer Norm
Linear
Add & Layer Norm
Multi-Head Attention
N×
Positional Encoding
Tokenization & Embedding
Hello I love you
Actor
Attention: weighted mean
Query
Similarity
Map weight to info (values)
Relevant info
Keys
Values
Gained Previous Memory
Stored Information
Output Probabilities
Softmax
Linear
Add & Layer Norm
Linear
Add & Layer Norm
Multi-Head Attention
Add & Layer Norm
Masked Multi-Head Attention
N×
Positional Encoding
Tokenization & Embedding
Bonjour je t'aime
$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$
$\mathbf{W}_Q$
$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$
$\mathbf{W}_K$
$\mathbf{K} = \mathbf{X}\mathbf{W}_K$
$\mathbf{W}_V$
$\mathbf{V} = \mathbf{X}\mathbf{W}_V$
$\mathbf{Q}$
$\mathbf{K}$
$\mathbf{V}$
$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

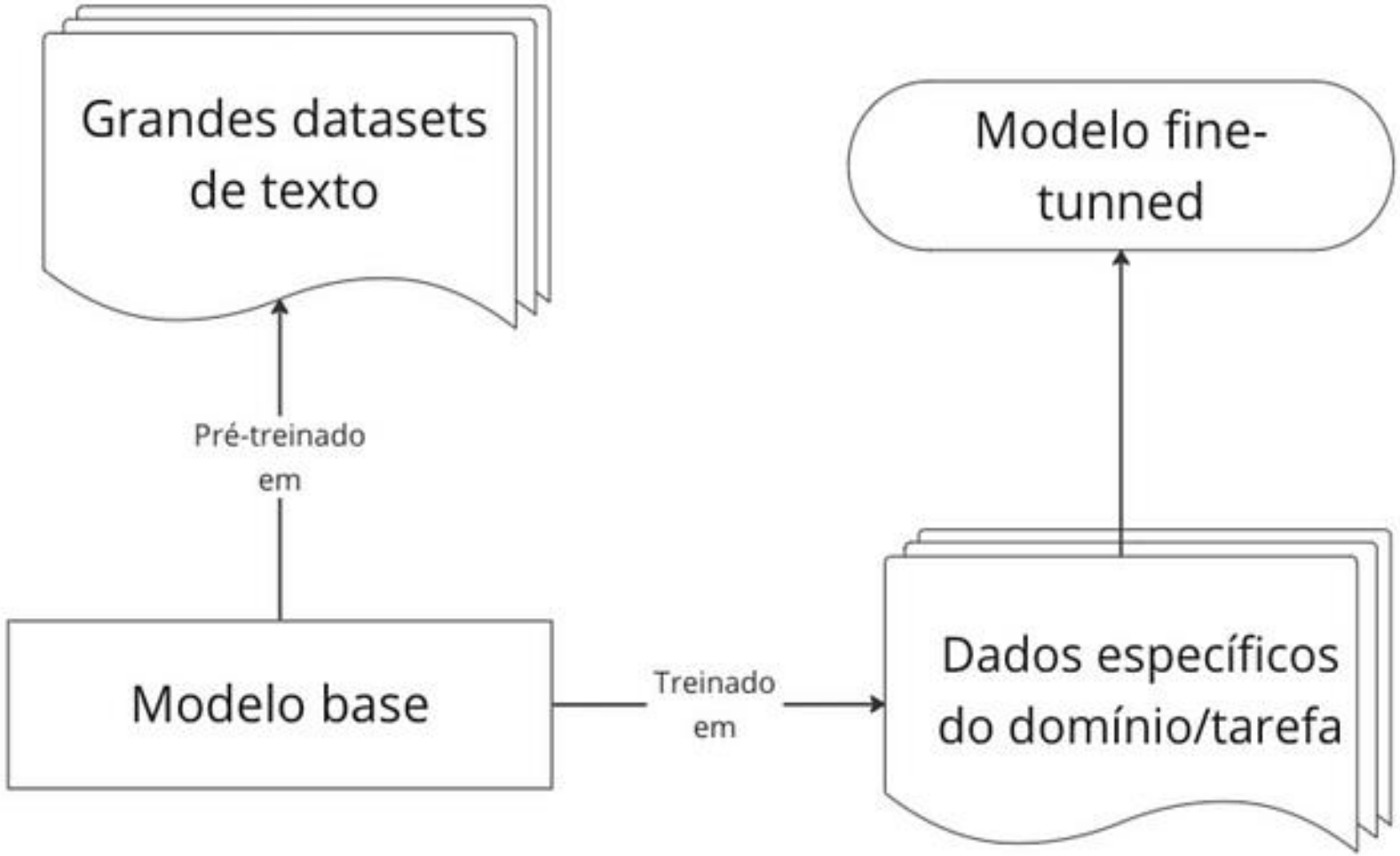
LLM Models


Large
Unlabeled
Dataset
Pre-training
LLM
(Computationally Demanding)
Small
labeled
Dataset
Fine-tuning
LLM
(Computationally Inexpensive)

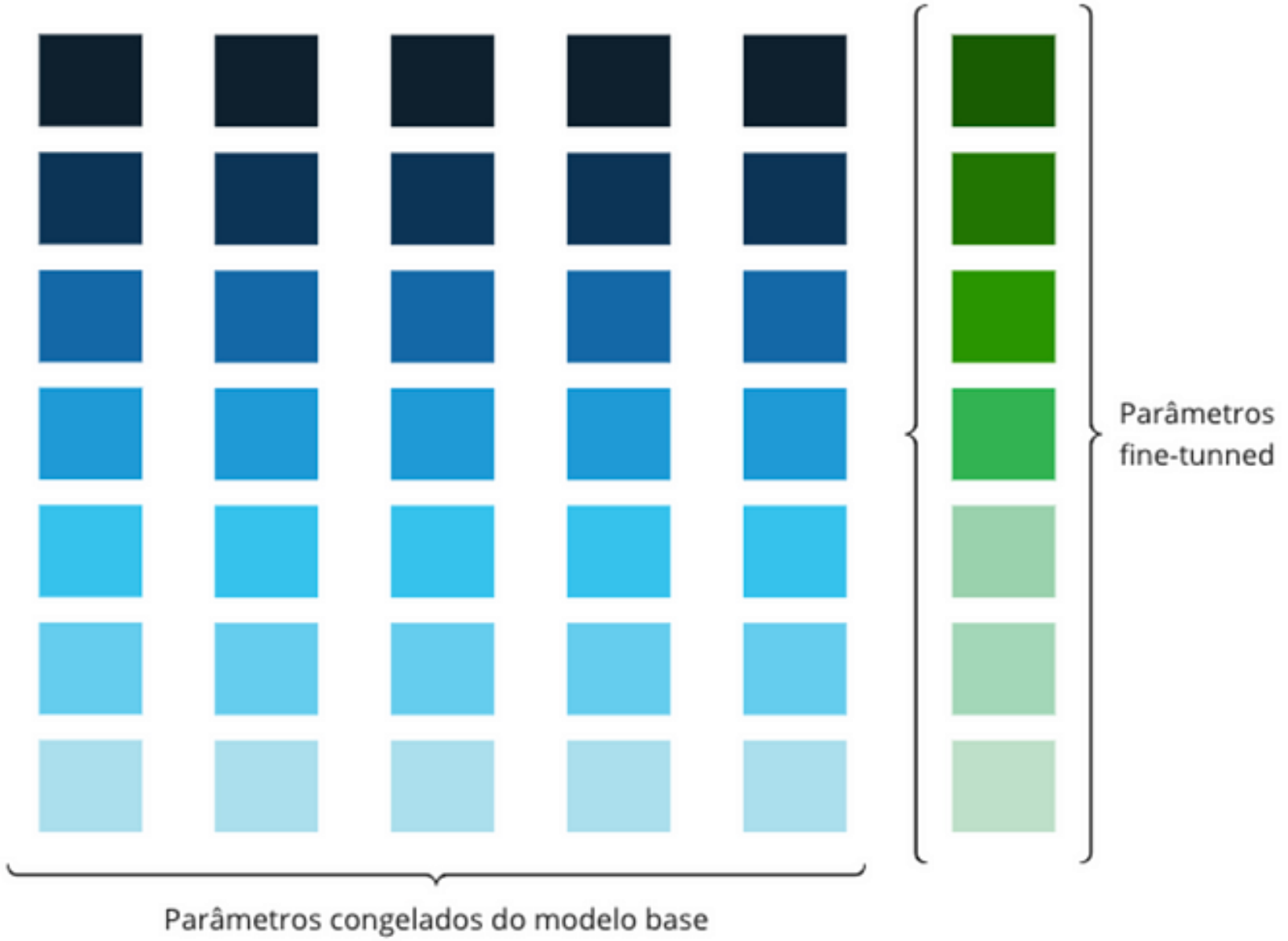
LLM – Fine Tunning



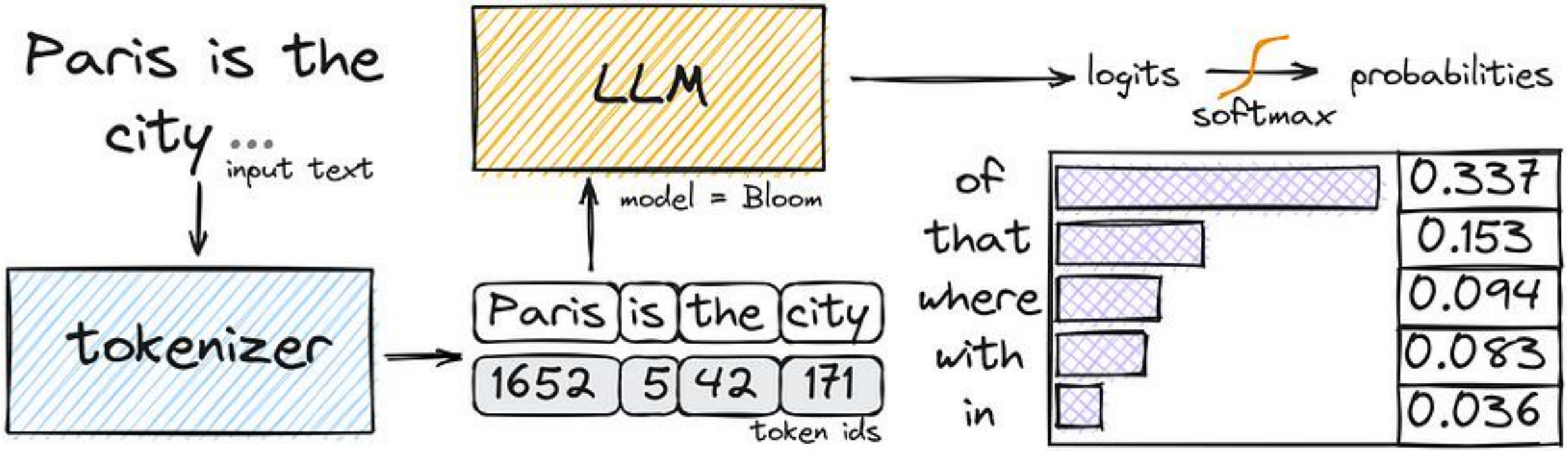
LLM – Fine Tunning



LLM – Fine Tunning



Grandes Modelos de Linguagem – LLM - Transformers

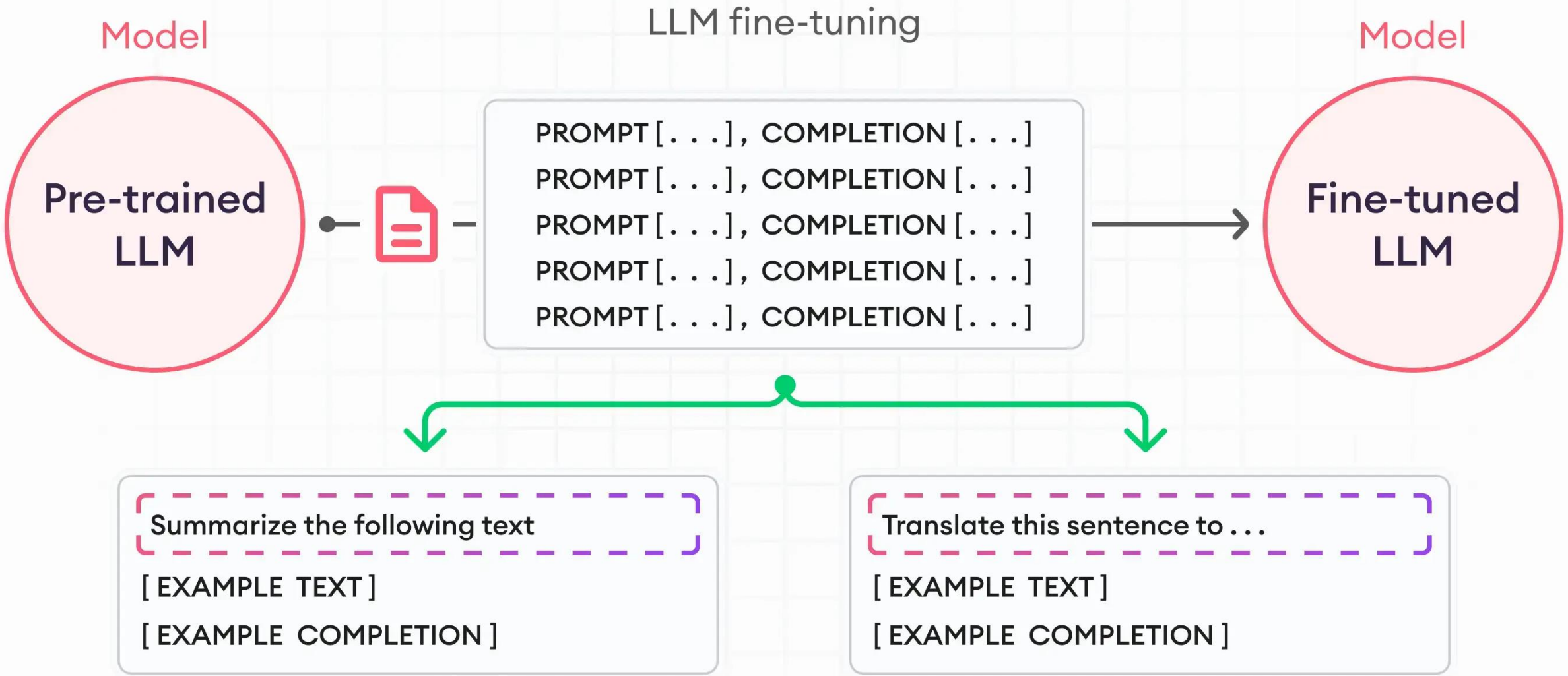


Embedding

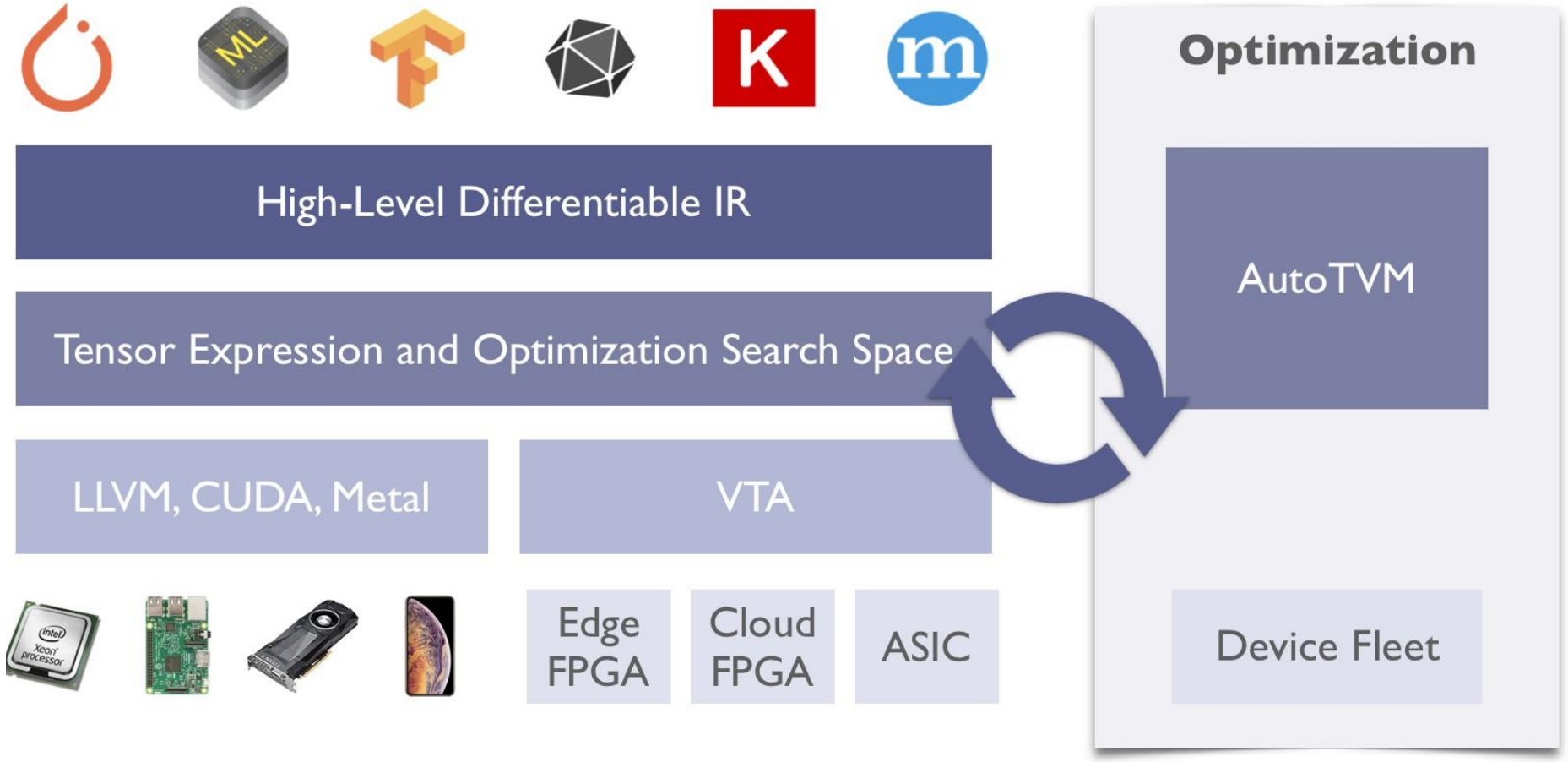
LLM – Fine Tunning

SuperAnnotate

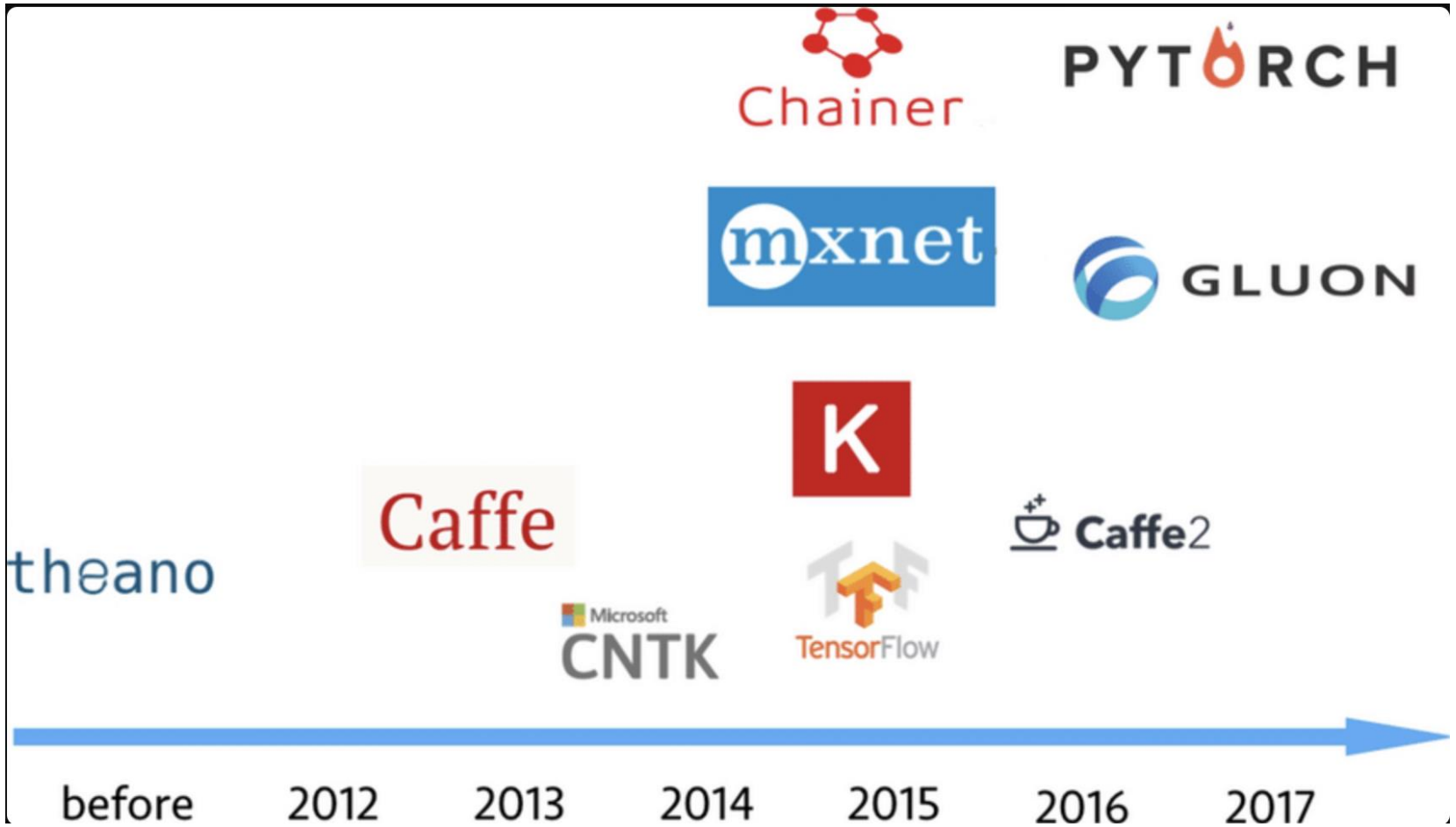
Using prompts to fine-tune LLMs with instruction



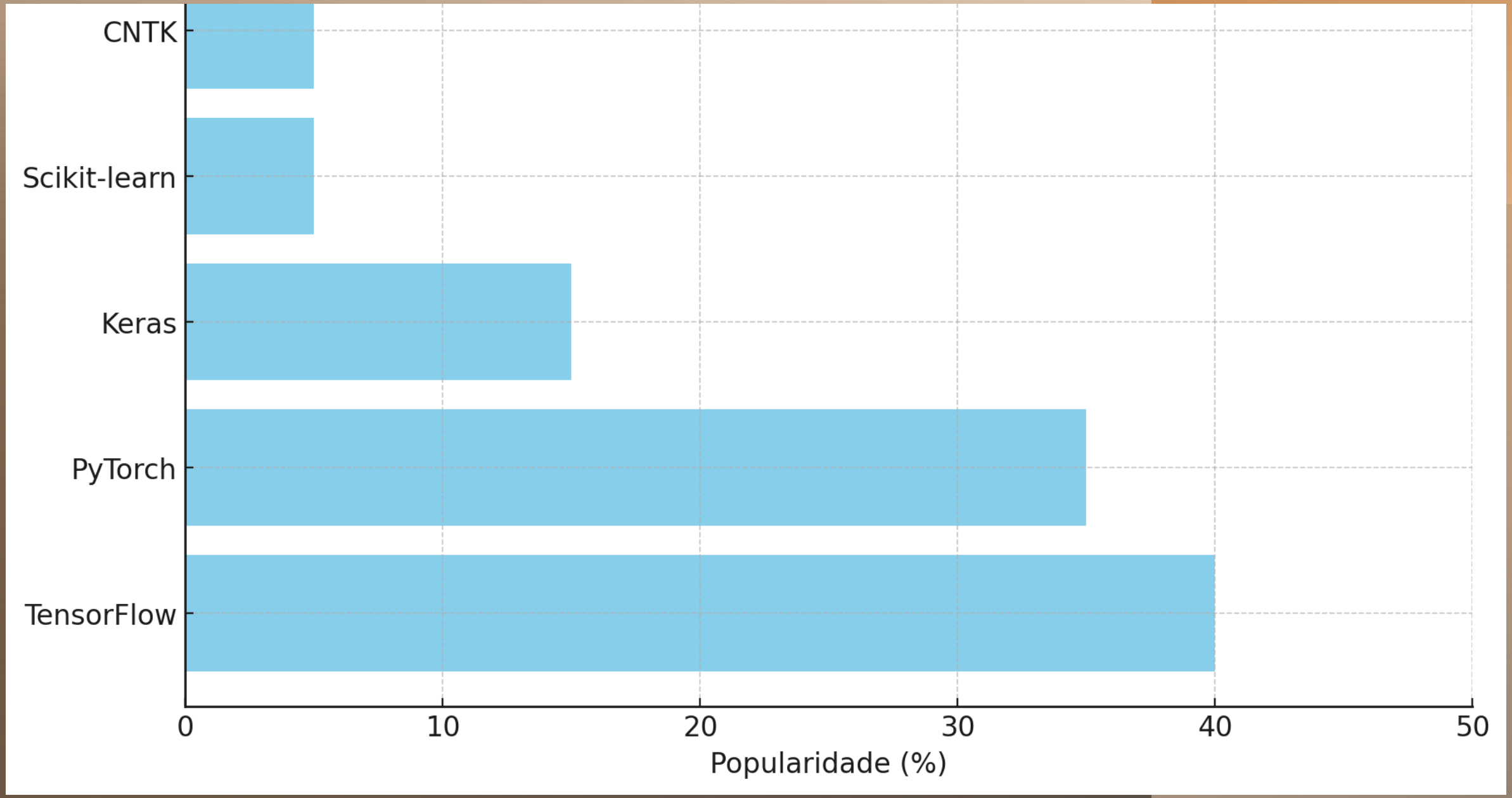
Frameworks de Deeplearning



Frameworks de Deep learning



Frameworks de Deep learning


CNTK
Scikit-learn
Keras
PyTorch
TensorFlow
0
10
20
30
40
50
Popularidade (%)

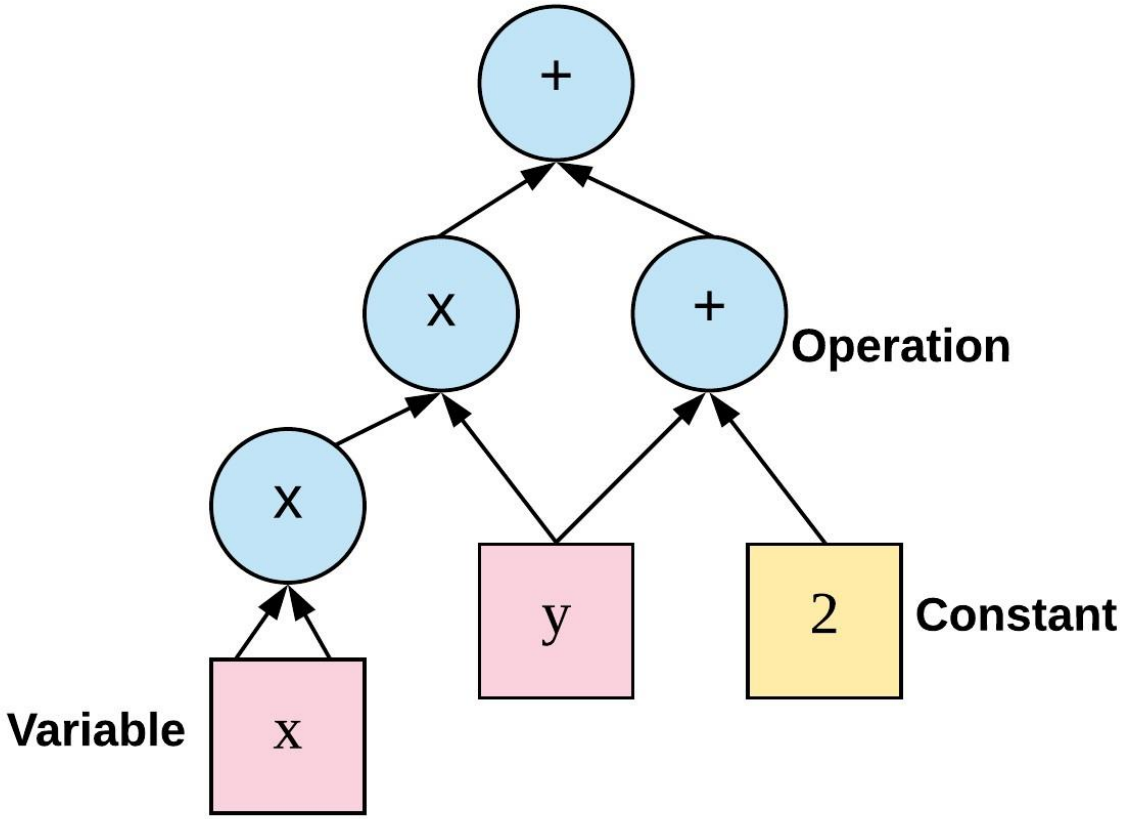
É uma biblioteca de software de código aberto para computação numérica usando grafos computacionais.

Foi originalmente desenvolvido pela Google Brain Team na organização de pesquisa Machine Intelligence do Google para aprendizado de máquina e pesquisa de redes neurais profundas (deep learning), mas a biblioteca é geral o suficiente para ser aplicada em uma grande variedade de outros domínios também.

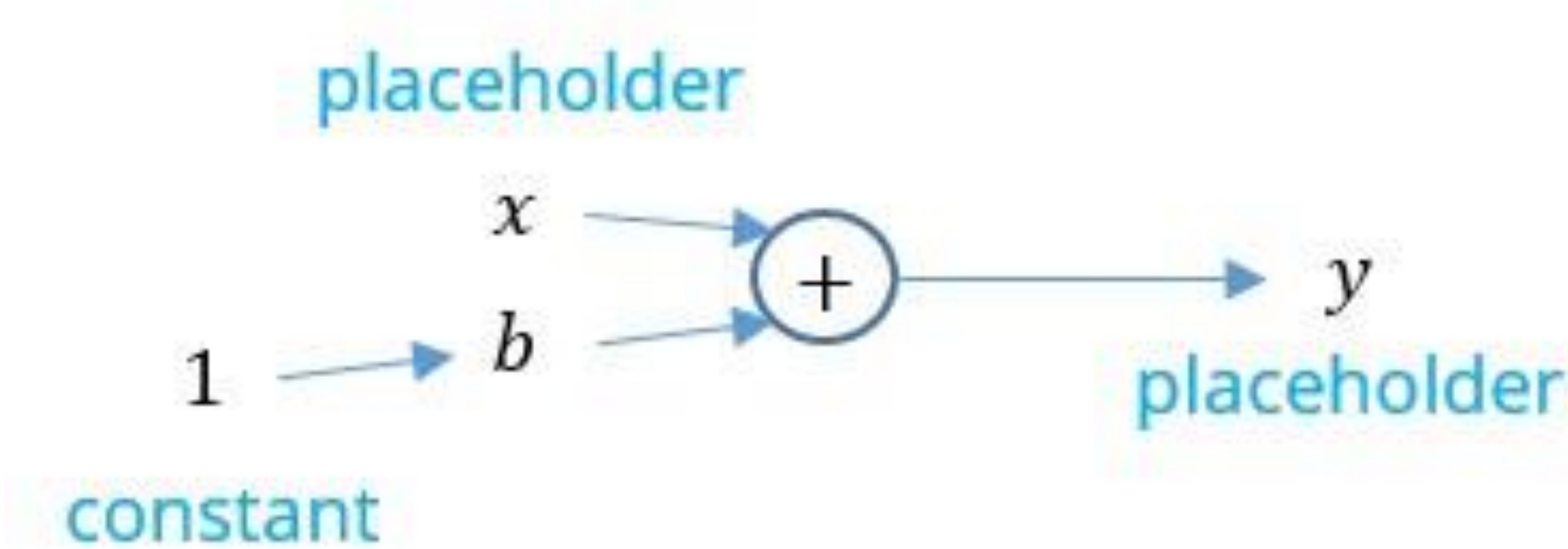
O TensorFlow permite que os desenvolvedores criem gráficos de fluxo de dados – estruturas que descrevem como os dados se movem em um gráfico ou uma série de nós de processamento.

Cada nó no gráfico representa uma operação matemática e cada conexão ou borda entre os nós é uma matriz de dados ou um tensor multidimensional.

Grafo Tensorflow

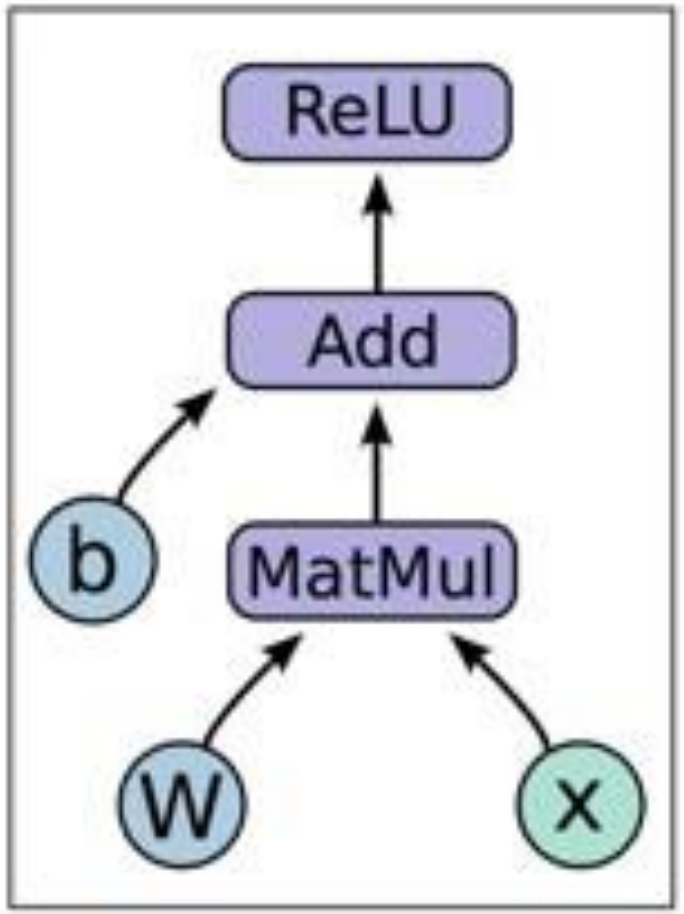


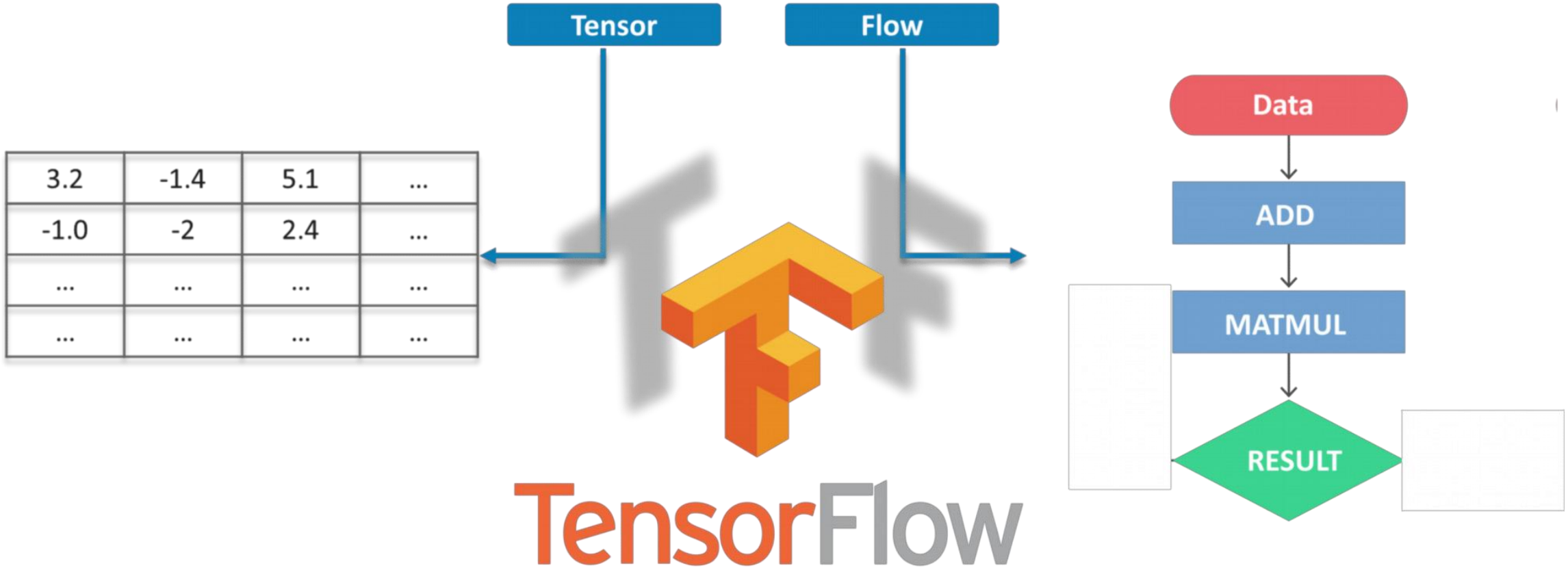
Grafo Tensorflow



Grafo Tensorflow

$$h = ReLU(Wx + b)$$




Tensor
Flow
3.2
-1.4
5.1
...
-1.0
-2
2.4
...
...
...
...
...
...
...
...
TensorFlow
Data
ADD
MATMUL
RESULT

```python
import tensorflow as tf
x = tf.placeholder("float", [None, 4])
y = x * 10 + 1
with tf.Session() as session:
    dataX = [[12, 2, 0, -2],
             [14, 4, 1, 0],]
    placeX = session.run(y, feed_dict={x: dataX})
    print(placeX)
```

```
[[ 121.   21.    1.  -19.]
 [ 141.   41.   11.    1.]]
```

TensorFlow with Python – Variables

- Design the Graph

```
import tensorflow as tf                    # import the Tensorflow library

a  = tf.Variable( tf.float )               # define some variables
b  = tf.Variable( tf.float )
x  = tf.placeholder( tf.float, shape=[4] )
yhat = a  + b * x                          # simple linear regression
```

Variables must be explicitly initialized prior to running the graph

- Run the Graph

```
sess = tf.session()                                  # connect session for execution
init  = tf_global_variables_initializer()            # initialize global variables
sess.run(init)

sess.run( yhat, {x: [1,2,3,4]} )                     # run the graph
```

Run simple linear regression for x = 1, 2, 3 and 4

[0, 0.30000001, 0.60000001, 0.90000001]

Example from tensorflow.org

Keras

Keras é uma biblioteca de rede neural de código aberto escrita em Python.

É capaz de funcionar com o TensorFlow, o Microsoft Cognitive Toolkit, o Theano ou o PlaidML.

Projetado para permitir a experimentação rápida com redes neurais profundas, ele se concentra em ser fácil de usar, modular e extensível

```
# Create your first MLP in Keras
from keras.models import Sequential
from keras.layers import Dense
import numpy

# fix random seed for reproducibility
numpy.random.seed(7)

# load pima indians dataset
dataset = numpy.loadtxt("pima-indians-diabetes.csv", delimiter=",")

# split into input (X) and output (Y) variables
X = dataset[0:8]
y = dataset[8]
```

```
# create model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile model
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# Fit the model
model.fit(X, y, epochs=150, batch_size=10)

# evaluate the model
scores = model.evaluate(X, y)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
Acurácia: 0.741379310344827
Matriz de Confusão
[[60 18]
[12 26]]
```

PyTorch

PyTorch é uma biblioteca de aprendizado de máquina baseada na biblioteca Torch, usada para aplicações como visão computacional e processamento de linguagem natural, originalmente desenvolvida pela Meta AI e agora parte da Linux Foundation

```python
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix

# Definindo a rede neural simples
class SimpleNN(nn.Module):
  def __init__(self,input_features=8,hidden1=20, hidden2=10, out_features=2):
      super(SimpleNN, self).__init__()
      self.f_connected1 = nn.Linear(input_features,hidden1)
      self.f_connected2 = nn.Linear(hidden1,hidden2)
      self.out = nn.Linear(hidden2,out_features)

  def forward(self,x):
    x = F.relu(self.f_connected1(x))
    x = F.relu(self.f_connected2(x))
    x = self.out(x)
    return x

# Criando o modelo
model = SimpleNN()
```

```
# Definindo o otimizador e a função de perda
optimizer = torch.optim.Adam(model.parameters(),lr=0.01)
loss_function = nn.CrossEntropyLoss()

# lê o arquivo
atributos = [ 'preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class' ]
url = 'https://raw.githubusercontent.com/vladimiralencar/Projects-
2024/refs/heads/main/pytorch/pima-indians-diabetes.csv'
df = pd.read_csv(url, header=None)
df.columns = atributos


# split into input (X) and output (Y) variables
X = df[ ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age'] ].values
y = df['class'].values


# Split para treino e teste
from sklearn.model_selection import train_test_split
test_size = 0.15
X_train,X_test,y_train,y_test = train_test_split(X,y , test_size=test_size,
random_state=0)
```

```
# Creating Tensors
X_train=torch.FloatTensor(X_train)
X_test=torch.FloatTensor(X_test)
y_train=torch.LongTensor(y_train)
y_test=torch.LongTensor(y_test)


# Treinamento
epochs=500
final_losses=[]
for i in range(epochs):
  i= i+1
  y_pred=model.forward(X_train) # Faz a previsão
  loss=loss_function(y_pred,y_train) # Calcula a perda
  final_losses.append(loss)
  if i % 100 == 1:
    print("Epoch number: {} and the loss : {}".format(i,loss.item()))
  optimizer.zero_grad()# zera os gradientes antes do Backpropagation
  loss.backward()# Backpropagation
  optimizer.step()# atualiza os pesos
```

```
# Previsões - Dados de Teste
predictions = []
with torch.no_grad():
  for i,data in enumerate(X_test):
    y_pred = model(data)
    predictions.append(y_pred.argmax().item())


accuracy = accuracy_score(y_test,predictions)
print('\nAcurácia: ', accuracy)
cm = confusion_matrix(y_test,predictions)
print('\nMatriz de Confusão')
print(cm)
```

```
Epoch number: 1 and the loss : 2.7328174114227295
Epoch number: 101 and the loss : 0.519047737121582
Epoch number: 201 and the loss : 0.451155453920364
Epoch number: 301 and the loss : 0.422844082117080
Epoch number: 401 and the loss : 0.394680172204971
Acurácia: 0.801724137931034
Matriz de Confusão
[[65 13]
[10 28]]
```