CENTRE FOR INNOVATION & ENTREPRENEURSHIP

PES UNIVERSITY

Machine Learning For DE-10: A Software Optimization Approach

(A session under DE-10 SoC Track)

Dheemanth R Joshi

Research Assistant

Centre For Innovation and Entrepreneurship, PES University

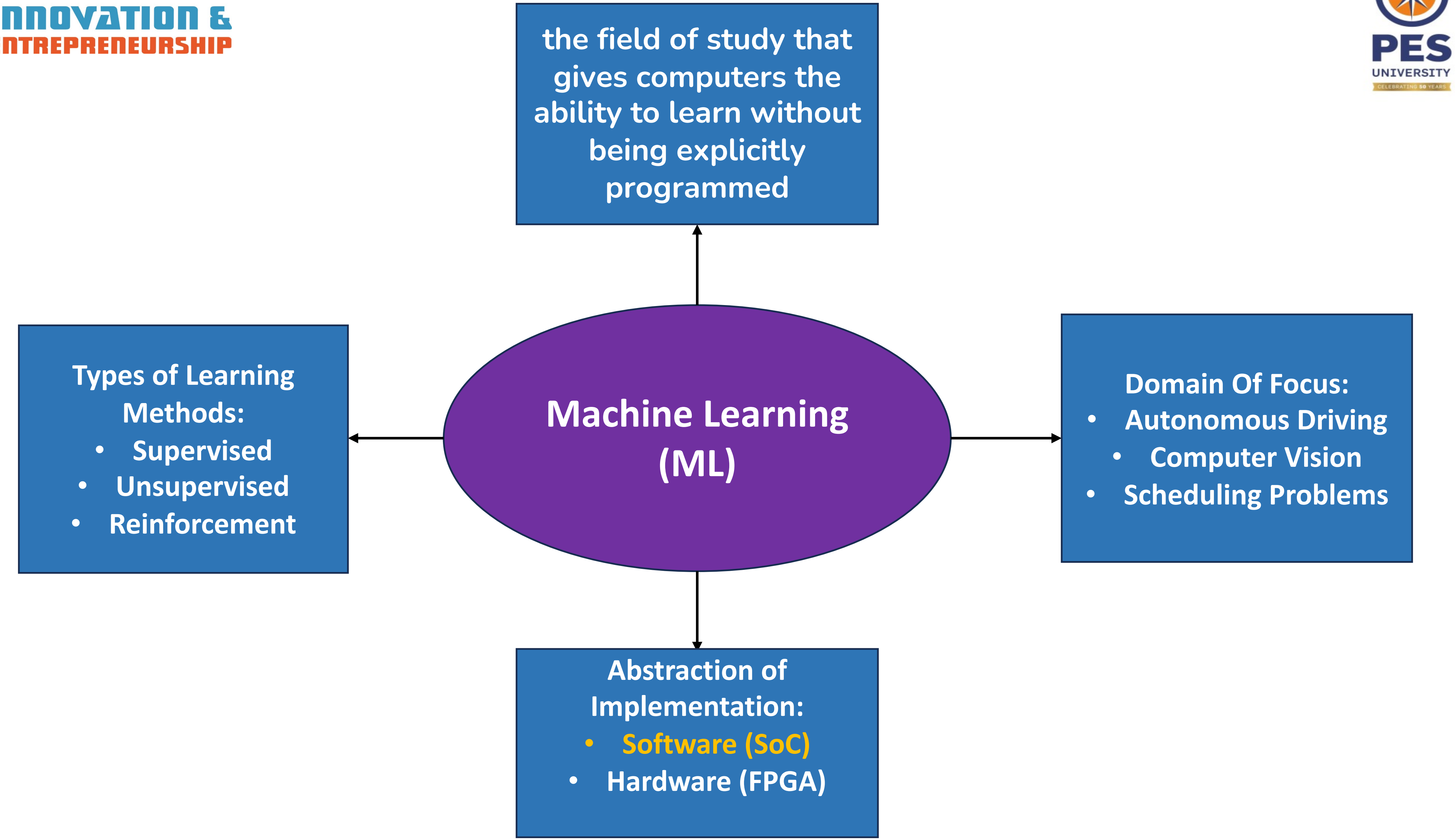
After this session you will:

- Have a foundation to build Machine Learning algorithms from scratch
- Get a brief idea about our Area of focus in embedded software development (DE-10 specific).
- Have a foundation on software optimizations for better hardware resource utilization
- Get an idea of system development through software’s frame of reference.

Pre-requisites

- Basic understanding on Machine Learning and Deep learning

- Python-3 programming language



Machine Learning (ML)

the field of study that gives computers the ability to learn without being explicitly programmed

Types of Learning Methods:

- Supervised
- Unsupervised
- Reinforcement

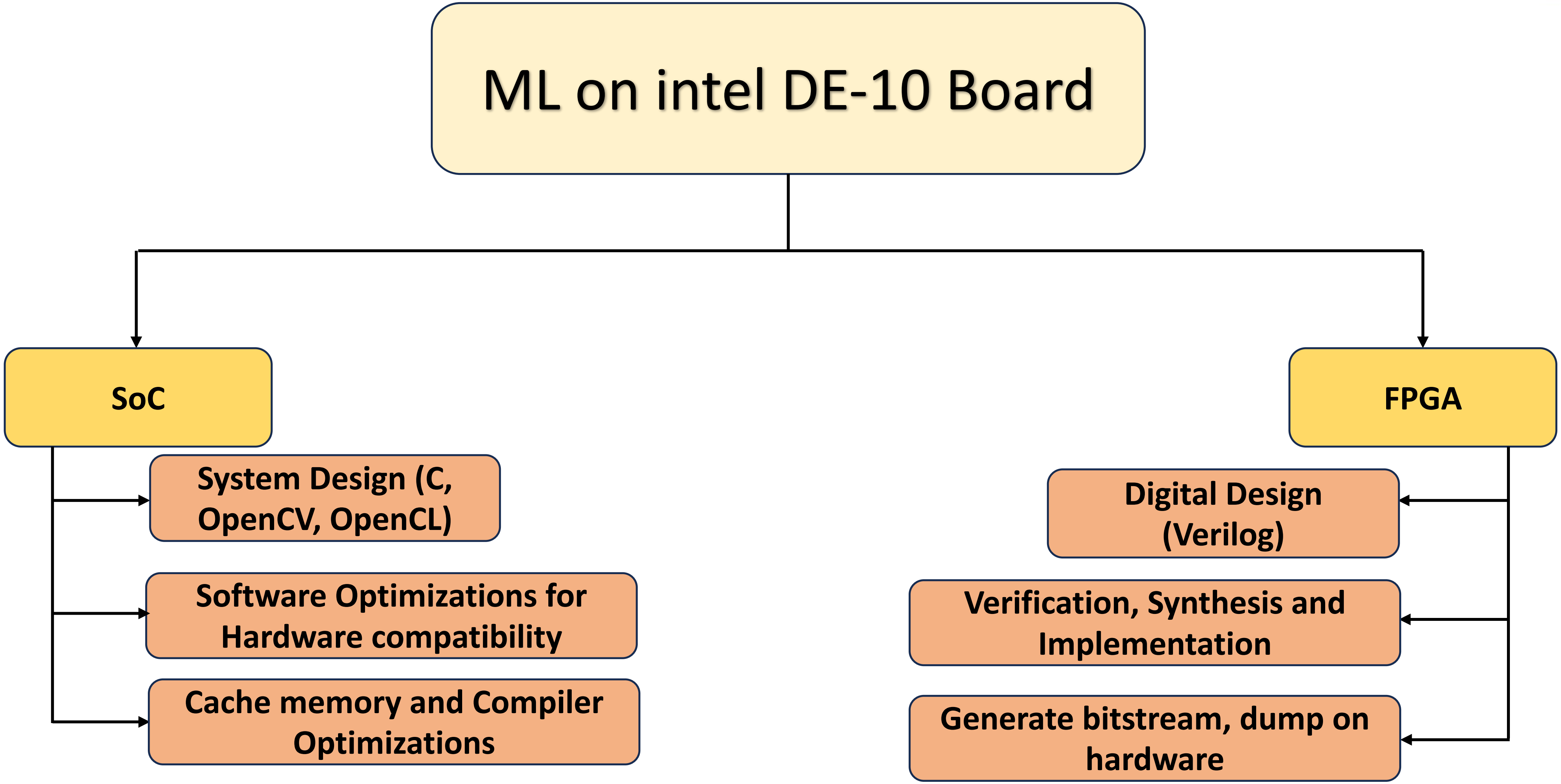
Domain Of Focus:

- Autonomous Driving
- Computer Vision
- Scheduling Problems

Abstraction of Implementation:

- Software (SoC)
- Hardware (FPGA)

ML and Hardware: The Bridge



ML on intel DE-10 Board

SoC

- System Design (C, OpenCV, OpenCL)
- Software Optimizations for Hardware compatibility
- Cache memory and Compiler Optimizations

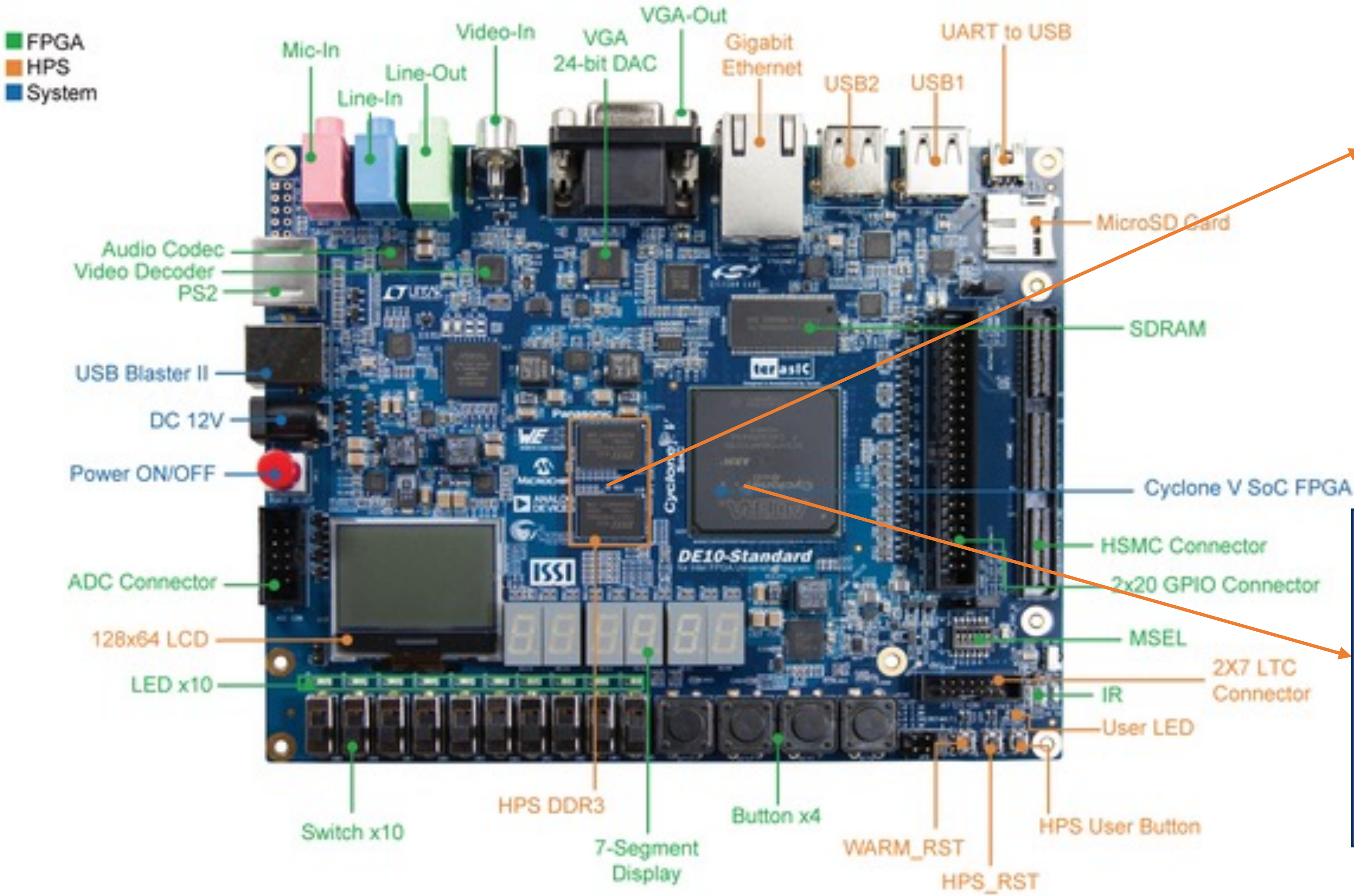
FPGA

- Digital Design (Verilog)
- Verification, Synthesis and Implementation
- Generate bitstream, dump on hardware

CENTRE FOR INNOVATION & ENTREPRENEURSHIP

Systems executing ML algorithms on the board

PES UNIVERSITY



Dual Core SoC units

- OpenCV support
- OpenCL for interface
- ML algorithms will execute as normal instructions on HPS

1 Cyclone V FPGA

- Verilog synthesized netlist implementation
- ML algorithms are executed as dedicated hardware units here

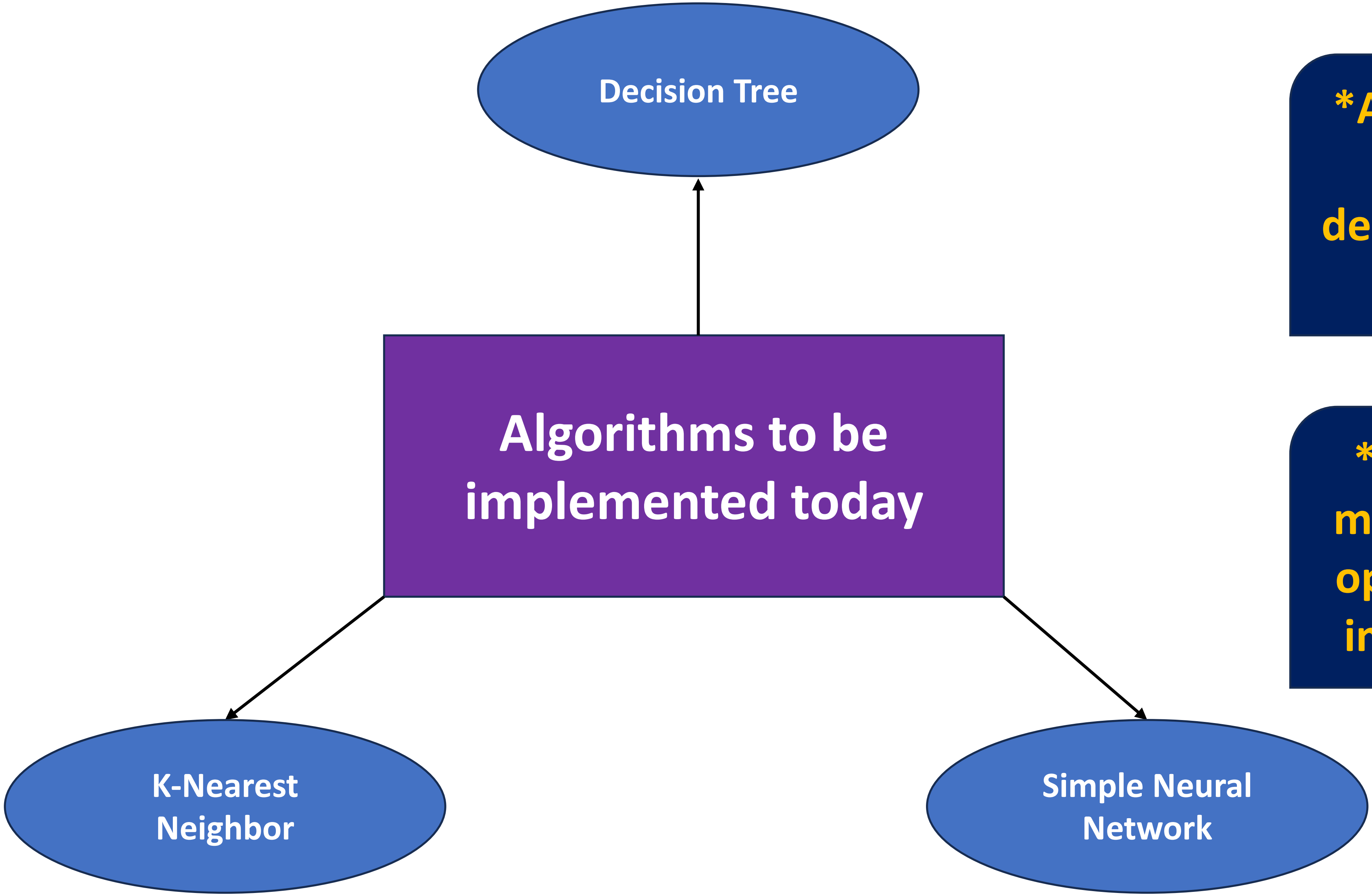
Python-3: Warmup

Problem: Implement DAXPY loop on Python-3 (Use minimum inbuilt libraries.)

Note: Daxpy function is

F= a*X+Y

(a=scalar, [X,Y]=vectors)



Algorithms to be implemented today

Decision Tree

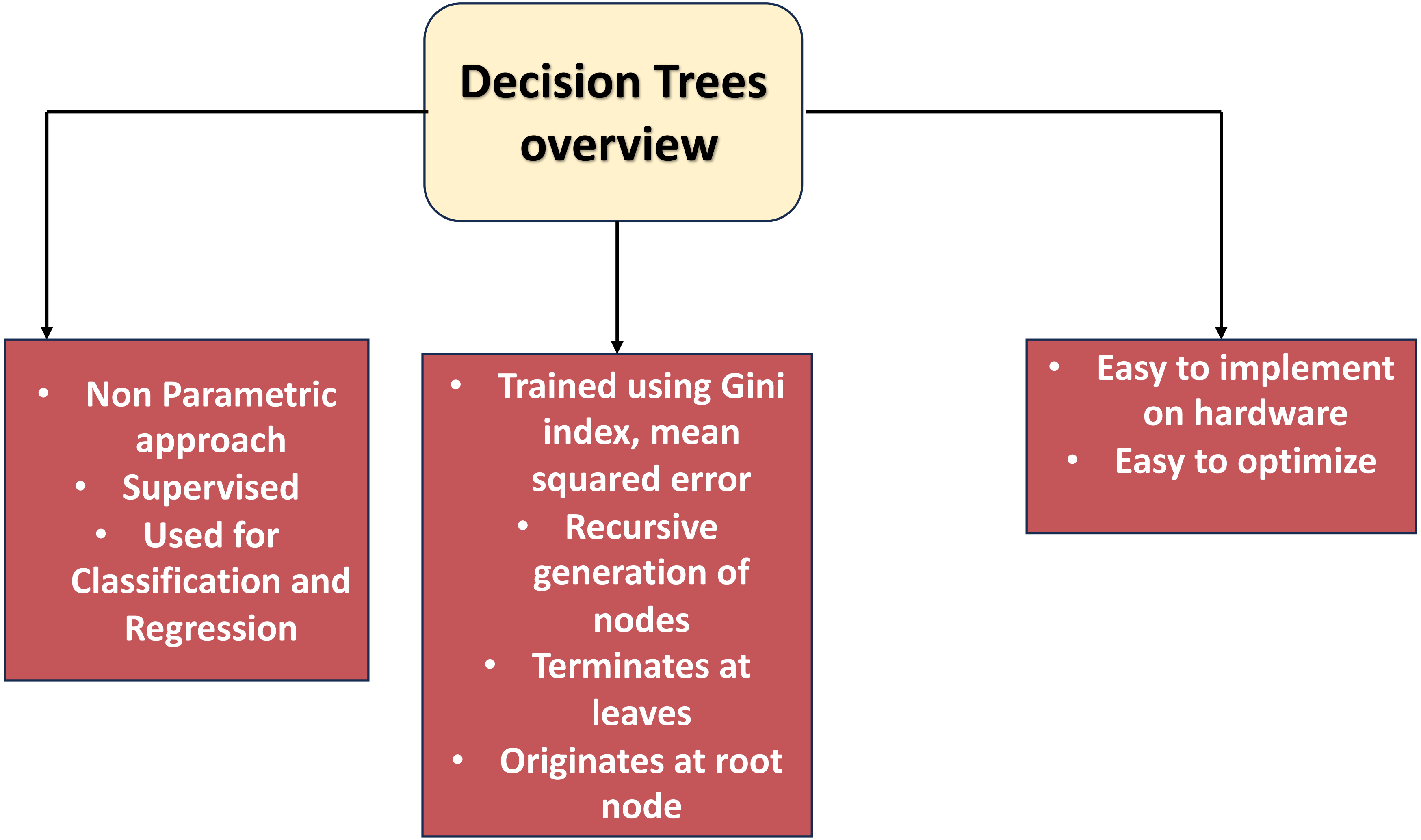
K-Nearest Neighbor

Simple Neural Network

*Architectures will not be designed in this session

*Pre-trained models will be optimized and implemented

Decision Trees



Decision Trees overview

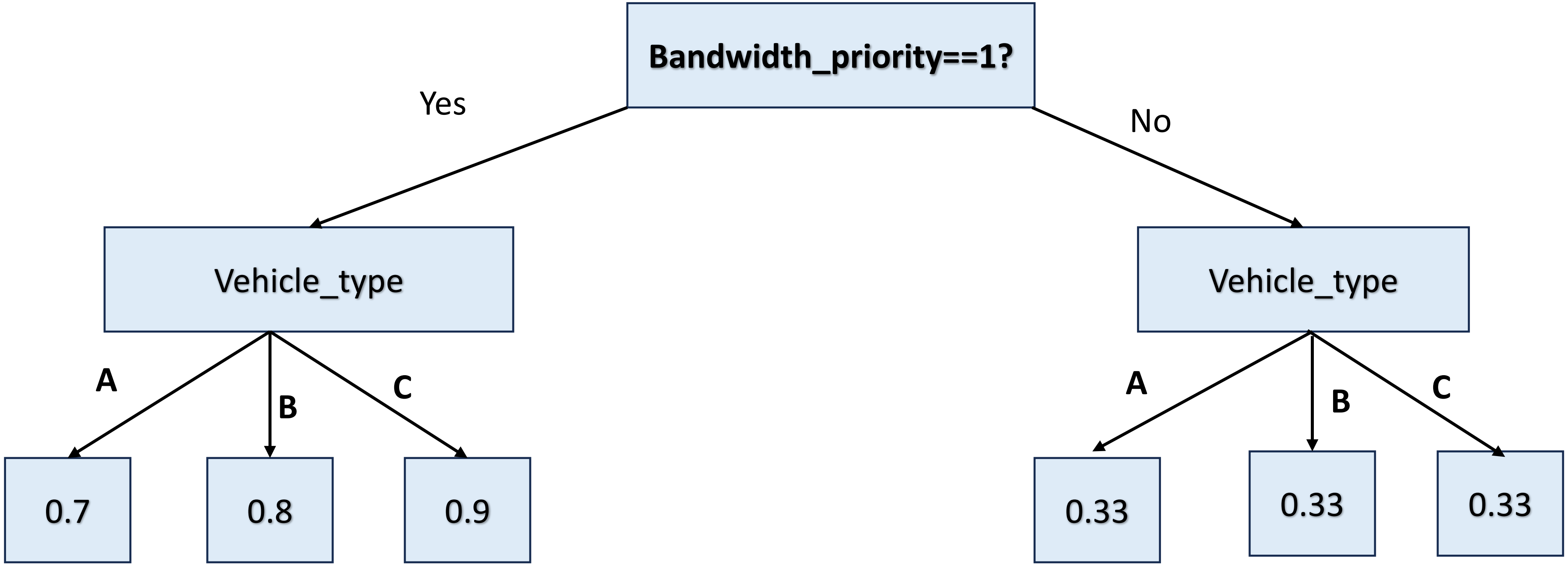
- Non Parametric approach
- Supervised
- Used for Classification and Regression

- Trained using Gini index, mean squared error
- Recursive generation of nodes
- Terminates at leaves
- Originates at root node

- Easy to implement on hardware
- Easy to optimize

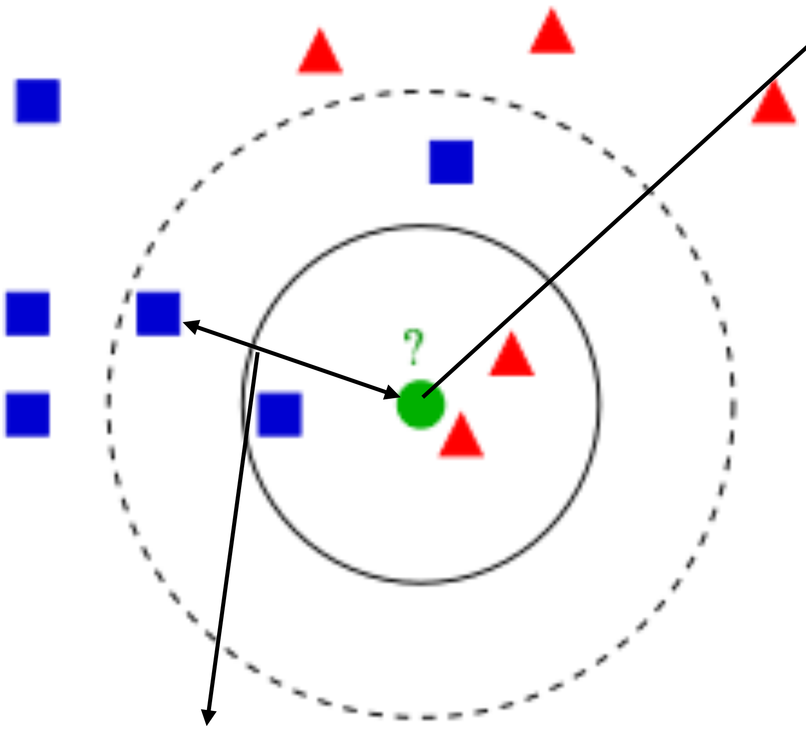
Implementation of Decision Trees

Consider the following generated decision tree:



Task: Realize this decision tree on python-3, how will it be implemented on hardware, if synthesized using HDL?

K-NN Classifier



Euclidean distance

Which Class should this datapoint be assigned to?

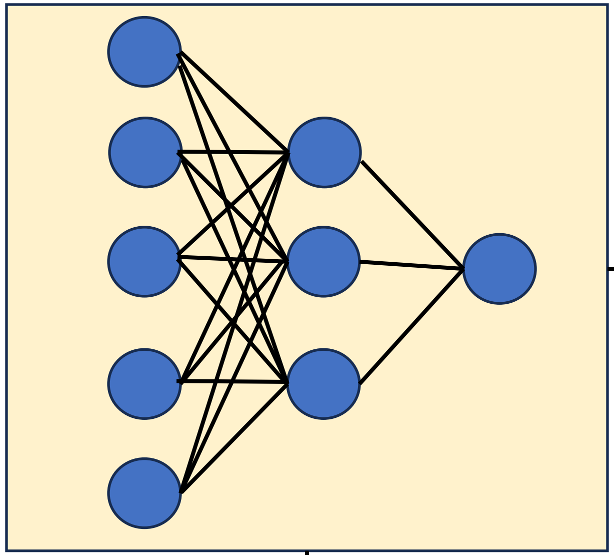
K-NN Approach :

1) Choose an odd integer K
2) Calculate k nearest datapoints from the point of reference
3) Assign the point to the class which has maximum datapoints in the k-nearest neighbors

How can this be optimized on hardware ?

Neural Network

Consider the following neural network, how is this implemented on software? :



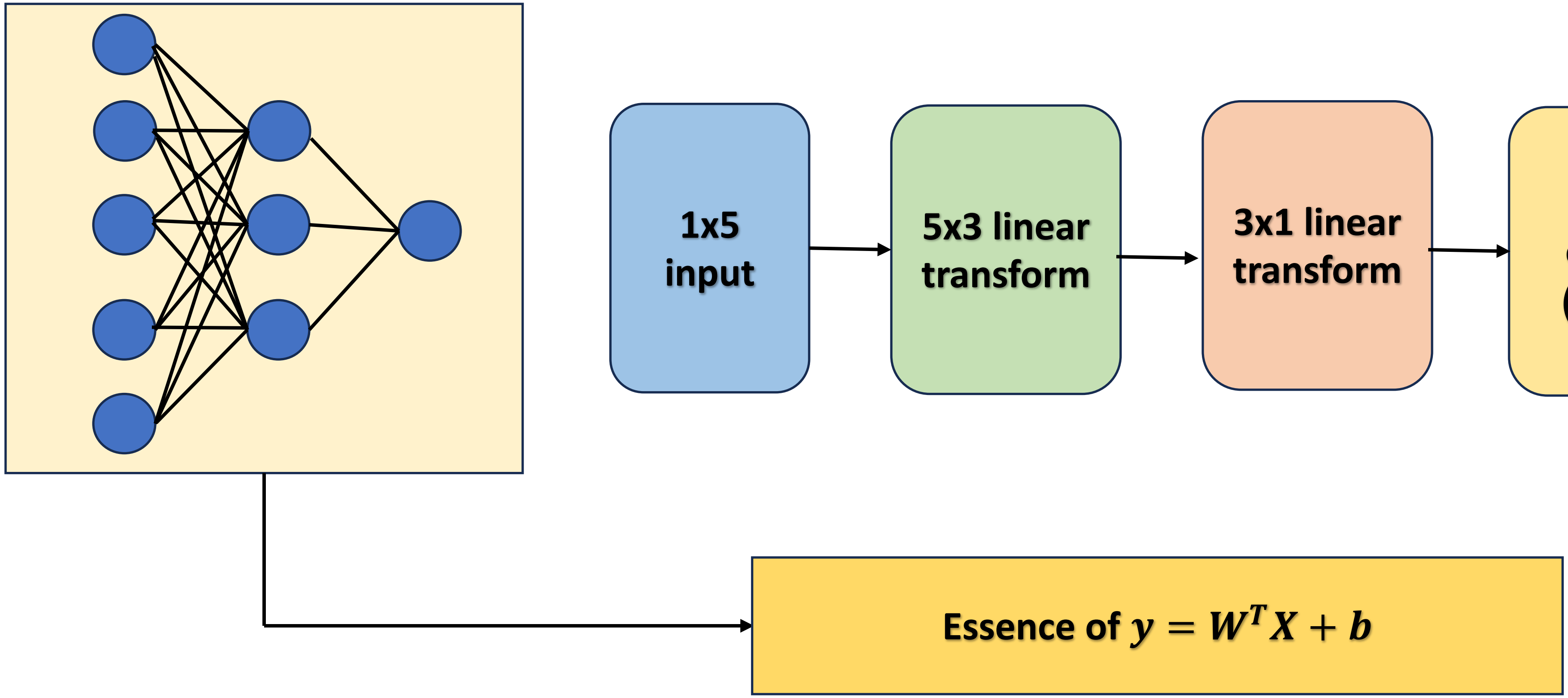
Why $y = W^T X + b$?

How do you implement this network on python 3 assuming that the network Is pretrained and all hyperparameters are given to you?

Hint: it is an 4x3x1 neural network

Neural Network

The Layers of the neural network (weights and biases) can be expressed in terms of matrices and can be forwarded by weight matrix multiplication and addition of bias vector



1x5 input → **5x3 linear transform** → **3x1 linear transform** → **1x1 output (scalar)**

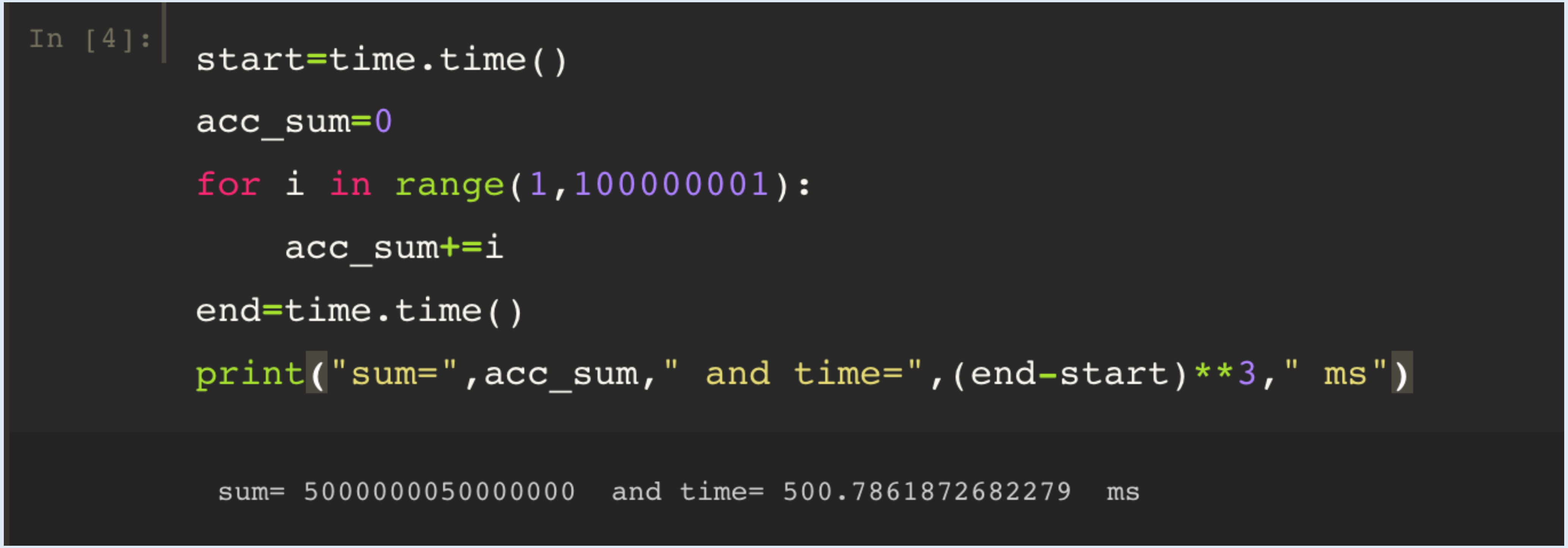
Essence of $y = W^T X + b$

Introduction to software optimization

software optimization, is the process of modifying a software system to make some aspect of it work more efficiently or use fewer resources.In general, a computer program may be optimized so that it executes more rapidly, or to make it capable of operating with less memory storage or other resources, or draw less power. [wiki]

Introduction to software optimization

Consider the following code: how would you optimize it?
{Hint: Loops is a huge overload for any conventional CPU}

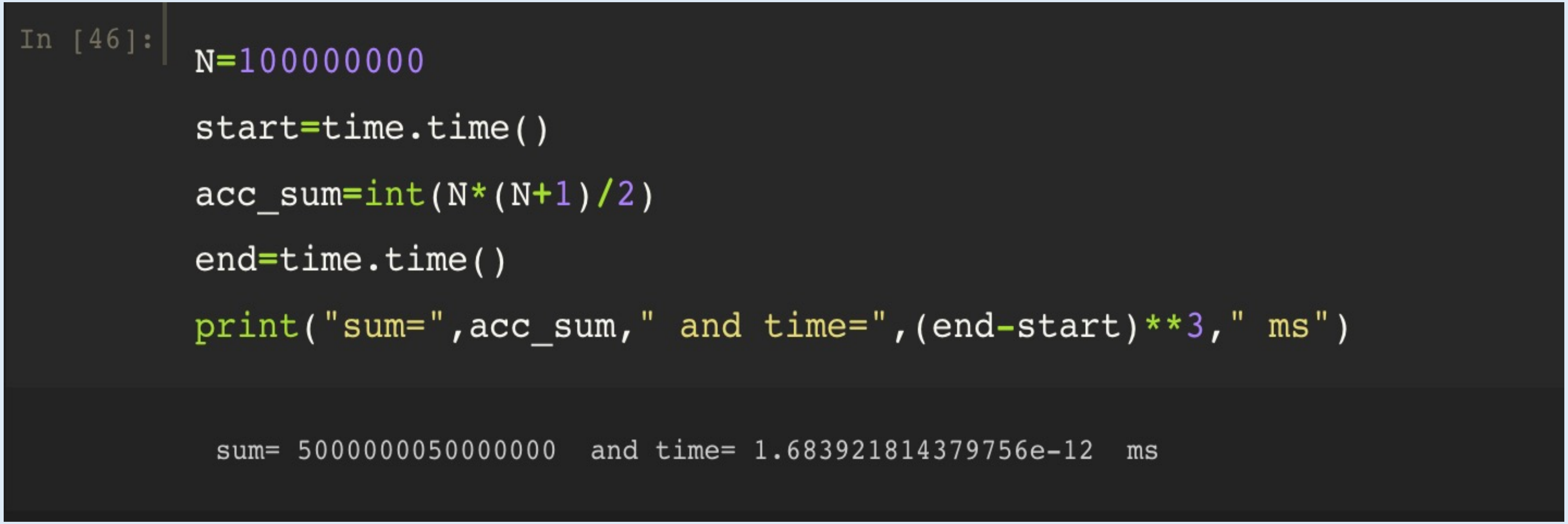


```
In [4]:
start=time.time()
acc_sum=0
for i in range(1,100000001):
    acc_sum+=i
end=time.time()
print("sum=",acc_sum," and time=",(end-start)**3," ms")
```

```
sum= 5000000050000000  and time= 500.7861872682279  ms
```

Introduction to software optimization

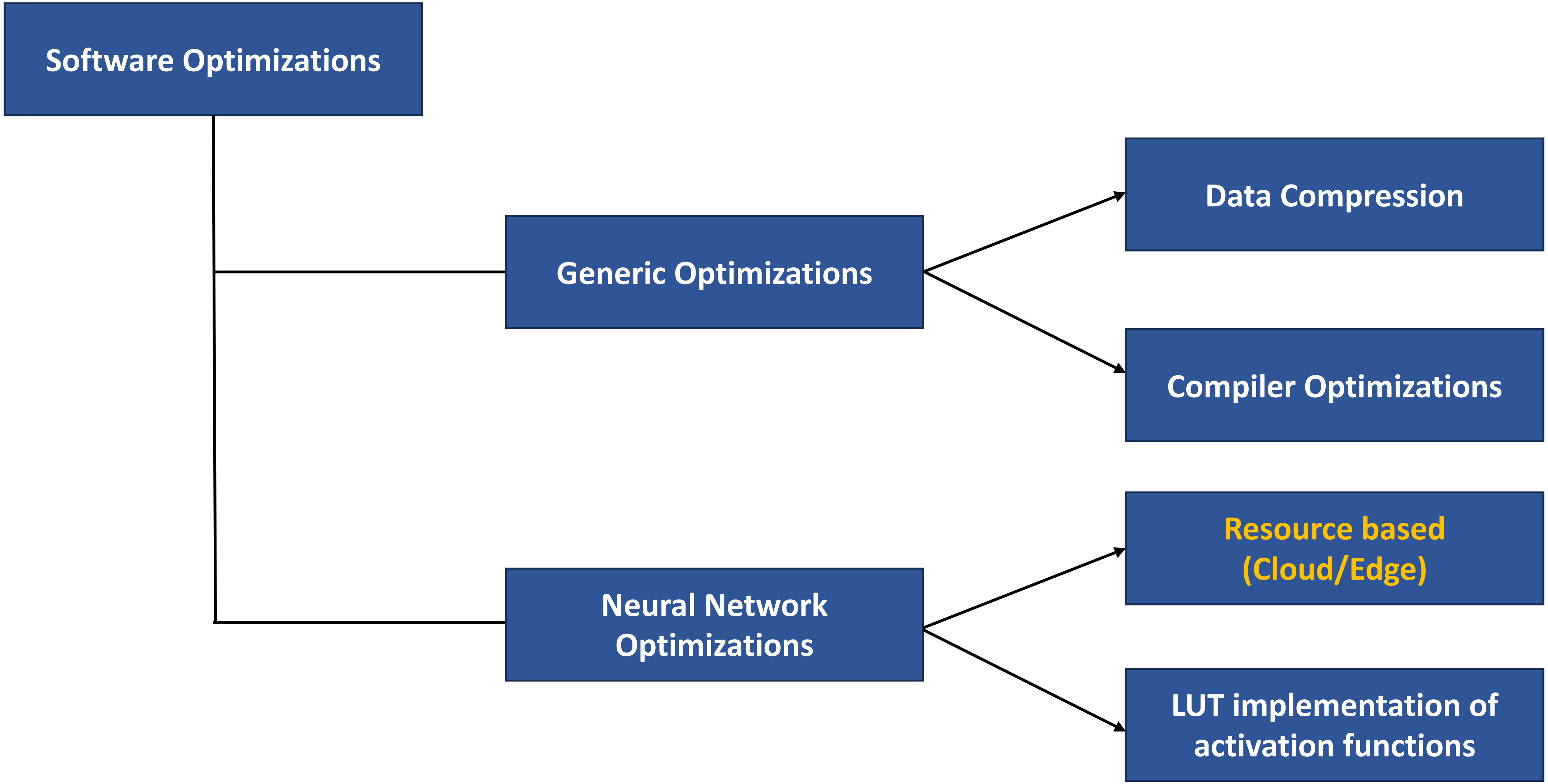
Consider the following code: how would you optimize it? {Solution: A loop elimination approach by changing the function definition}



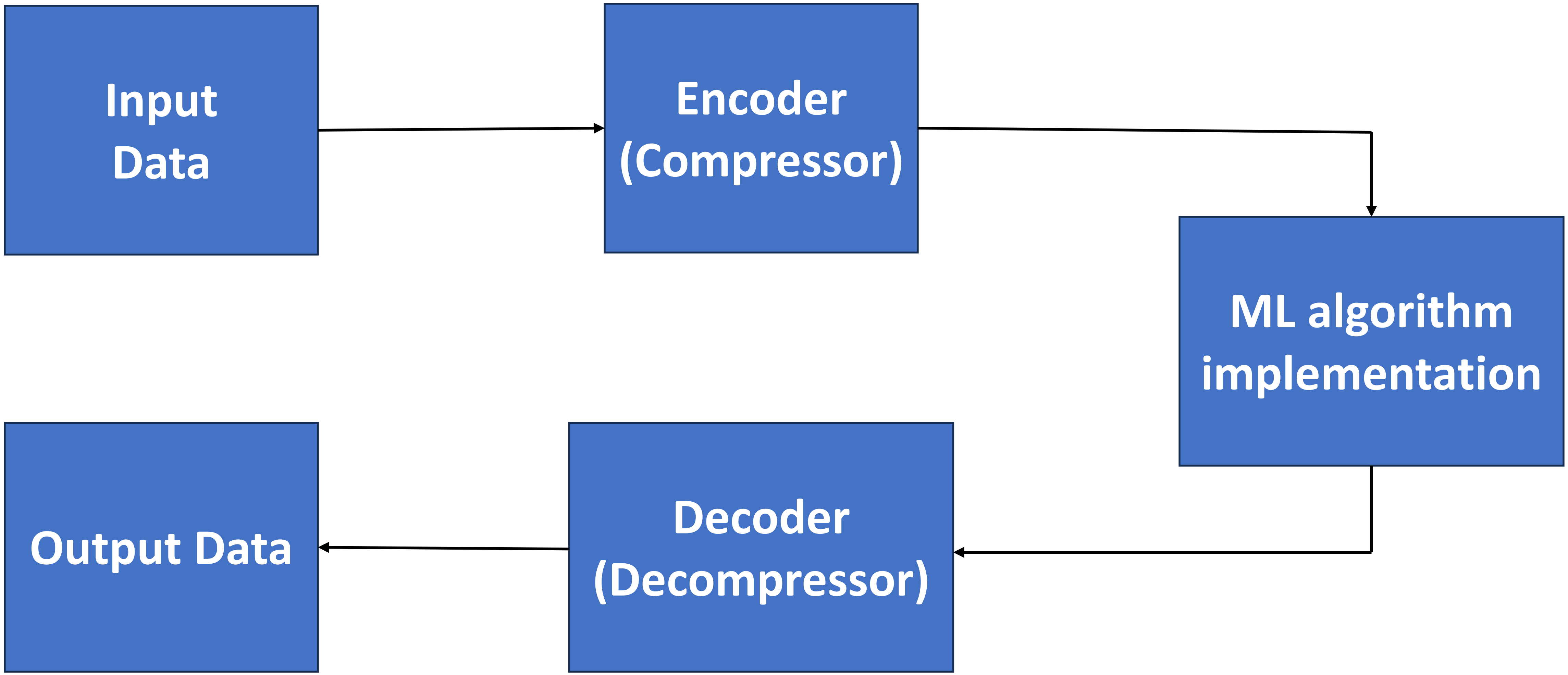
```
In [46]:  N=100000000

          start=time.time()

          acc_sum=int(N*(N+1)/2)

          end=time.time()

          print("sum=",acc_sum," and time=",(end-start)**3," ms")
```

```
sum= 5000000050000000  and time= 1.683921814379756e-12  ms
```

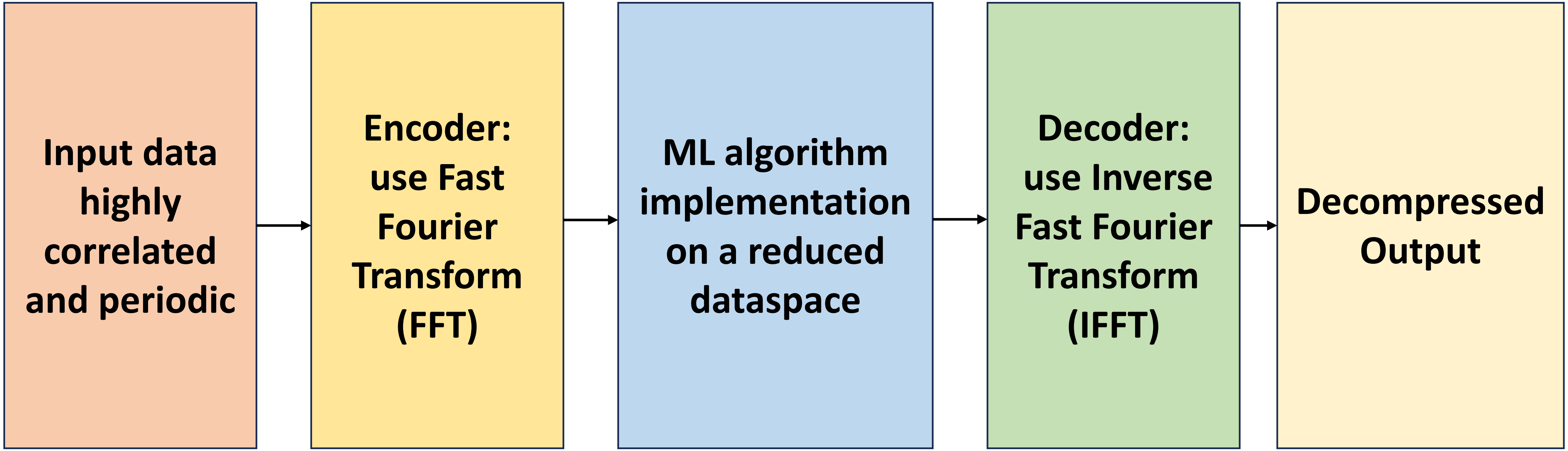
Software Optimizations for ML


Software Optimizations
Generic Optimizations
Data Compression
Compiler Optimizations
Neural Network Optimizations
Resource based (Cloud/Edge)
LUT implementation of activation functions

Data Compression



Data Compression: Example



Input data highly correlated and periodic

Encoder: use Fast Fourier Transform (FFT)

ML algorithm implementation on a reduced dataspace

Decoder: use Inverse Fast Fourier Transform (IFFT)

Decompressed Output

Compiler Optimizations

Various Compiler Optimizations include:

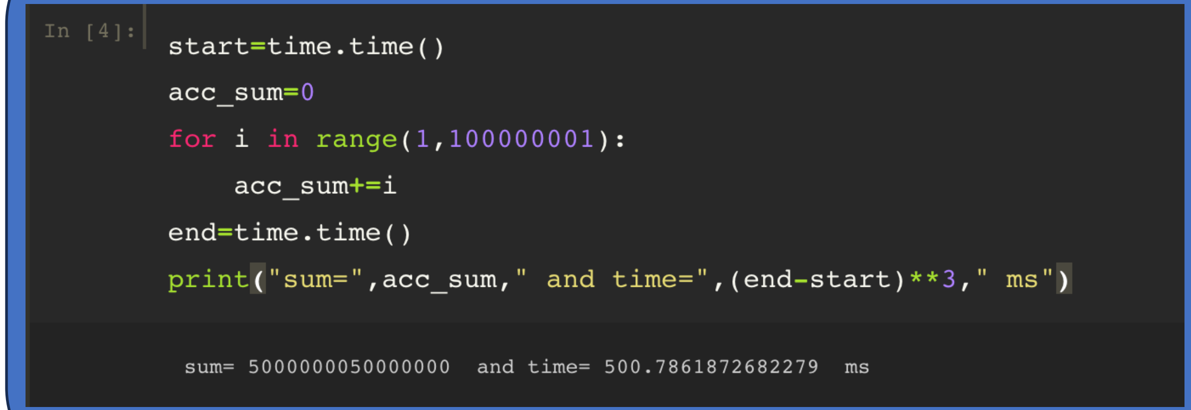
- **Dead Code Elimination**
- **Loop Exchange**
- **Loop Unrolling**
- **Instruction scheduling**

Is there a way to customize compilers to transform code into some other set of instructions which perform same functionality but is highly optimized?
For example: The sample program demonstrated in section “introduction to software optimization”

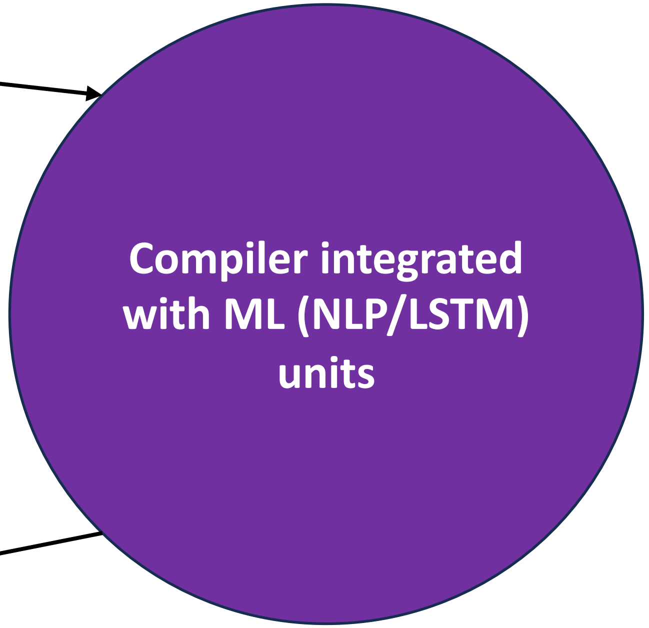
CENTRE FOR INNOVATION & ENTREPRENEURSHIP

Compiler Optimization : Customized Compiler (ML integrated compiler)

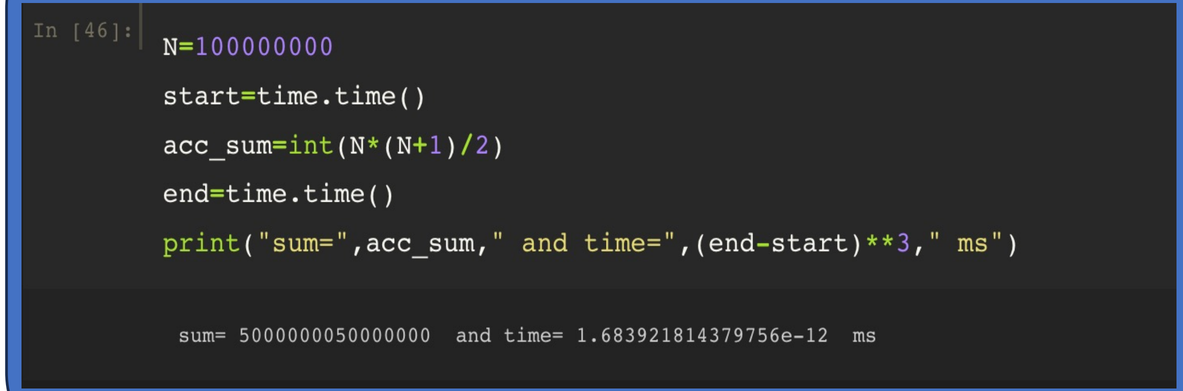
PES UNIVERSITY



```
In [4]:  start=time.time()
         acc_sum=0
         for i in range(1,100000001):
             acc_sum+=i
         end=time.time()
         print("sum=",acc_sum," and time=",(end-start)**3," ms")

          sum= 5000000050000000  and time= 500.7861872682279  ms
```



Compiler integrated with ML (NLP/LSTM) units



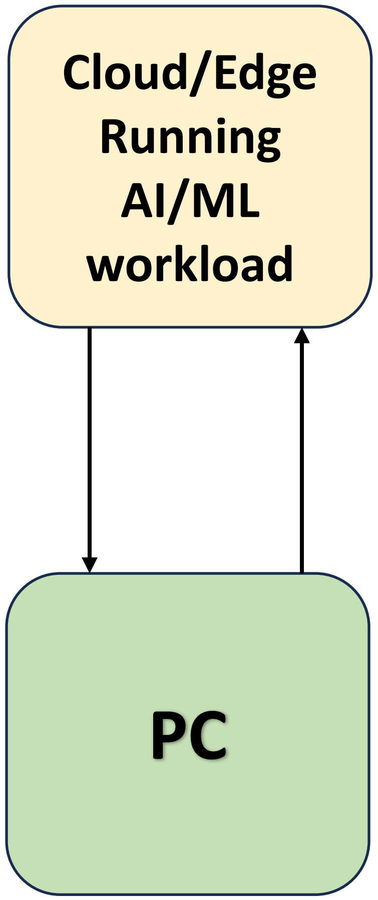
```
In [46]:  N=100000000
          start=time.time()
          acc_sum=int(N*(N+1)/2)
          end=time.time()
          print("sum=",acc_sum," and time=",(end-start)**3," ms")

           sum= 5000000050000000  and time= 1.683921814379756e-12  ms
```

Interesting problem statement

Resource Allocation For Neural Networks: A network attached storage approach

- Not all programs can be executed on a PC.
- **Time, Space** and power restrictions
- Therefore, computation intensive tasks are offloaded to a cloud or an edge to compute tasks.
- For personal computers, a IoT network can be formed where each PC can store partial computational loads and data can be processed through the IoT.
- This Method is known as **Network Attached Storage (NAC)**.

Resource Allocation For Neural Networks: A network attached storage approach



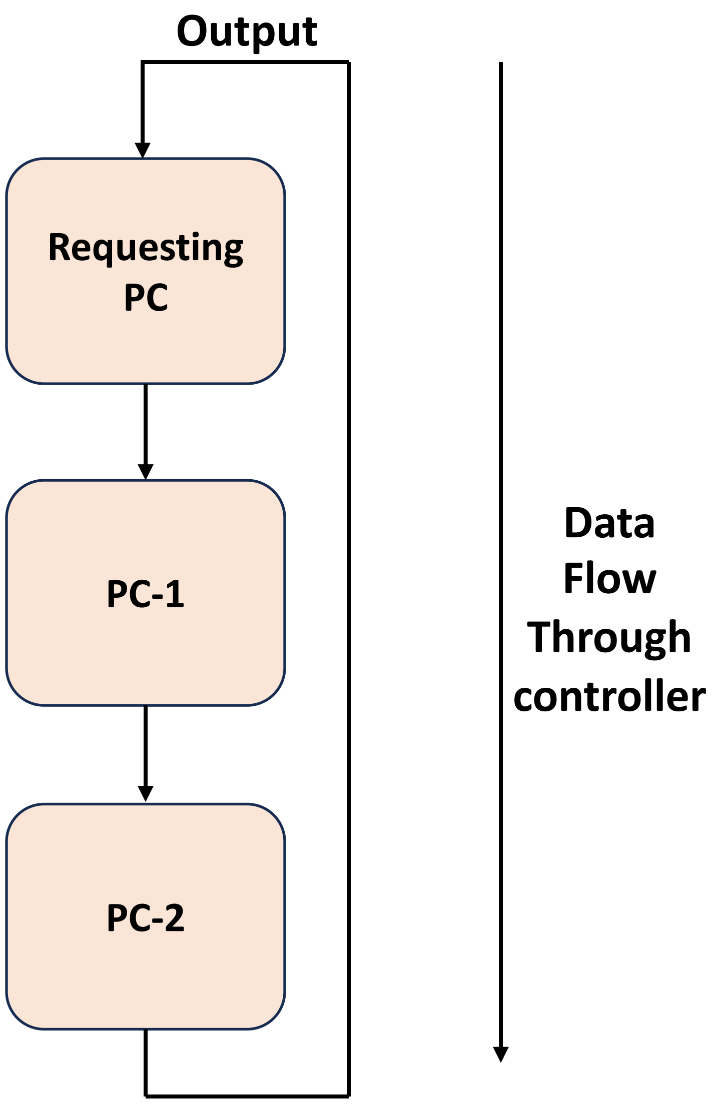
Cloud/Edge Running AI/ML workload

PC

PC offloading its dataset to a cloud or an edge server

V/S

PC forming a NAS environment and getting its data processed.



Output

Requesting PC

PC-1

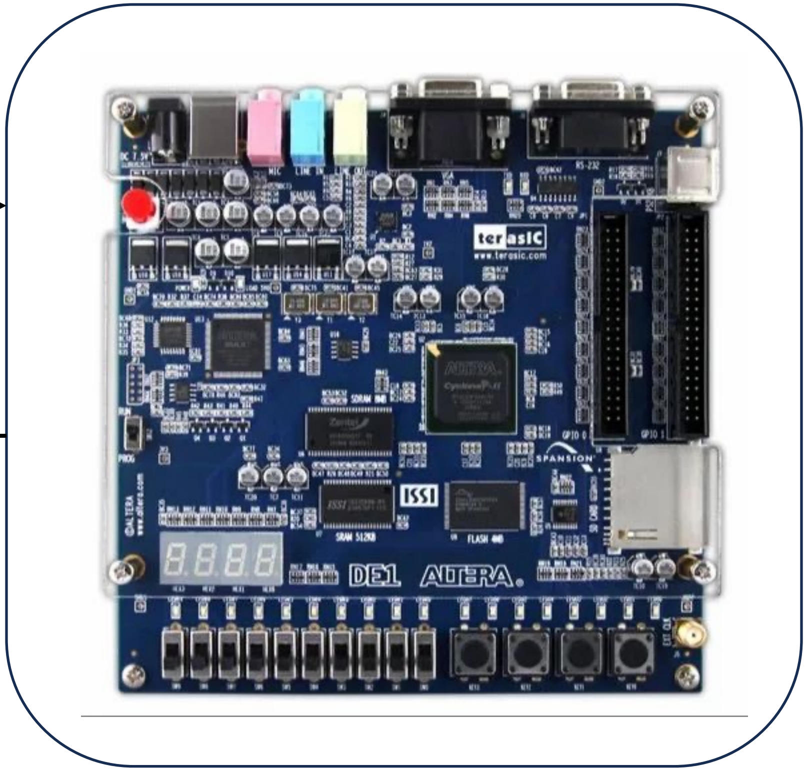
PC-2

Data Flow Through controller

Resource Allocation For Neural Networks: (implications on DE-10 board)

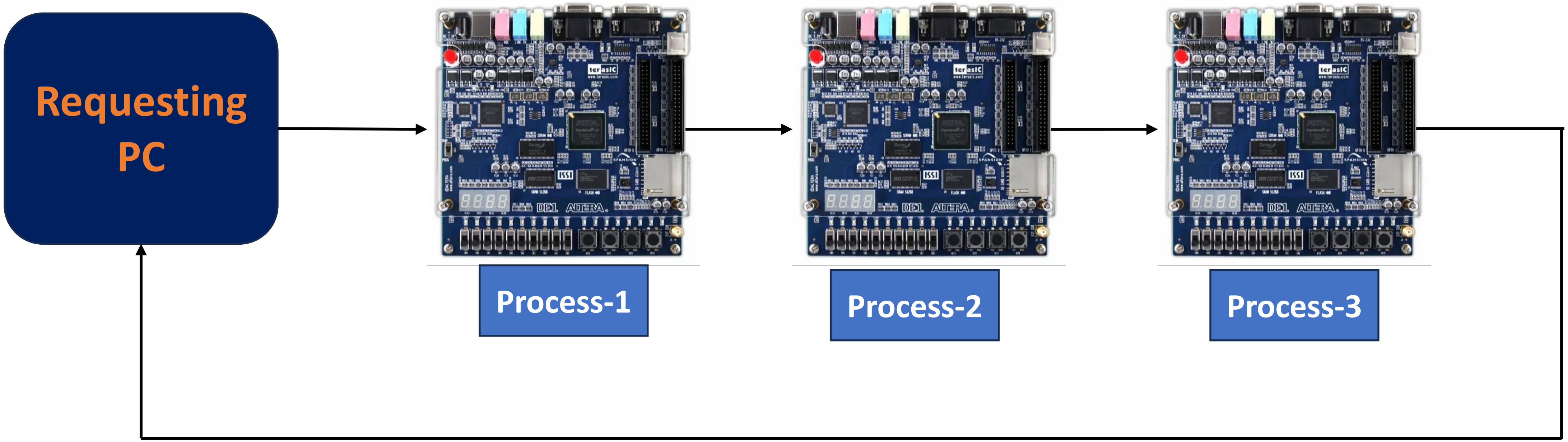
- What if previous concepts were used to implement a workload on DE-10 boards?

PC



DE-10 used as an edge server
EDGE On Chip (EoC)

Resource Allocation For Neural Networks: (implications on DE-10 board)



Requesting PC

Process-1

Process-2

Process-3

DE-10 Implemented as an IoT of Edge servers running various processes (may or may not be neural networks)

Exercise

- Using Socket programming in python-3, build a 2-unit Network attached computing system with your teammate.
- Assume one of you as a client and other as a server
- Consider a (5x5) three-layer neural network was broken into 2 networks, one stored on client and other stored on server.
- Make request to the server as a client by sending 1st layer processed data to the server.

Activation function response through LUT’s

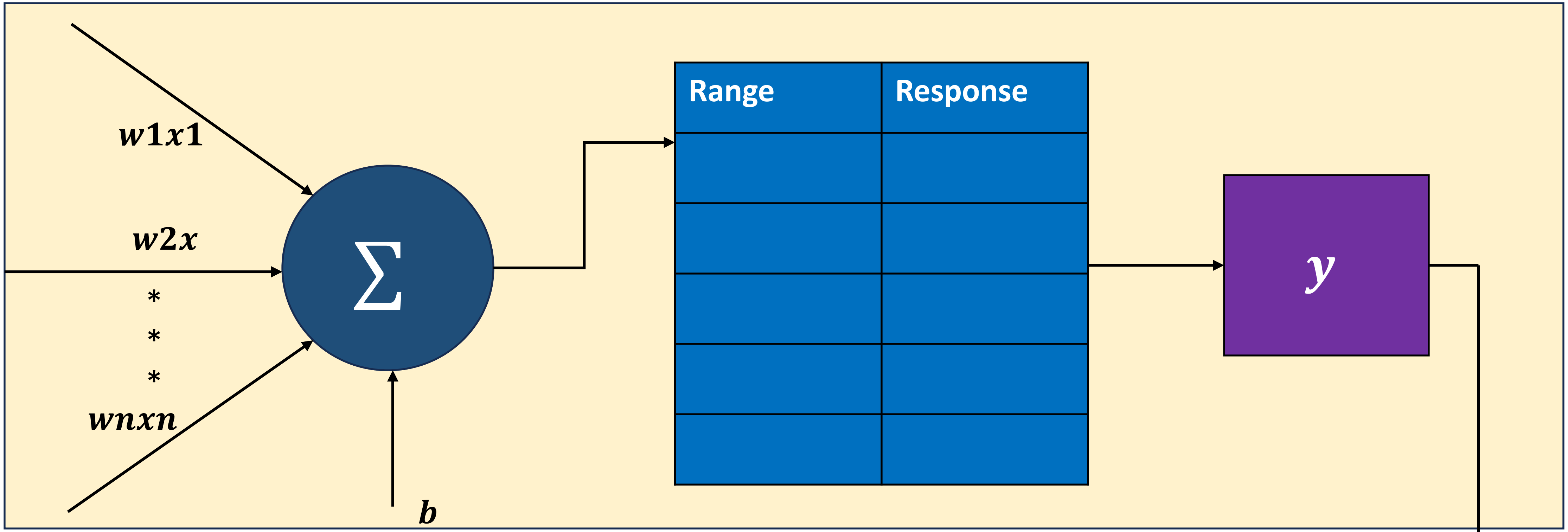
- Activation functions such as sigmoid, ReLU are used to shift and squish the neuron response to a desired range.

- If number of neurons increase, computing these activation functions for each neuron is time consuming.

- Therefore, Lookup Tables (LUT’s) are designed which contain values of the response for an input range.

- It is convenient for hardware to fetch a value from the LUT rather than calculating the response through the functional hardware unit.

LUT Implementation



Range	Response

LUT integrated neuron architecture

$$W^TX+b$$

QNA
Thanks

Dheemanth R Joshi
Research Assistant
Centre For Innovation and Entrepreneurship, PES University