ECE618
Hardware Accelerators for Machine Learning
(Spring 2022)

Lecture 1: Course Information &
Machine Learning and FPGA Accelerator Recap

Weiwen Jiang, Ph.D.

Electrical and Computer Engineering

George Mason University

wjiang8@gmu.edu

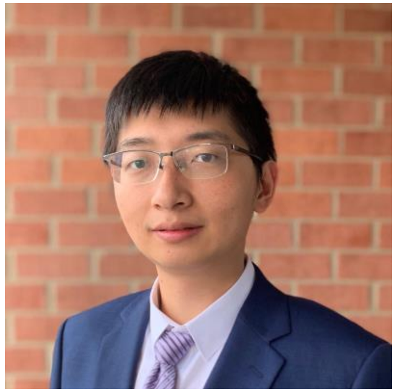
Agenda

- **Course Information**
- Course Resources
- Course Policy
- Tools for Lab
- Motivation and Schedule

Course Information

Instructor	Dr. Weiwen Jiang
E-Mail	wjiang8@gmu.edu
Phone	(703)993-5083
Lecture Time	**Monday 19:20 - 22:00**
Location	**Room 1002, Music/Theater Building**
Office Hour	Monday 16:30 - 17:30
Office	Room 3247, Nguyen Engineering Building
Zoom	http://go.gmu.edu/zoom4weiwen
Backup Course Zoom	https://go.gmu.edu/ece618 **(Need Permission First)**

About Me.



Dr. Weiwen Jiang

- **Background**
  - Researcher at University of Pittsburgh (2017-2019)
  - Postdoc at University of Notre Dame (2019-2021)
  - George Mason University (2021 - present)
- **Research Interests**
  - HW/SW Co-Design
  - Quantum Machine Learning
- **Contacts:**

  - wjiang8@gmu.edu
  - Nguyen Engineering Building, Room3247
  - (703)993-5083
  - https://jqub.ece.gmu.edu/

Teaching Assistant

Yi Sheng (Ph.D. Candidate)

ysheng2@gmu.edu

https://jqub.ece.gmu.edu/yi/

Office Hours: TBD

Course Description

Covers the **hardware design** principles to **deploy** different machine learning algorithms. The emphasis is on understanding the fundamentals of **machine learning and hardware architectures** and determine plausible methods to **bridge them**.

Topics include precision scaling, in-memory computing, hyperdimensional computing, architectural modifications, GPUs and vector architectures, quantum computing as well as recent hardware programming tools such as **Xilinx AI Vitis, Xilinx HLS, and IBM Qiskit**.

Recommend Prerequisite

- **ECE 554: Machine Learning for Embedded Systems**

- Good C programming
  - Especially required for FPGA-related project

- Familiar with Python and PyTorch

Agenda

- Course Information
- **Course Resources**
- Course Policy
- Tools for Lab
- Motivation and Schedule

Course Resources

- **Blackboard:**
  - Assignments will be posted and submitted here!
  - Online discussion, shared documents, announcements.
    - Do NOT upload codes in discussion.
- **Course Website:**
  - https://jqub.ece.gmu.edu/2022/01/01/HA4ML/
  - Course information (TA time, location, zoom, etc.)
  - Slides, readings, and documents will be posted here!

Agenda

- Course Information
- Course Resources
- **Course Policy**
- Tools for Lab
- Motivation and Schedule

Grading Policy

• Midterm Exam	10%
• Final Exam	20%
• Research Paper Presentation	20%
• Assignments and Labs	20%
• Project	30%

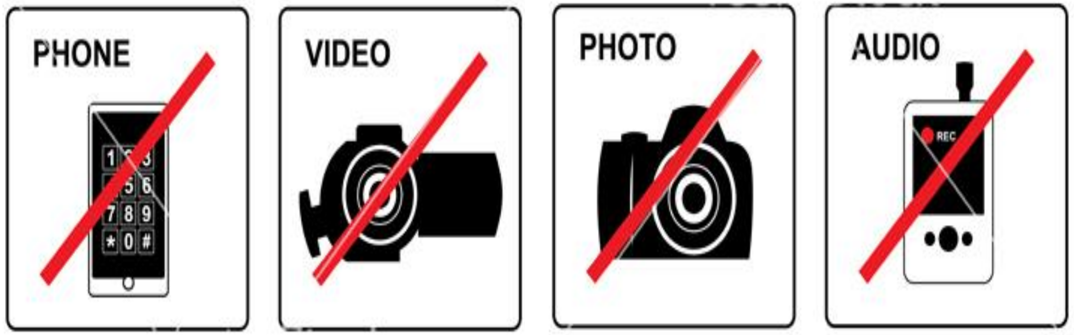
You Have Been Warned.
Zero Tolerance!

- **No matter vaccinated or not, face mask is required in class**

- **Request to a Zoom access for a few classes if needed**

You Have Been Warned.
Zero Tolerance!

- **Lecture content and materials should NOT go online without explicit permission**



- **No plagiarism!**

  The most common sense of way interpreting no plagiarism:
  You need to DO your work.

Agenda

- Course Information
- Course Resources
- Course Policy
- **Tools for Lab**
- Motivation and Schedule

Tools for lab

Google Colab

Xilinx High-Level Synthesis

IBM Qiskit

Agenda

- Course Information
- Course Resources
- Course Policy
- Tools for Lab
- **Motivation and Schedule**

What Software to Be Accelerated? --- MLP/CNN

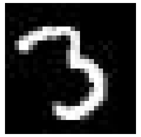
Supervised Learning

Example: Classification

Training

Given: Labeled data as training dataset

(x_i, y_i): x_i training data, y_i: label

$x_i =$  $y_i = 3$

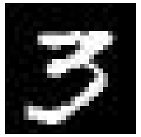
Output: A learned function $\boldsymbol{f}$ from X to Y

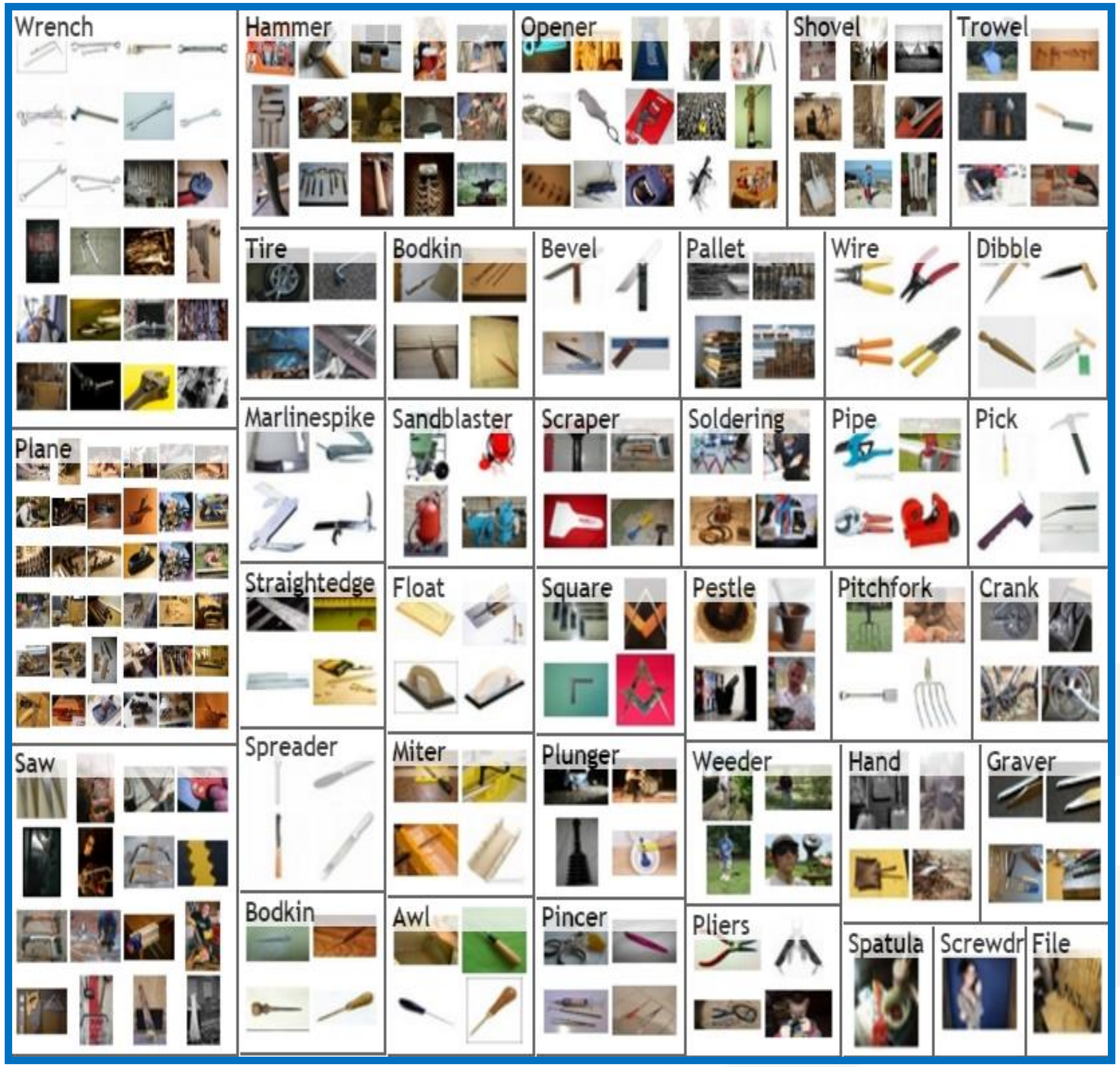
$$\boldsymbol{f}: x \mapsto y$$

Inference/Execution

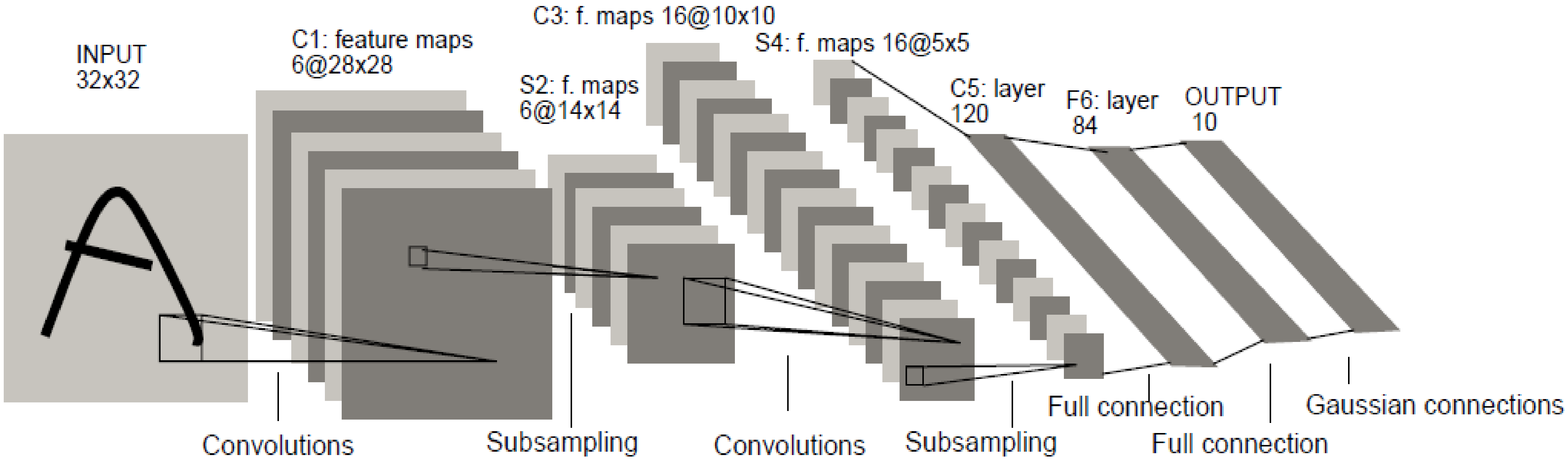
Given: Unseen data test dataset

A learned function $\boldsymbol{f}$

Do: $\boldsymbol{f}($  $) = 3$



What Software to Be Accelerated? --- MLP/CNN



- Local receptive fields
- Shared weights
- Pooling (subsampling)

Cat?

Dog?

What Software to Be Accelerated? --- RNN

Supervised Learning

Example: Classification

Training

Given: Labeled data as training dataset

(x_i, y_i): x_i training data, y_i: label

$x_i =$  $y_i =$"can I"

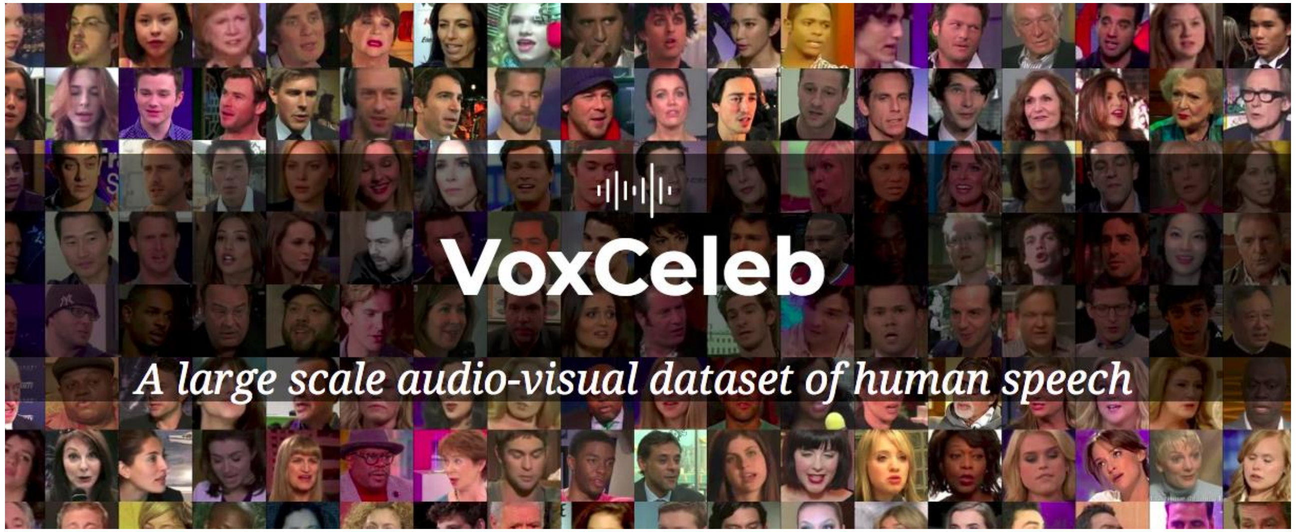
Output: A learned function $\boldsymbol{f}$ from X to Y

$$\boldsymbol{f}: x \mapsto y$$

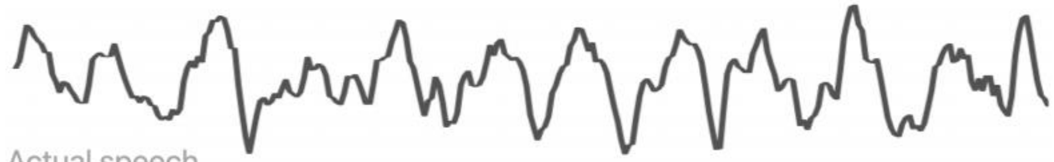
Inference/Execution

Given: Unseen data test dataset
A learned function $\boldsymbol{f}$

Do: $\boldsymbol{f}($  $)$ = "brown fox"



SEP-28k Dataset



What Software to Be Accelerated? --- RNN

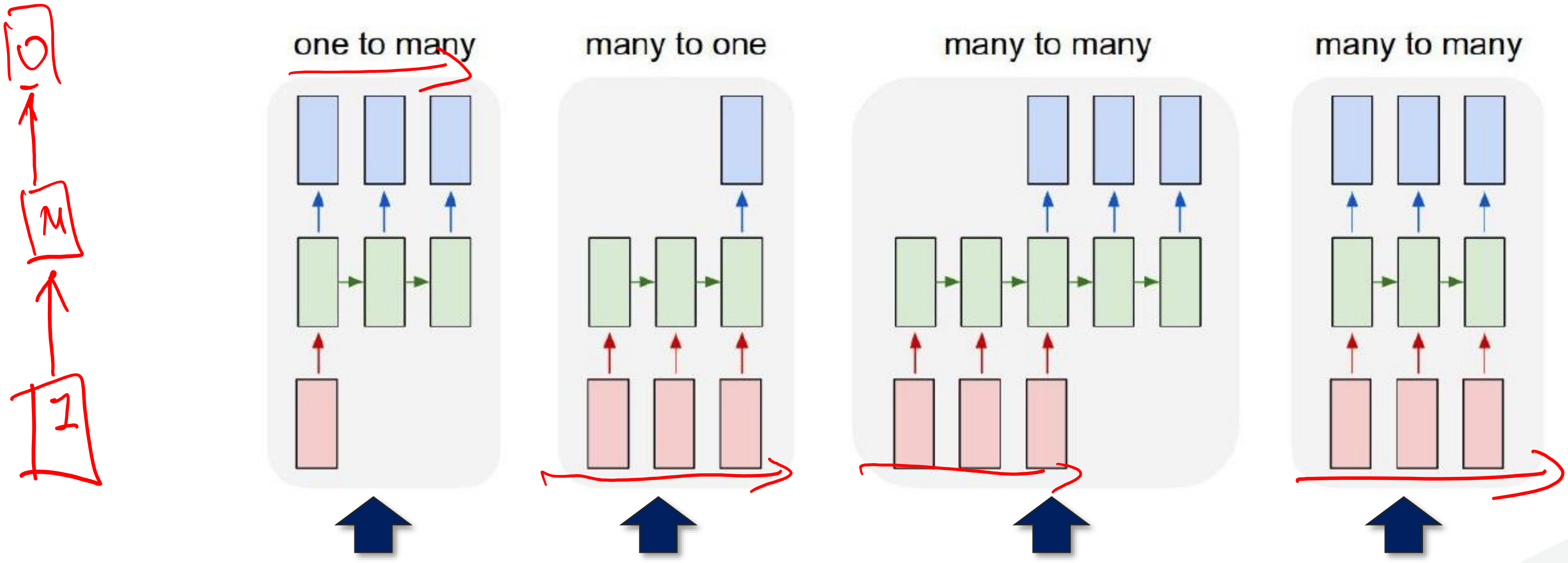


Image captioning (image → words)	Sentiment classification (words → sentiment)	Translation (words → words)	Video frame classification (frames → classes)

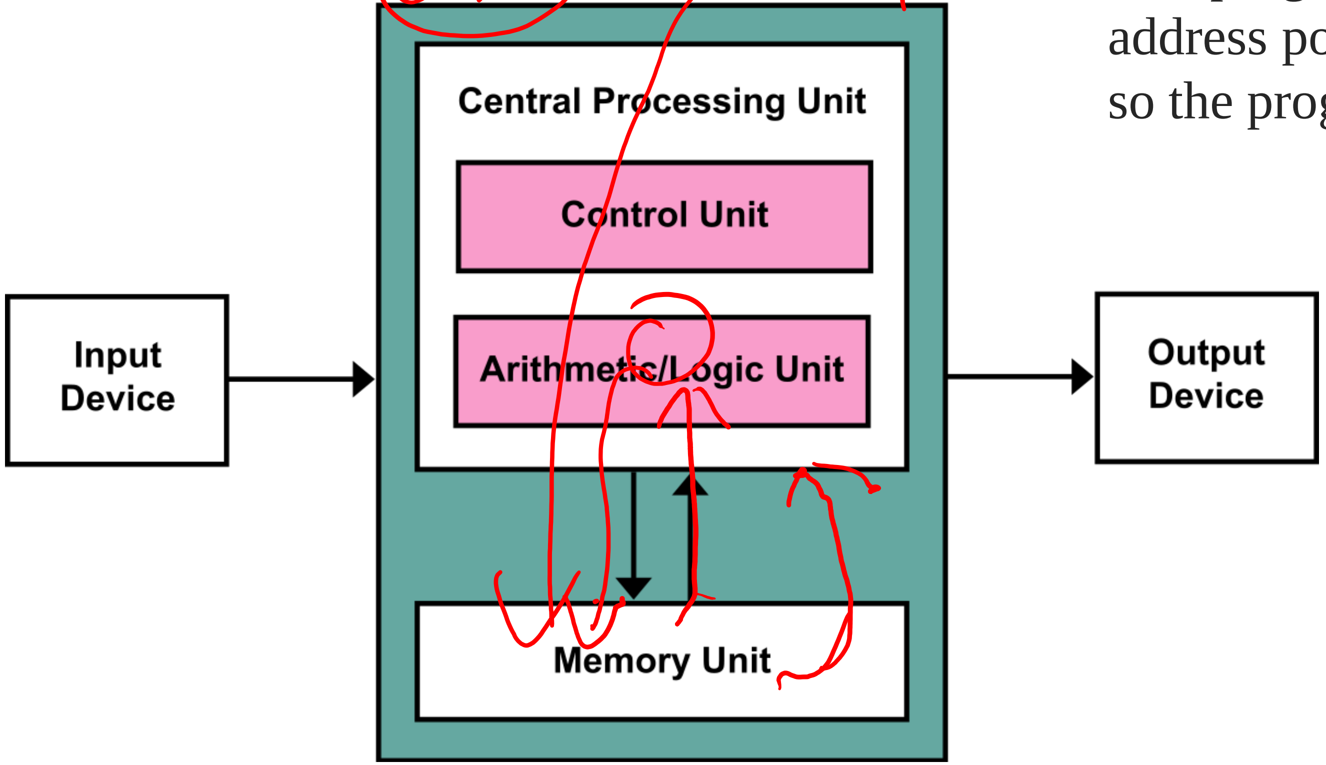
What Hardware Will Be Covered in This Course?

ALU

Cache

Conv

pool



The **von Neumann structure**, also known as the Princeton structure, is a **memory structure** that merges **program instruction memory and data memory together**.

The program instruction memory address and the data memory address point to different physical locations in the same memory, so the program instruction and data are of the same width.

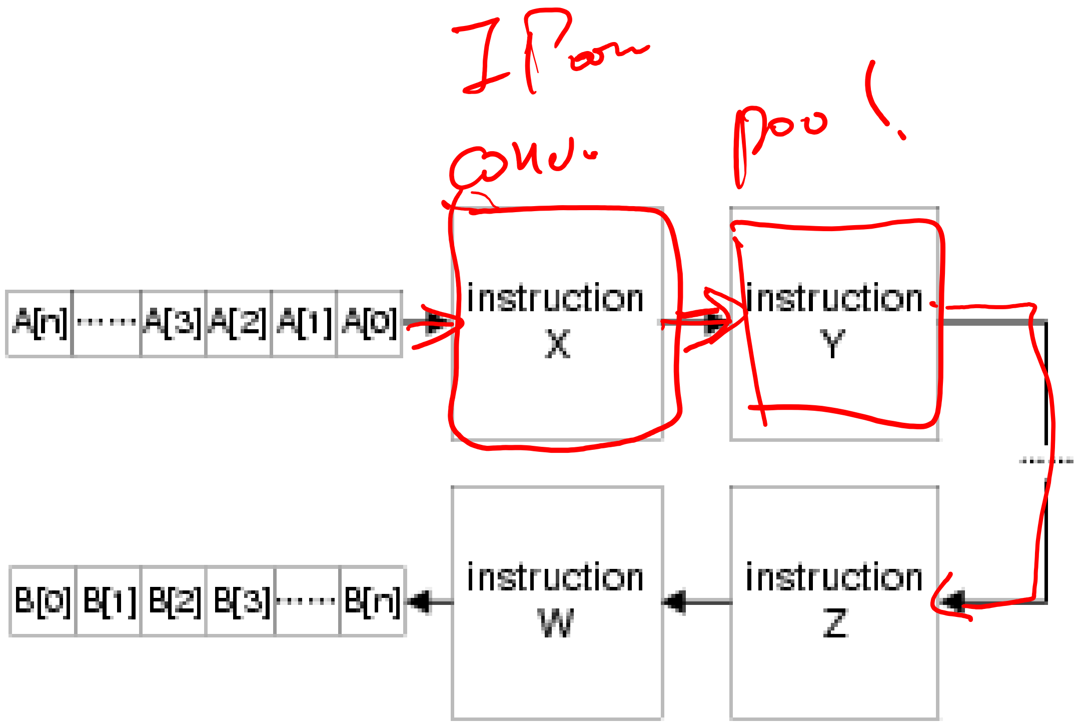
Intel's 12th Gen "Alder Lake" 10nm Desktop CPU

NVIDIA RTX A6000 Workstation Graphics Card (in my lab)

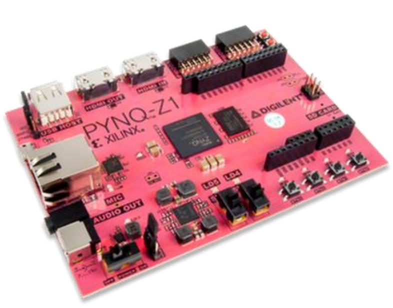
ODROID-XU4 Single Board Computer with Quad Core 2GHz A15, 2GB RAM

NVIDIA Jetson Nano

What Hardware Will Be Covered in This Course?



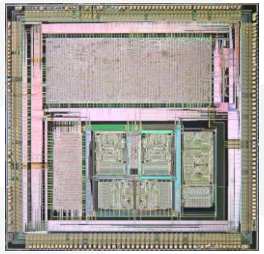
Streaming architecture: data items are pushed in and out as sequential streams, the **instructions are mapped into programmable circuit units** along the path from the input ports to output ports. Therefore, instead of fetching instructions and data back and forth from the memory, the computation gets performed as the **data streams flow** through the circuit units in one pass.



PYNQ

ZCU Series (102, 104, 106)

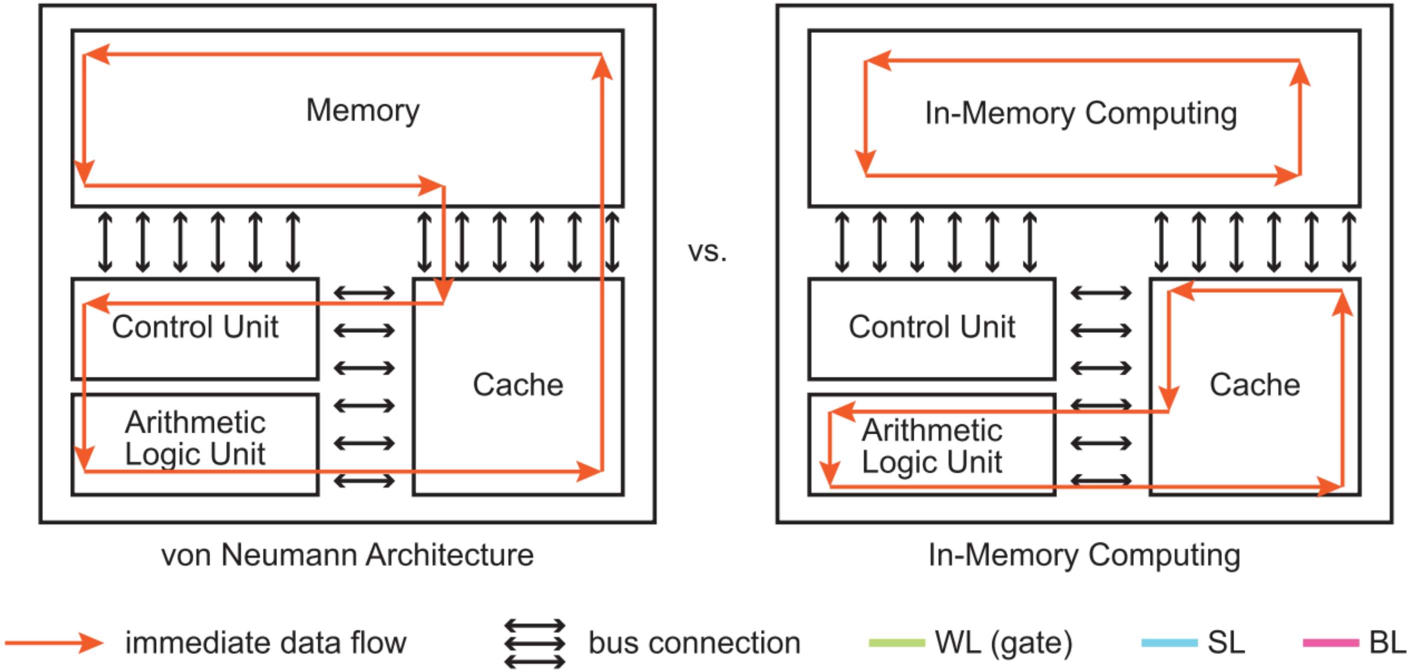
Xilinx Alveo U280 Data Center Accelerator Card



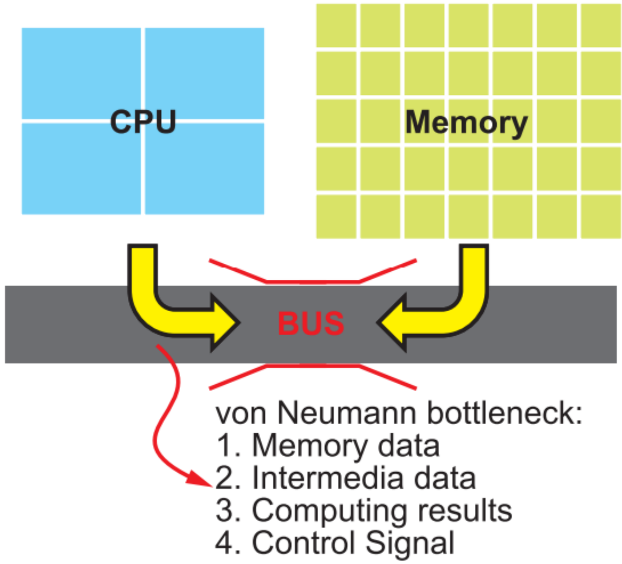
ASIC

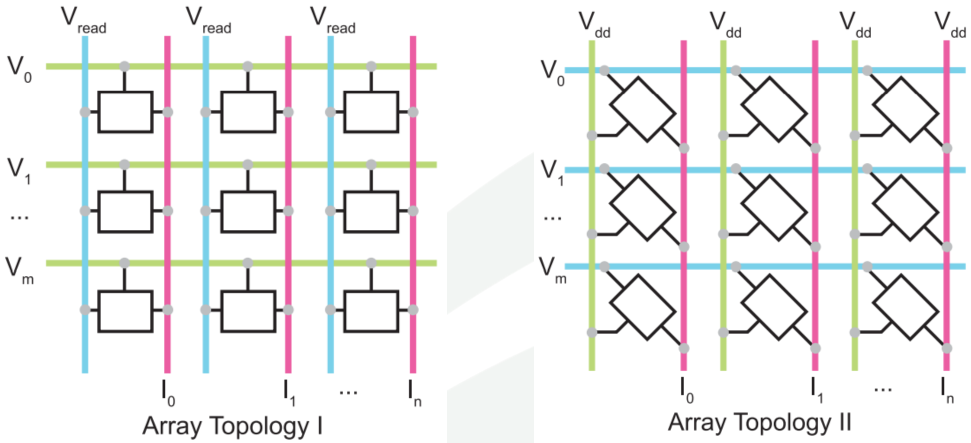
What Hardware Will Be Covered in This Course?

In-memory computing is the technique of **running computer calculations entirely in computer memory** (e.g., in RAM).



[**Yan**, Advanced Intelligent Systems 2019]





What Hardware Will Be Covered in This Course?

Quantum computing is a type of computation that harnesses the collective properties of **quantum states**, such as **superposition**, **interference**, and **entanglement**, to perform calculations.

Classical Bit

$X = 0 \text{ } \boldsymbol{or} \text{ } 1$

Quantum Bit (Qubit)

$|\psi\rangle = |0\rangle \quad \boldsymbol{and} \; |1\rangle$

$|\psi\rangle = a_0|0\rangle + a_1|1\rangle$

Reading out Information from Qubit (Measurement)

$|\psi\rangle$ 0 1

$\xrightarrow{a_0^2}$ 0 0

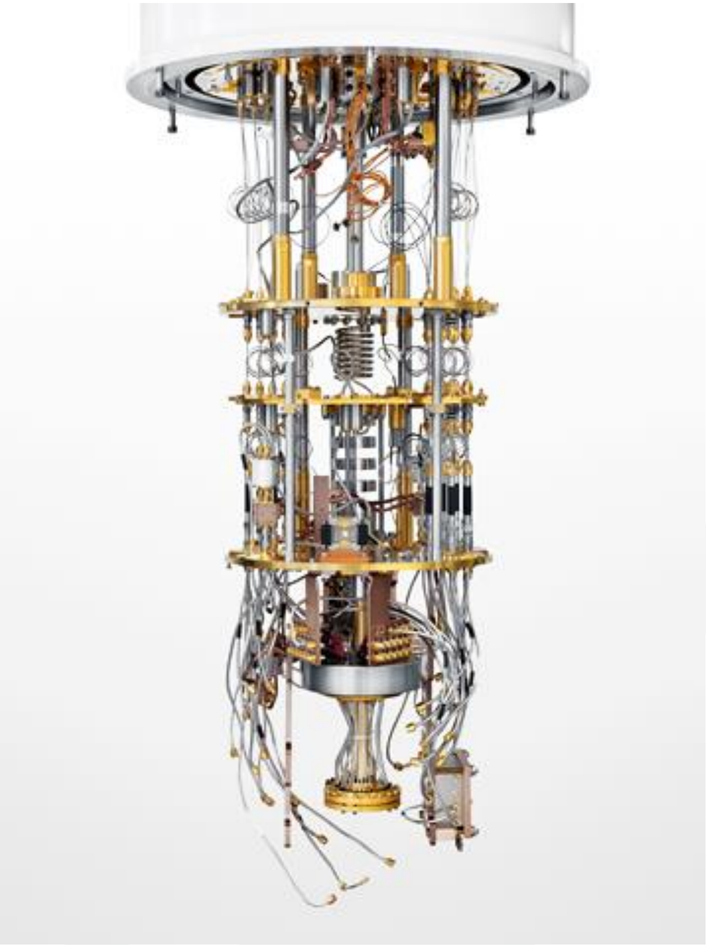
$\xrightarrow{a_1^2}$ 1 1

Probability

Non–Deterministic Computing

$a_0^2 + a_1^2 = 100\%$

$40\% + 60\% = 100\%$



Why Need Specialized Hardware Accelerators?

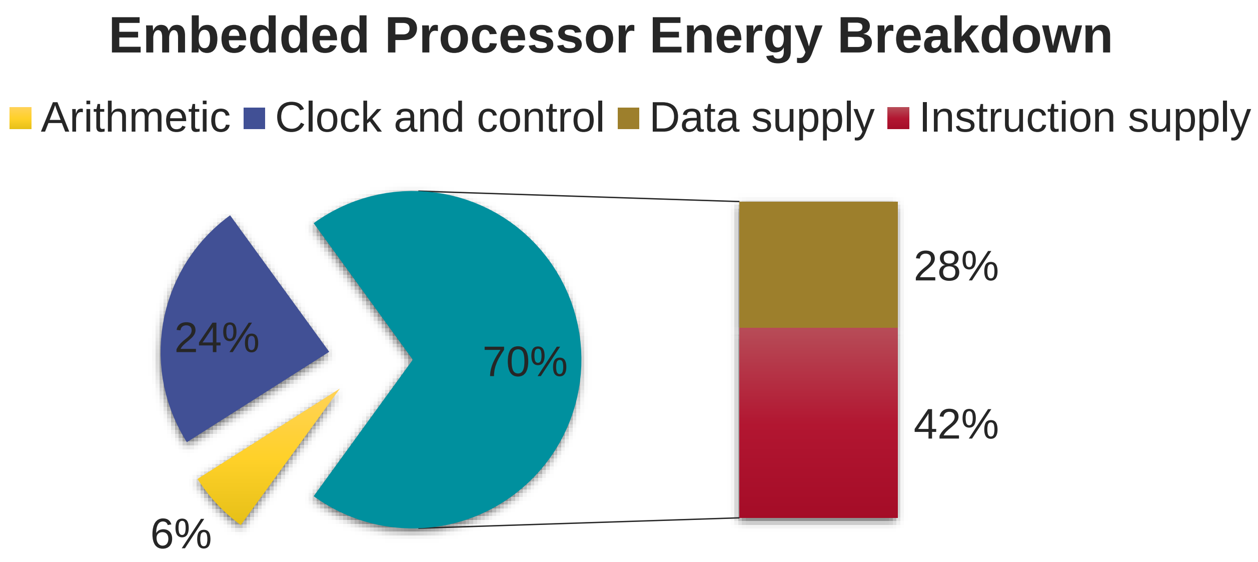
- Specialized High-Efficiency Computing!
- Why specialization?
  - **Power constraint of modern computers**

<< 1W | ~ 1W | ~ 15W | ~ 50W | ~ 100W | ~ 100W

[Image credit]: Prof. Zhiru Zhang @ Cornell

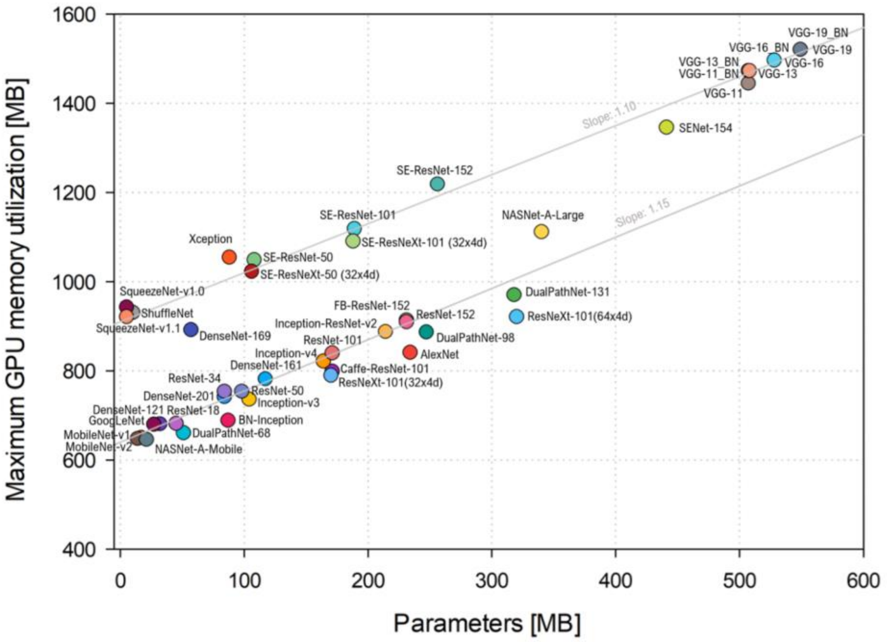
Why Need Specialized Hardware Accelerators?

- Specialized High-Efficiency Computing!
- Why specialization?
  - Power constraint of modern computers
  - **In-efficiency of general–purpose computing**

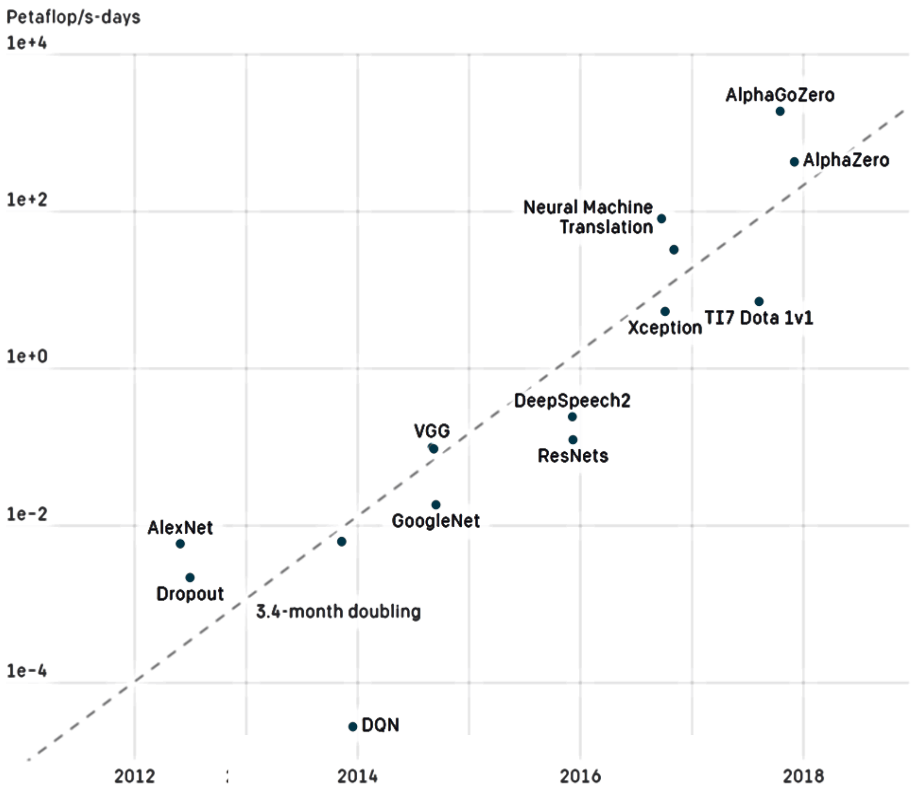


Why Need Specialized Hardware Accelerators?

- Specialized High-Efficiency Computing!
- Why specialization?
  - Power constraint of modern computers
  - In-efficiency of general–purpose computing
  - **Data and computation explosion (big data, AI)**



[**Bianco**, IEEE Access 2018]

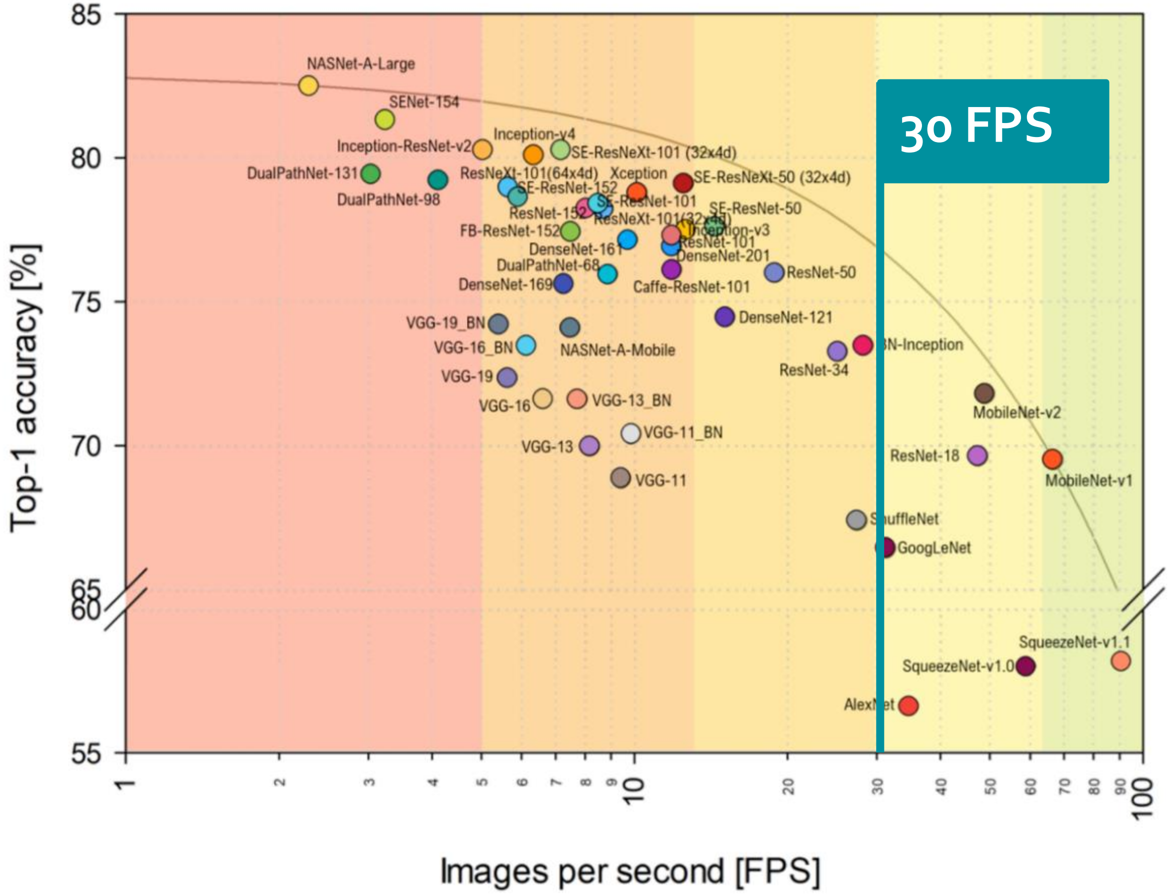


https://openai.com/blog/ai-and-compute/

[Images credit]: Prof. Callie Hao @ GATech

Why Need Specialized Hardware Accelerators?

- Specialized High-Efficiency Computing!
- Why specialization?
    - Power constraint of modern computers
    - In-efficiency of general–purpose computing
    - Data and computation explosion (big data, AI)
    - **Real-time processing requirement**



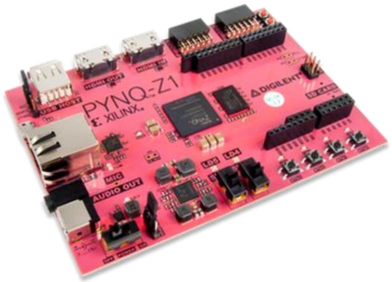
[**Bianco**, IEEE Access 2018]

[Images credit]: Prof. Callie Hao @ GATech

An Overview of Hardware Accelerators

Intel's 12th Gen "Alder Lake" 10nm Desktop CPU

NVIDIA RTX A6000 Workstation Graphics Card (in my lab)



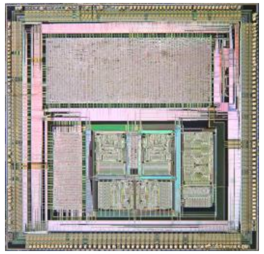
PYNQ

ZCU Series (102, 104, 106)

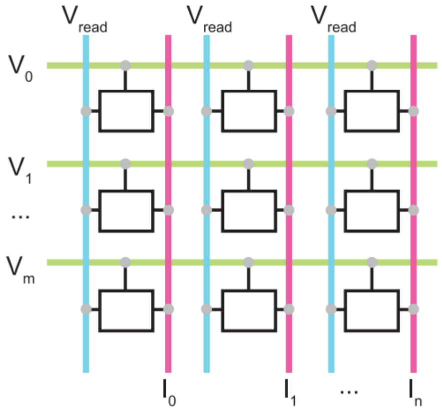
ODROID-XU4 Single Board Computer with Quad Core 2GHz A15, 2GB RAM

NVIDIA Jetson Nano

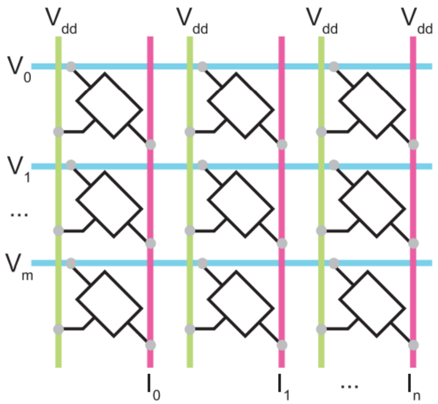
Xilinx Alveo U280 Data Center Accelerator Card



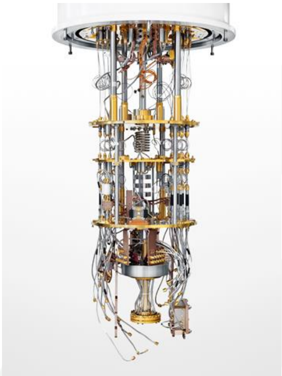
ASIC



Array Topology I



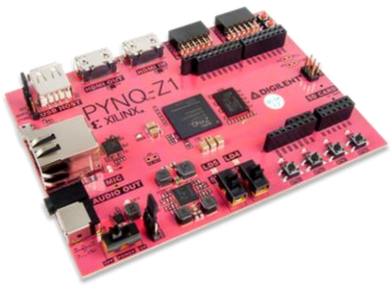
Array Topology II



Schedule

Intel's 12th Gen "Alder Lake" 10nm Desktop CPU

NVIDIA RTX A6000 Workstation Graphics Card (in my lab)



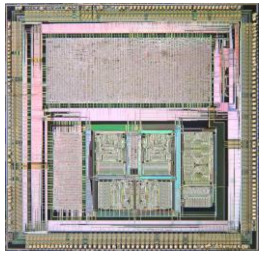
PYNQ

ZCU Series (102, 104, 106)

ODROID-XU4 Single Board Computer with Quad Core 2GHz A15, 2GB RAM

NVIDIA Jetson Nano

Xilinx Alveo U280 Data Center Accelerator Card

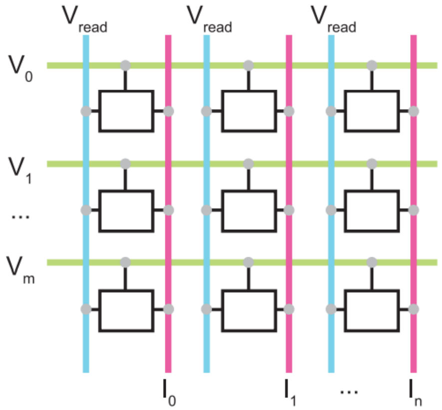


ASIC

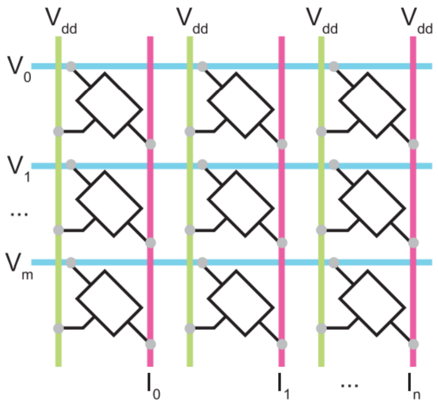
Session I: Classical Computing Accelerators for Machine Learning

Date	Topic
Jan. 24	Course Information & Machine Learning and FPGA Accelerator Recap
Jan. 31	Vector Architectures, FPGAs and GPU Architectures
Feb. 7	ASIC Accelerators

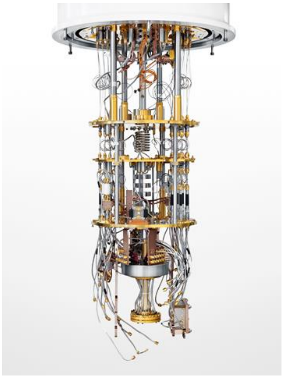
Schedule



Array Topology I



Array Topology II



Session II: Novel Post-Moore Computing Accelerators for ML

Date	Topic
Feb. 14	In-Memory Computing Accelerator Design
Feb. 21	Neuromorphic Accelerators
Feb. 28	Hyperdimensional Computing Accelerators
Mar. 07	Quantum Neural Network Accelerators

Schedule

Session III: Other Accelerator Related Topics

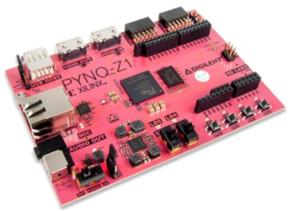
Date	Topic
Mar. 28	Project Proposal
Apr. 04	Distributed Learning
Apr. 11	Hands-on Accelerator Design (1)
Apr. 18	Project Overview
Apr. 25	Hands-on Accelerator Design (2)
May 02	Project Presentations
May 11-18	Final exam

Expectation & Final Project

- Implement ML on any hardware in a team with 1-3 students

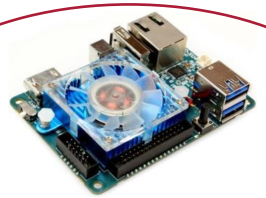
Intel's 12th Gen "Alder Lake" 10nm Desktop CPU

NVIDIA RTX A6000 Workstation Graphics Card (in my lab)



PYNQ

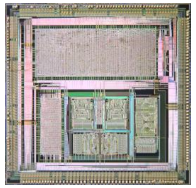
ZCU Series (102, 104, 106)



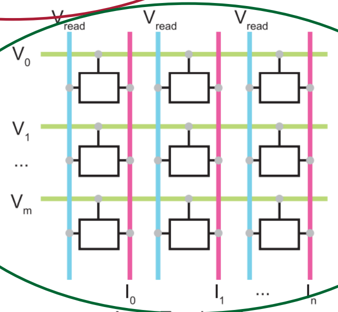
ODROID-XU4 Single Board Computer with Quad Core 2GHz A15, 2GB RAM

NVIDIA Jetson Nano

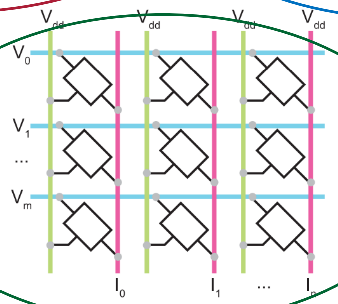
Xilinx Alveo U280 Data Center Accelerator Card



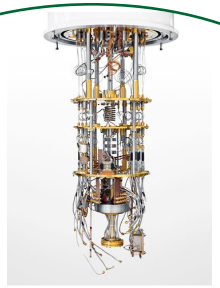
ASIC



Array Topology I



Array Topology II



What Did We Learn in ECE 554? (Recap)

Application
- Computer Vision
- Natural Language
- Games
- …

ML/DL Algorithms
- AN
- MLP
- CNN
- RNN
- LSTM
- RL
- Transformer
- MLP-Mixer
- …

Optimization
- Network Inference
- Network Training
- Model Compression
- Network Design

Embedded Systems

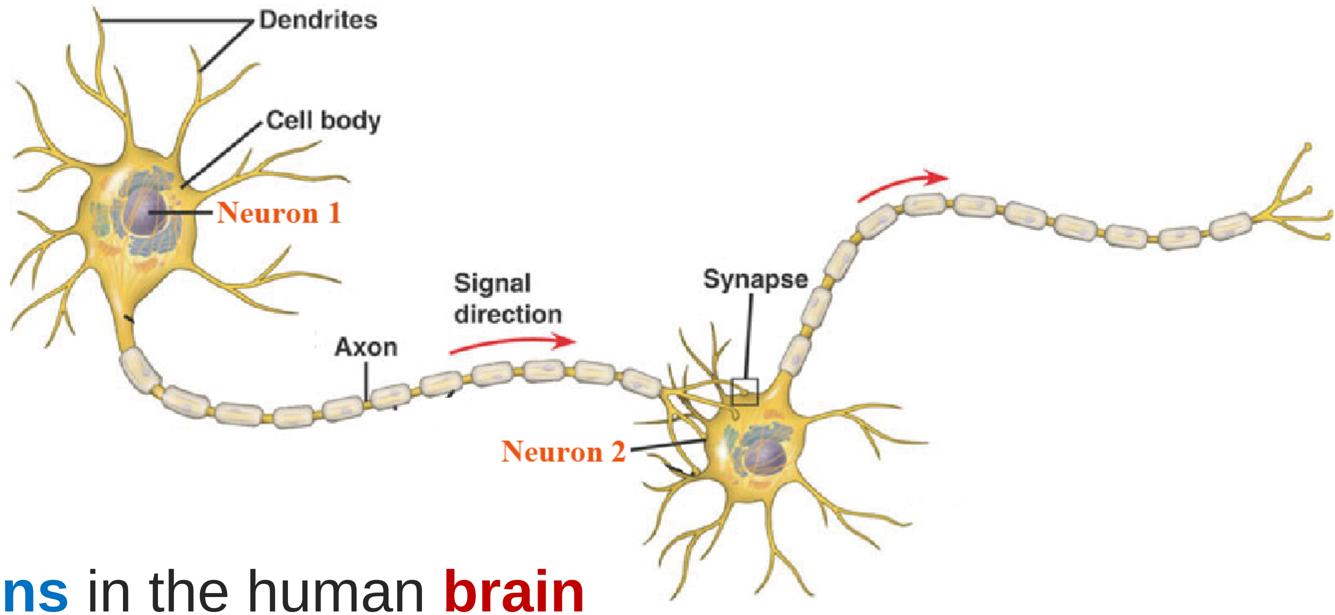
Performance Model
- Resource Usage
- Power
- Latency

Optimization
- Hardware Design
- Mapping/Scheduling
- Communication

ECE 554 Course Recap

- **Machine Learning Basis:**
  - Different neural networks: **MLP**, **CNN**, **RNN**, **RL**
  - Training (**Gradient Descent**) and inferencing neural networks using Pytorch
  - Implement convolution using "**for loops**"

Biological Neuron



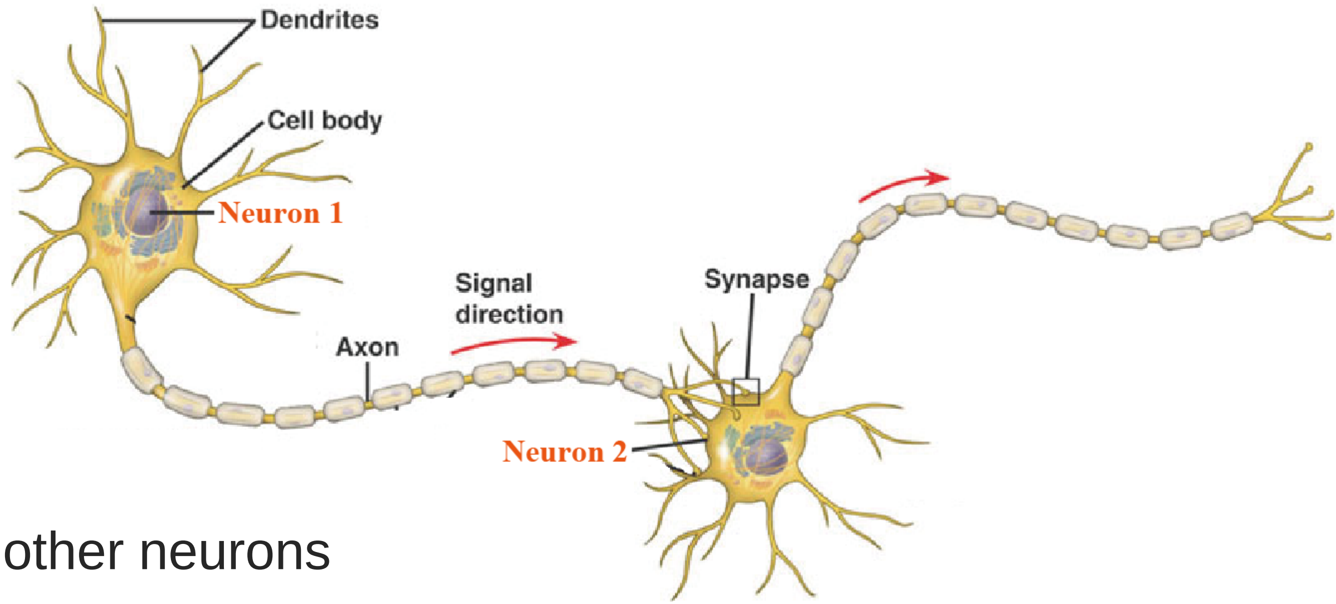
Human intelligence reside in the brain:

- Approximately **86 billion neurons** in the human **brain**
- The brain is a **network** of **neurons**, connected with nearly $10^{14} - 10^{15}$ **synapses**

How to equip intelligence in the machine?

- **To understand how the brain network is constructed**
- **To mimic the brain**

Biological Neuron



Neurons work together:

- **Cell body** process the information
- **Dendrites** receive messages from other neurons
- **Axon** transmit the output to many smaller branches
- **Synapses** are the contact points between axon (Neuron 1) and dendrites (Neuron 2) for message passing

Cell body receives input signal from **dendrites** and produce output signal along **axon**, which interact with the next neurons via **synaptic weights**

Synaptic weights are **learnable** to perform useful computations

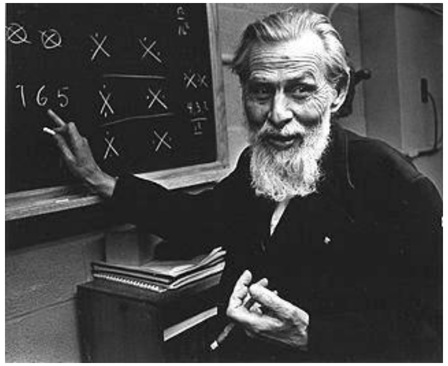
(e.g., Recognizing objects, understanding language, making plans, controlling the body.)

Artificial Neuron Design

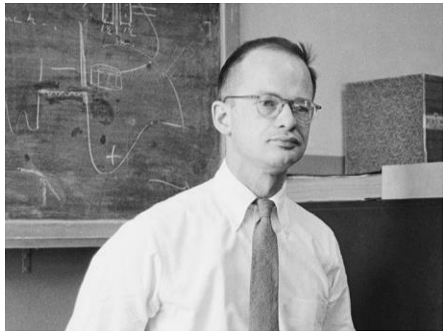
- **Idealized neuron models**
  - Idealization **removes complicated details** that are not essential for understanding the main principles.
  - It allows us to apply **mathematics** and to make **analogies**.

McCulloch-Pitts (MP) Neuron

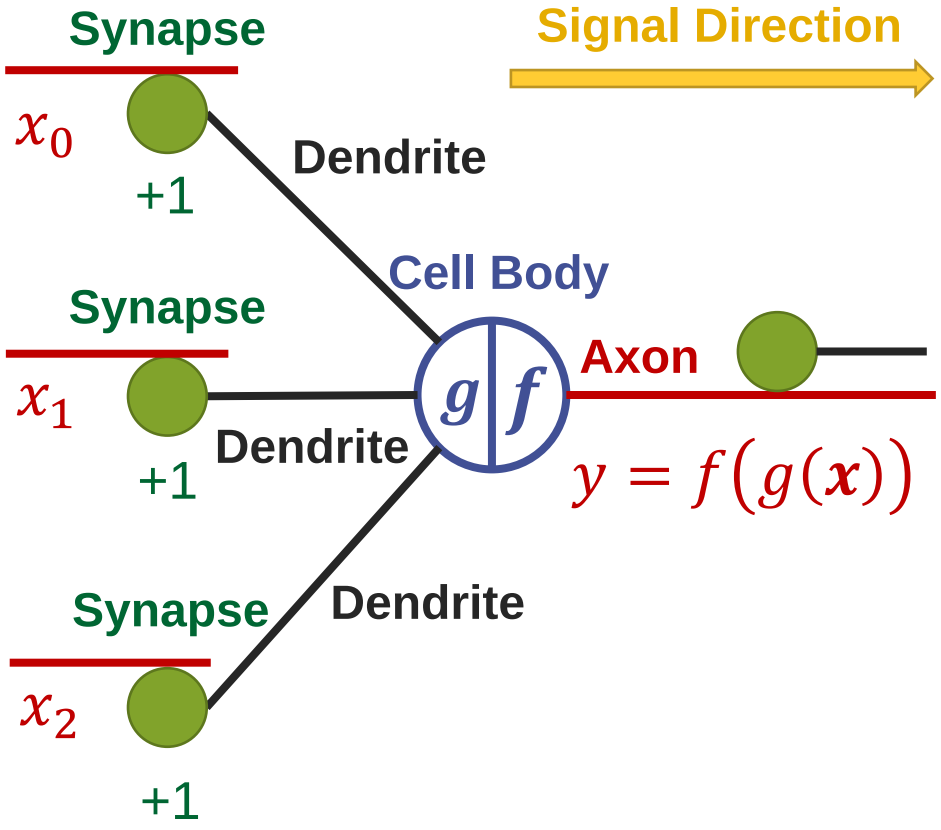
The first computational model of a biological neuron @ 1943



Warren McCulloch

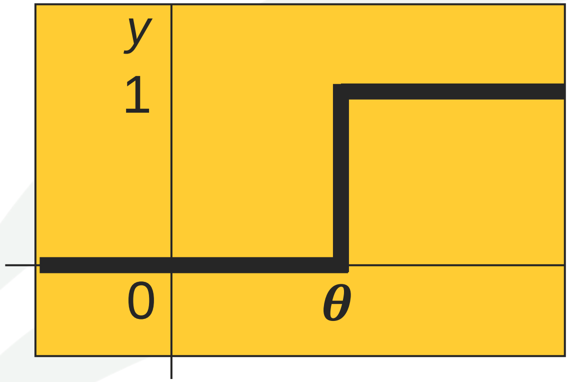


Walter Pitts



Assumptions:

- Binary devices (i.e., $x_i \in \{0,1\}$ and $y \in \{0,1\}$)
- Identical synaptic weights (i.e., +1)
- Activation function f has a fixed threshold $\boldsymbol{\theta}$



0 0 0
0 1 1
1 0 1
1 1 0

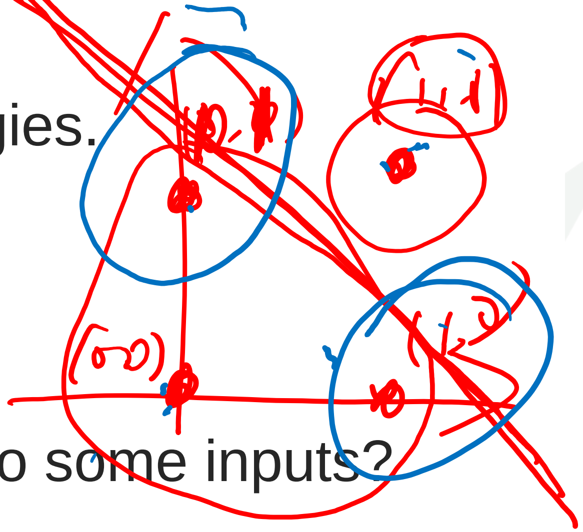
XOR

~~(AND)~~

0

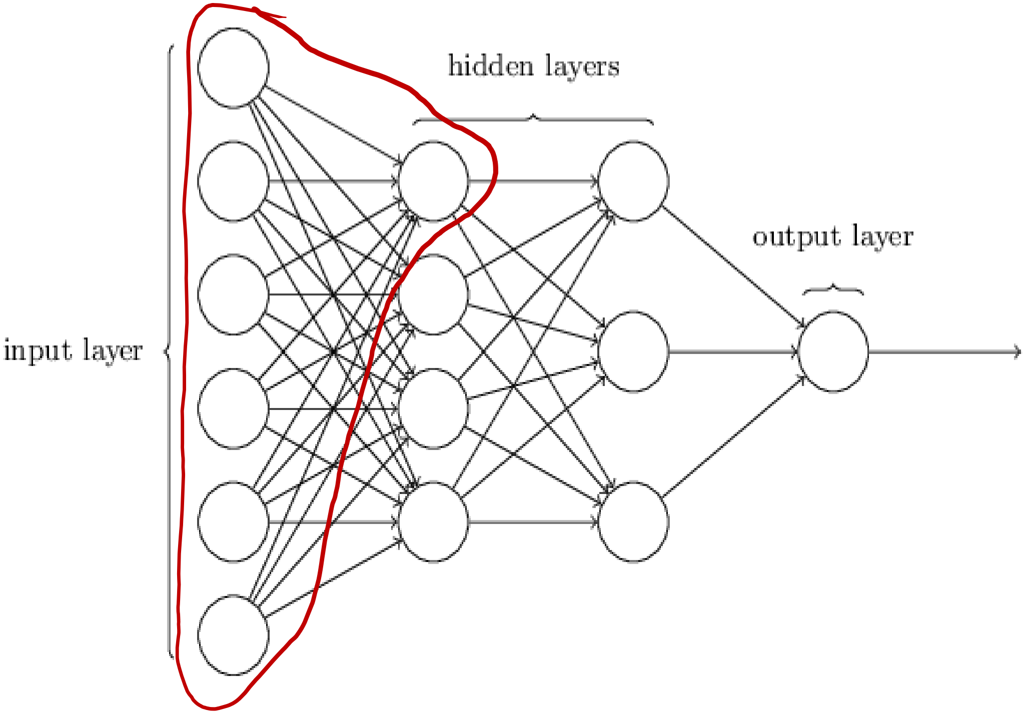
Artificial Neuron Design

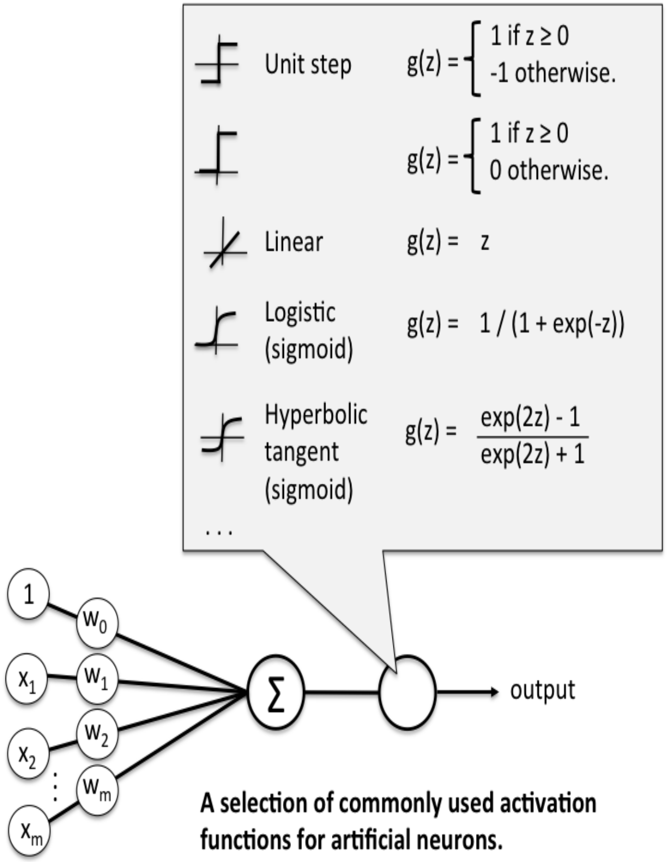
- **Idealized neuron models**
  - Idealization removes complicated details that are not essential for understanding the main principles.
  - It allows us to apply mathematics and to make analogies.
- **Break the limitations on MP Neuron**
  - What about non-boolean inputs (say, real number)?
  - What if we want to assign more weight (importance) to some inputs?
  - What about functions which are not linearly separable ?
  - Do we always need to hand code the threshold?



Multi-Layer Perceptron (MLP) – Lecture 2

- Input layer, output layer and hidden layers



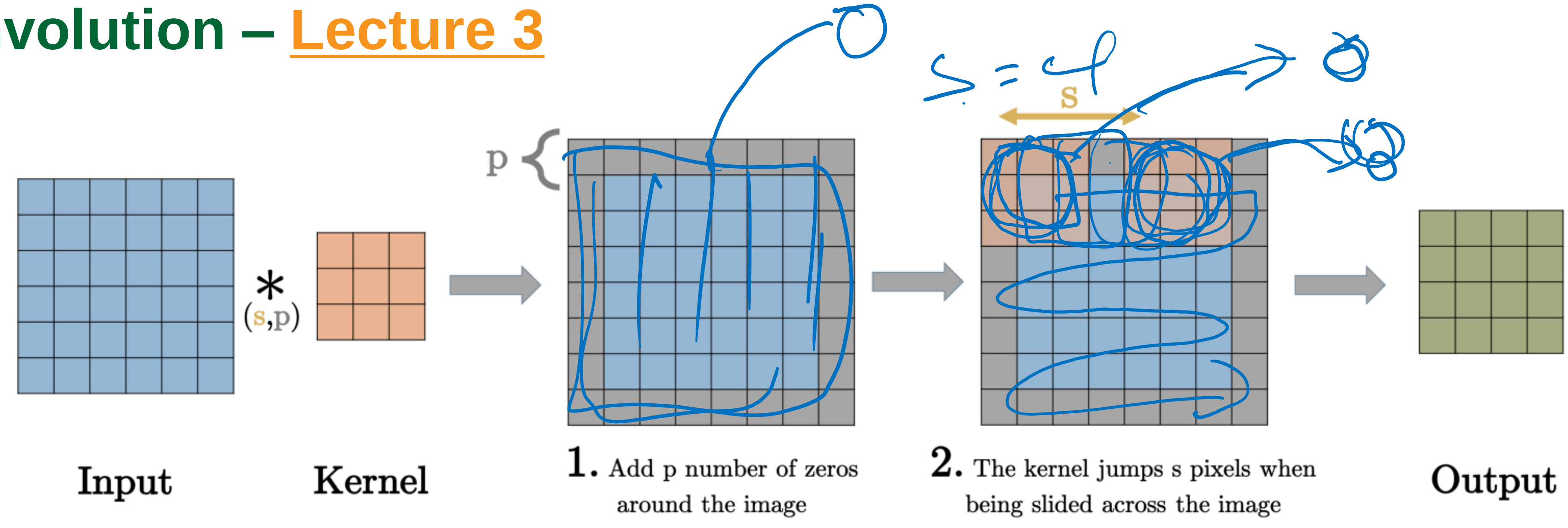


A selection of commonly used activation functions for artificial neurons.

Deep Convolutional Neural Networks (CNN) – Lecture 3

- One of the most widely used types of deep network
- Fully-connected nets treat far apart input pixels same as those close by
  - Hence spatial information must be inferred from the training data
- In contrast, CNN proposes an architecture that inherently tries to take advantage of the spatial structure
  - Such an architecture makes convolutional networks fast to train
  - This, in turn, helps us train even deeper, many-layer networks
- Today, deep convolutional networks or some close variants are used in solving many interesting problems that go beyond image classification
- We will use image classification as a driving use case to explain the main concepts behind CNN

Convolution – Lecture 3



Parameters:

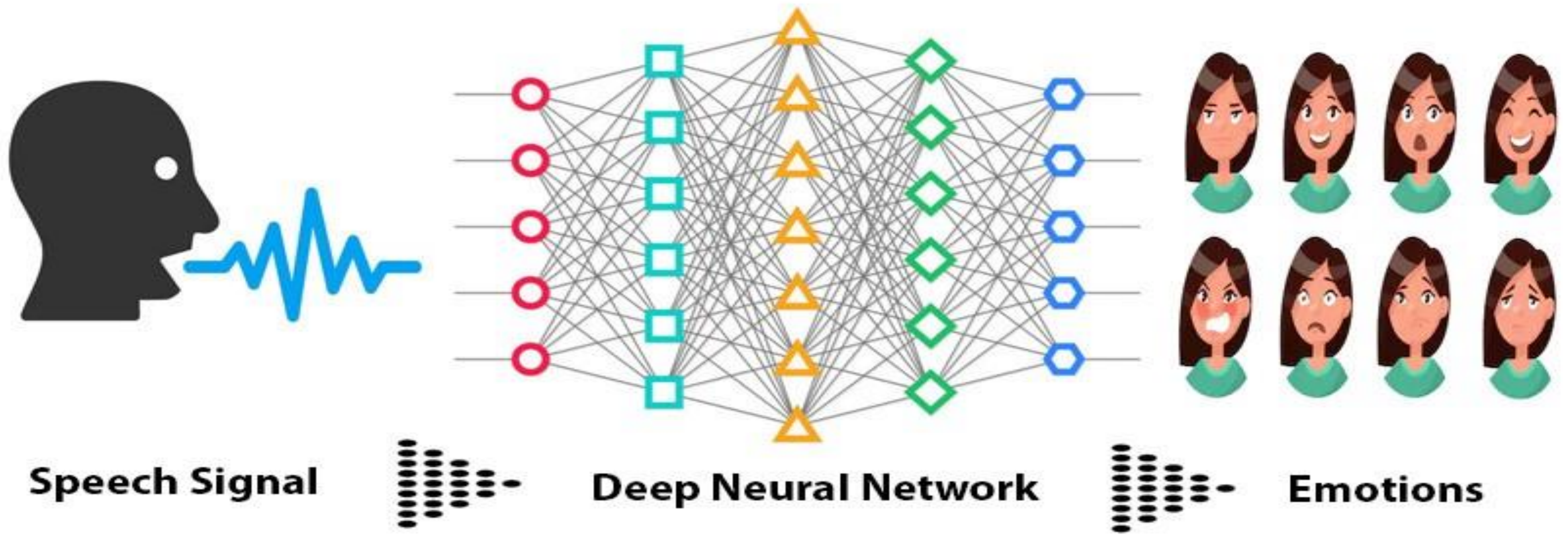
- N: input channels
- M: output channels
- K: kernel size
- P: padding size
- S: stride
- *D: dilation*
- *R: rows*
- *C: columns*

CLASS

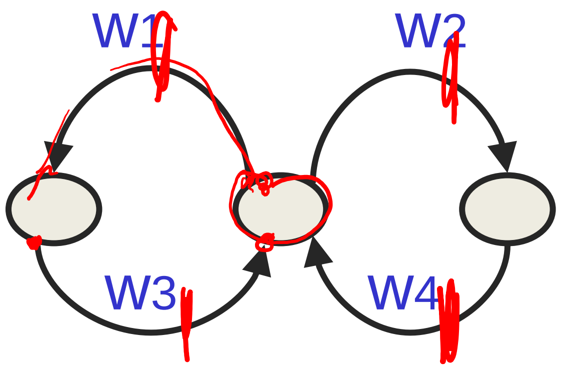
torch.nn.Conv2d(*in_channels*, *out_channels*, *kernel_size*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*, *bias=True*, *padding_mode='zeros'*, *device=None*, *dtype=None*)

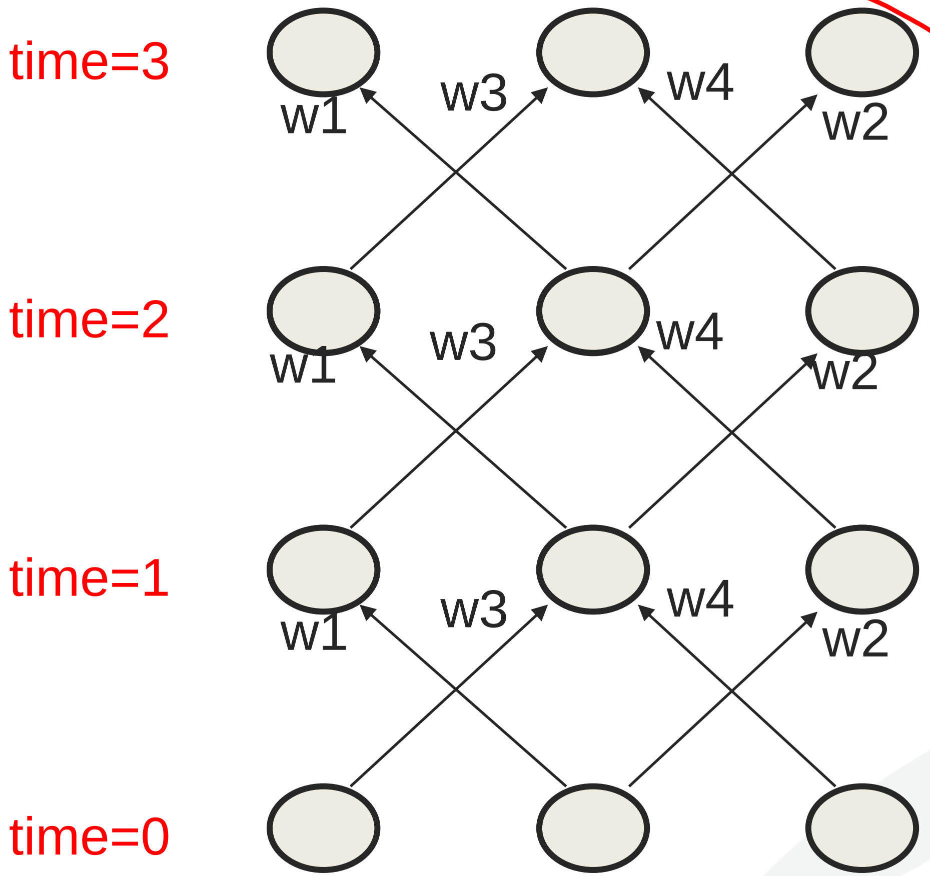
[ref] Aqeel Anwar, What is Transposed Convolutional Layer? https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11

From Static Image to Sequences of Data



RNN and Feedforward Network – Lecture 5

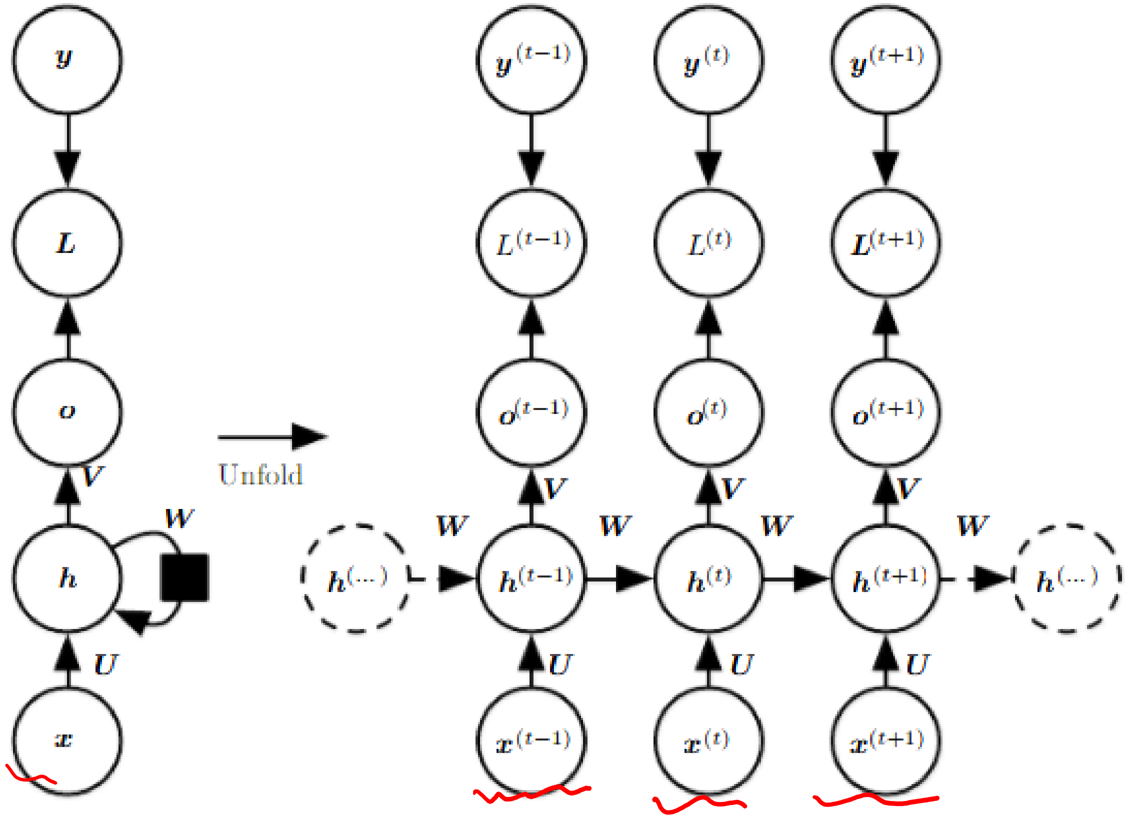




Courtesy to Geff. Hinton

- Assume each connection has 1 unit delay
- RNN can be unrolled into feedforward networks
  - Each layer keeps on reusing the same weights

RNN and Feedforward Network – Lecture 5



$$
\begin{aligned}
\boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\
\hat{\boldsymbol{y}}^{(t)} &= \mathrm{softmax}(\boldsymbol{o}^{(t)}),
\end{aligned}
$$

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

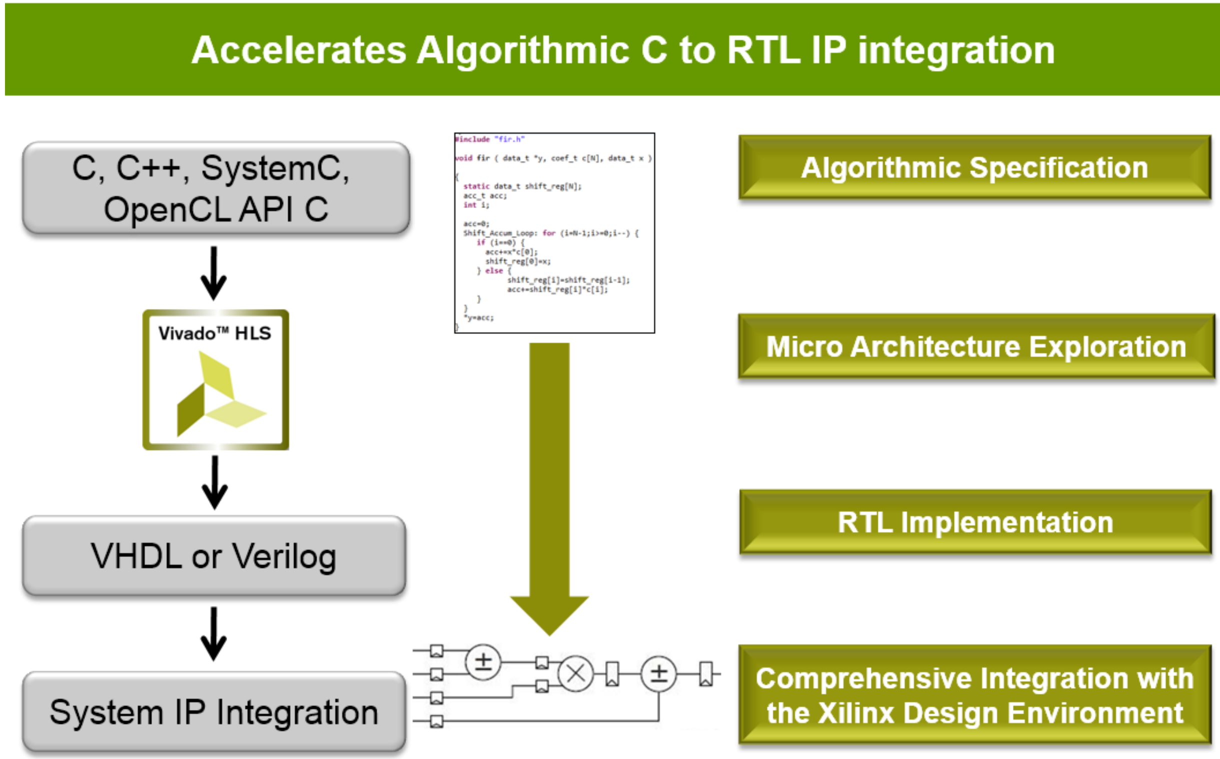
From GoodFellow et al.
Deep Learning

ECE 554 Course Recap

- **Machine Learning Basis:**
  - Different neural networks: **MLP**, **CNN**, **RNN**, **RL**
  - Training (**Gradient Descent**) and inferencing neural networks using Pytorch
  - Implement convolution using “**for loops**”
- **Put Machine Learning onto Embedded Systems:**
  - Introduction to HLS (Lec 8-9)
    - Using **MLP** as example in class
    - Using **CNN** as example in Labs, which is based on the “**for loop**” implementation
  - Model compression on FPGA: pruning and quantization (Lec 10-11)
  - Neural architecture search (Lec 12)
    - Using **RNN-based RL** as controller/optimizer
    - Using **Gradient Descent** approach for optimization
  - Data movement in HLS-based FPGA implementation (Lec 13)
  - Co-explore neural architectures and FPGA design (Lec 14)

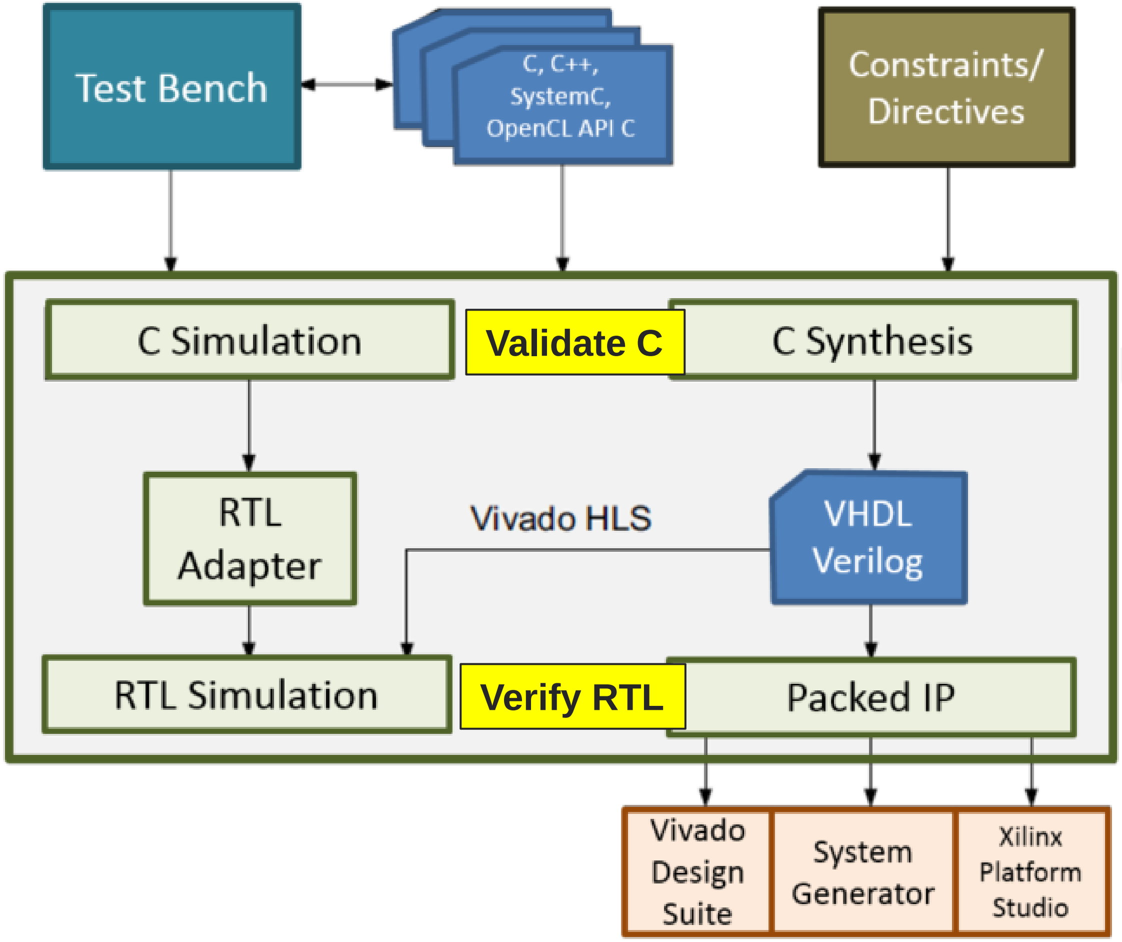
High-Level Synthesis: HLS – Lecture 8

- High-Level Synthesis
  - Creates an RTL implementation from C, C++, System C, OpenCL API C kernel code
  - Extracts control and dataflow from the source code
  - Implements the design based on defaults and user applied **directives**
- Many implementation are possible from the same source description
  - Smaller designs, faster designs, optimal designs
  - Enables design exploration

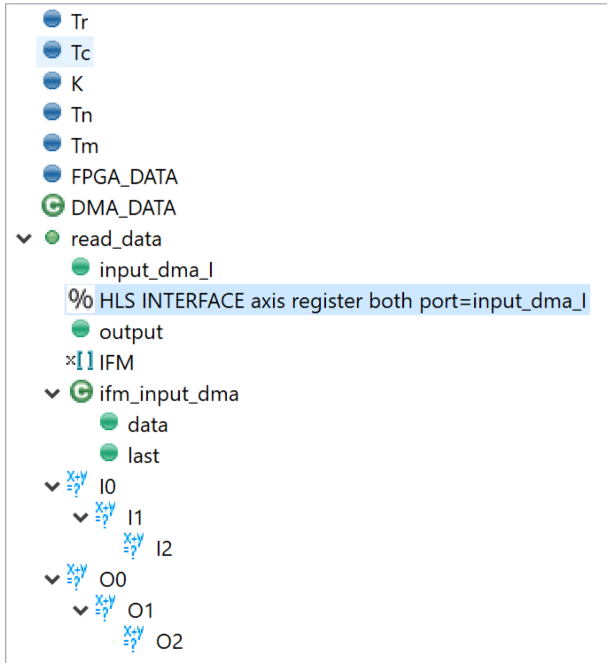


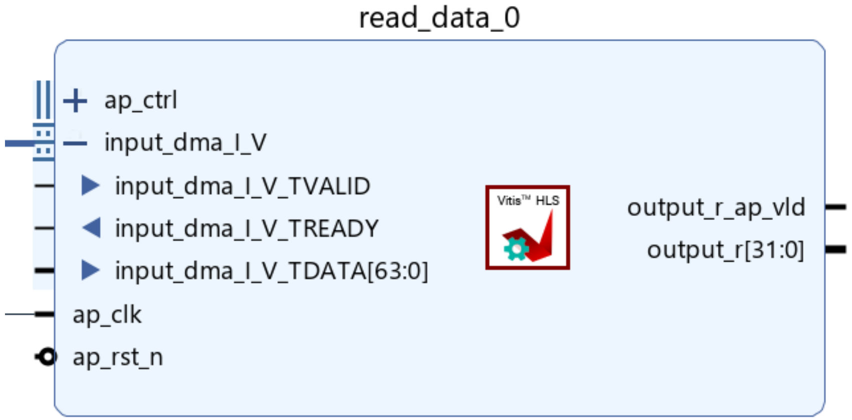
C Validation and RTL Verification – Lecture 8

- There are two steps to verifying the design
  - Pre-synthesis: C ***Validation***
    - Validate the algorithm is correct
  - Post-synthesis: RTL ***Verification***
    - Verify the RTL is correct
- C validation
  - A **HUGE** reason users want to use HLS
    - Fast, free verification
  - Validate the algorithm is correct **before** synthesis
    - Follow the test bench tips given over
- RTL Verification
  - Vivado HLS can co-simulate the RTL with the original test bench



AXI_Stream – Lecture 13





Read_data (Pre-Production)

```
#include <ap_fixed.h>
#include <hls_stream.h>

const int Tr=4;
const int Tc=4;
const int K=3;
const int Tn=3;
const int Tm=6;

//typedef ap_fixed<16,8,AP_TRN_ZERO, AP_SAT> FPGA_DATA;
typedef float FPGA_DATA;
struct DMA_DATA{
    FPGA_DATA data;
    bool last;
};

void read_data(hls::stream<DMA_DATA> &input_dma_I, FPGA_DATA *output){

    static FPGA_DATA IFM[Tn][Tr+K-1][Tc+K-1];

    DMA_DATA ifm_input_dma;
    I0:for(int i=0;i<Tn;i++){
        I1:for(int j=0;j<Tr+K-1;j++){
            I2:for(int m=0;m<Tc+K-1;m++){
                ifm_input_dma=input_dma_I.read();
                IFM[i][j][m]=ifm_input_dma.data;
            }
        }
    }

    O0:for(int i=0;i<Tn;i++){
        O1:for(int j=0;j<Tr+K-1;j++){
            O2:for(int m=0;m<Tc+K-1;m++){
                output[i*(Tr+K-1)*(Tc+K-1)+j*(Tc+K-1)+m] = IFM[i][j][m]+2;
            }
        }
    }

}
```

Test Bench – Lecture 13

```c
FPGA_DATA input[Tn*(Tr+K-1)*(Tc+K-1)] = { 0.47902363538742065,0.5932260751724243,0.59

void read_data(hls::stream<DMA_DATA> &input_dma_W, FPGA_DATA *output);

int main(){
    hls::stream<DMA_DATA> input_dma_I("input_dma_I");

    FPGA_DATA y[Tn*(Tr+K-1)*(Tc+K-1)]={0};

    DMA_DATA ifm;
    for(int i=0;i<Tn;i++){
        for(int j=0;j<Tr+K-1;j++){
            for(int m=0;m<Tc+K-1;m++){
                ifm.data = input[i*(Tr+K-1)*(Tc+K-1)+j*(Tc+K-1)+m];
                if(i==Tn-1 && j==Tr+K-1-1 && m==Tc+K-1-1)
                    ifm.last = true;
                else
                    ifm.last = false;
                input_dma_I.write(ifm);
            }
        }
    }
    read_data(input_dma_I,y);

    for(int i=0;i<Tn;i++){
        for(int j=0;j<Tr+K-1;j++){
            for(int m=0;m<Tc+K-1;m++){
                printf("gap: %f\n", input[i*(Tr+K-1)*(Tc+K-1)+j*(Tc+K-1)+m]-y[i*(Tr+K
            }
        }
    }
    printf("Done!\n");
    return 0;
}
```

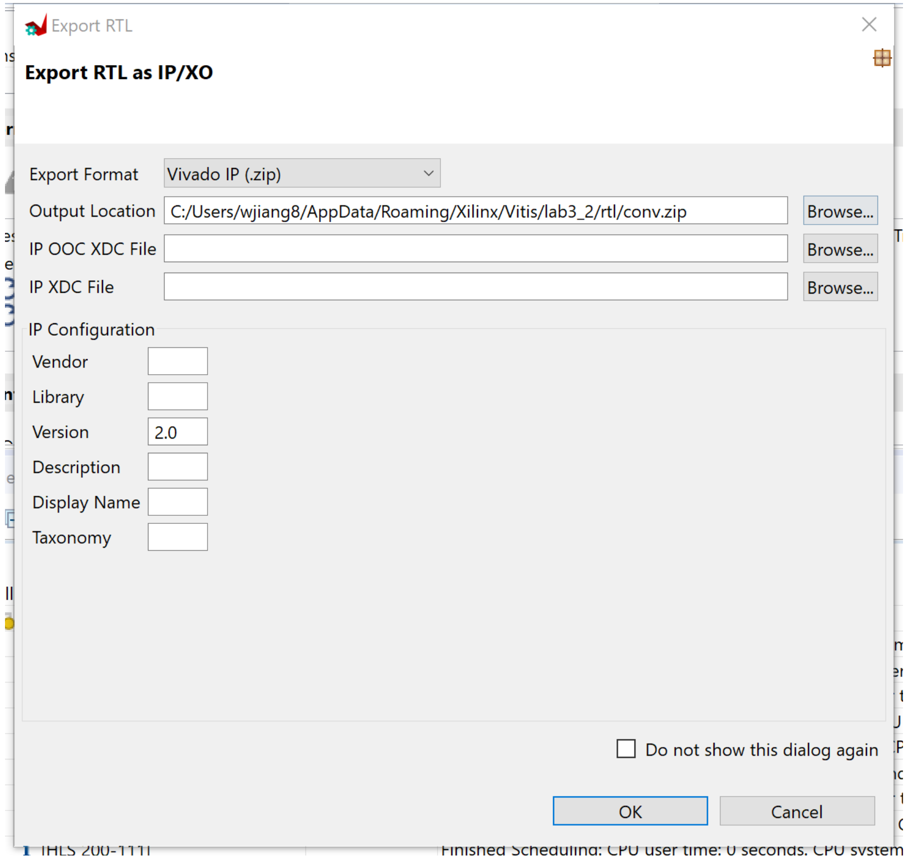
Golden results generation code for Lab 4

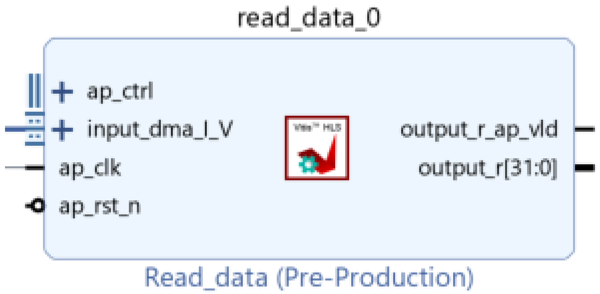
```python
import torch
import torch.nn as nn

Tr=16
Tc=16
K=3
Tn=3
Tm=16

input = torch.rand(1,Tn,Tr+K-1,Tc+K-1)
weight = torch.rand(Tm, Tn, K, K)

conv = nn.Conv2d(3,2,3,bias=False)
conv.weight = torch.nn.Parameter(weight)

output = conv(input)

torch.set_printoptions(precision = 16)
torch.set_printoptions(profile="full")

print("FPGA_DATA input[Tn*(Tr+K-1)*(Tc+K-1)] = {", end=" ")
for elem in list(input.flatten()):
  print(float(elem), end=",")
print("};")

print("FPGA_DATA weight[Tm*Tn*K*K] = {", end=" ")
for elem in list(weight.flatten()):
  print(float(elem), end=",")
print("};")


print("FPGA_DATA output[Tm*Tr*Tc] = {", end=" ")
for elem in list(output.flatten()):
  print(float(elem), end=",")
print("};")

print()
```

https://colab.research.google.com/drive/1TufHcDNMftm3bwAfcKEF5Njev0y_v6rM#scrollTo=X3pBQmyNW4rs

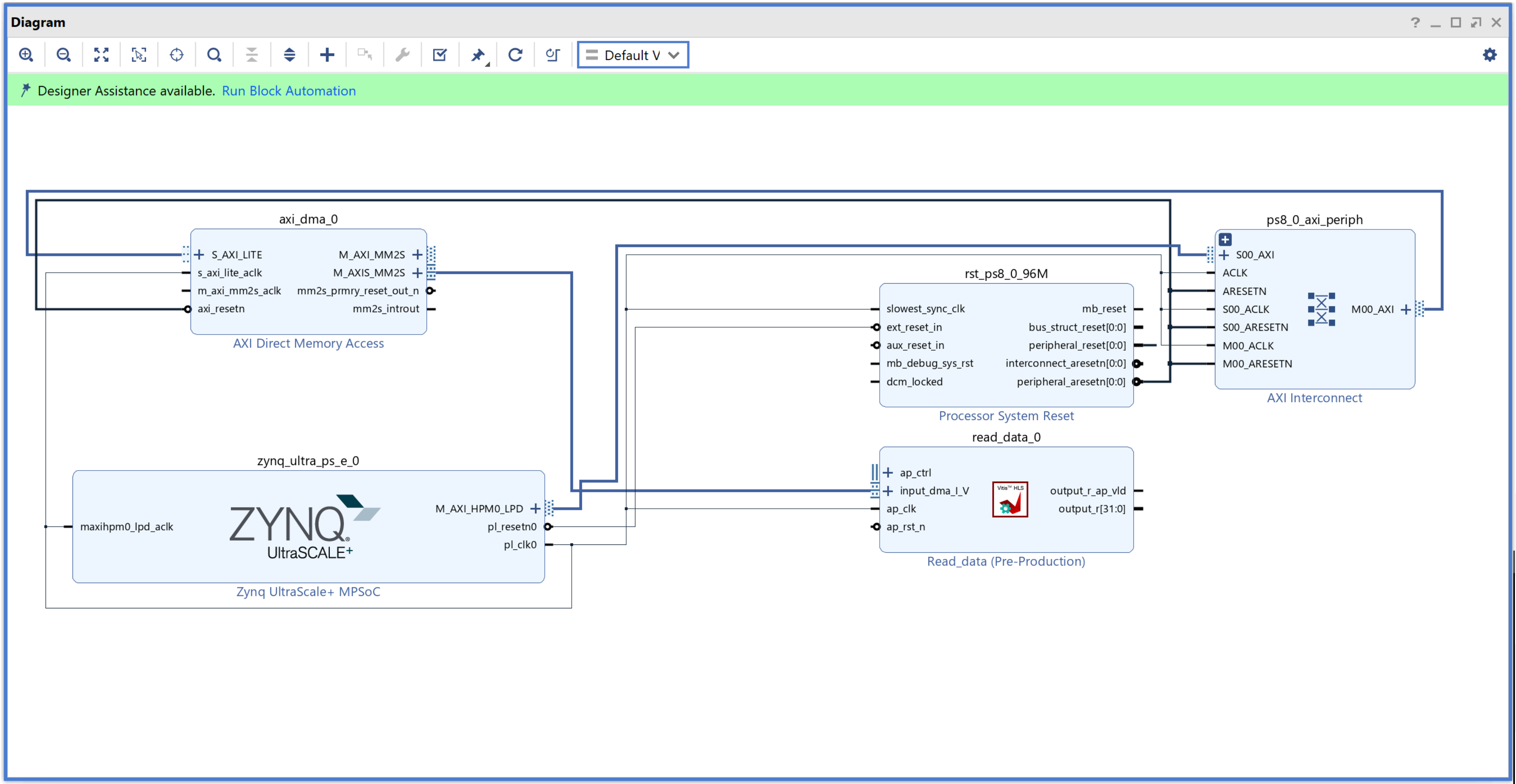
Export RTL as IP Core – Lecture 13

Synthesis





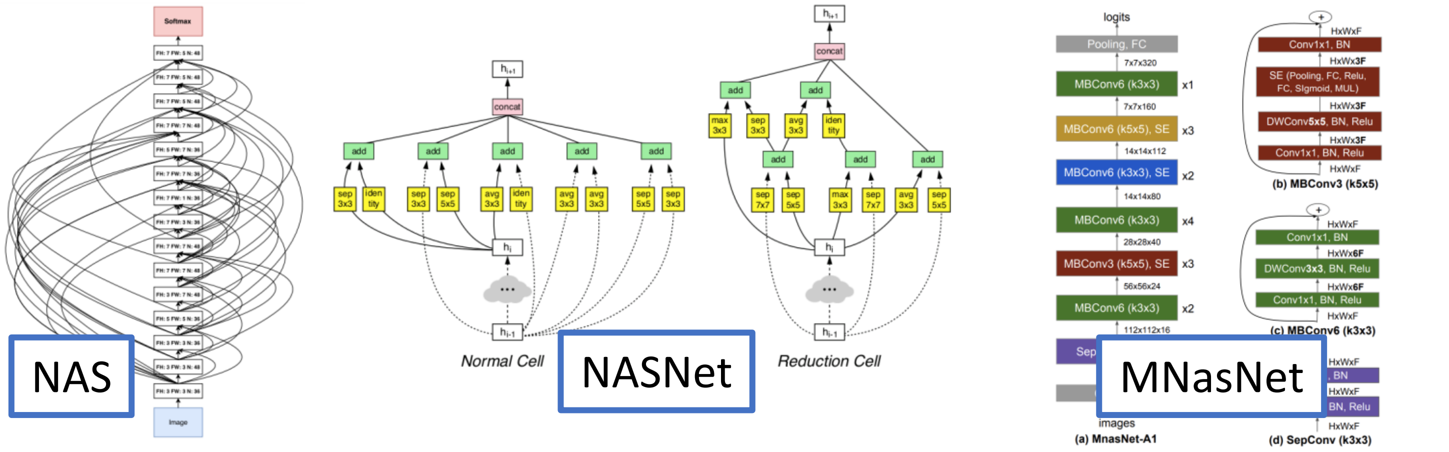
Import the IP into Block Design – Lecture 13



Goal: Enable AI for Everyone – Lecture 14

AI Democratization —— Two Levels

Level 1: Automation of Neural Network Design



NAS

NASNet

MNasNet

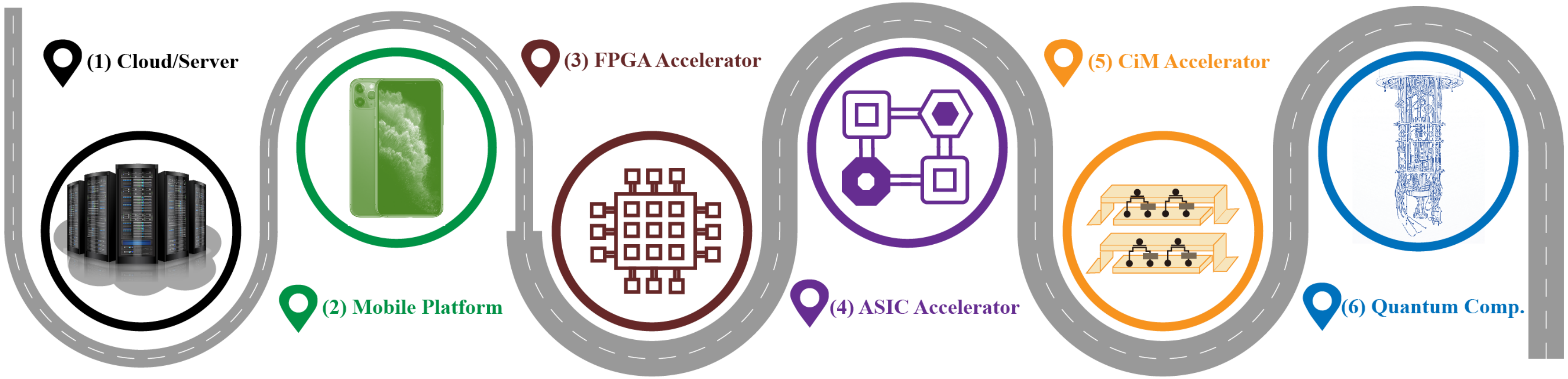
ProxylessNAS

FBNet

FNAS

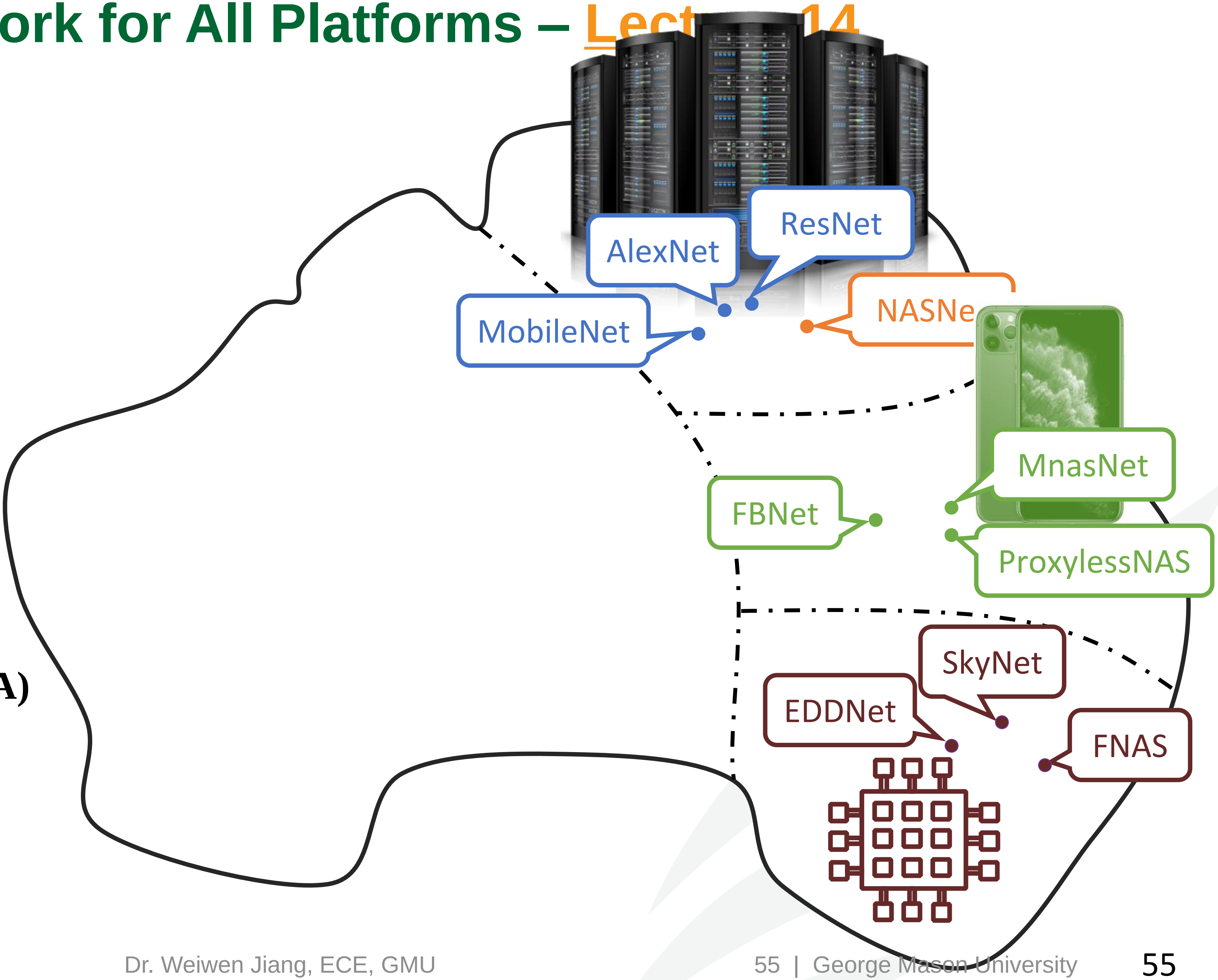
……

Level 2: Automation of AI System Design



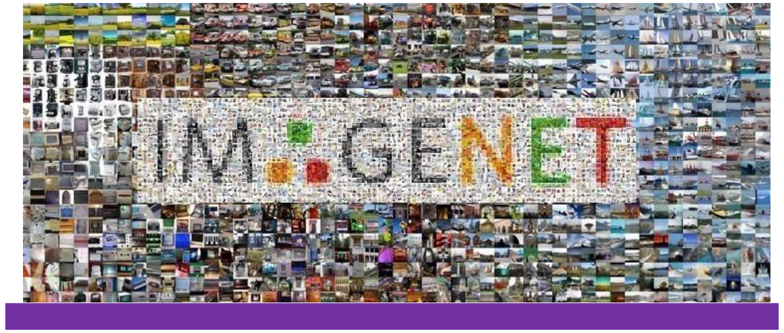
One Network Cannot Work for All Platforms – Lecture 14

- **Cloud / Server**
  - Unlimited Resource
  - Maximizing Accuracy
  - AlexNet, VGGNet, ResNet, …
- **Mobile Phones**
  - Fixed Hardware
  - Accuracy v.s. Latency
  - MnasNet, ProxylessNAS, …
- **Hardware Accelerators (e.g., FPGA)**
  - Hardware Design Flexibility
  - Accuracy, Latency, and Energy
  - FNAS, SkyNet, EDDNet…



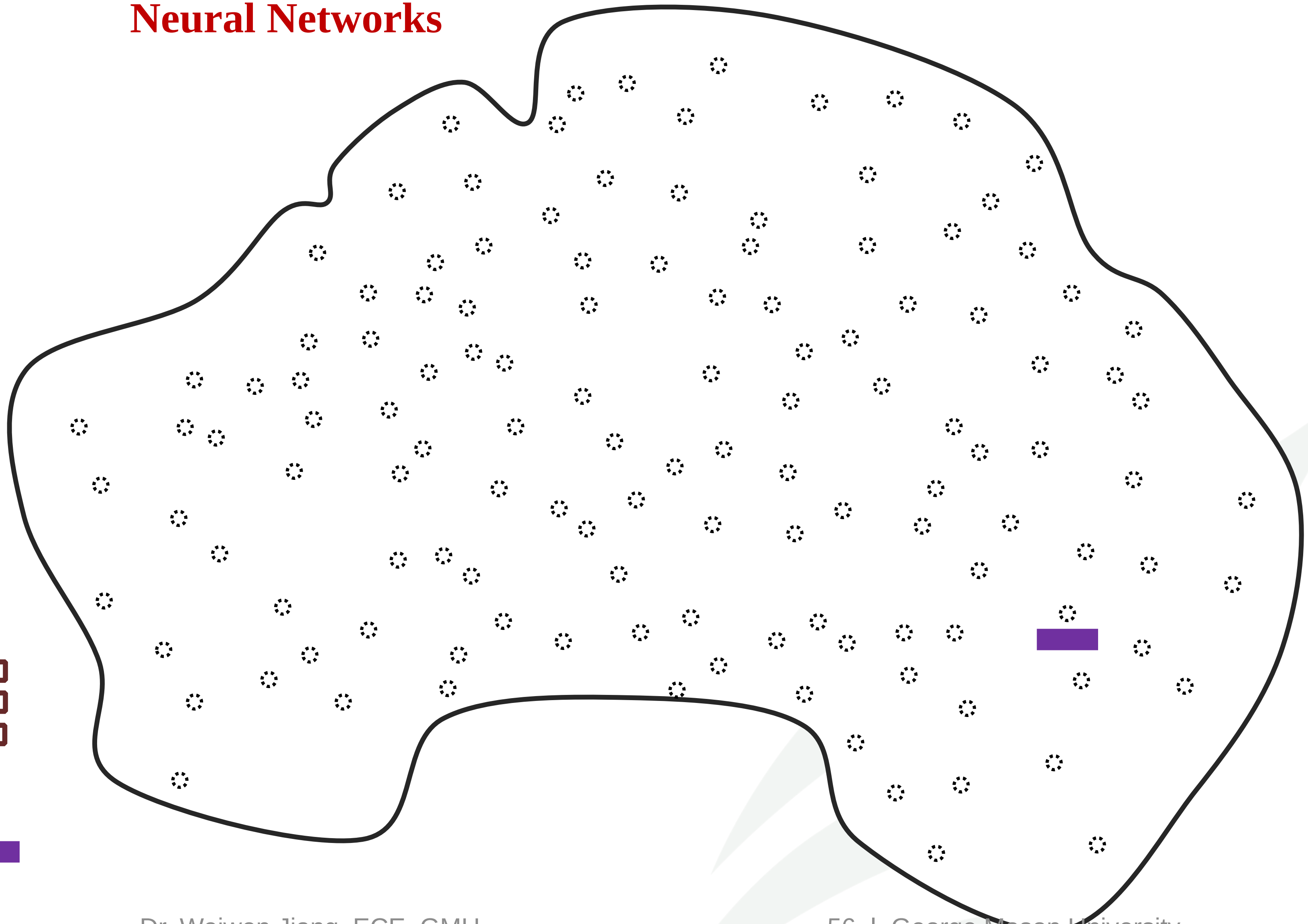
Datasets/Applications, Hardware, and Neural Networks – Lecture 14

Datasets / Applications



Hardware Platforms

Neural Networks



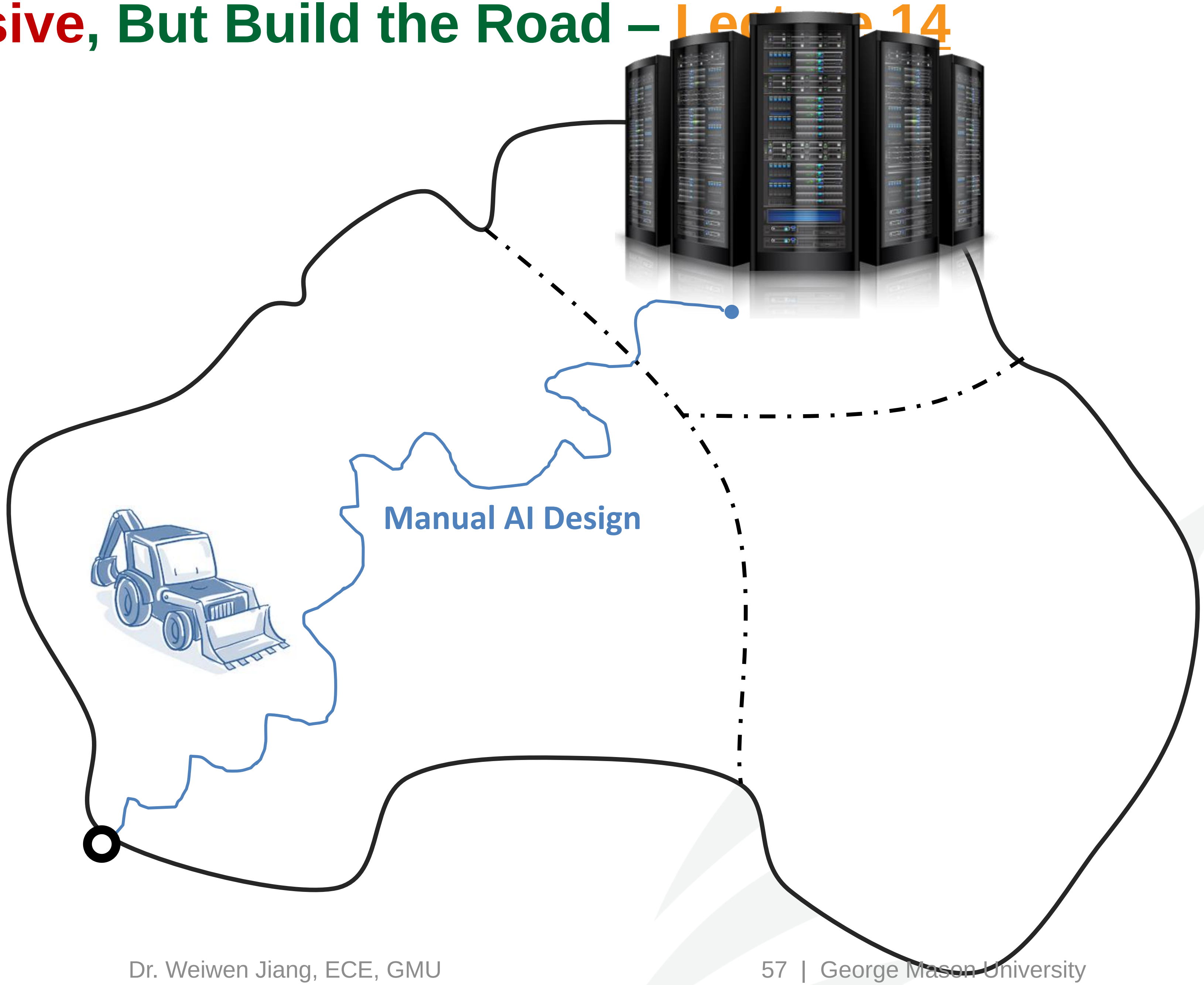
Manual Design: Expensive, But Build the Road – Lecture 14

1 year for only 1 application

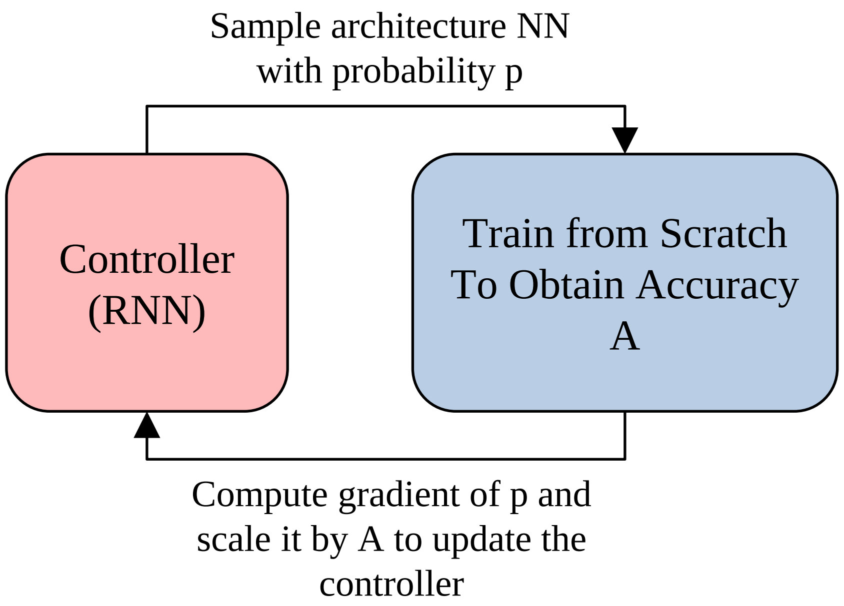
Name	Year	Acc.(T5)
AlexNet	2012	83.4%
ZFNet	2013	88.3%
VGGNet	2014	92.7%
RestNet	2015	96.4%
GoogleNet	2016	96.9%

Problem

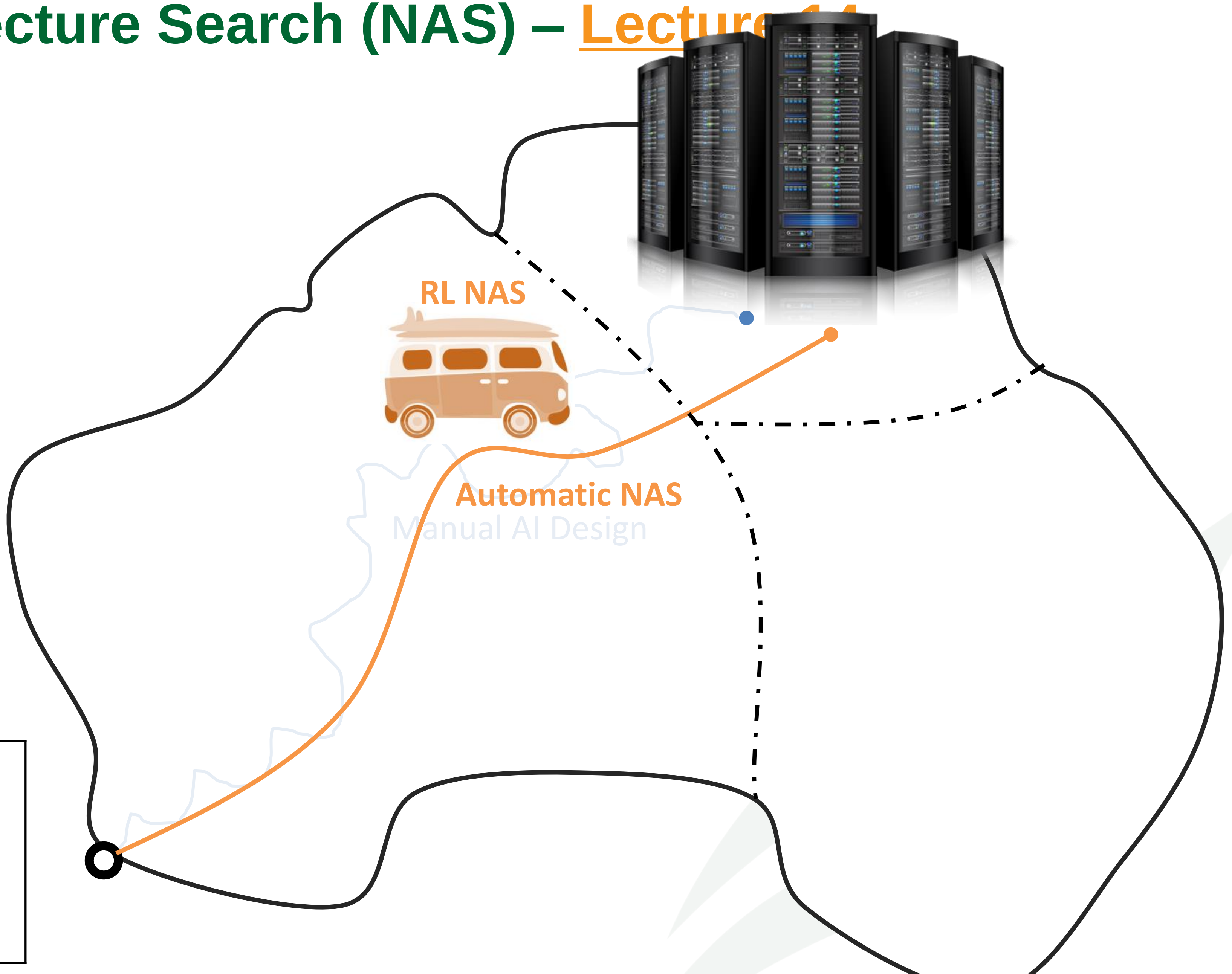
- **Domain knowledge and excessive labor**
- **It takes too long to devise new architectures**



AutoML: Neural Architecture Search (NAS) – Lecture



Reinforcement Learning Based NAS

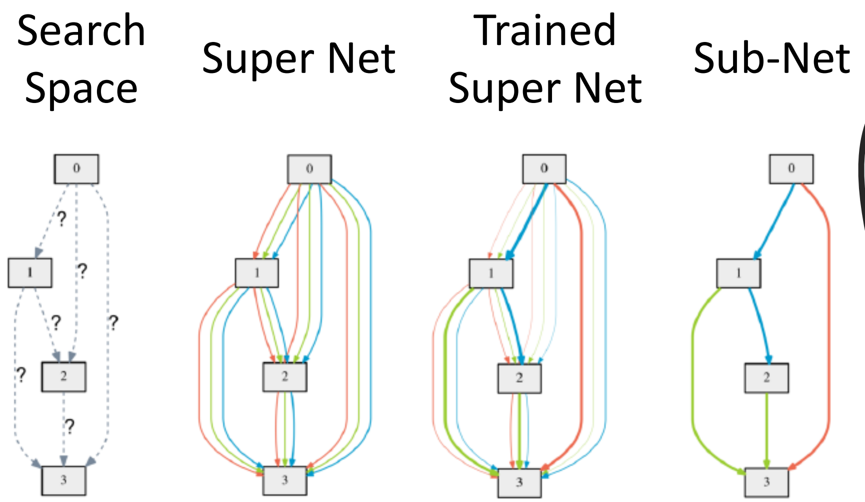


Problem

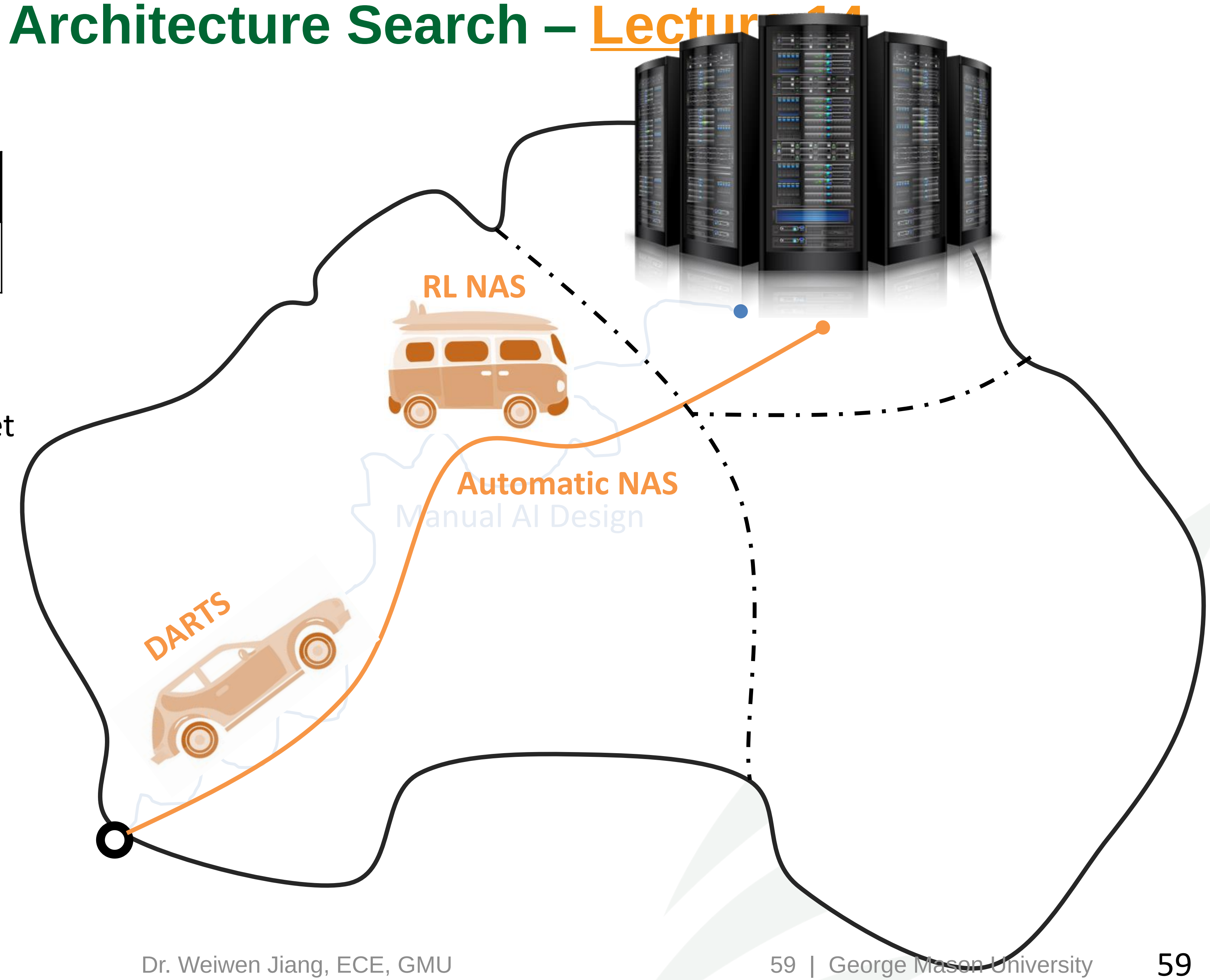
- **Low Efficiency, hundreds even thousands of GPU hours**
- **Mono-Objective: Accuracy, leading network too complicated**

AutoML: Differentiable Architecture Search – Lecture 14

Name	Time
DARTS	Jun. 2018

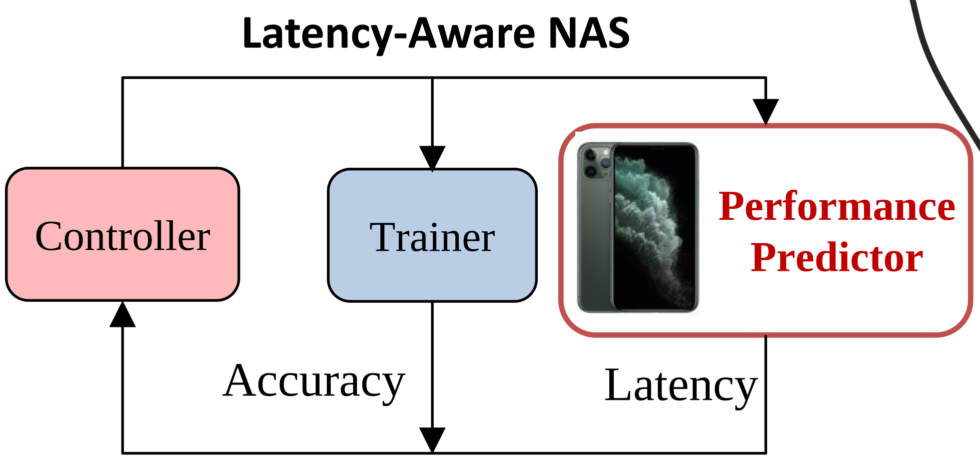


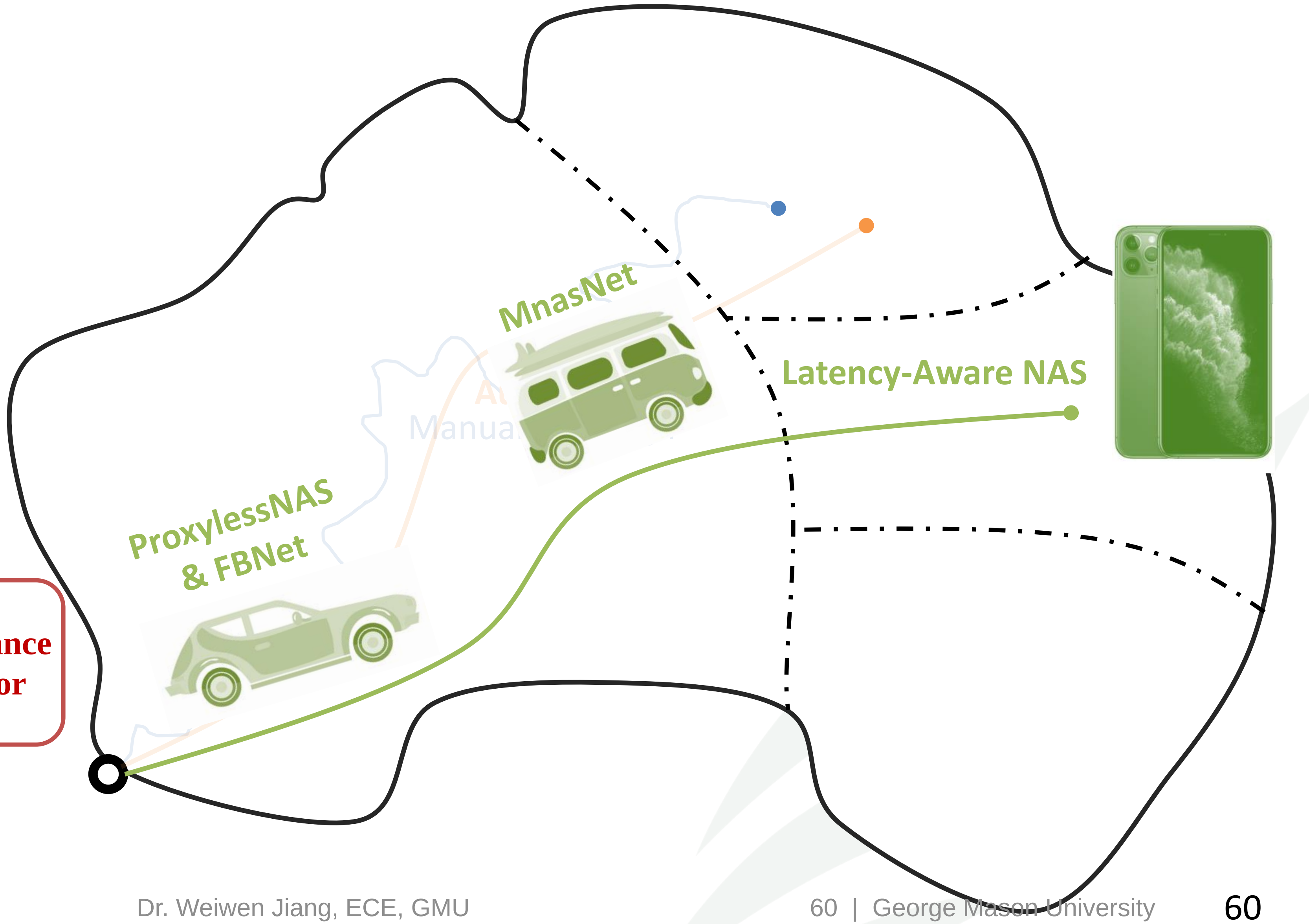
DARTS



AutoML: Hardware-Aware NAS – Lecture 14

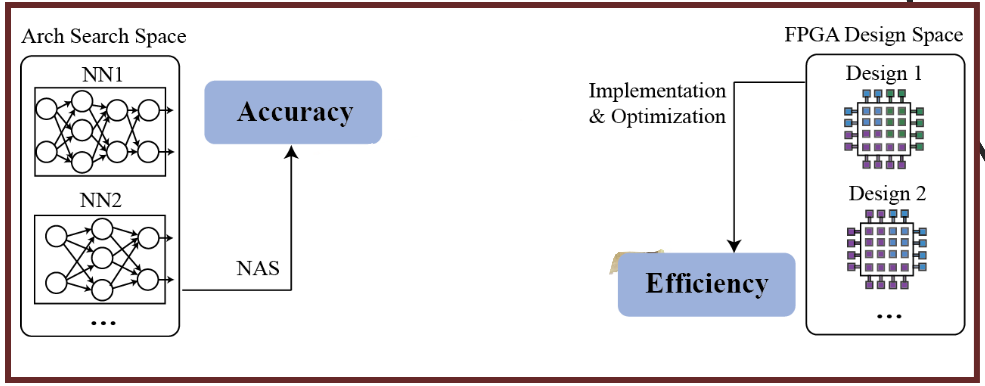
Name	Time
MnasNet	Jul. 2018
ProxylessNAS	Dec. 2018
FBNet	Dec. 2018



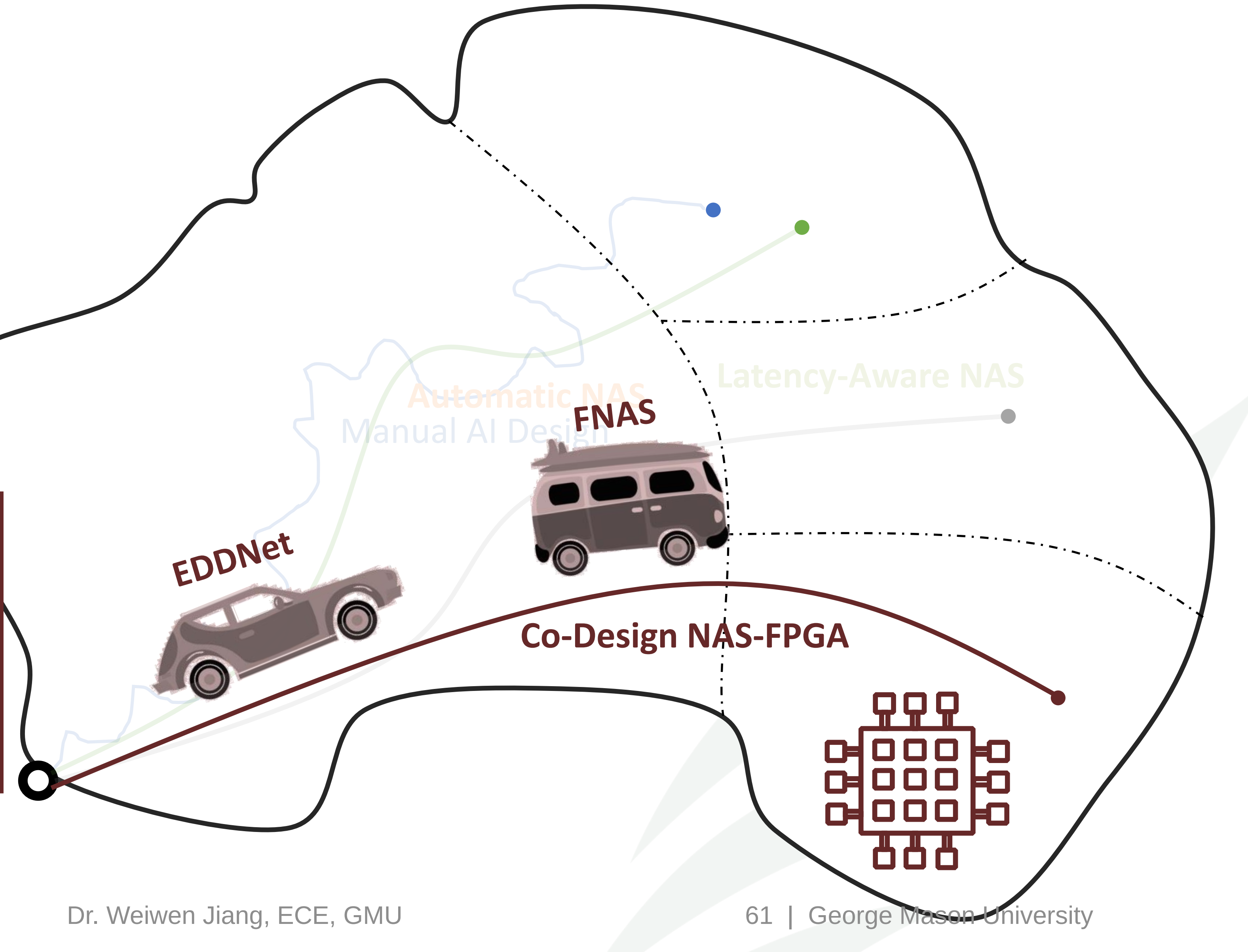


AutoML: Network-FPGA Co-Design Using NAS – Lecture 14

Name	Time
FNAS (ours)	Jan. 2019
DNN/FPGA	Apr. 2019
SkyNet	Sep. 2019
EDDNet	May. 2020



FNAS - DAC’19 (Best Paper Nomination)
TCAD’20 (Best Paper Award)



How to Conduct Neural Architecture Search – Lecture 14

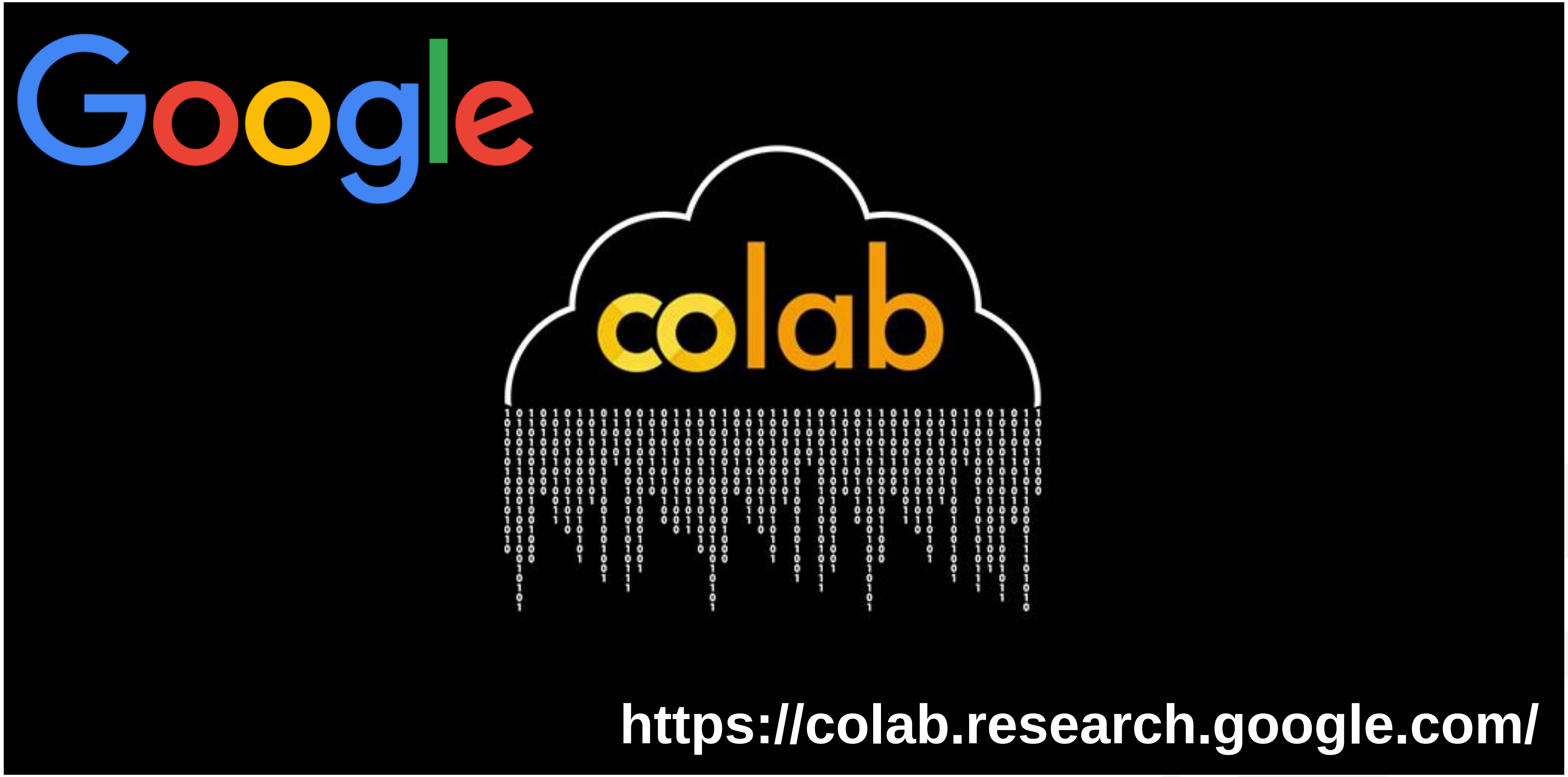
- **Selection of the Backbone Architecture**
  - **VGG** (NAS with RL, FNAS), **GoogLeNet** (NASNet), **MobileNet** (FBNet, ProxylessNAS), etc.

- **Determination of the Search Space**
  - **Software:** Number of Channels, Kernel Size, Convolution Type, etc.
  - **Hardware**: Loop Titling Parameters, Loop Order, Schedule, etc.

- **Optimization Approaches**
  - **Deep Reinforcement Learning:** RNN based controller
  - **Gradient Descent:** DARTS
  - **Metaheuristics:** Swarm

- **Optimization Objective(s):**
  - **Software:** Accuracy, Robustness, Fairness, etc.
  - **Hardware:** Latency, Chip Area, Energy Efficiency, etc.

Programming Platform



GMU.EDU

George Mason University

4400 University Drive
Fairfax, Virginia 22030

Tel: (703)993-1000