



<https://haoailab.com/cse291-s26/>

# CSE/DSC 291: Deep Learning Systems Spring 2026

---

LLM, diffusion, and case studies

Optimizations and Parallelization

Basics

# Forms worth your attention

- Beginning of quarter survey
  - Please fill the survey by end of week 2
    - If  $\geq 80\%$  of you filled the survey, all of you get 1%
    - If  $< 80\%$ , all of you do not get 1%
- Scribe sign-up:
  - Do it as a team.
  - Do it as detailed as possible.
  - Most efficient way to get 5%!

# Enrollment Update

Current enrollment:

- DSC: 65
- CSE: 86
- We are going to enroll 20 more students by this week (based on FCFS)
- We are working with CSE to discuss a solution to increase the room to 200 with more teaching/facility support, too.

# Enrollment Update

If for whatever reason, you cannot enroll but want to learn

- Ask TAs to add you as observer in Canvas and piazza
- All other materials are publicly accessible including PA, excluding exams
- The only difference is your assignments/summary won't be graded.

# Other FAQs

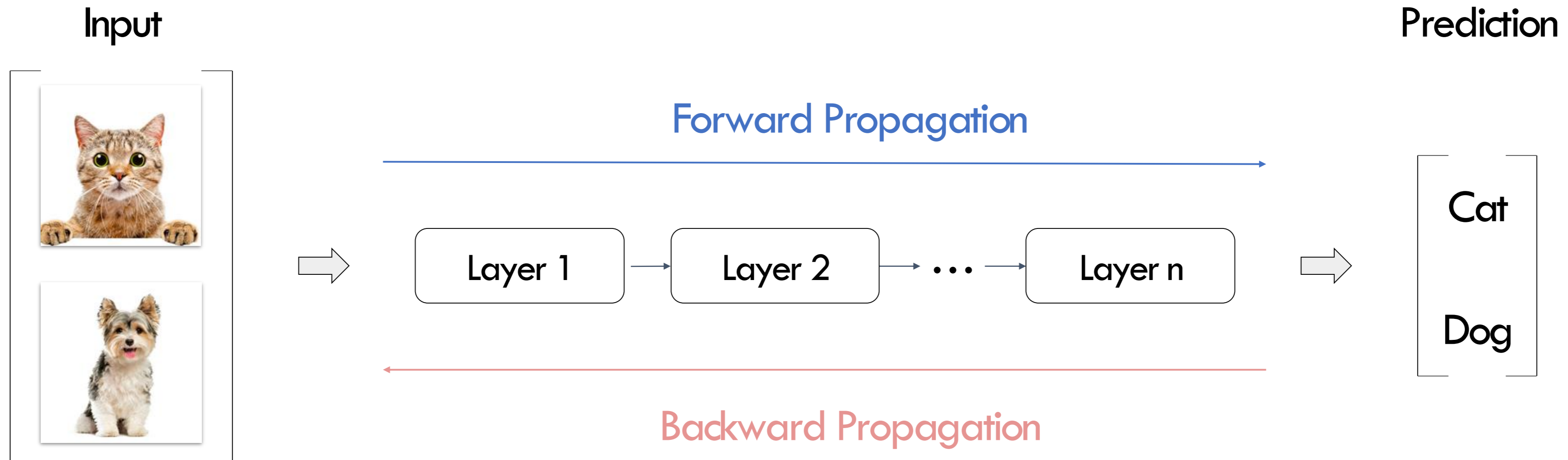
- **Difference from this year CSE 234:**
  - More applications, more data management, more agents, more RAG
  - This course: runtimes, kernels, distributed systems, model architectures, more rationale behind the system and model
- **Comp exam:**
  - $\geq$  B- on your grade (all components accumulated)
- **Recording and slides for lecture 1:**
  - No 😊
- **Can we make this source a core elective at cse**
  - Unlikely for this offering. But will be in the next offering.

# Today

- Understand workloads
- Dataflow graph representation
  - Flavors of different ML frameworks

# Background: Deep Learning Computation

- Idea: Composable Layers



$$\theta^{(t+1)} = f(\theta^{(t)}, \nabla_L(\theta^{(t)}, D^{(t)}))$$

# Dive in to Models: Three parts

$$\theta^{(t+1)} = f(\theta^{(t)}, \nabla_L(\theta^{(t)}, D^{(t)}))$$

parameter                      weight update (sgd, adam, muon, etc.)                      model (GPT, diffusion, etc.)                      data

- Model: A parameterized function that describes how do we map inputs to predictions
  - Parameters: to be optimized
  - Loss function: How “well” are we doing for a given set of parameters
    - L2 loss, hinge loss, softmax loss, ranking loss
- Optimization method: A procedure to find a set of parameters that minimizes the loss
  - Adam, Muon, 2<sup>nd</sup> order methods(e.g., Newton methods),

# Three important components

## Data

- Images
- Text
- Audio
- Video
- etc.

## Model

- CNNs
- LLMs
- MoEs
- Diffusions
- Ertc.

## Compute

- CPUs
- GPUs/TPUs/LPUs
- M1 /M2/M3/M4
- FPGA/etc.

# Today

- **Understand our Workloads: Deep Learning**
- Dataflow graph representation

# How we prioritize in a fast-evolving world?

- There are many great models developed in history
- *We will not be able to* build systems that can support all models
- What are the most important workloads that solve 80% of the problems?

# What are the most important models and optimization algos

Most important model architectures today?

- Convolutional neural networks
- Transformers
- Mixture-of-Experts

Most important optimization algorithms?

- SGD and its variants, e.g., Adam, Muon

# Ambiguity: Model building block vs. Models

- Is LLM a model or a building block ?
  - A family of Models
- Is transformer a model or a building block ?
  - Building block
- Is diffusion a model or a building block?
  - Neither, it is a learning process
  - Name a diffusion model?
  - Name the building block of a diffusion model?

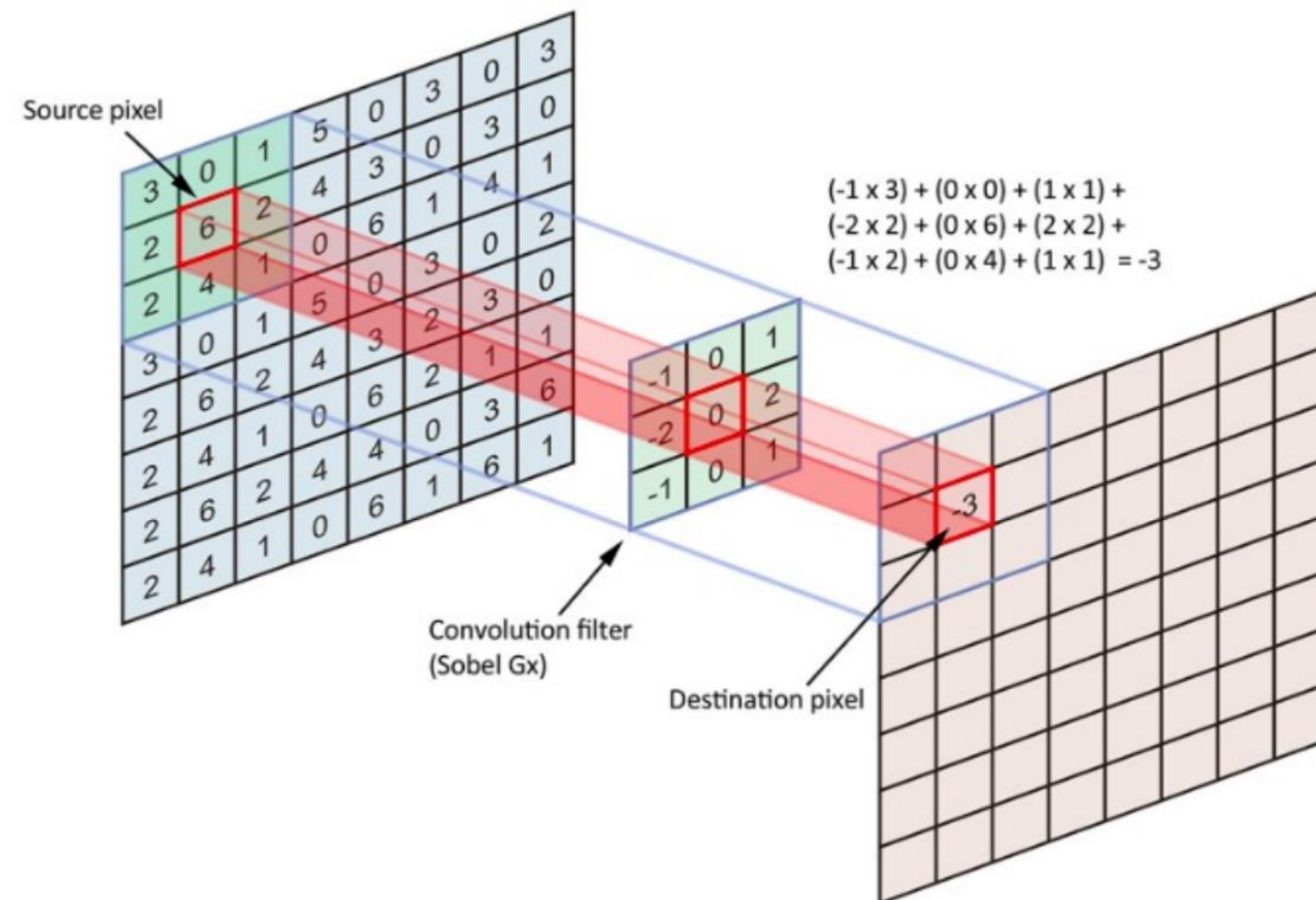
# Understand Our Workload (a.k.a. DL course in 20 mins)

- In this class, we review the most important 3 building blocks
  - Convolutional Neural Networks
  - Transformers
  - Mixture-of-Experts
- We will keep asking ourselves: what are the most important  $X$  in  $Y$  ( $X \subset Y$ ) to spec out system building abstraction
  - E.g.,  $X = \text{ResNet}$ ,  $Y = \text{CNNs}$
- If you have trouble following this session, spend time reading deep learning book or learn <https://sites.google.com/view/cse251b>

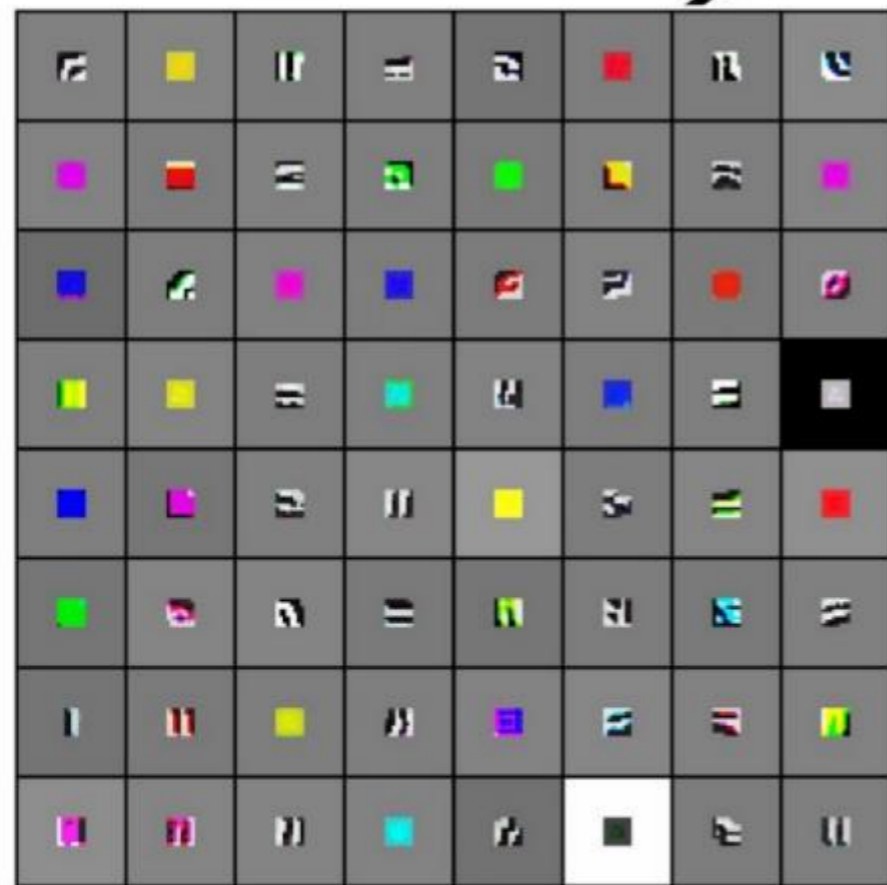


# CNN: Key components

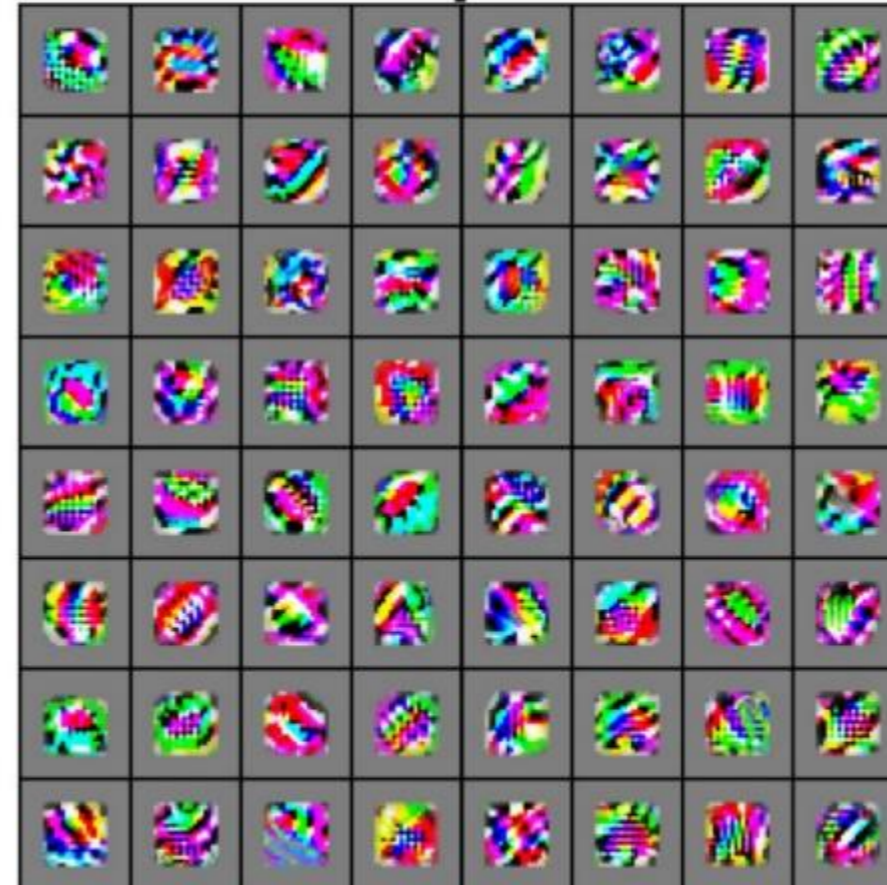
- Convolve the filter with the image: slide over the image spatially and compute dot products



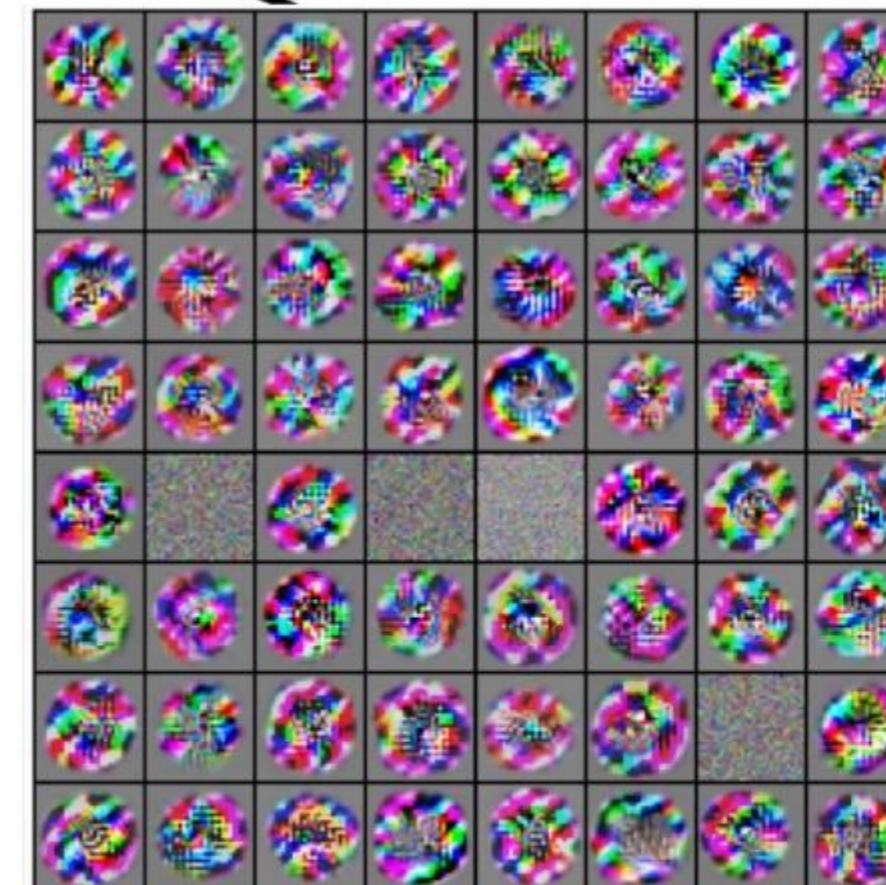
# Principle: Stacking Conv layers



VGG-16 Conv1\_1



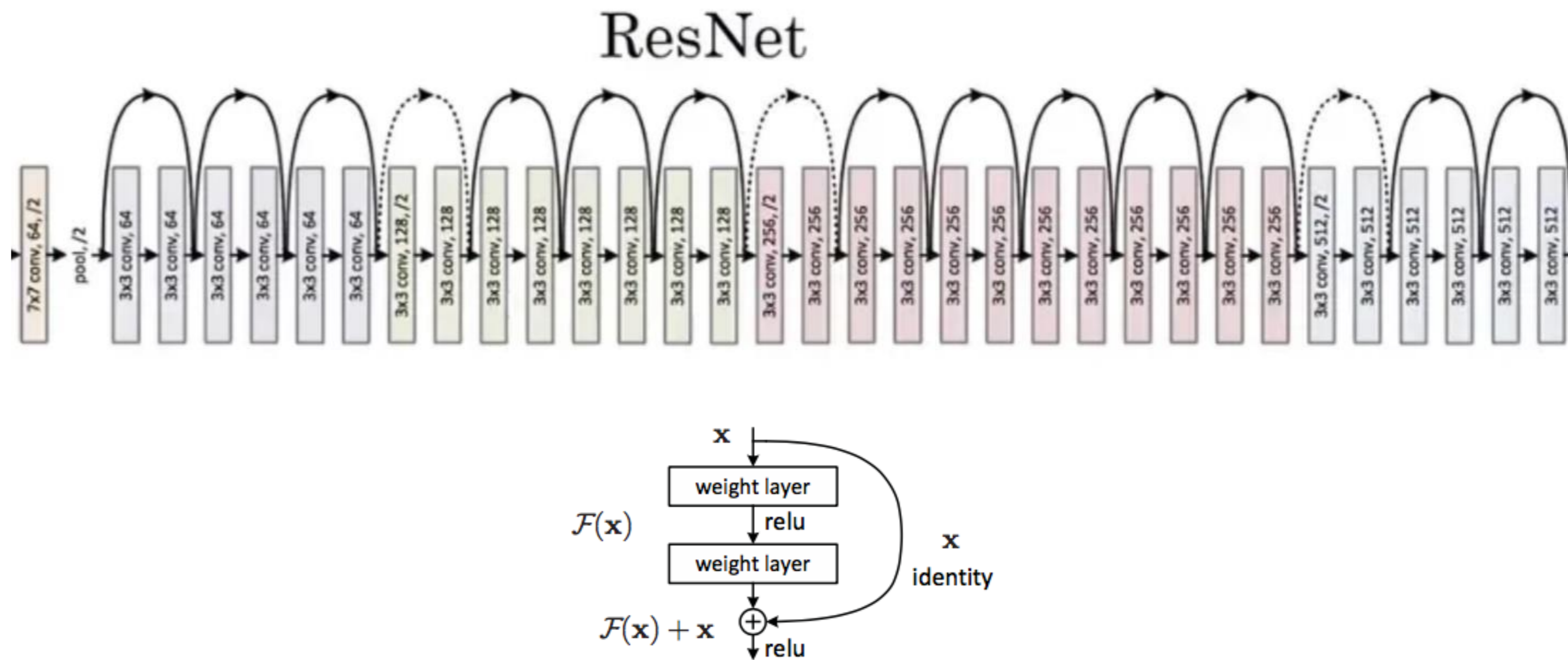
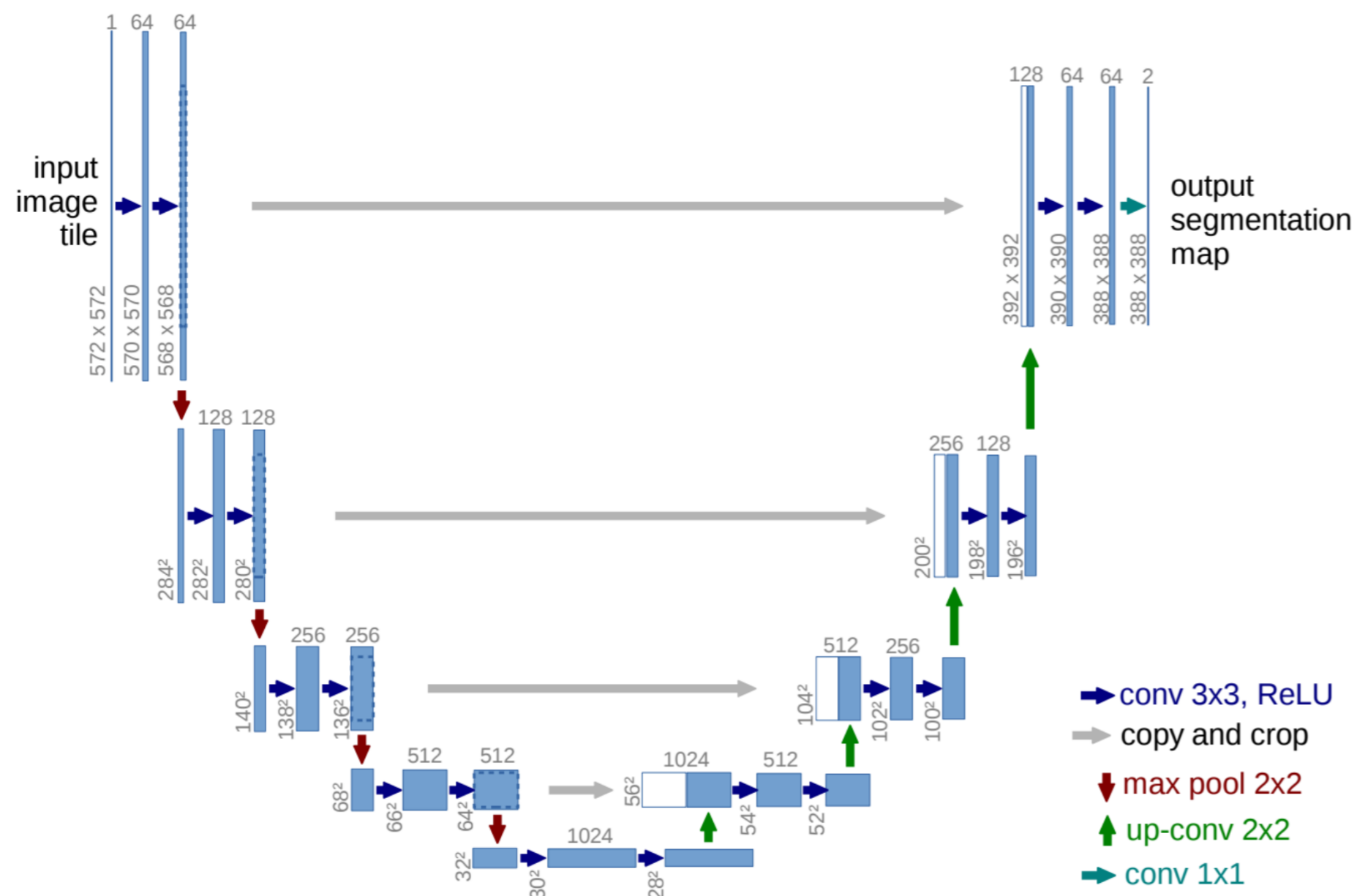
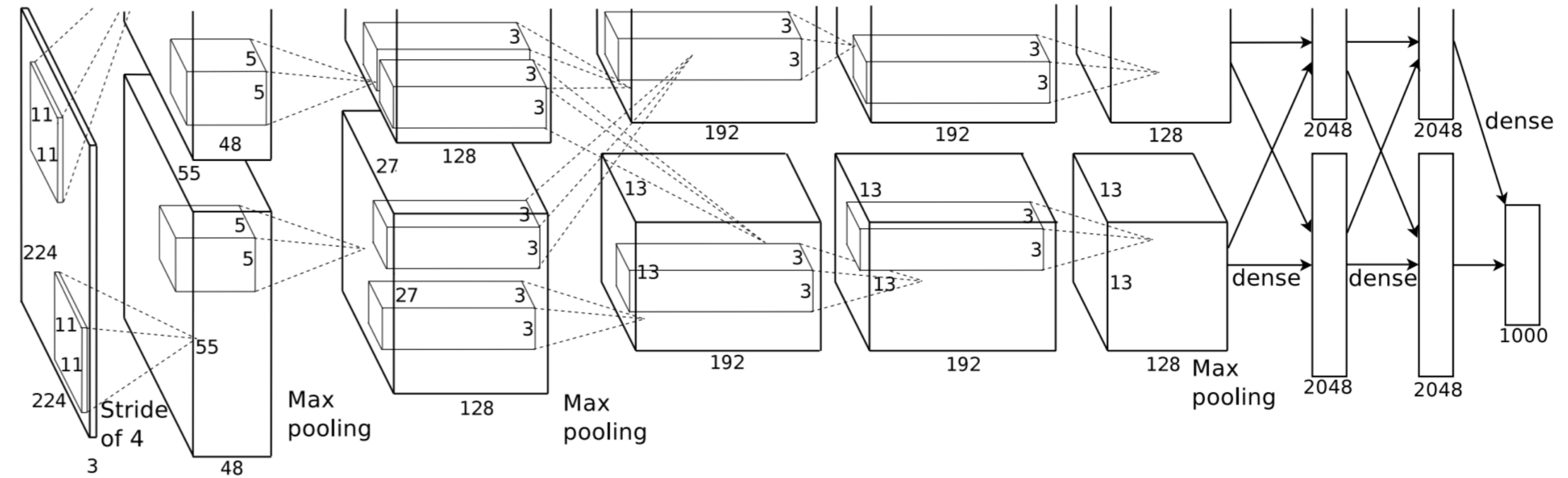
VGG-16 Conv3\_2



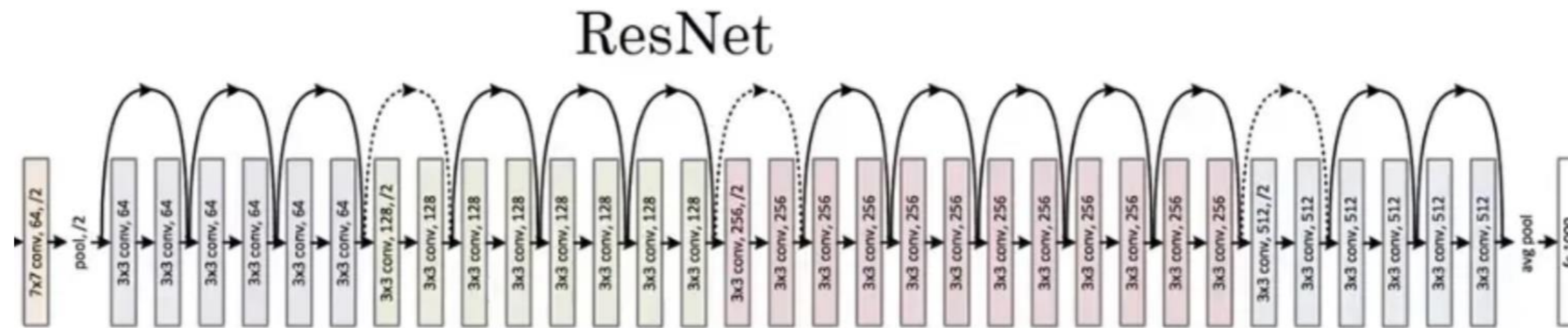
VGG-16 Conv5\_3

# CNN: Top3 models

- AlexNet [Alex/Iliya/Hinton]
- ResNet [Kaiming etc.]
- U-Net [Olaf etc.]



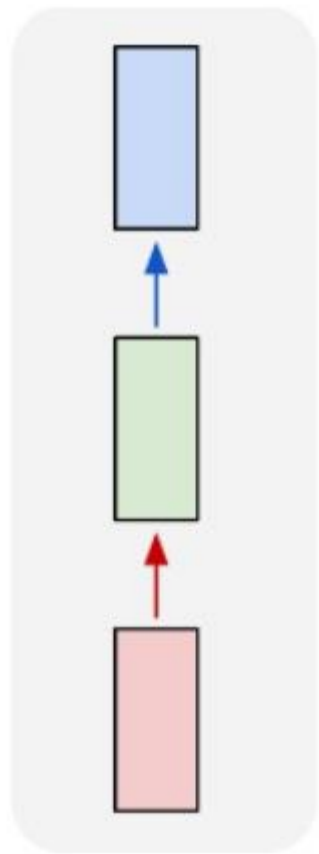
# Most important components in CNNs?



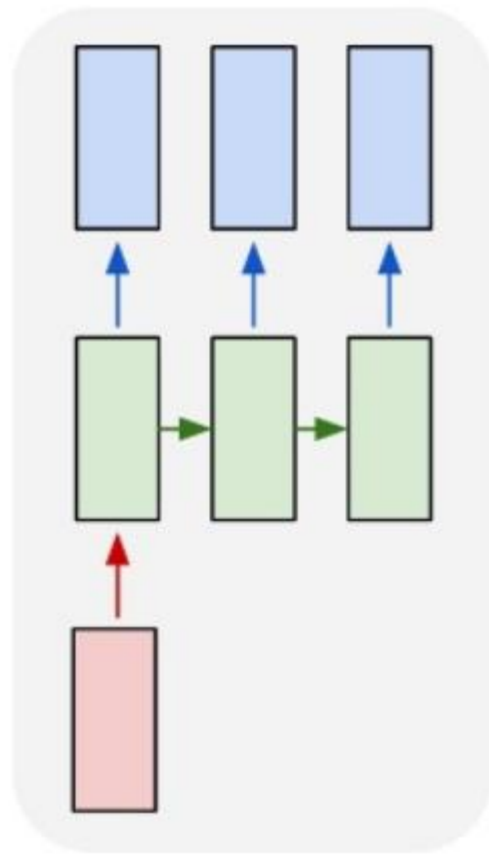
- Conv
  - Conv1d, Conv2d, conv3d, esp. 3x3-conv2d
- Matmul (linear) :
  - $C = A * B$
- Softmax
- Elementwise operations:
  - ReLU, add, sub
  - Pooling, normalization, etc.

# Recurrent Neural Networks

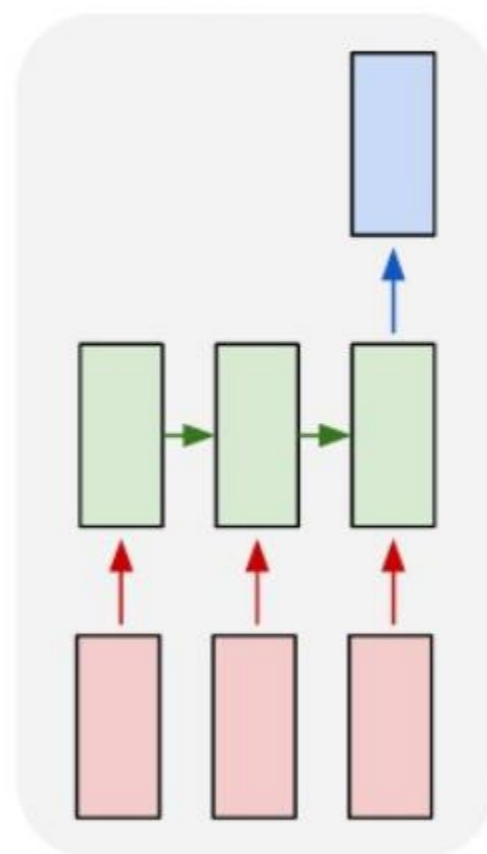
one to one



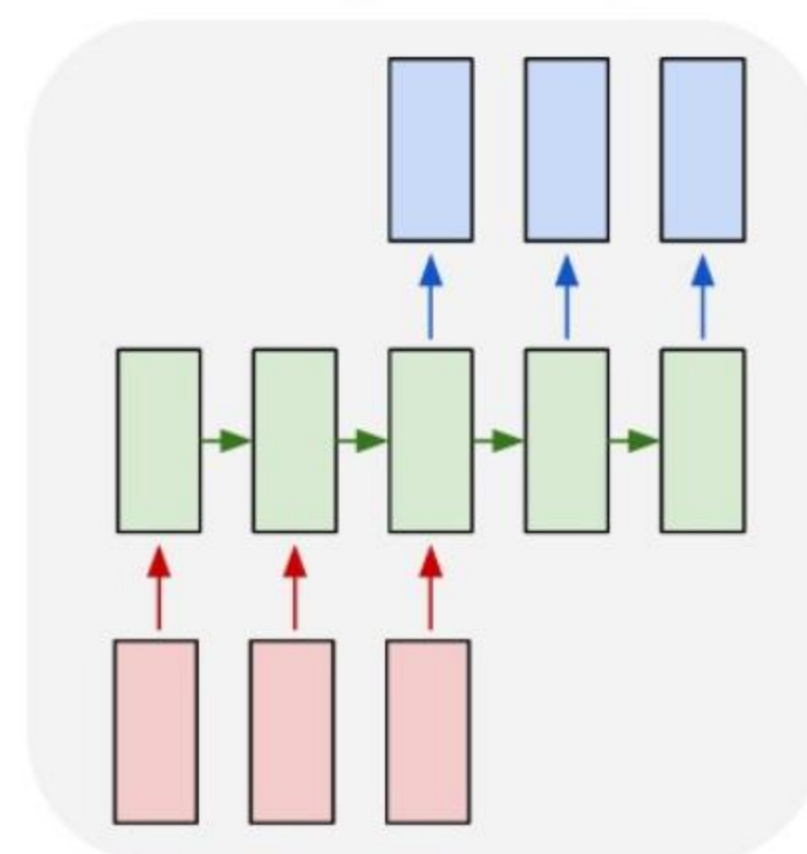
one to many



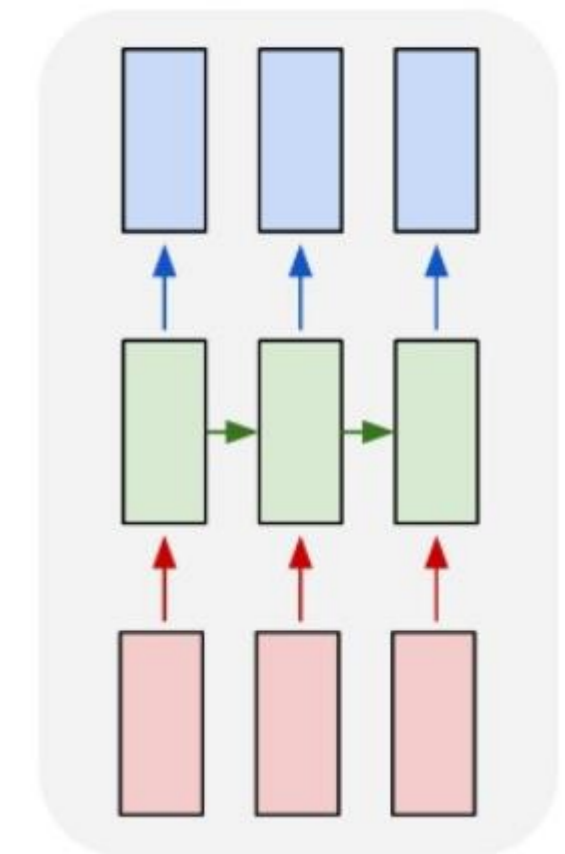
many to one



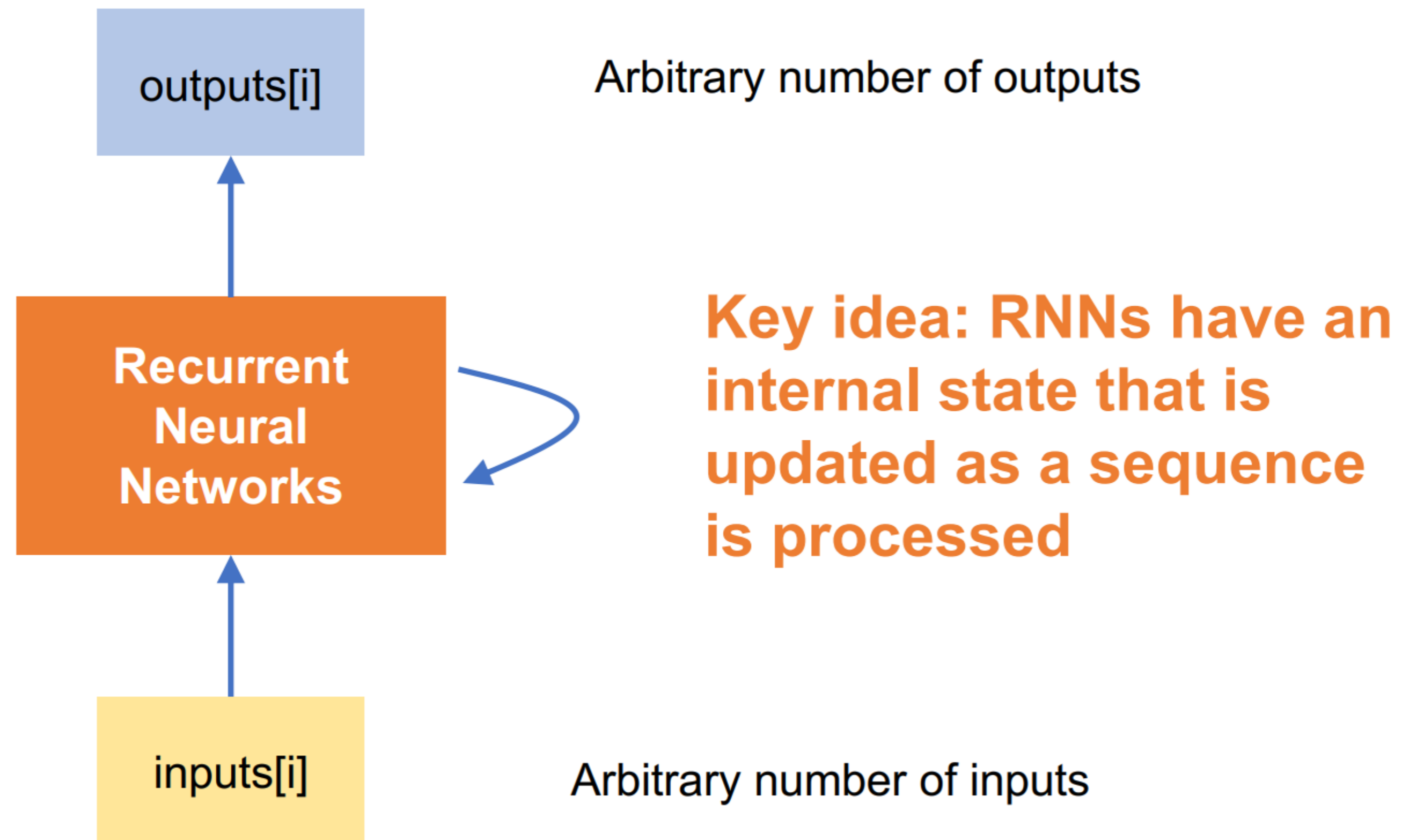
many to many



many to many

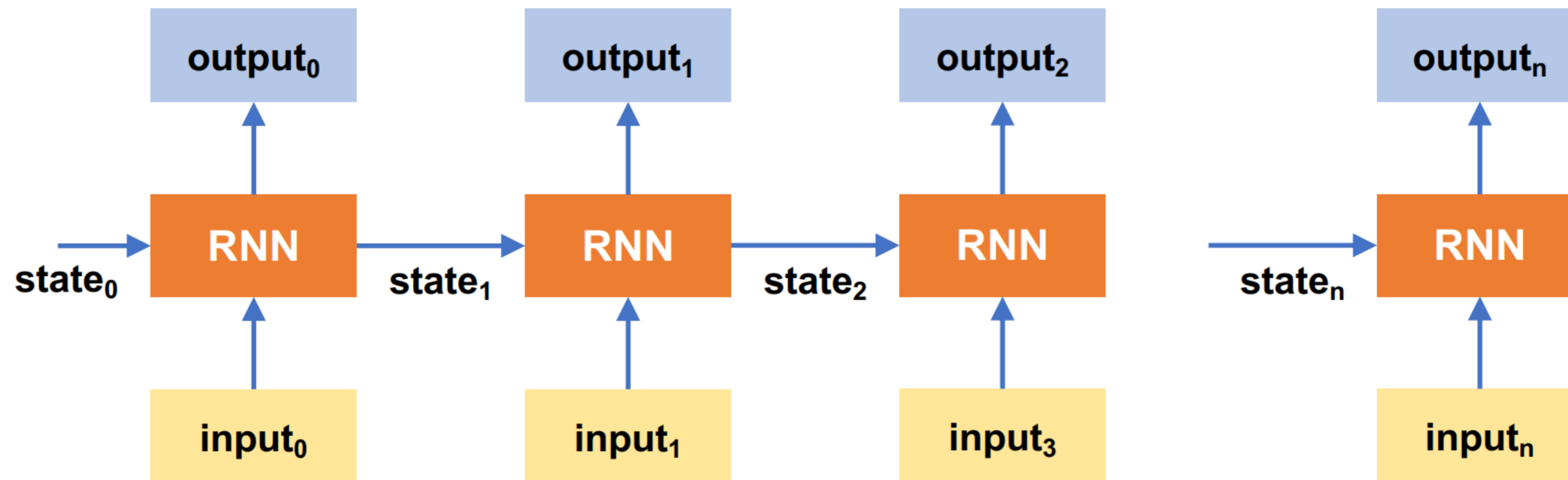


# Recurrent Neural Networks



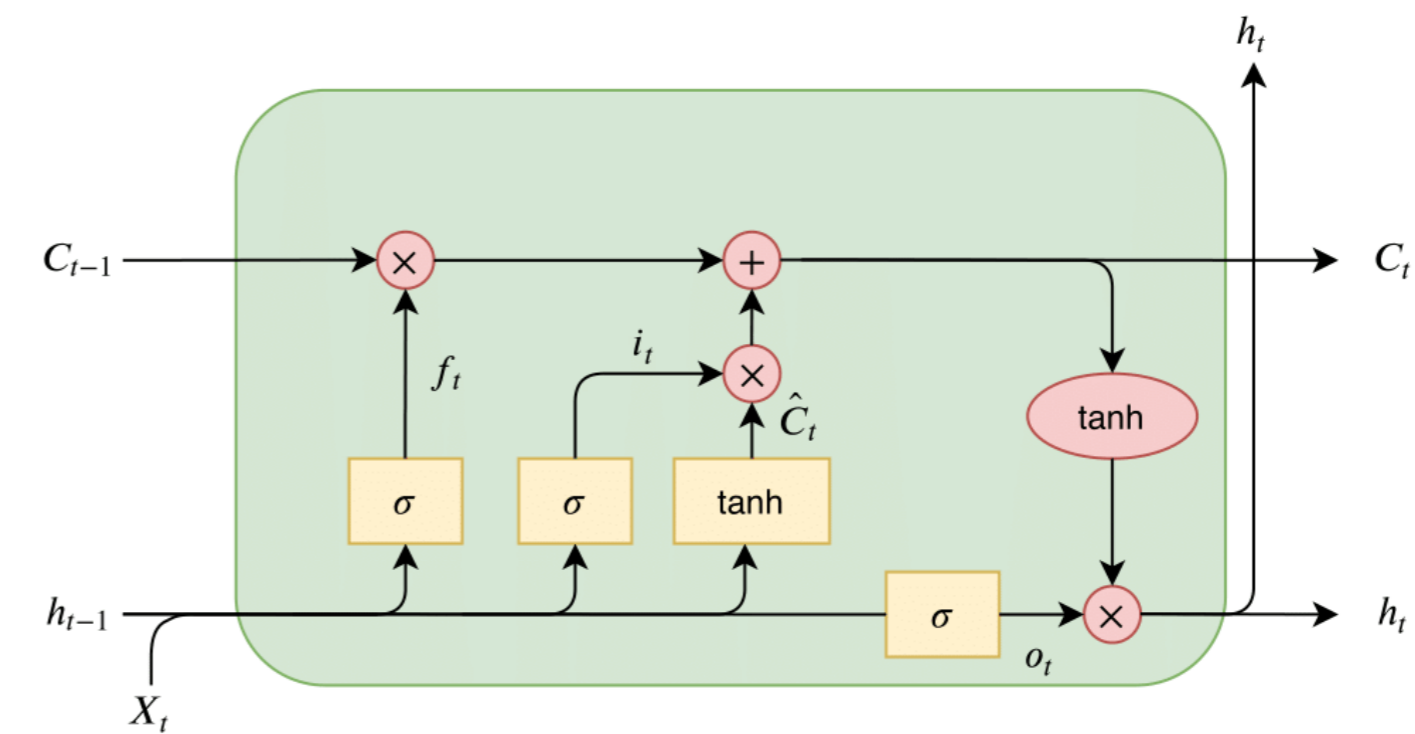
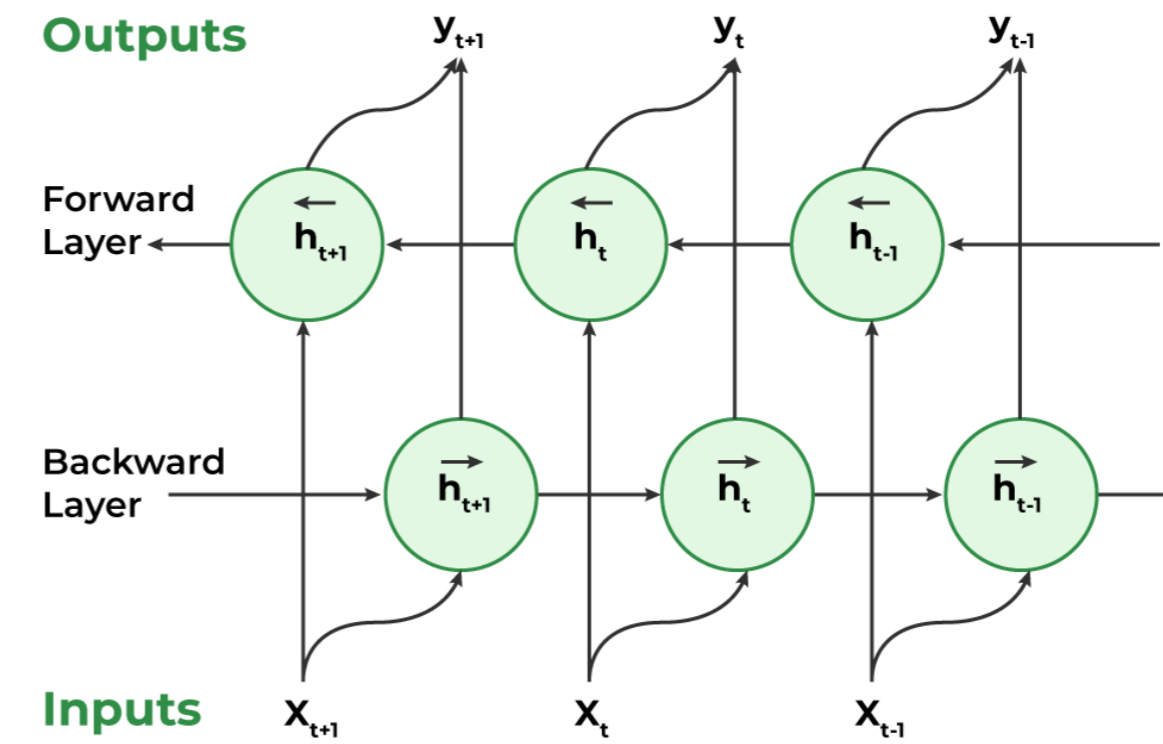
# Recurrent Neural Networks: unrolling the computation

- One can make any NN recurrent

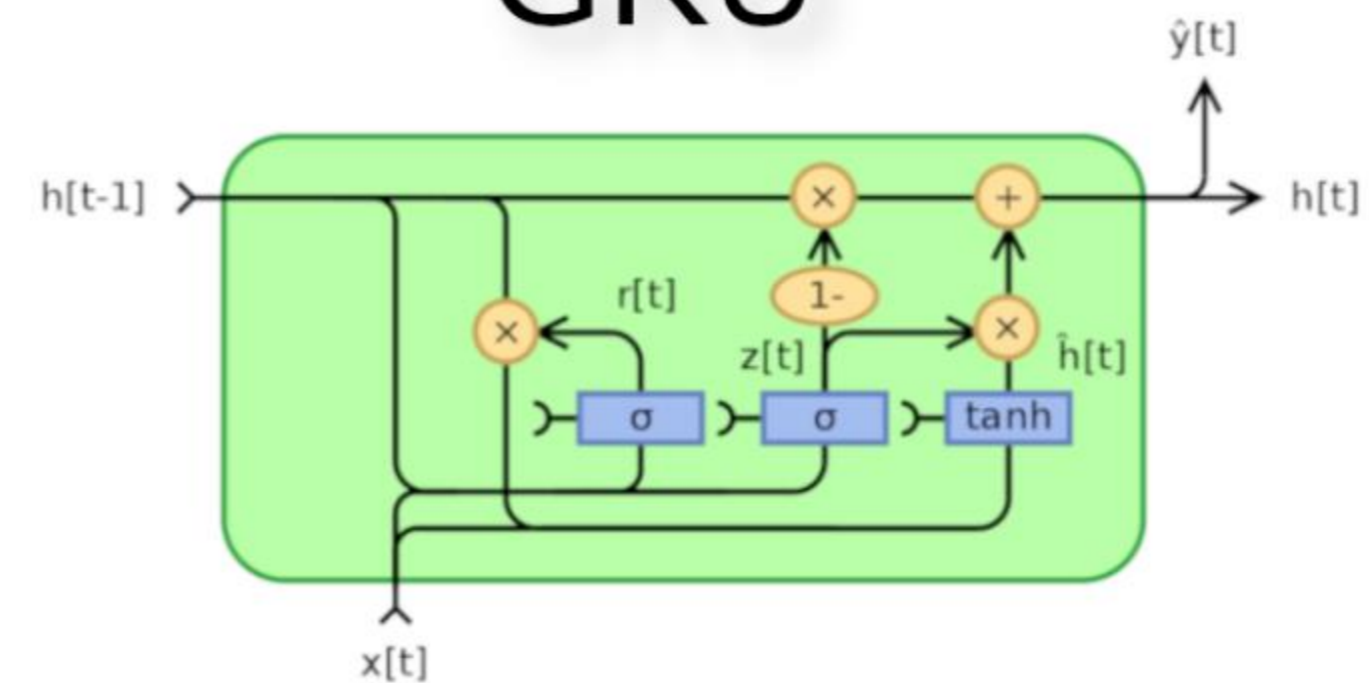


# RNN: top3 models

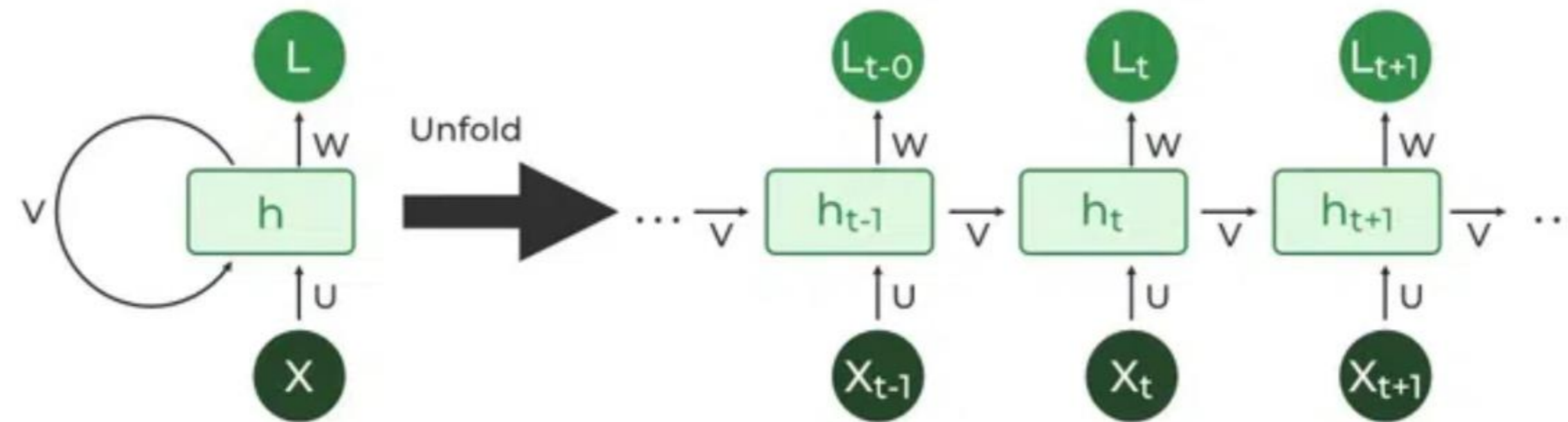
- Bidirectional RNNs
- LSTM
- GRU



## GRU



# Most Important Components in RNNs



- Matmul
- Elementwise nonlinear
  - ReLU, Tanh, sigmoid, etc.

# MCQ Example (A difficult one)

Who Invented LSTMs?



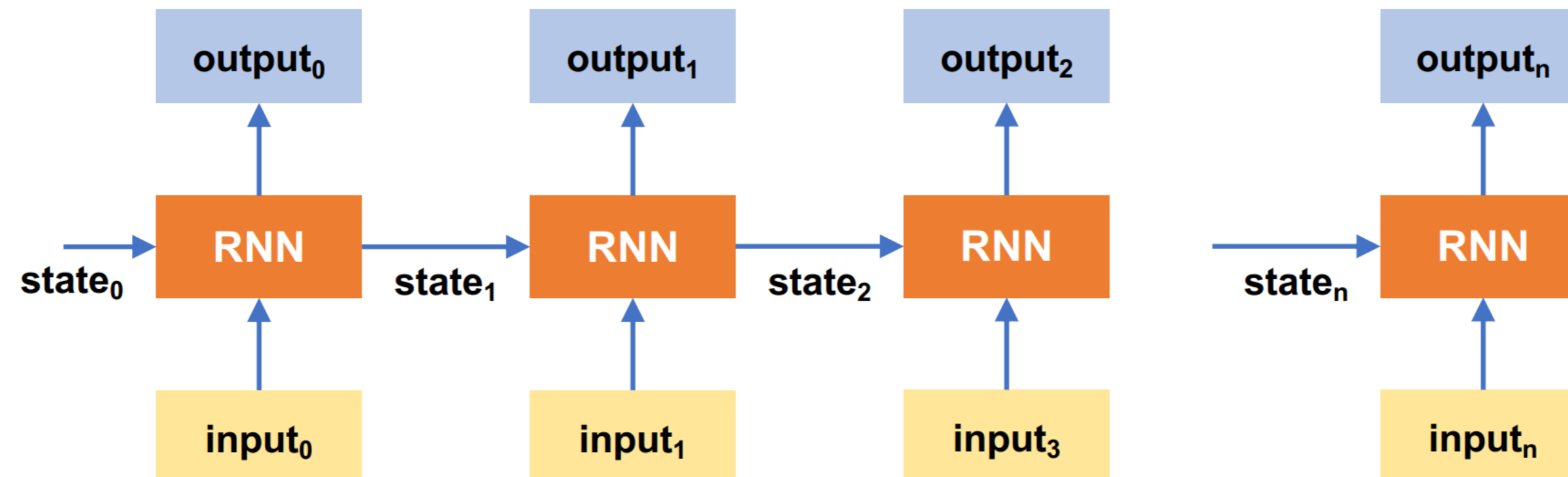
Jürgen Schmidhuber

**Jürgen Schmidhuber** (born 17 January 1963) is a German computer scientist noted for his work in the field of artificial intelligence, specifically artificial neural networks.



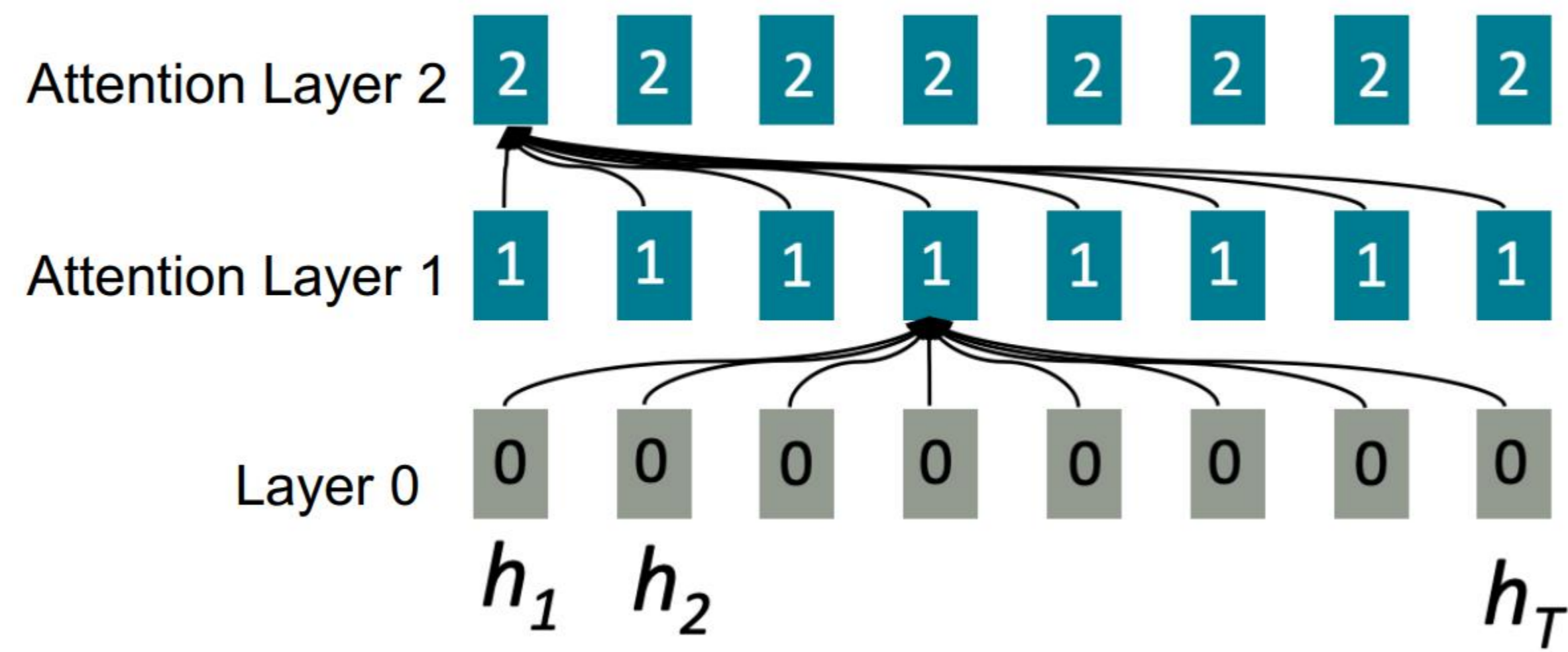
# Why ChatGPT was not built using RNNs?

- Problem 1: forgetting.
  - $h * 0.9 * 0.9 * \dots \rightarrow 0$
- Problem 2: **lack of parallelizability**.
  - Both forward and backward passes have  $O(\text{sequence length})$  unparallelizable operators. i.e., a state cannot be computed before all previous states have been computed. Inhibits training on very long sequence



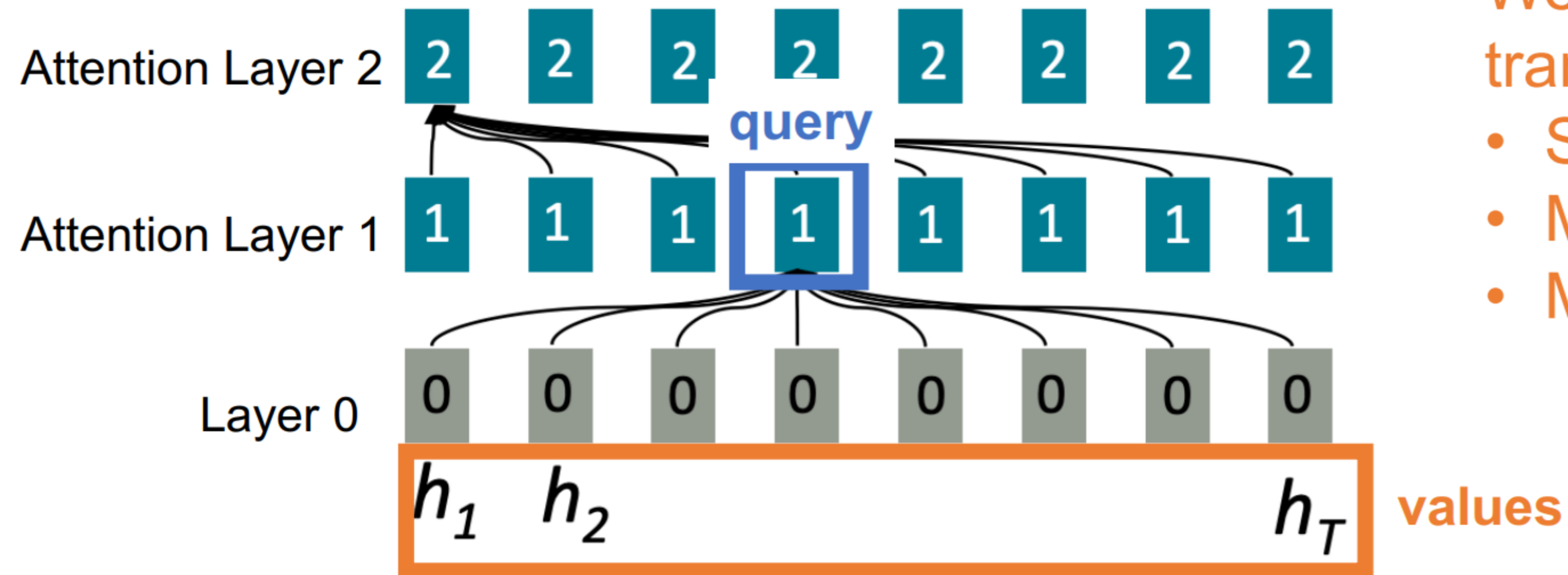
# Attention: Enable parallelism

- Idea: treat each position's representation as a query to access and incorporate information from a set of values



# Attention

- Massively parallelizable: number of unparallelizable operations does not increase sequence length



We will learn attention and transformers in depth later:

- Self-attention
- Masked attention
- Multi-head attention

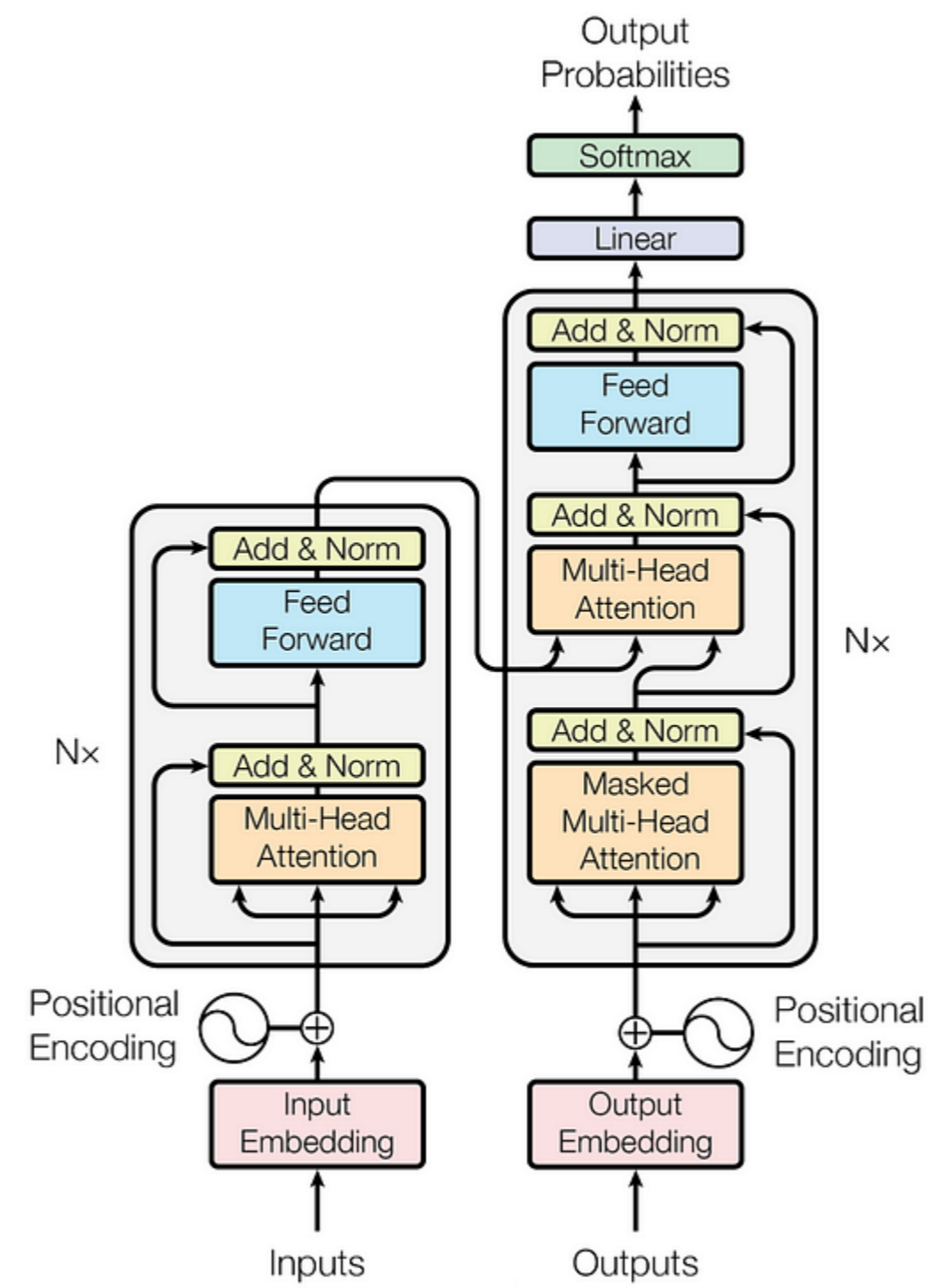
# Transformers: Attention + MLP

**BERT**

Encoder

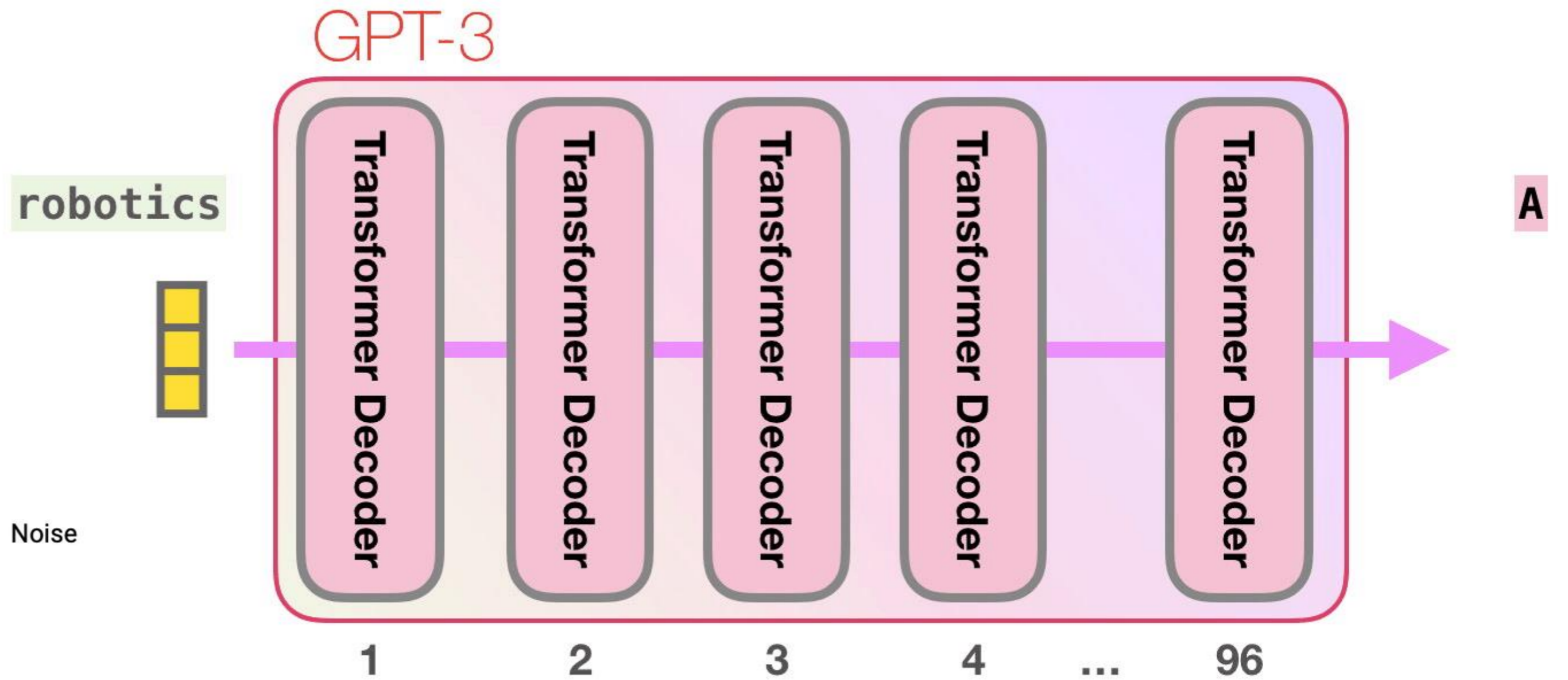
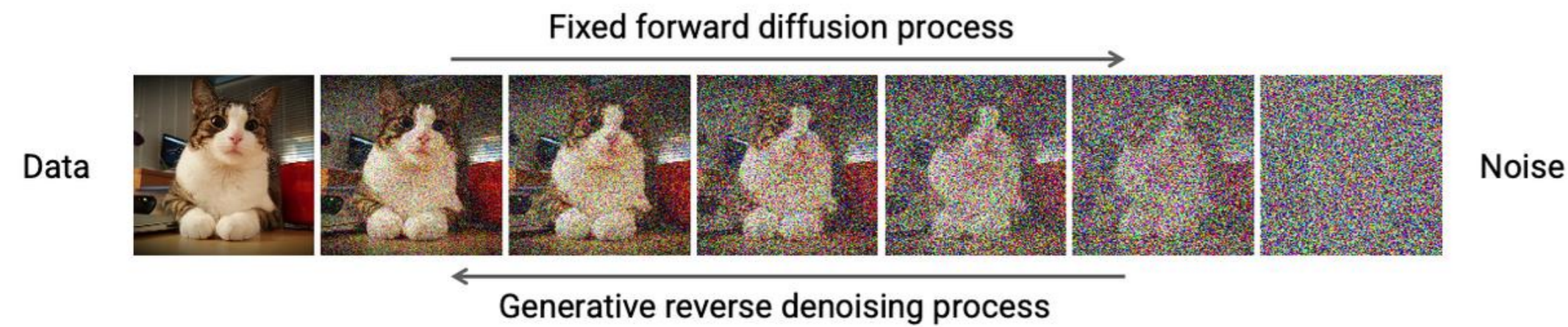
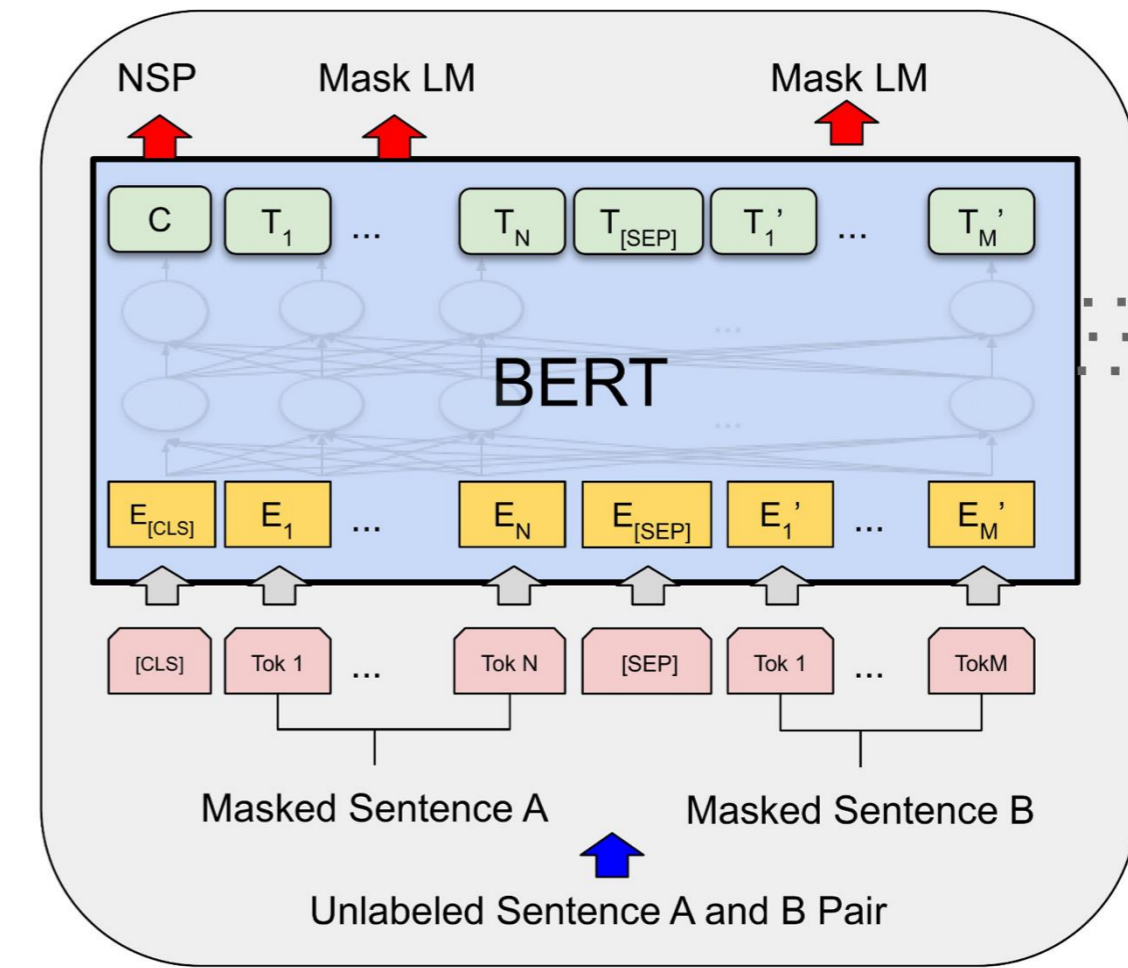
**GPT**

Decoder



# Transformers: Top3 models?

- Bert
- GPT/LLMs
- DiT: diffusion



Transformer becomes the chosen one... (for a reason)



## Conclusions

- If you have a large big dataset
- And you train a very big neural network
- Then success is guaranteed!

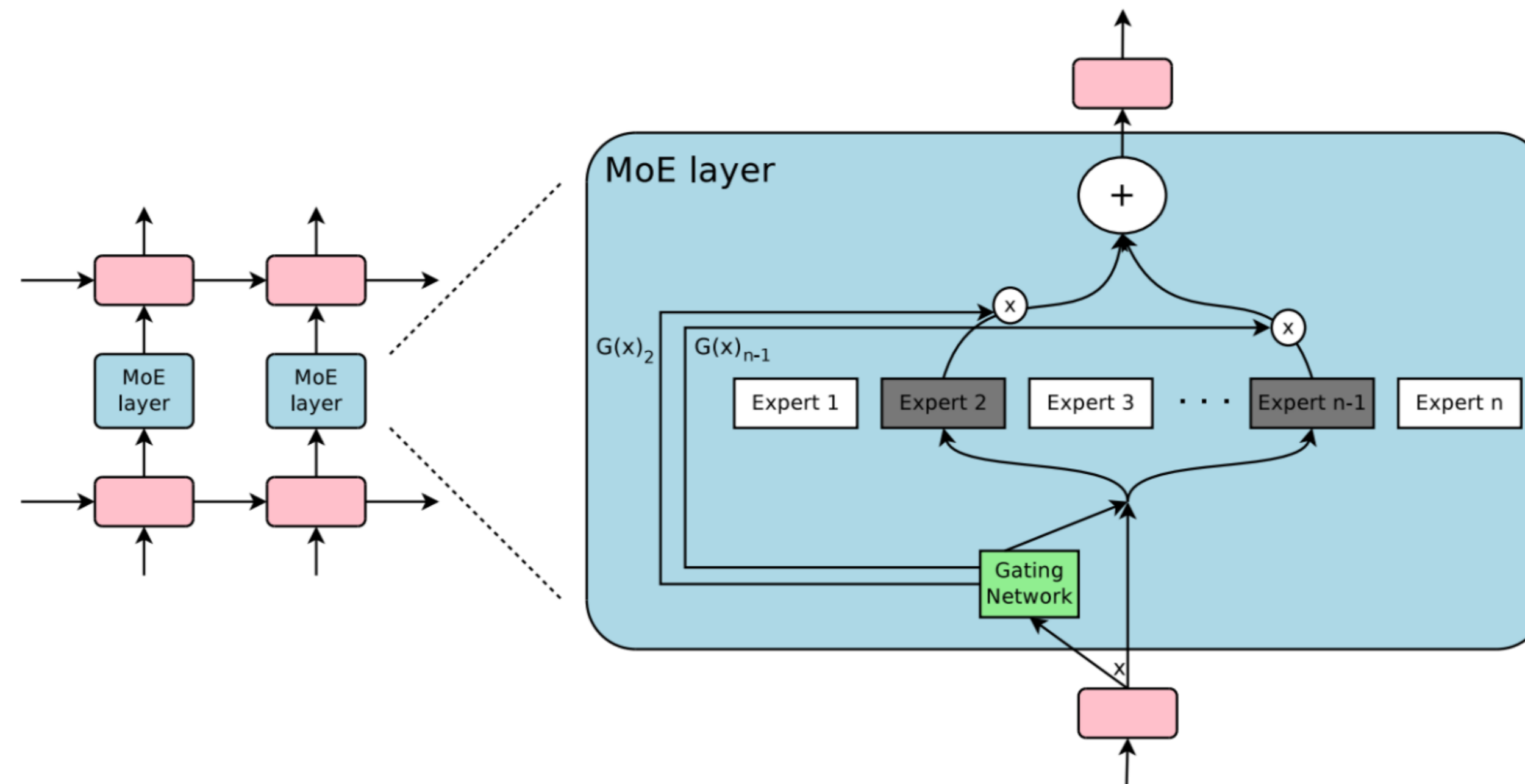
<https://www.youtube.com/watch?v=9I9xGoyDhMM>

# Most Important Components in Transformers?

- Transformers = Attention + MLP + something else
- Attention:
  - Matmul
  - Softmax
  - Normalization
- MLP :
  - Matmul
- Something else:
  - Layernorm, GeLU, etc.

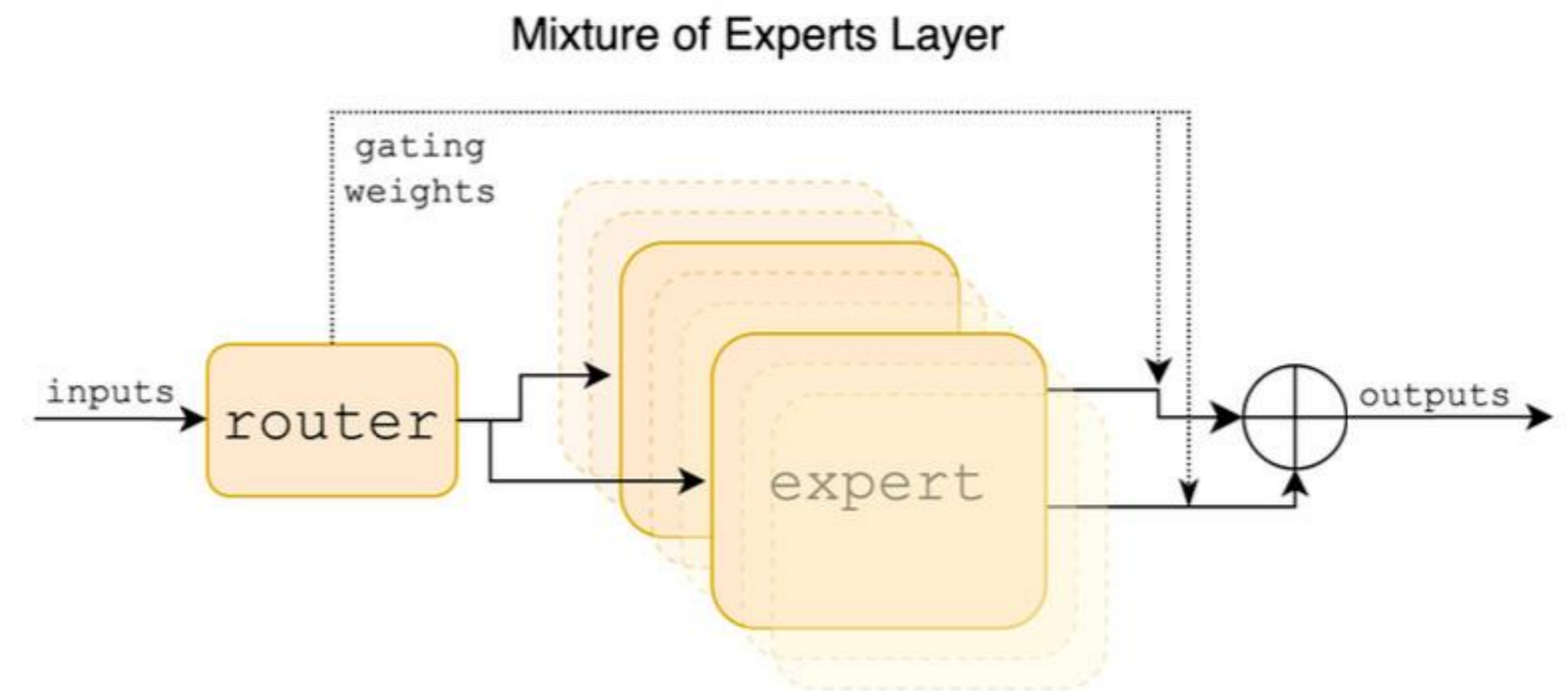
# MoE: mixture of experts

- Ideas: Voting from many experts could be better than one expert



# Novel Component in MoE?

- All Latest LLMs are mostly MoEs
  - Grok, Mixtral, GPT4, Deepseek-v3
- Novel Components in MoE:
  - Router
- What constitutes Router?
  - Matmul, Softmax
- After-class Q:
  - Why router makes it hard



# Rundown from a compute perspective

- CNNs: Conv, Matmul, Softmax, ReLU, batchnorm.
- RNNs: Matmul, sigmoid, tanh
- Transformers: Matmul, Softmax, GeLU, layernorm
- MoE: Matmul, softmax

Or:

Matmul: 4 times

Softmax: 3 times

Others...

# Summary of DL class in 30 mins

Matmul (plus softmax) are all you need

MLSys  $\approx$  Matmul Sys

# High-level Picture

Data

?  $\{x_i\}_{i=1}^n$

Model



Math primitives  
(mostly matmul)

? A repr that expresses the  
computation using primitives

Compute

? Make them run on (clusters  
of ) different kinds of  
hardware

# Today

- Understand our Workloads: Deep Learning
- **Dataflow graph representation**
- Flavors of different ML frameworks

# Recall our Goal

- Goal: we want to express as many as model as possible using one set of programming interface by connecting math primitives
- What constitutes a model from math primitives?
  - Model and architecture: connecting math primitives
  - Objective function
  - Optimizer
  - Data

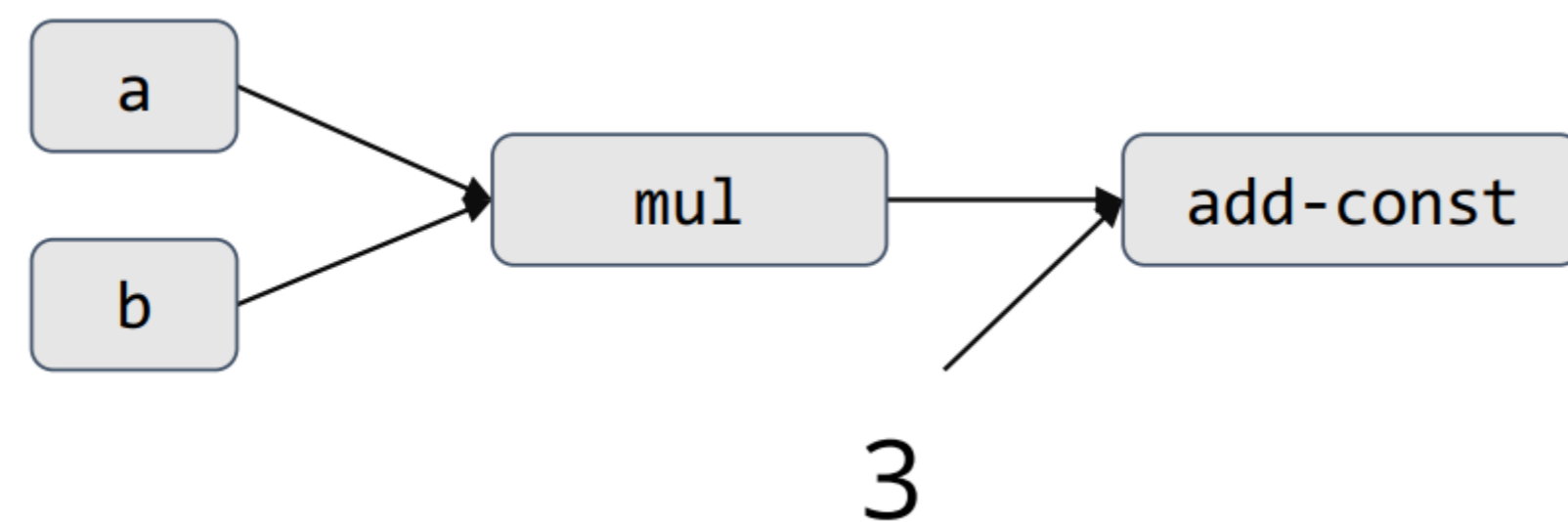
# Discussion: how we express computation in history

## Applications <-> System Design

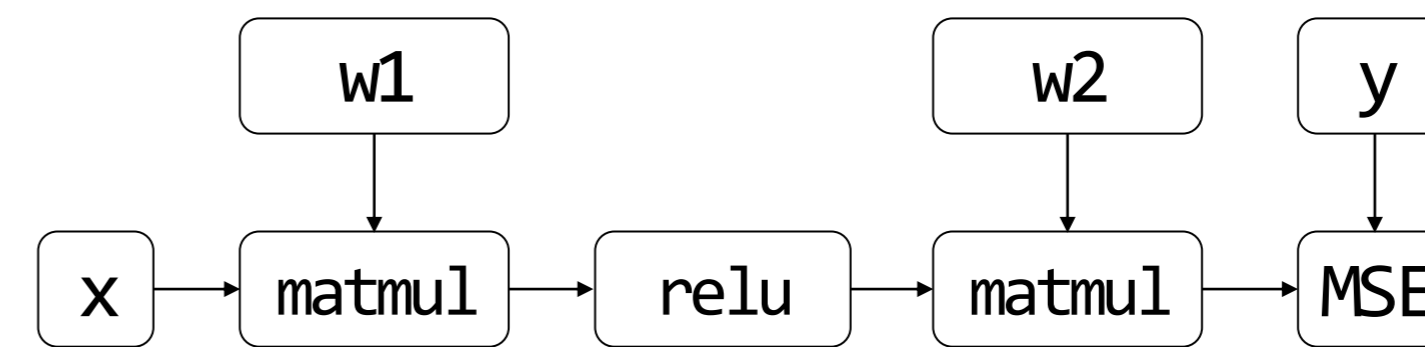
| Application | Data management<br>(OLTP)                              | Big data processing<br>(OLAP)  |
|-------------|--|--|
| Systems     | SQL<br>Query planner<br>Relational database<br>Storage | Spark/mapreduce<br>Dataflow, lineage<br>Data warehousing<br>Column storage |

# Computational Dataflow Graph

- Node: represents the computation (operator)
- Edge: represents the data dependency (data flowing direction)
- Node: also represents the *output tensor* of the operator
- Node: also represents an input constant tensor (if it is not a compute operator)



$$a \times b + 3$$



$$L = \text{MSE}(w_2 \cdot \text{ReLU}(w_1 x), y)$$

# Case Study: TensorFlow Program

- In the next few slides, we will do a case study of a deep learning program using TensorFlow v1 style API (classic Flavor).
- Note that today most deep learning frameworks now use a different style, but share the same mechanism under the hood
- Think about abstraction and implementation when going through these examples

# One linear NN: Logistic Regression

**Input**

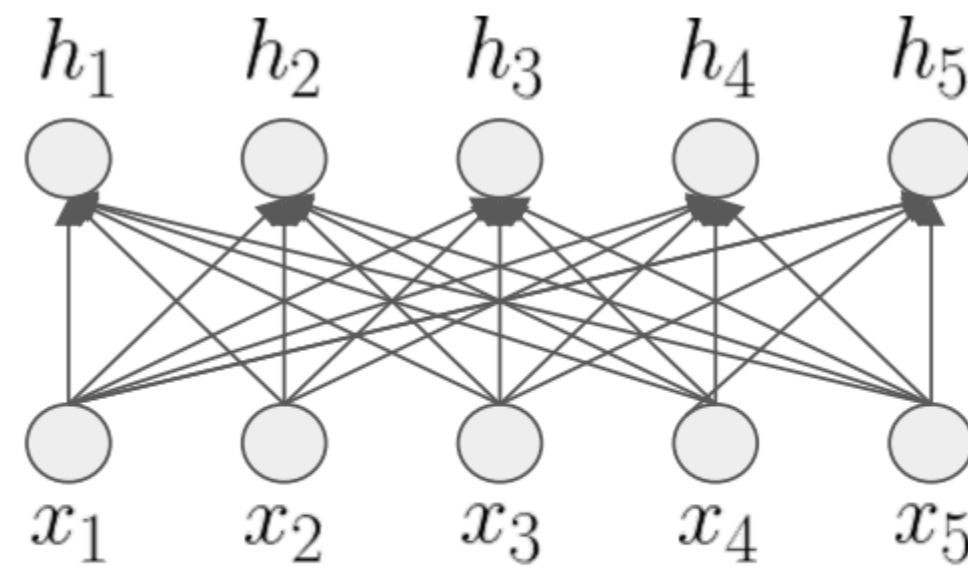
$$x_i = \begin{bmatrix} \text{pixel}_1 \\ \text{pixel}_2 \\ \dots \\ \text{pixel}_m \end{bmatrix}$$

**One Linear Layer**

$$h_k = w_k^T x_i$$

**Softmax**

$$P(y_i = k | x_i) = \frac{\exp(h_k)}{\sum_{j=1}^{10} \exp(h_j)}$$



# Whole Program

```
import tinyflow as tf
from tinyflow.datasets import get_mnist
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
y = tf.nn.softmax(tf.matmul(x, W))
# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
# Update rule
learning_rate = 0.5
W_grad = tf.gradients(cross_entropy, [W])[0]
train_step = tf.assign(W, W - learning_rate * W_grad)
# Training Loop
sess = tf.Session()
sess.run(tf.initialize_all_variables())
mnist = get_mnist(flatten=True, onehot=True)
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_:batch_ys})
```

Forward Computation  
Declaration



# Loss Function

```
import tinyflow as tf
from tinyflow.datasets import get_mnist
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
y = tf.nn.softmax(tf.matmul(x, W))
# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
# Update rule
learning_rate = 0.5
W_grad = tf.gradients(cross_entropy, [W])[0]
train_step = tf.assign(W, W - learning_rate * W_grad)
# Training Loop
sess = tf.Session()
sess.run(tf.initialize_all_variables())
mnist = get_mnist(flatten=True, onehot=True)
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_:batch_ys})
```

Loss function **Declaration**

$$P(\text{label} = k) = y_k$$
$$L(y) = \sum_{k=1}^{10} I(\text{label} = k) \log(y_i)$$

# Auto-diff

```
import tinyflow as tf
from tinyflow.datasets import get_mnist
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
y = tf.nn.softmax(tf.matmul(x, W))
# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
# Update rule
learning_rate = 0.5
W_grad = tf.gradients(cross_entropy, [W])[0]
train_step = tf.assign(W, W - learning_rate * W_grad)
# Training Loop
sess = tf.Session()
sess.run(tf.initialize_all_variables())
mnist = get_mnist(flatten=True, onehot=True)
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_:batch_ys})
```

Automatic Differentiation:  
Next incoming topic



# SGD Update

```
import tinyflow as tf
from tinyflow.datasets import get_mnist
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
y = tf.nn.softmax(tf.matmul(x, W))
# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
# Update rule
learning_rate = 0.5
W_grad = tf.gradients(cross_entropy, [W])[0]
train_step = tf.assign(W, W - learning_rate * W_grad)
# Training Loop
sess = tf.Session()
sess.run(tf.initialize_all_variables())
mnist = get_mnist(flatten=True, onehot=True)
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_:batch_ys})
```

SGD update rule



# Trigger the Execution

```
import tinyflow as tf
from tinyflow.datasets import get_mnist
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
y = tf.nn.softmax(tf.matmul(x, W))
# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
# Update rule
learning_rate = 0.5
W_grad = tf.gradients(cross_entropy, [W])[0]
train_step = tf.assign(W, W - learning_rate * W_grad)
# Training Loop
sess = tf.Session()
sess.run(tf.initialize_all_variables())
mnist = get_mnist(flatten=True, onehot=True)
for i in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x: batch_xs, y_:batch_ys})
```

Real execution happens here!

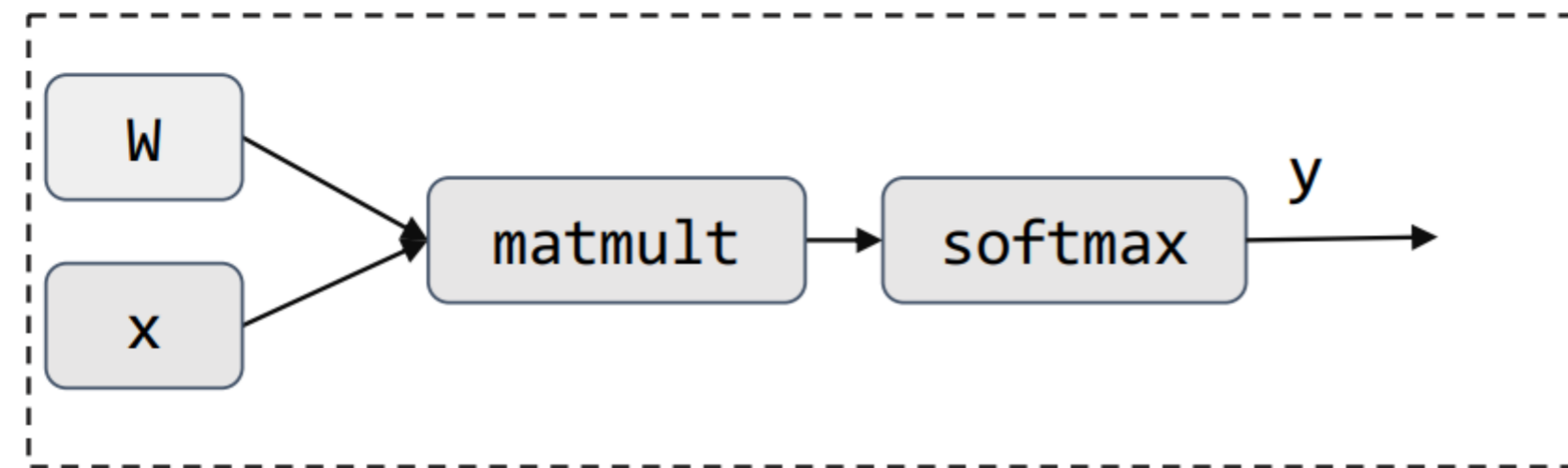


# What happens behind the Scene

```
x = tf.placeholder(tf.float32, [None, 784])
```

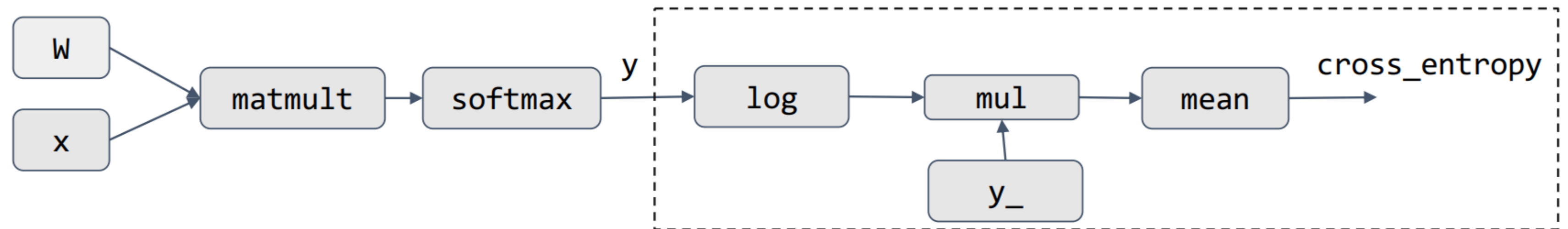
```
W = tf.Variable(tf.zeros([784, 10]))
```

```
y = tf.nn.softmax(tf.matmul(x, W))
```



# What happens behind the Scene (Cond.)

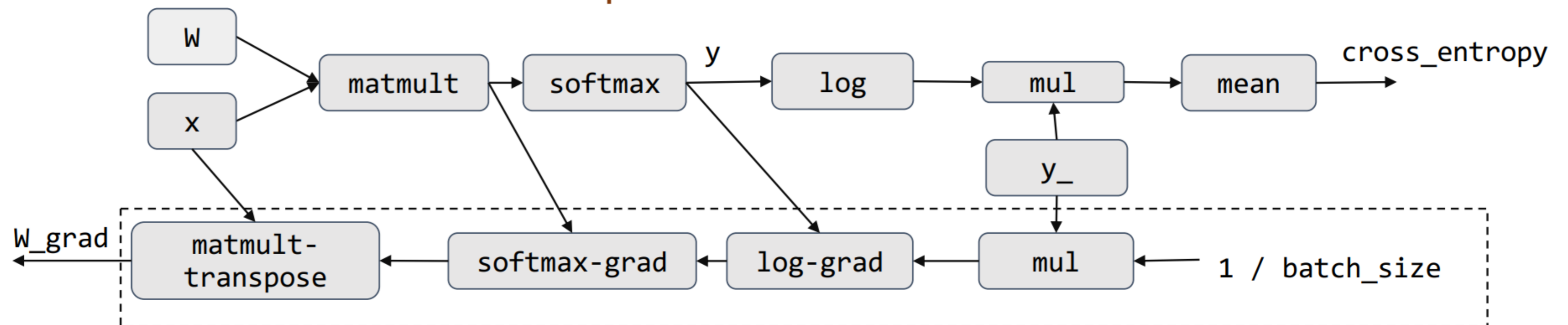
```
y_ = tf.placeholder(tf.float32, [None, 10])  
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
```



# What happens behind the Scene (Cond.)

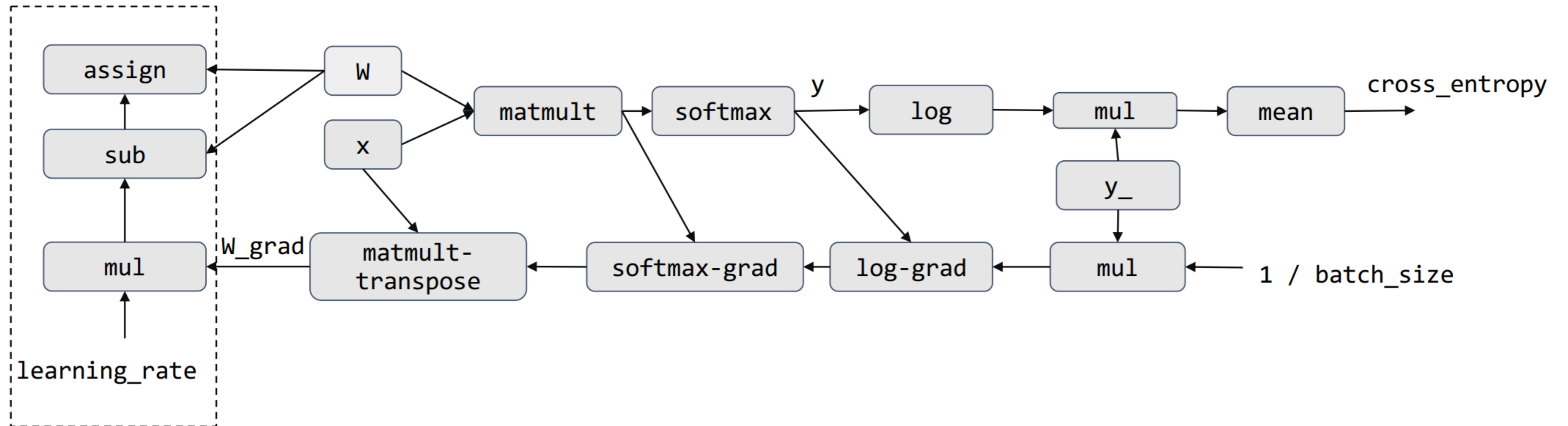
```
W_grad = tf.gradients(cross_entropy, [W])[0]
```

Automatic Differentiation, more details in follow up lectures



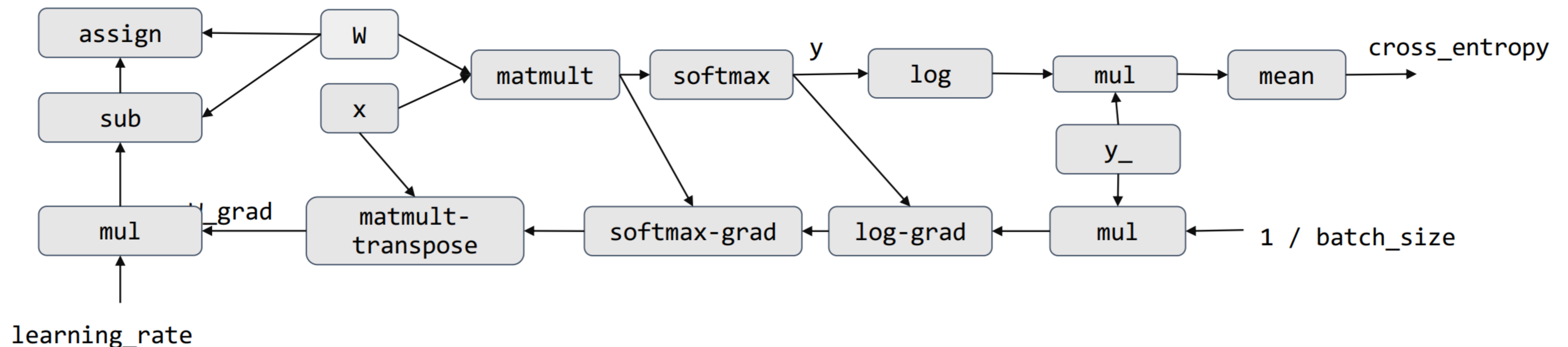
# What happens behind the Scene (Cond.)

```
sess.run(train_step, feed_dict={x: batch_xs, y_:batch_ys})
```



# Discussion

- What are the benefits for computational graph abstraction?
- What are possible implementations and optimizations on this graph?
- What are the cons for computational graph abstraction?



# A different flavor: PyTorch

A graph is created on the fly

```
W_h = torch.randn(20, 20, requires_grad=True)
W_x = torch.randn(20, 10, requires_grad=True)
x = torch.randn(1, 10)
prev_h = torch.randn(1, 20)
```



# Topic: Symbolic vs. Imperative

- Symbolic vs. imperative programming
- Define-then-run vs. Define-and-run

```
# Create the model
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
y = tf.nn.softmax(tf.matmul(x, W))
# Define loss and optimizer
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
```

Symbolic

```
x = torch.Tensor([3])
y = torch.Tensor([2])
z = x - y
loss = square(z)
loss.backward()
print(x.grad)
```

Imperative

# Discussion: Symbolic vs. Imperative

- Symbolic
  - Good
    - easy to optimize (e.g. distributed, batching, parallelization) for developers
    - Much more efficient: can be 10x more efficient
  - Bad
    - The way of programming might be counter-intuitive
    - Hard to debug for user programs
    - Less flexible: you need to write symbols before actually doing anything
- Imperative:
  - Good
    - More flexible: write one line, evaluate one line (that's why we all like Python)
    - Easy to program and easy to debug
  - Bad
    - Less efficient
    - More difficult to optimize

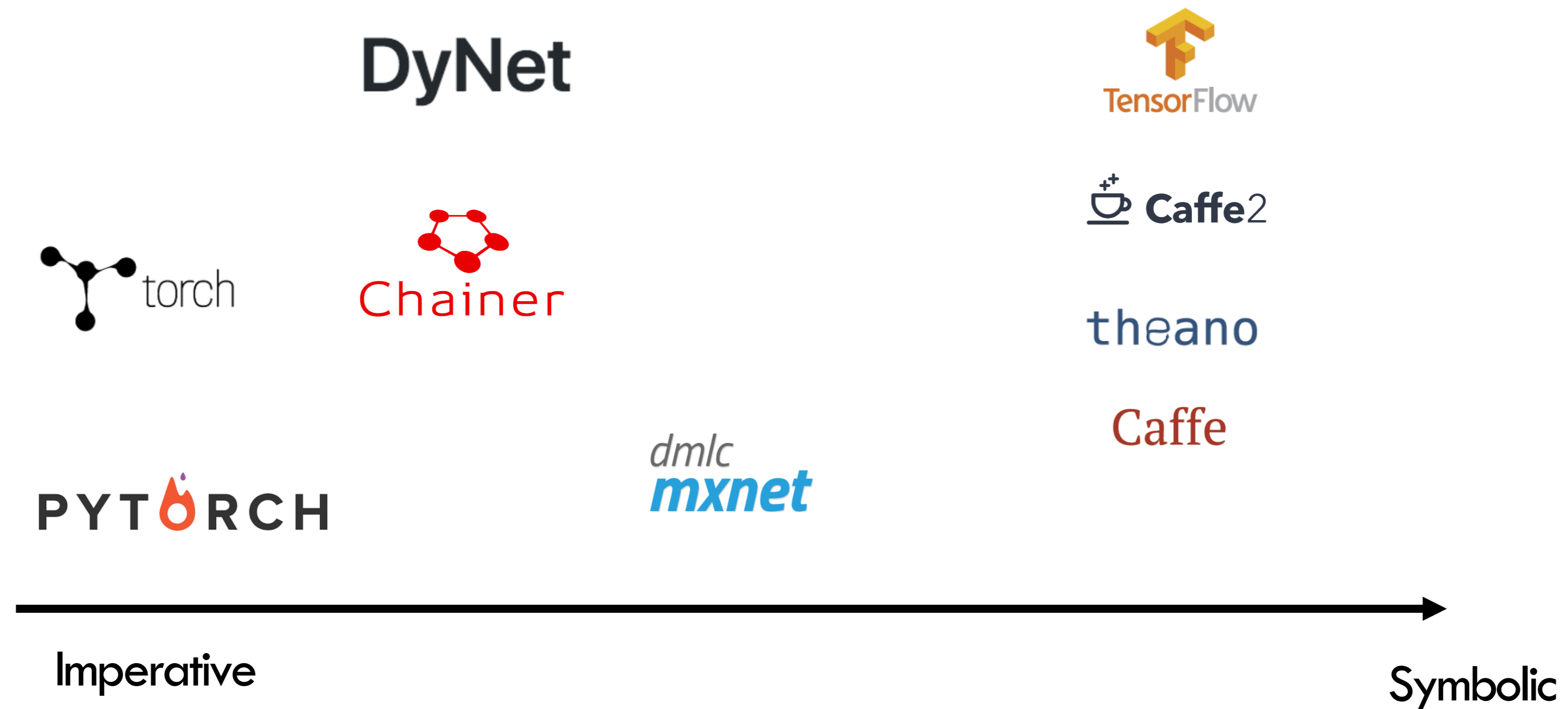
# MCQ Time

- Which category, symbolic vs. imperative, is the following PL belonging to?
  - C++
  - Python
  - SQL

# Something Interesting Here?

- Python is a *define-and-run* PL
- Tensorflow is *define-then-run* ML framework
- Tensorflow has Python as the primary interface language
  
- You are indeed using a DSL built on top of Python
  - But PyTorch DSL is more *pythonic* than Tensorflow DSL.

# Symbolic vs. Imperative (2016)

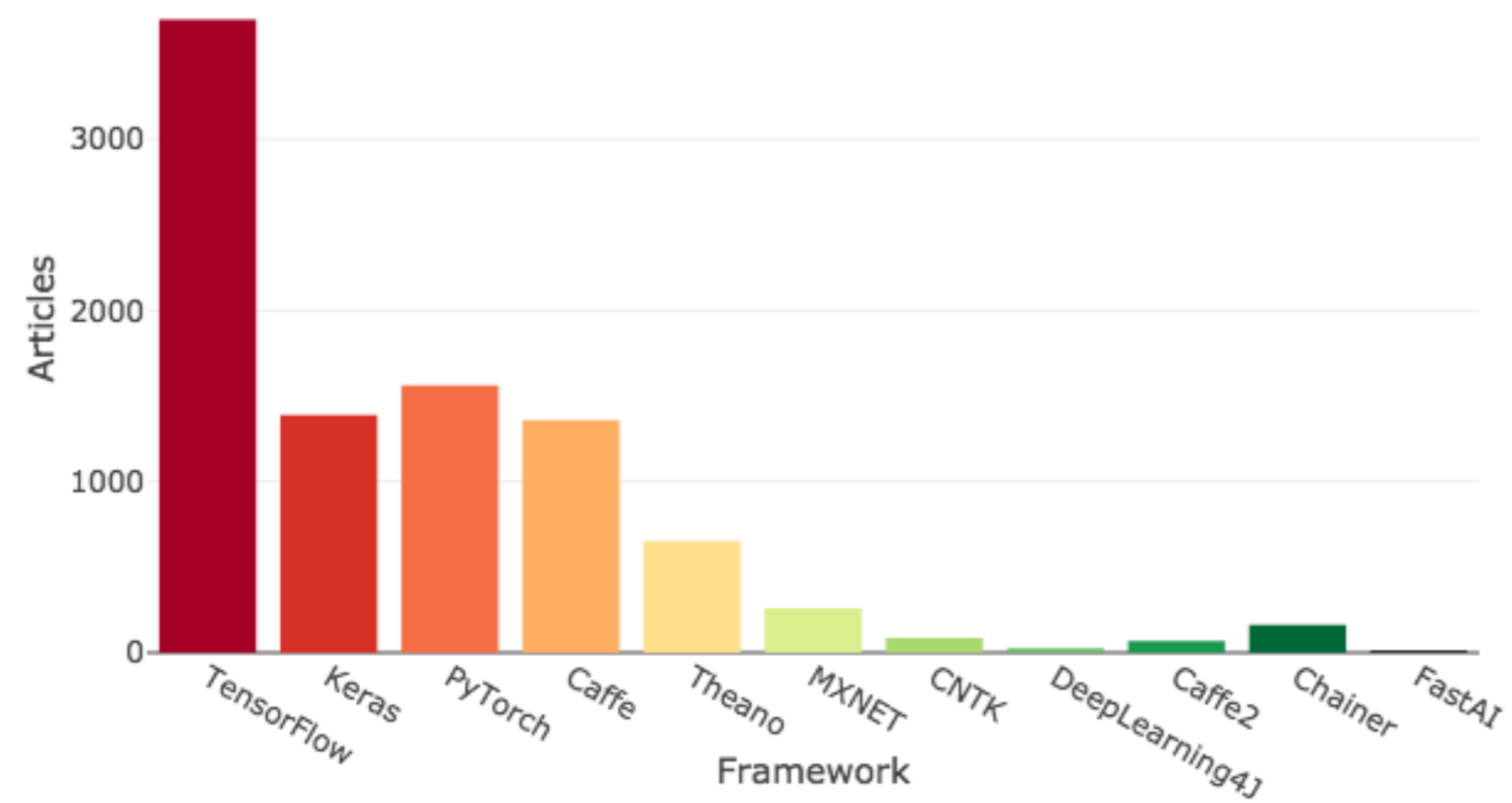


# Symbolic vs. Imperative (2024)

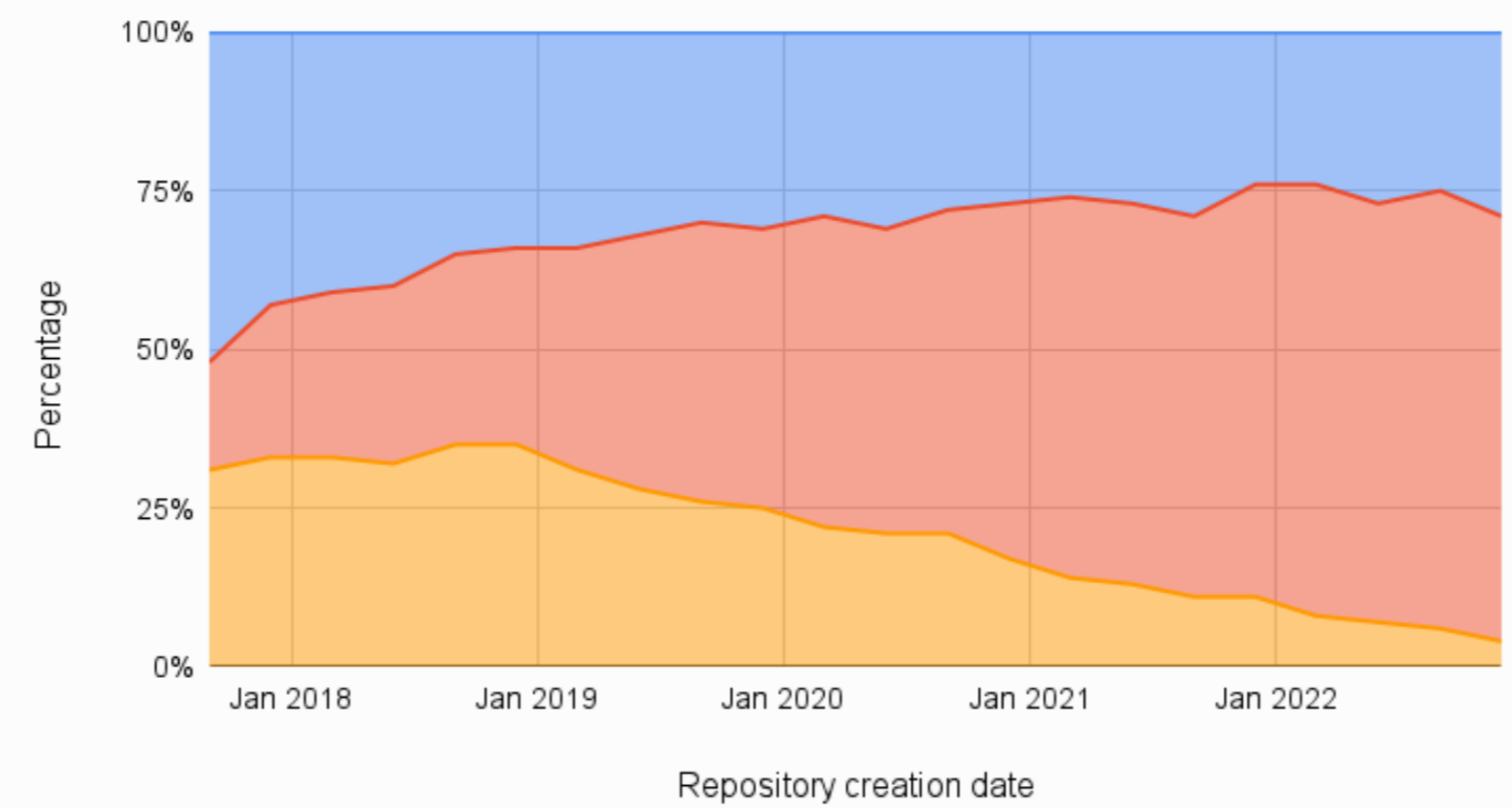


# Market size of frameworks

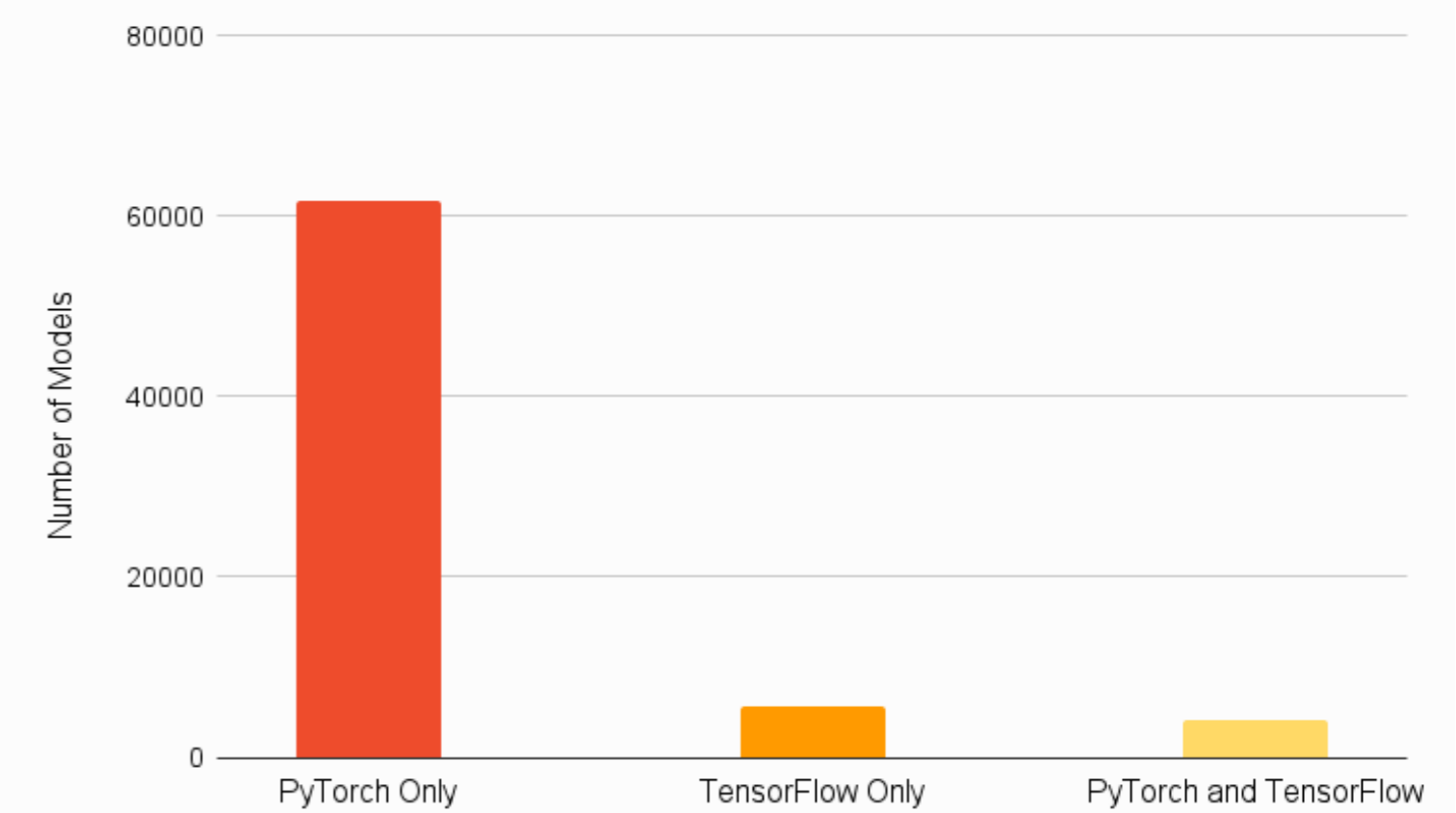
ArXiv Articles



Percentage of Repositories by Framework



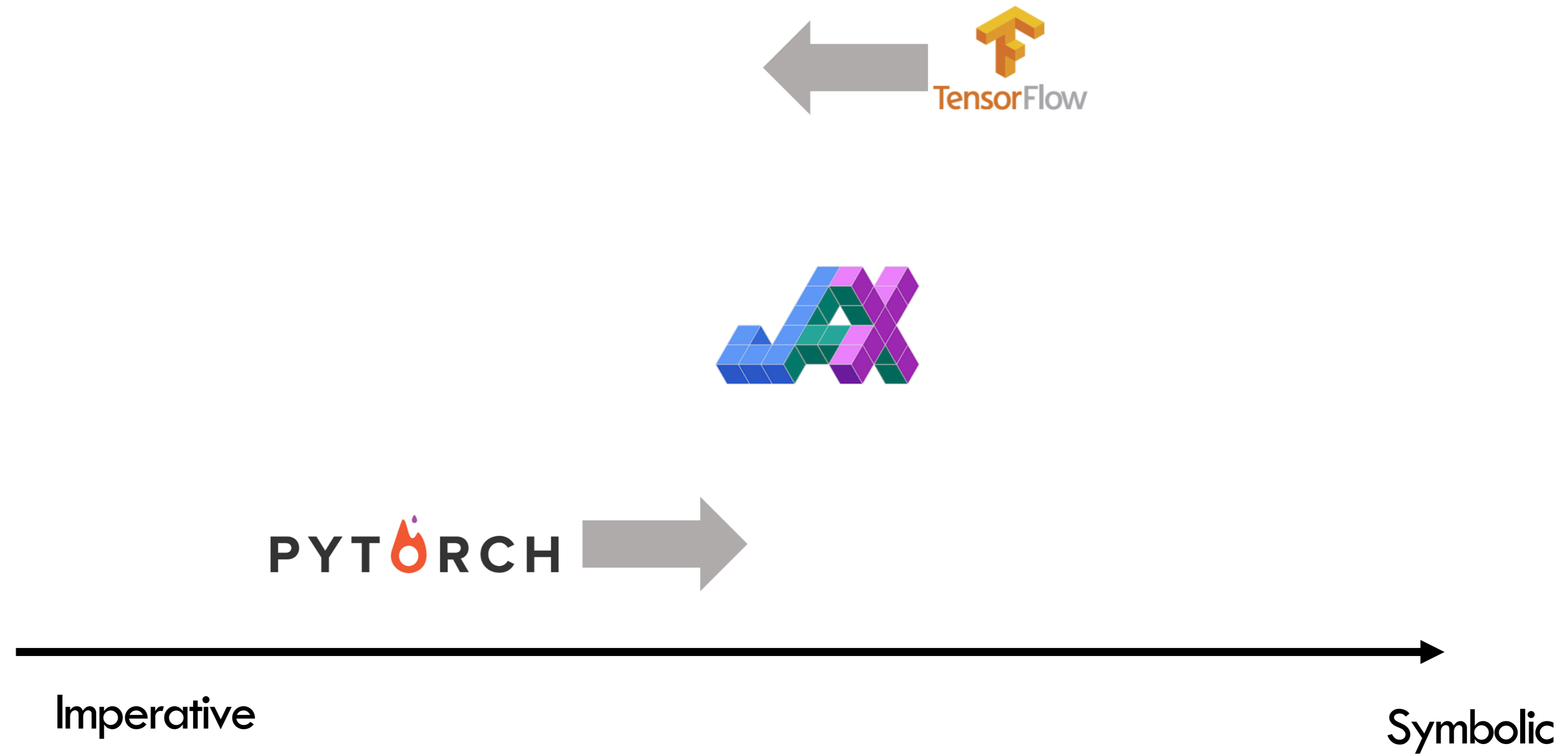
Number of Models on HuggingFace



# After-class Question

Why PyTorch wins the market even if it was a later framework?

# Symbolic vs. Imperative (2024)



# Just-in-time (JIT) Compilation

- Ideally, we want define-and-run during \_\_\_\_\_
- We want define-then-run during \_\_\_\_\_
- Q: how can combine the best of both worlds?

```
x = torch.Tensor([3])
y = torch.Tensor([2])
z = x - y
loss = square(z)
loss.backward()
print(x.grad)
```

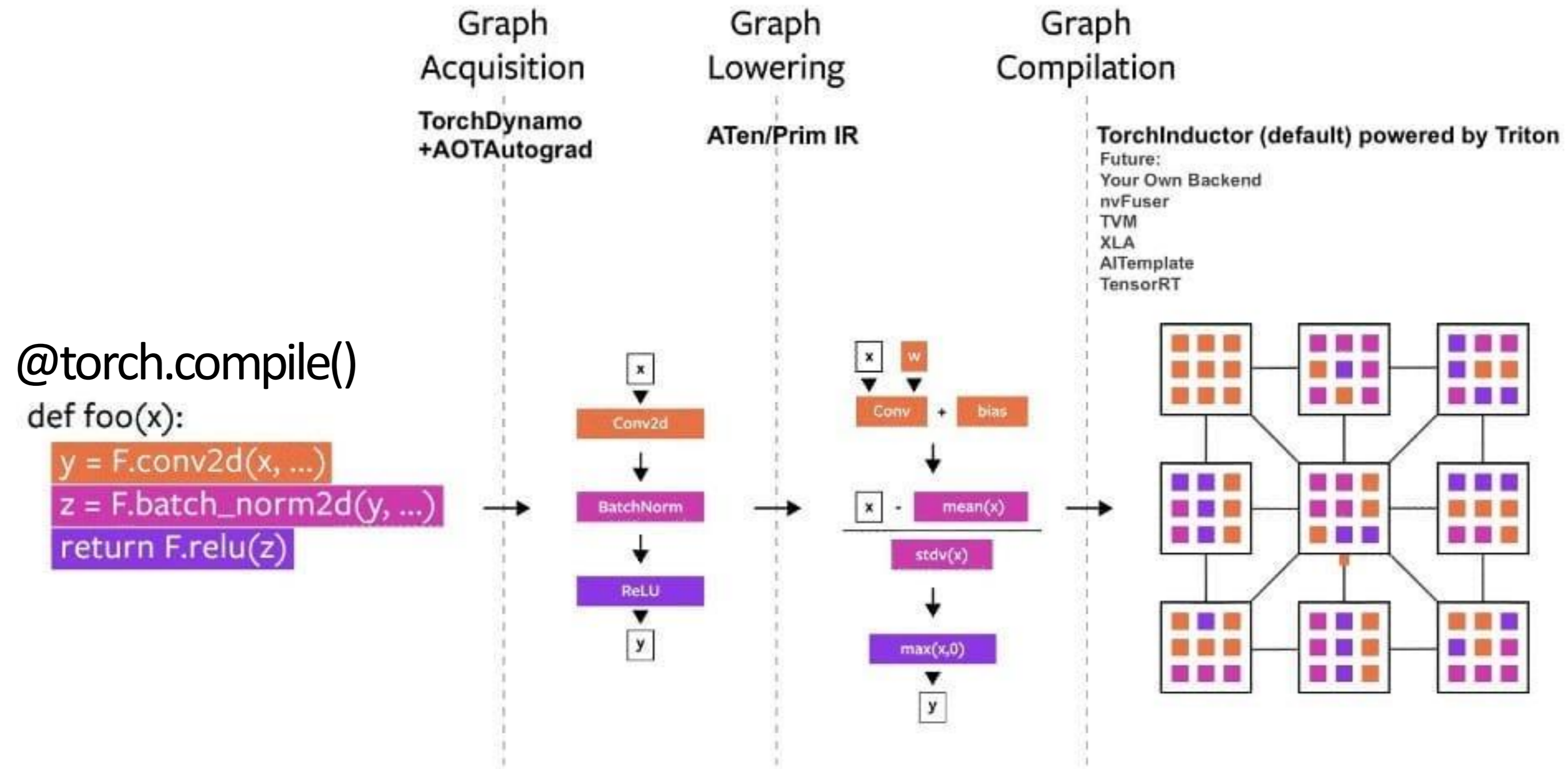
**Dev mode**

@torch.compile()

```
x = torch.Tensor([3])
y = torch.Tensor([2])
z = x - y
loss = square(z)
loss.backward()
print(x.grad)
```

**Deploy mode:  
Decorate torch.compile()**

# What happens behind the scene

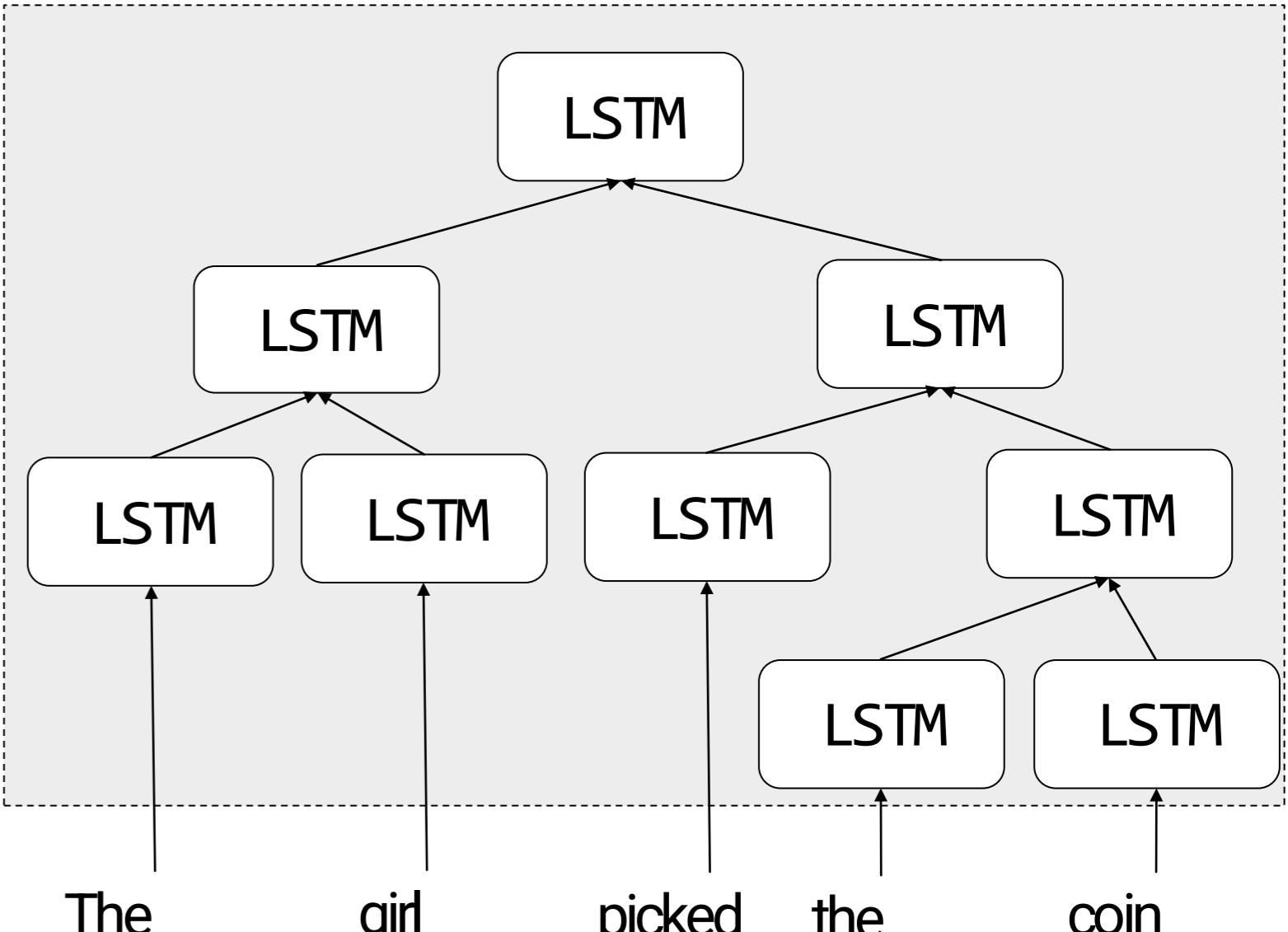
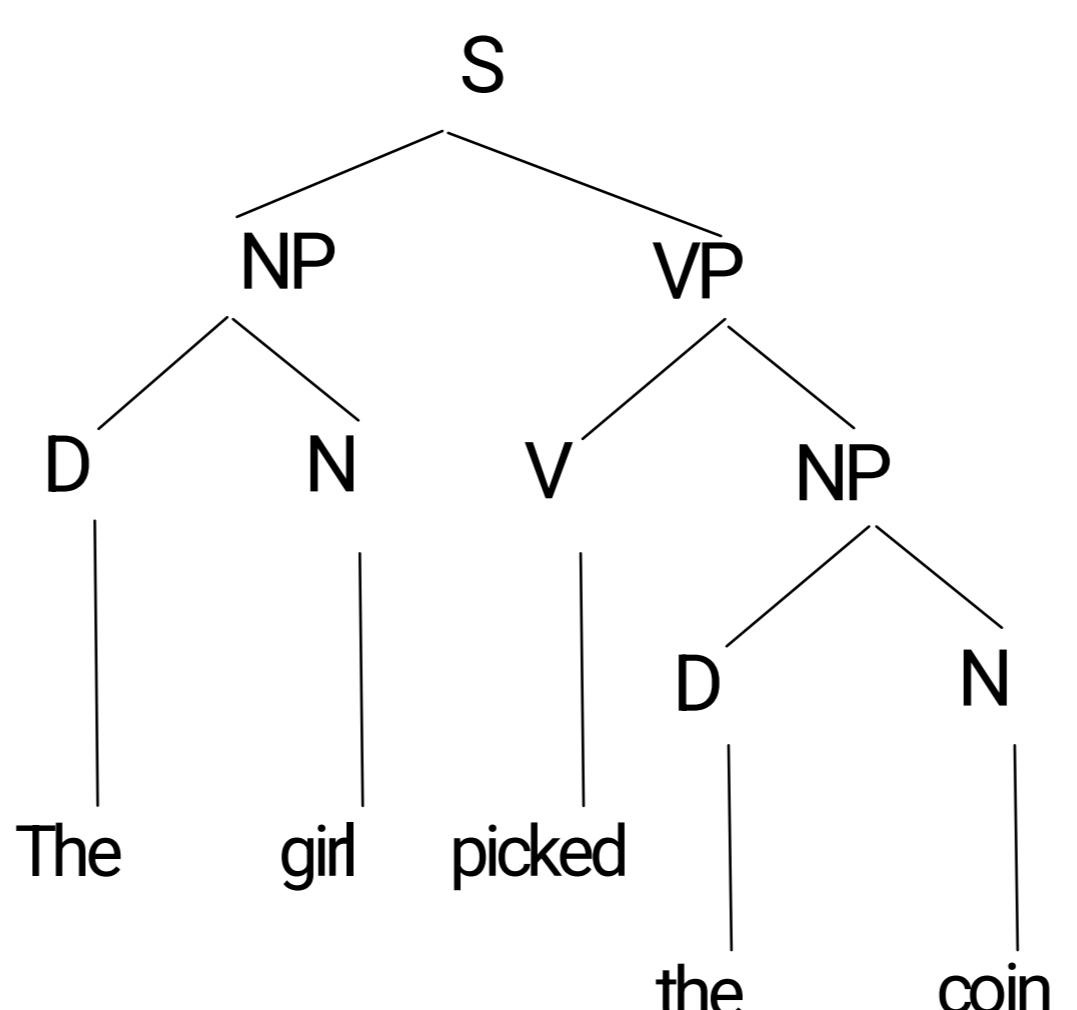
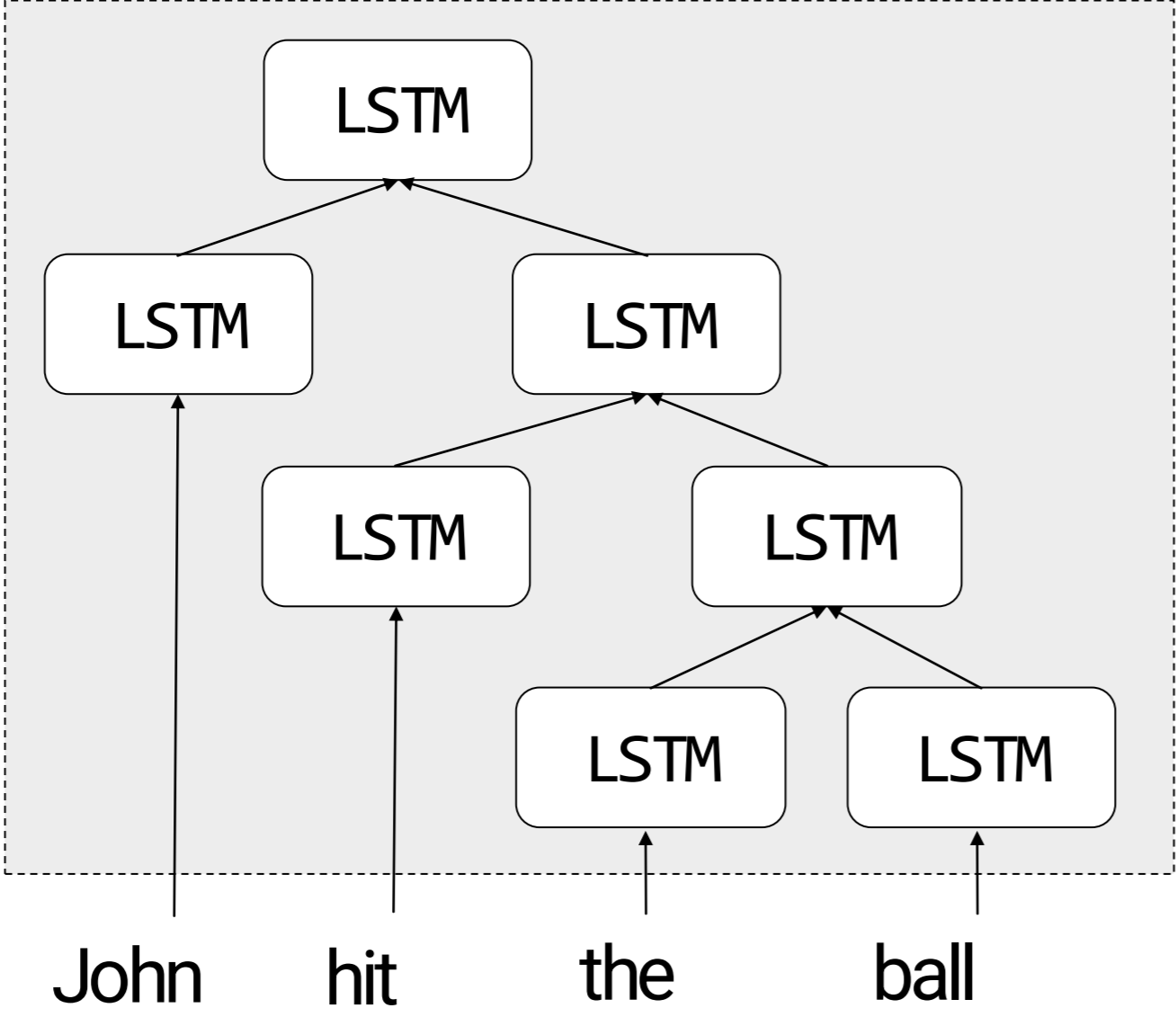
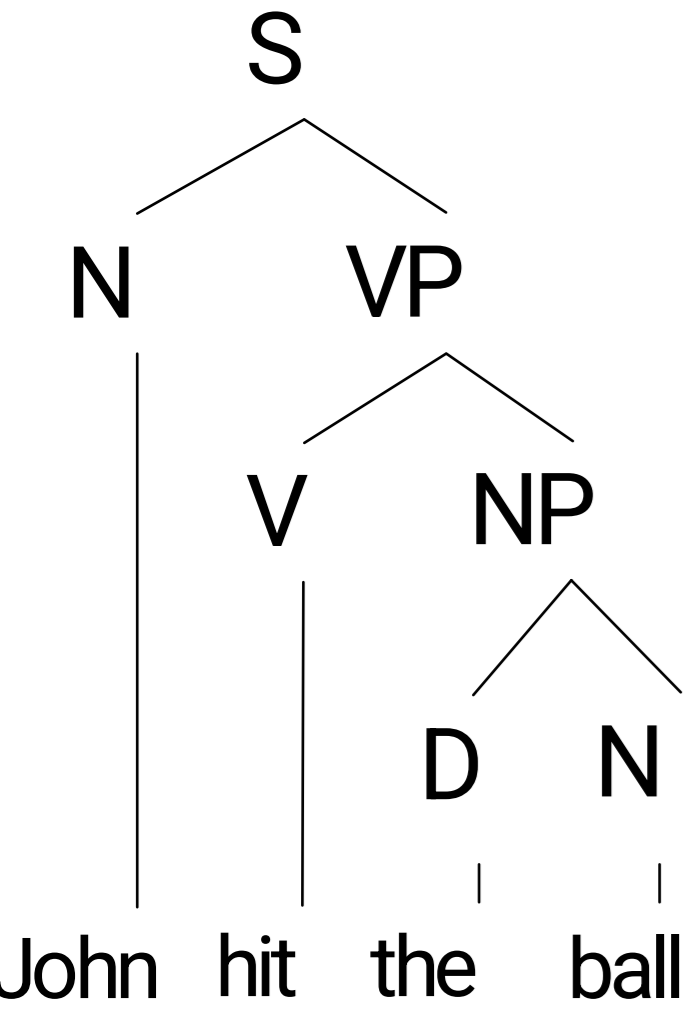
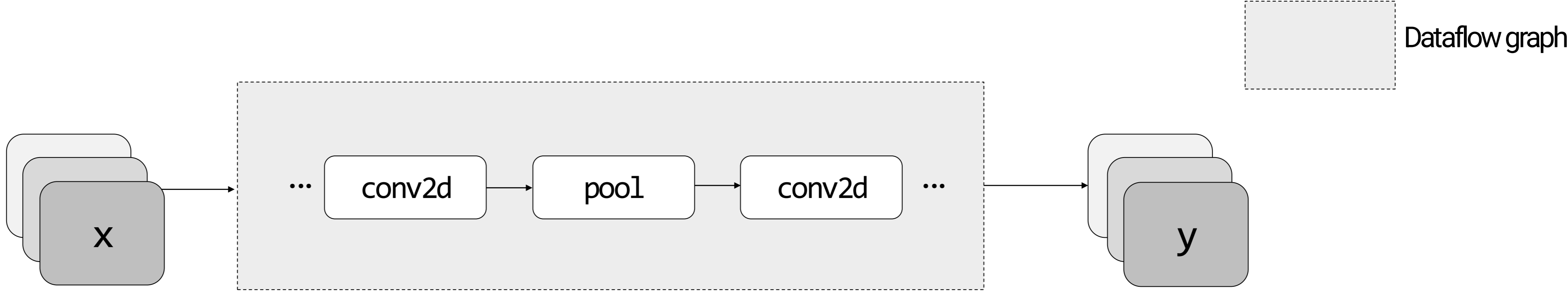


What is the problem of JIT?  
Requirements for static graphs

Q: What is the problem of JIT?

A: Requirements for static graphs

# Static Models vs. Dynamic Models



# Static vs. Dynamic Dataflow Graphs

- Static Dataflow graphs
  - Define once, optimized once, execute many times
  - Execution: Once defined, all following computation will **follow** the defined computation

# Static vs. Dynamic Dataflow Graphs

- Dynamic Dataflow Graphs
  - Difficulty in expressing complex flow-control logic
  - Complexity of the computation graph implementation
  - Difficulty in debugging

- Is LLM static or dynamic?

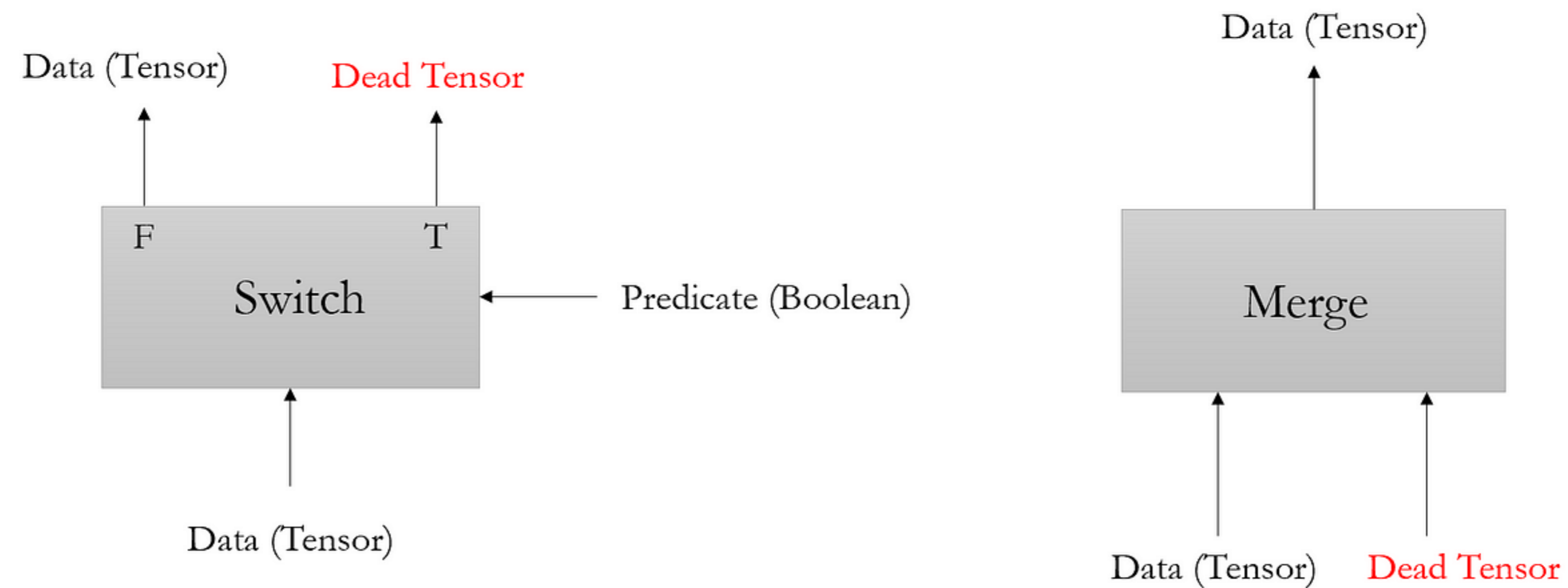
# Open Research: How to Handle Dynamics?

Three ways:

- Just do Define-and-run and forget about JIT
  - As long as you do not care about performance...
- Introduce Control flow Ops
- Piecewise compilation and guards

# Control flow primitives

- Example primitive: Switch and Merge

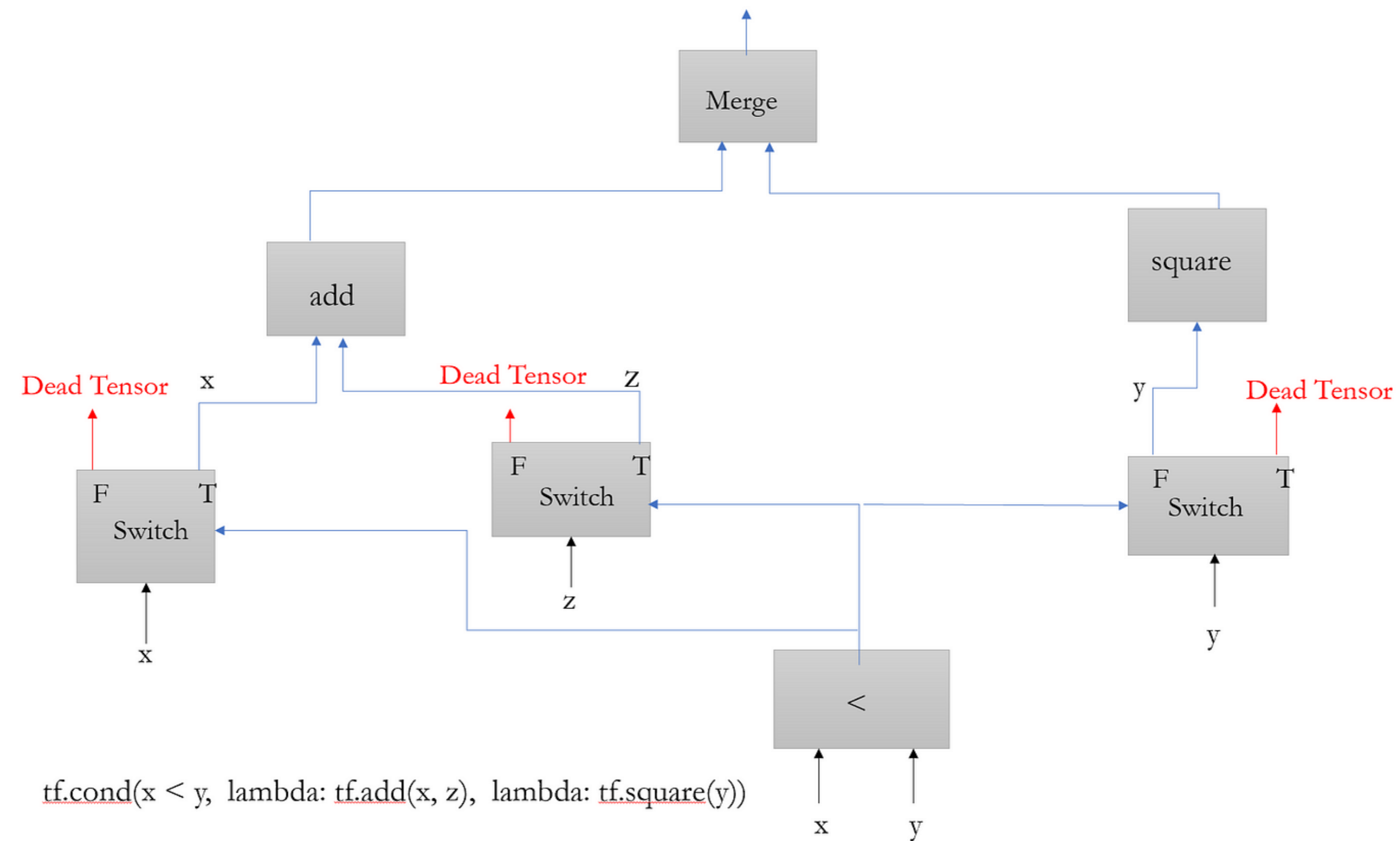


Switch receives two arguments: Data and predicate (boolean), and has two outputs: data and dead Tensor

Merge receives two arguments: Data and dead Tensor, and has one outputs: data

# Control flow primitives

- Example compute: `tf.cond(x < y, lambda: tf.add(x, z), lambda: tf.square(y))`



# Control flow primitives

Control flow is natural idea in all PLs:

- if...then...,
- for,
- while

What is the potential problem of using control flow in dataflow graphs?

# Piecewise Compilation

- Case 1: a graph accepting input shapes of  $[x, c1, c2]$ 
  - $c1, c2$ : constants
  - $x$ : variable
  - Q: how to statically JIT this graph?
- Case 2: a graph with is static, then dynamic, then static.
  - Q: how to statically JIT this graph?

# High-level Picture

Data

?  $\{x_i\}_{i=1}^n$

Model



Math primitives  
(mostly matmul)


? A repr that expresses the  
computation using primitives


Compute

? Make them run on (clusters  
of ) different kinds of  
hardware

## Next class

A repr that expresses the computation using primitives

 A repr that expresses the **forward** computation using primitives

 A repr that expresses the **backward** computation using primitives