https://hao-ai-lab.github.io/cse234-w25/

CSE 234: Data Systems for Machine Learning
Winter 2025

LLMSys

Optimizations and Parallelization

MLSys Basics

Forms worth your attention

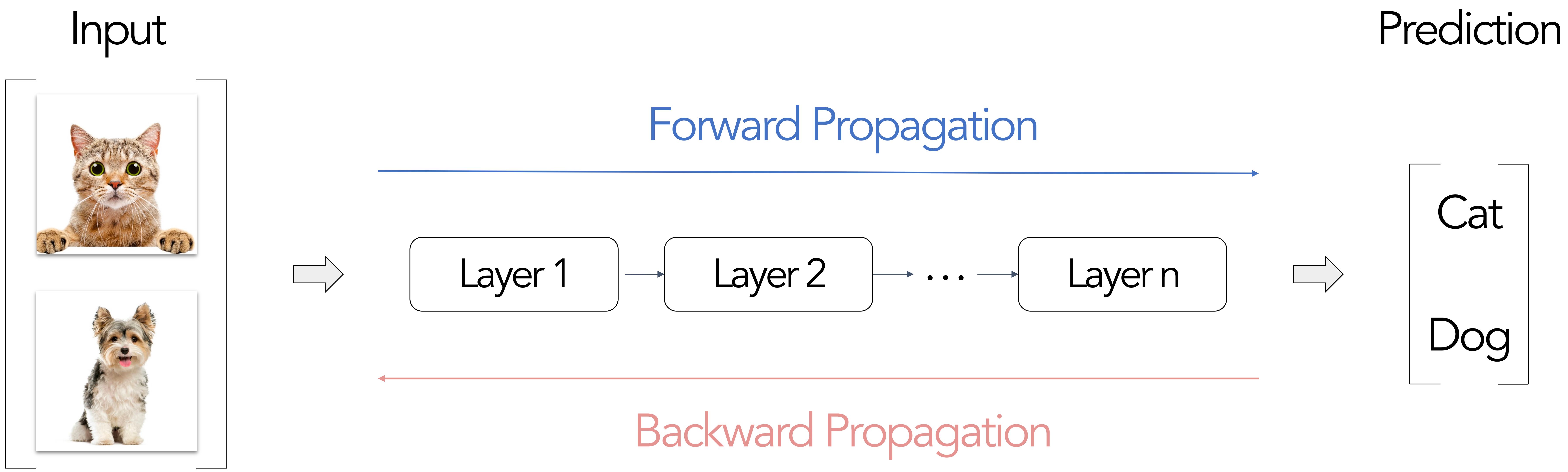
- Beginning of quarter survey
  - Please fill the survey by end of week 2
    - If >=80% of you filled the survey, all of you get 1%
    - If <80%, all of you do not get 1%
- Scribe sign-up:
  - Do it as a team.
  - Do it as detailed as possible.
  - Most efficient way to get 8%!
- There is a slack for your convenience:
  - https://join.slack.com/t/cse234-w25/shared_invite/zt-2xc1bsmxz-xFXiufPMM8Fv8eCsiaT2Rg

Today

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

Background: DL Computation

- Idea: Composable Layers



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

Dive in to Models: Three parts

$$\underbrace{\theta^{(t+1)}}_{\text{parameter}} = \underbrace{f}_{\substack{\text{weight update} \\ \text{(sgd, adam, etc.)}}}\Big(\theta^{(t)},\ \underbrace{\nabla_L}_{\substack{\text{model} \\ \text{(CNN, GPT, etc.)}}}\big(\theta^{(t)},\ \underbrace{D^{(t)}}_{\text{data}}\big)\Big)$$

- 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
  - SGD, Newton methods,

Three important components

Data	Model	Compute
• Images	• CNNs	• CPUs
• Text	• RNNs	• GPUs/TPUs/LPUs
• Audio	• Transformers	• M1/M2/M3/M4
• Table	• MoEs	• FPGA/etc.
• etc.	• 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?
- System building is the process to reveal the most important factors

What are the most important models and optimization algos

Most important models?

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

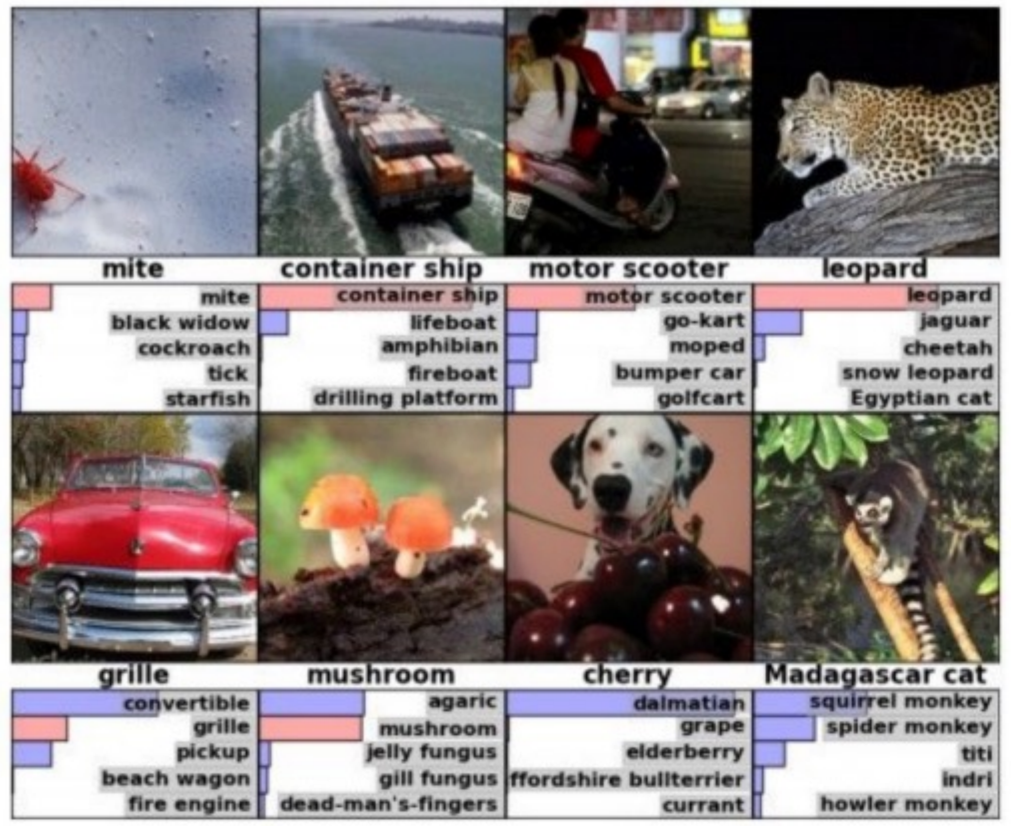
Most important optimization algorithms?

- SDG and its variants, e.g., Adam

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

- In this class, we review the most important 4 model families
  - Convolutional Neural Networks
  - Recurrent 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 = ResNet, Y = CNNs
- If you have trouble following this session, spend time reading deep learning book or learn https://sites.google.com/view/cse251b

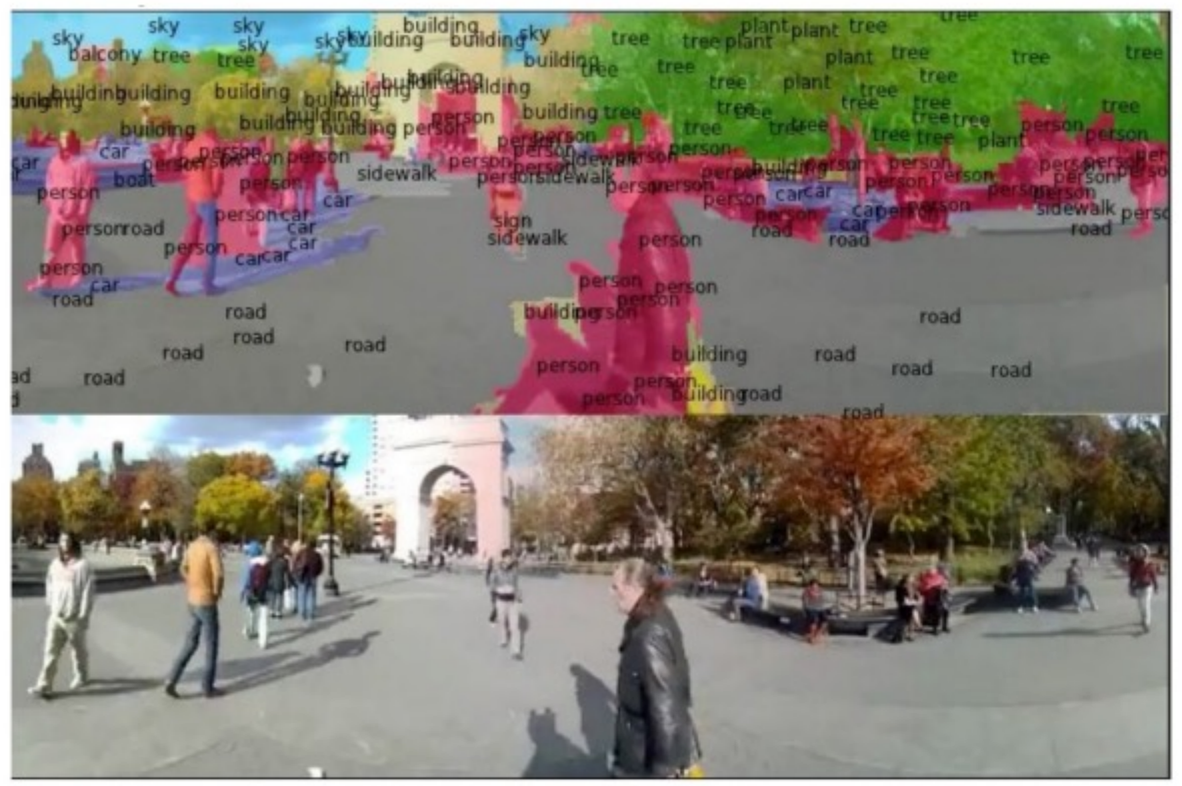
CNNs: Applications



Classification

Retrieval

Detection



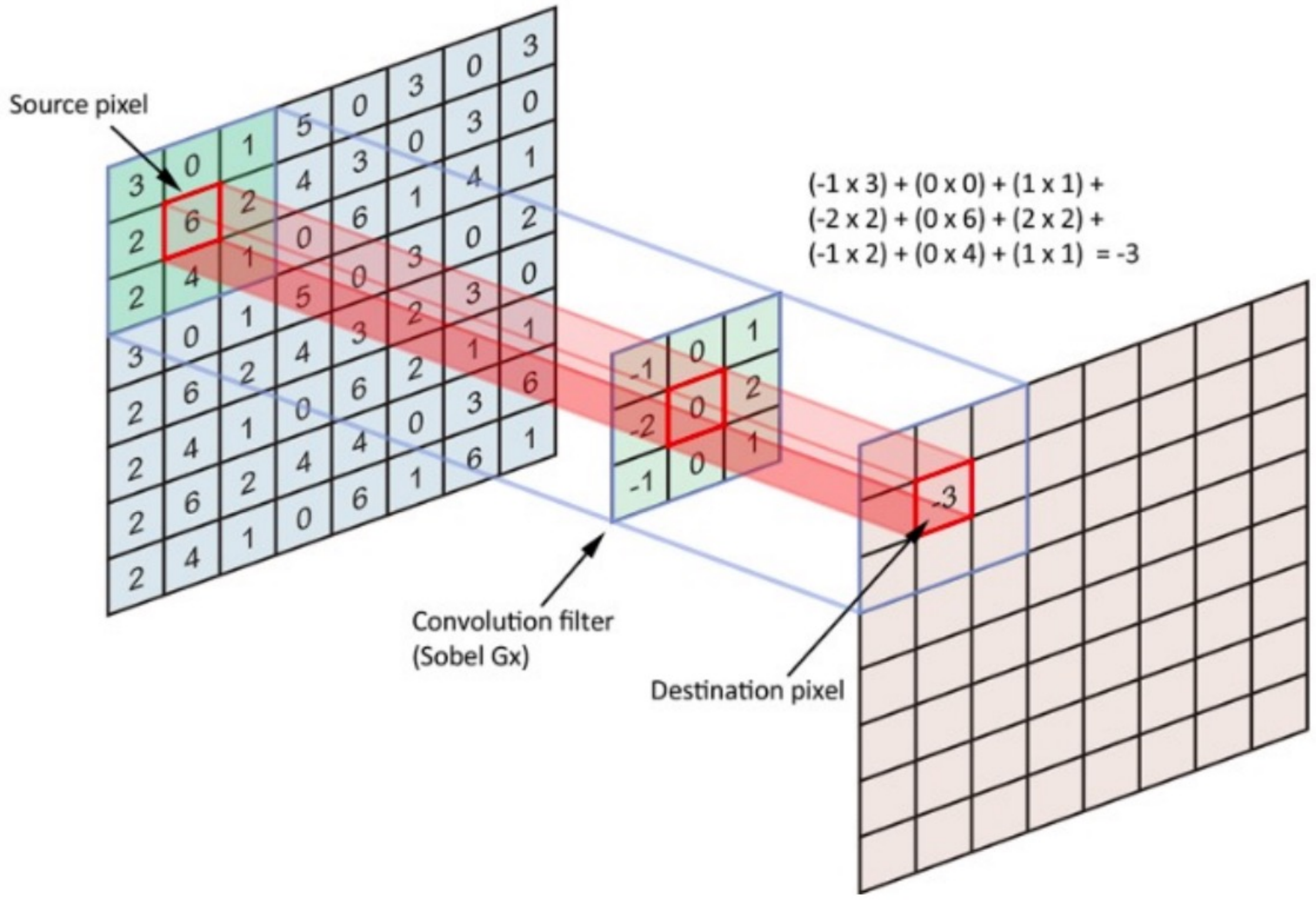
Segmentation

Self-Driving

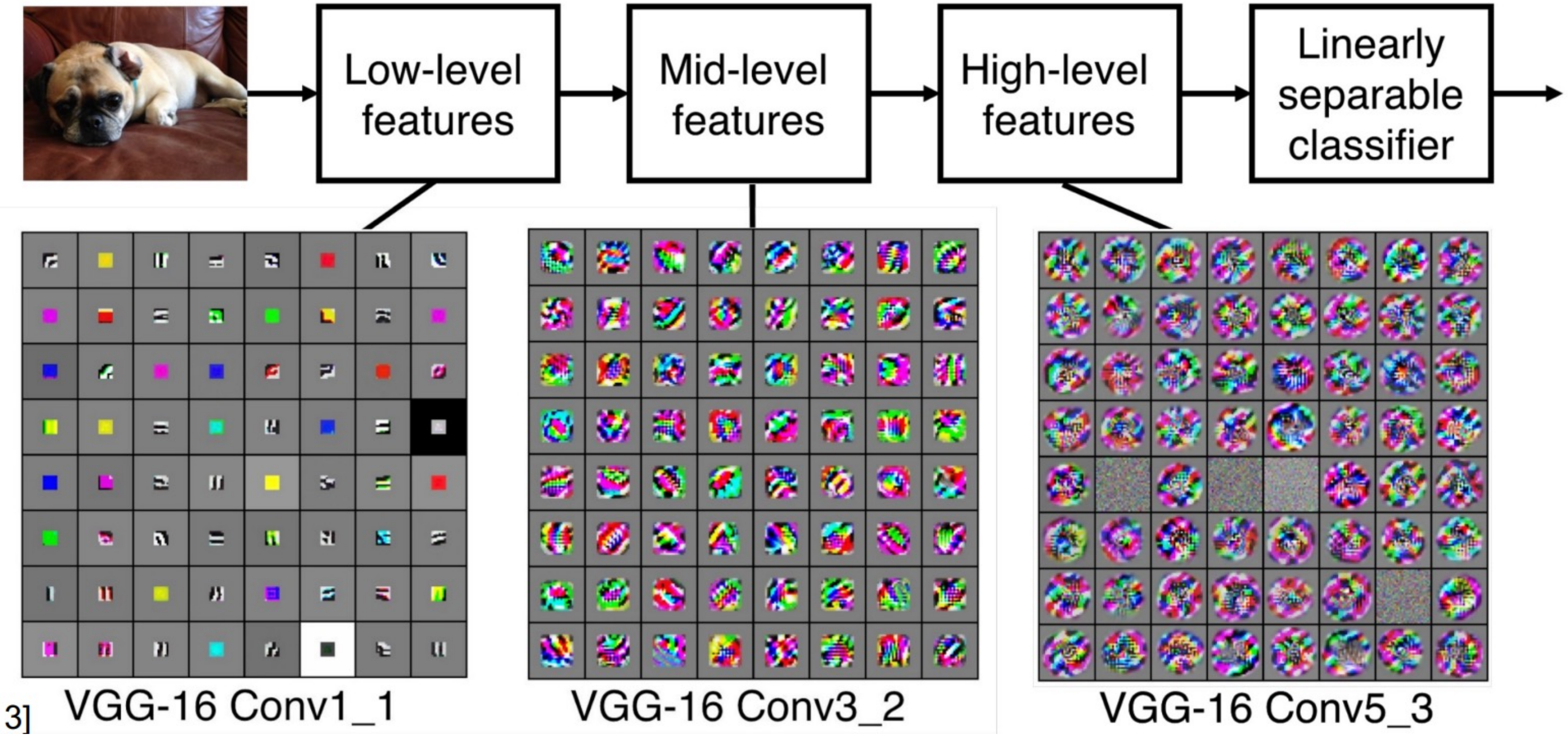
Synthesis

CNN: Key components

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



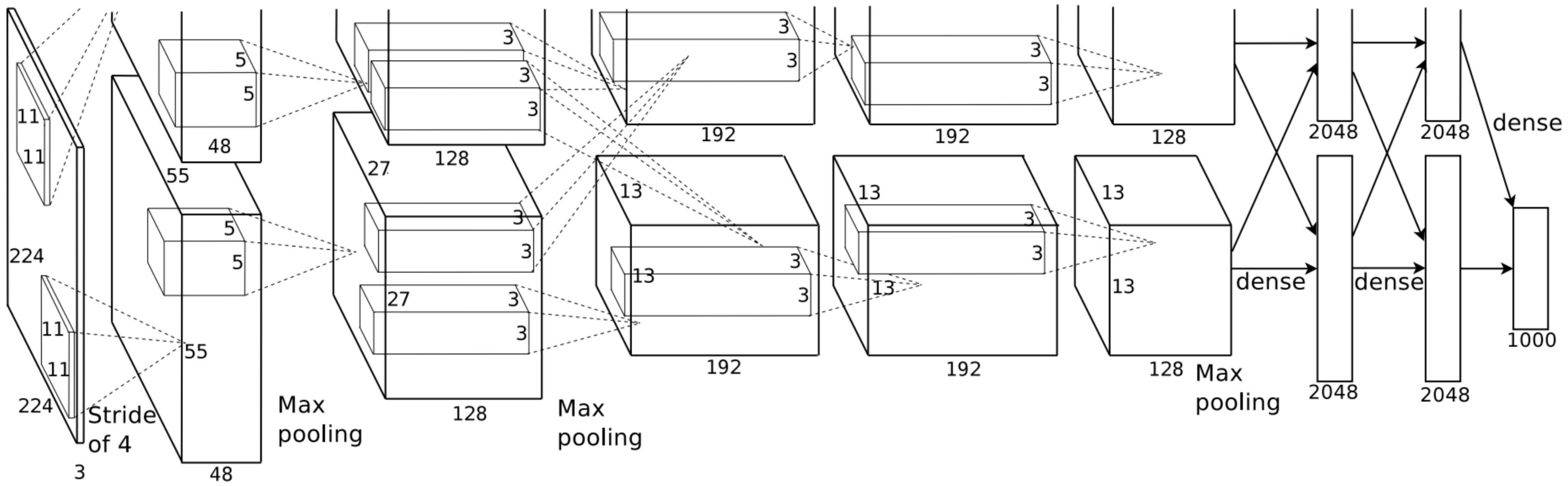
Principle: Stacking Conv layers

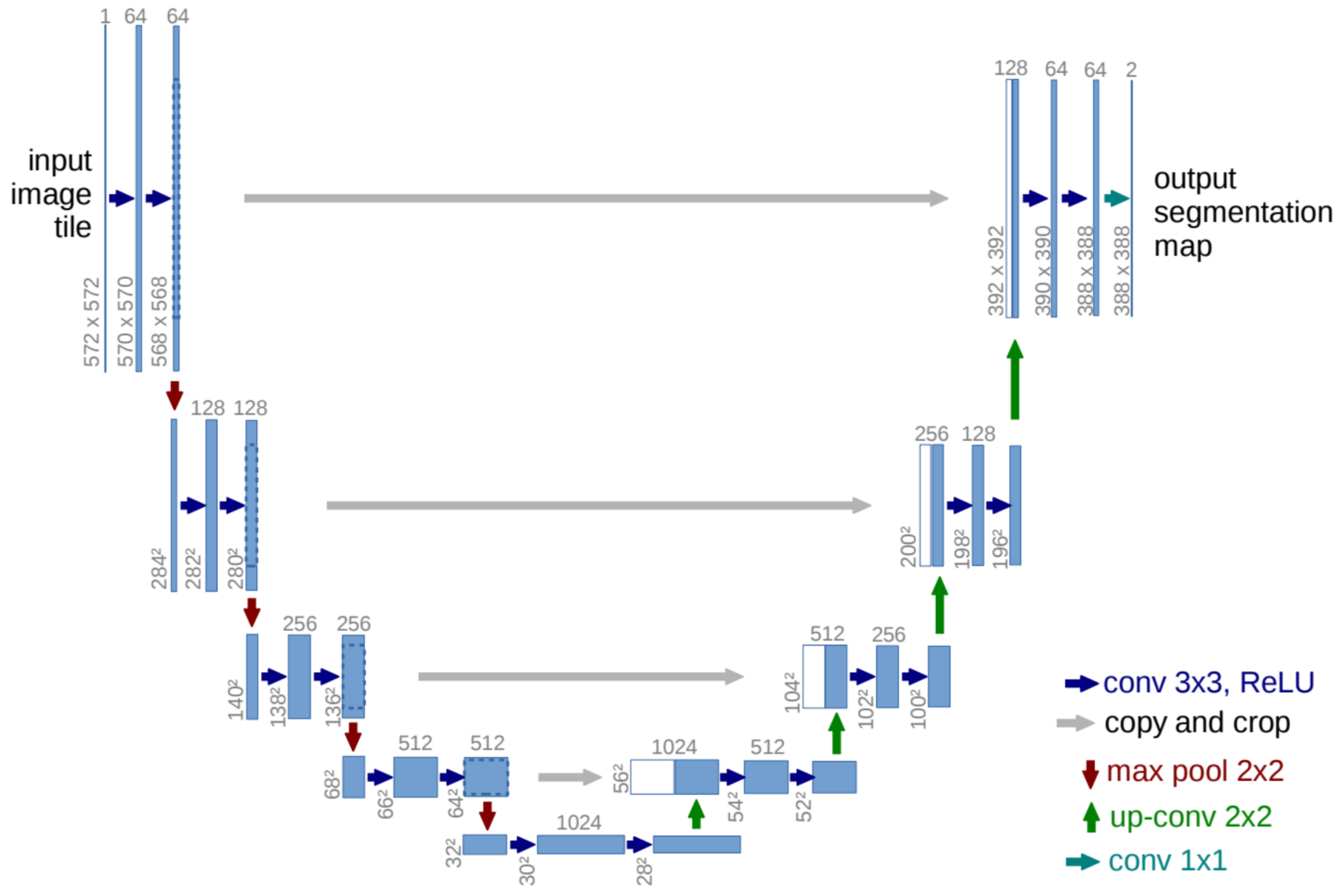


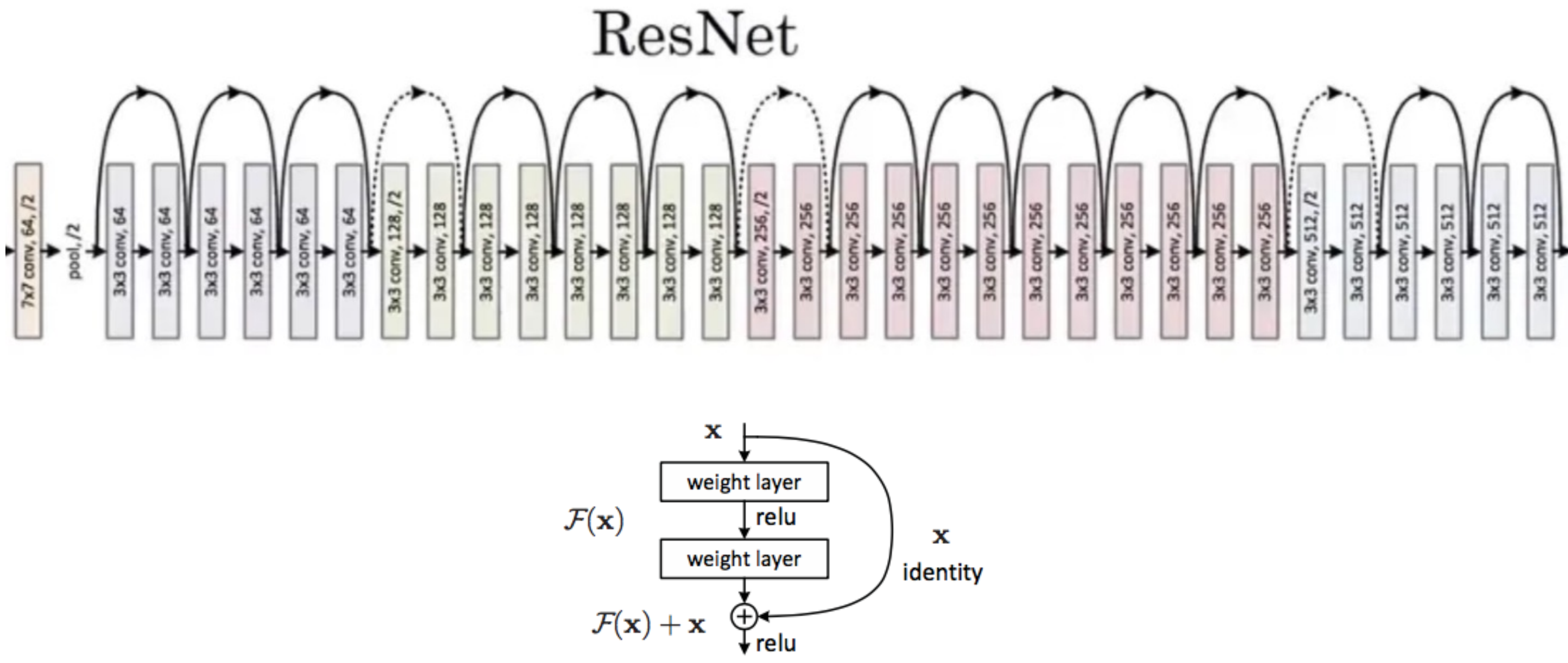
Zeiler and Fergus 2013]

CNN: Top3 models

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

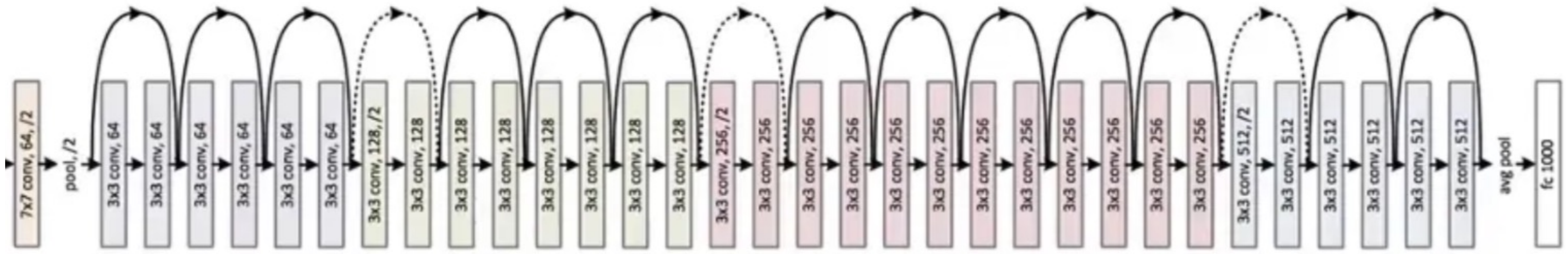






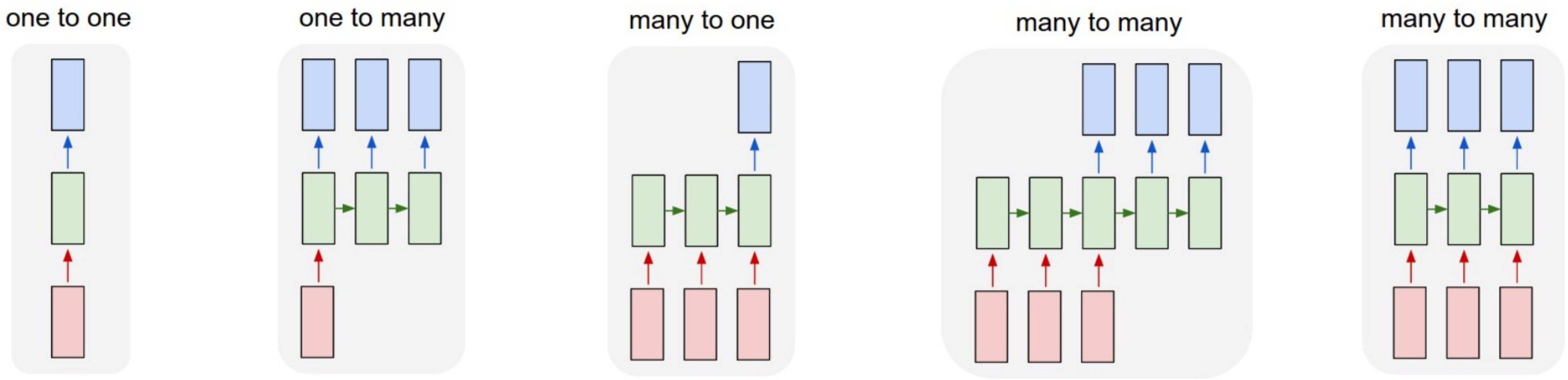
Most important components in CNNs?

ResNet

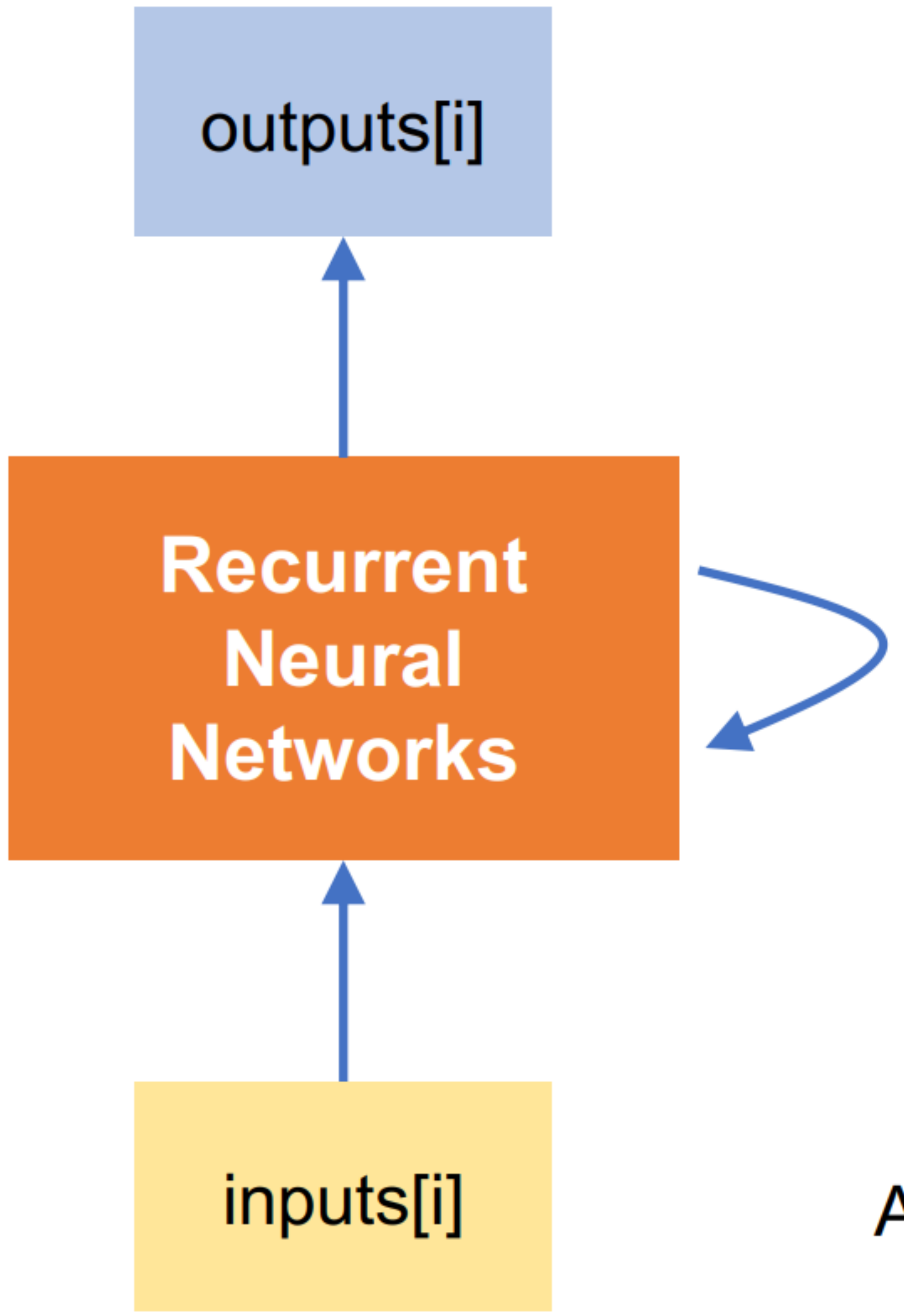


- Conv
  - Conv1d, Conv2d, conv3d, esp. 3x3-conv2d
- Matmul (linear) :
  - C = A * B
- Softmax
- Elementwise operations:
  - ReLU, add, sub
  - Pooling, normalization, etc.

Recurrent Neural Networks



Recurrent Neural Networks



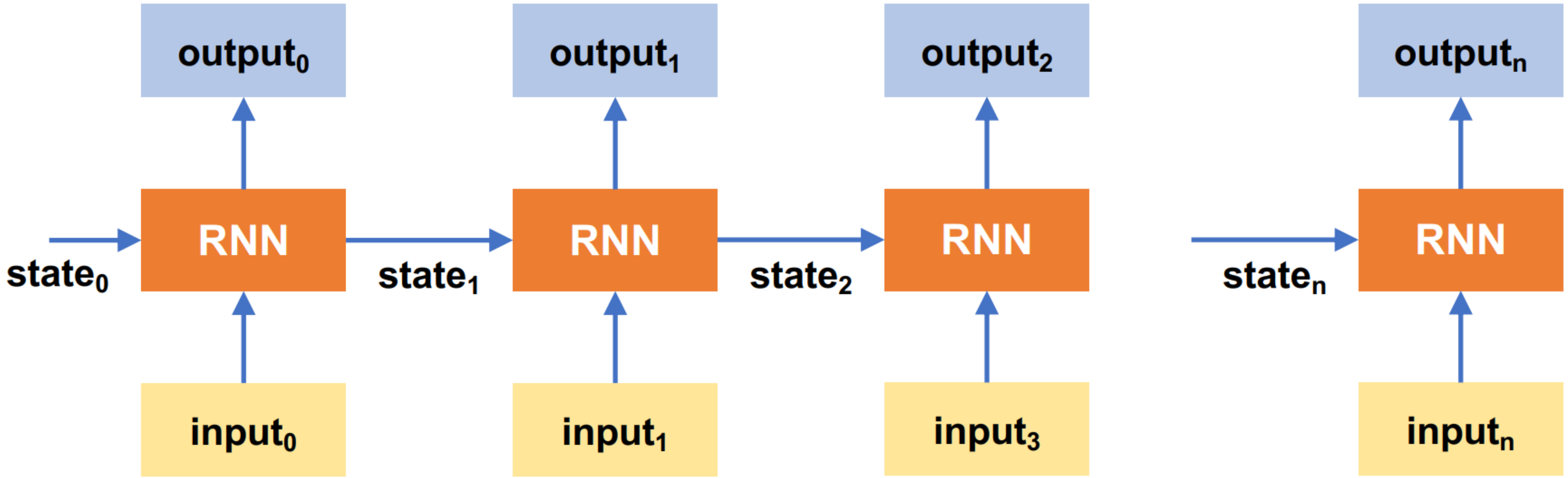
Arbitrary number of outputs

Key idea: RNNs have an internal state that is updated as a sequence is processed

Arbitrary number of inputs

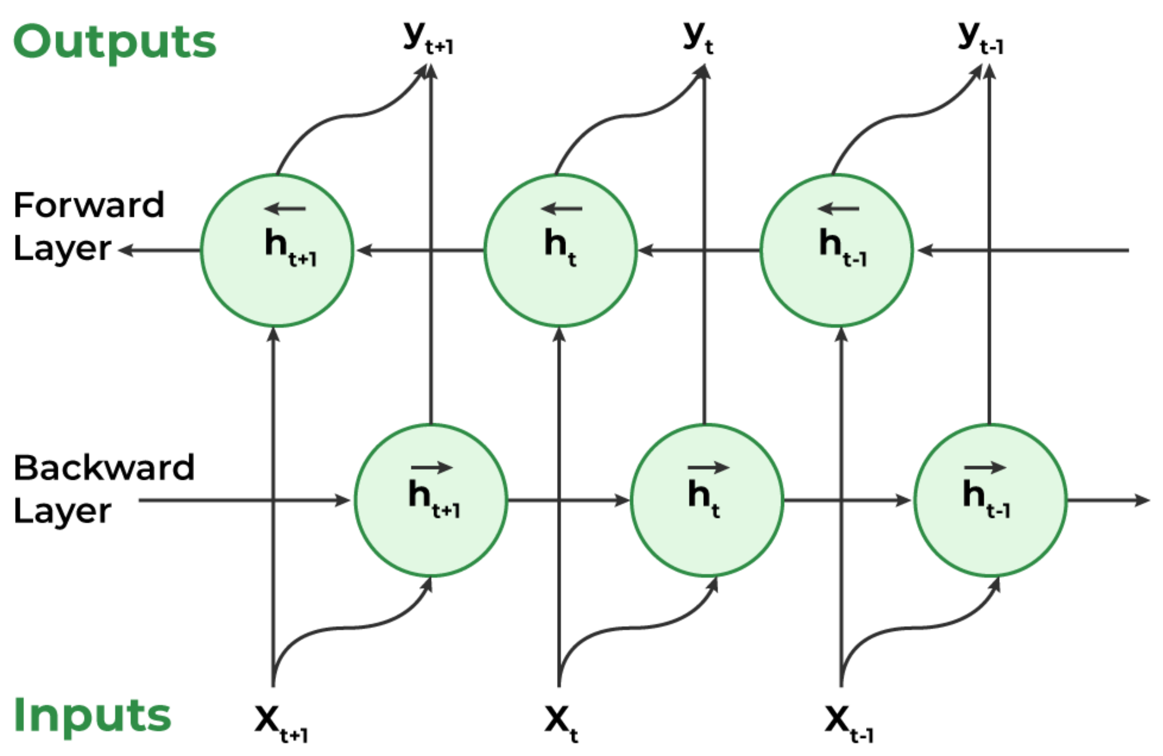
Recurrent Neural Networks: unrolling the computation

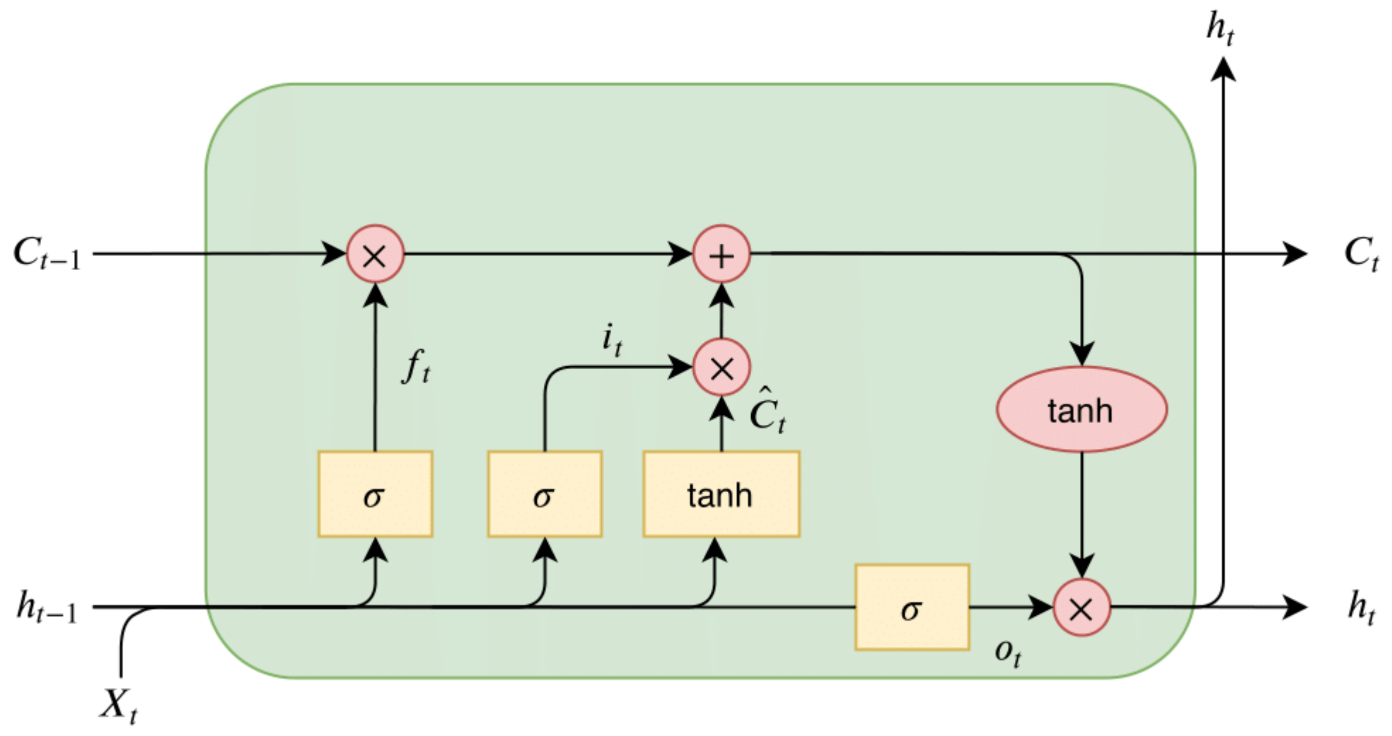
- One can make any NN recurrent

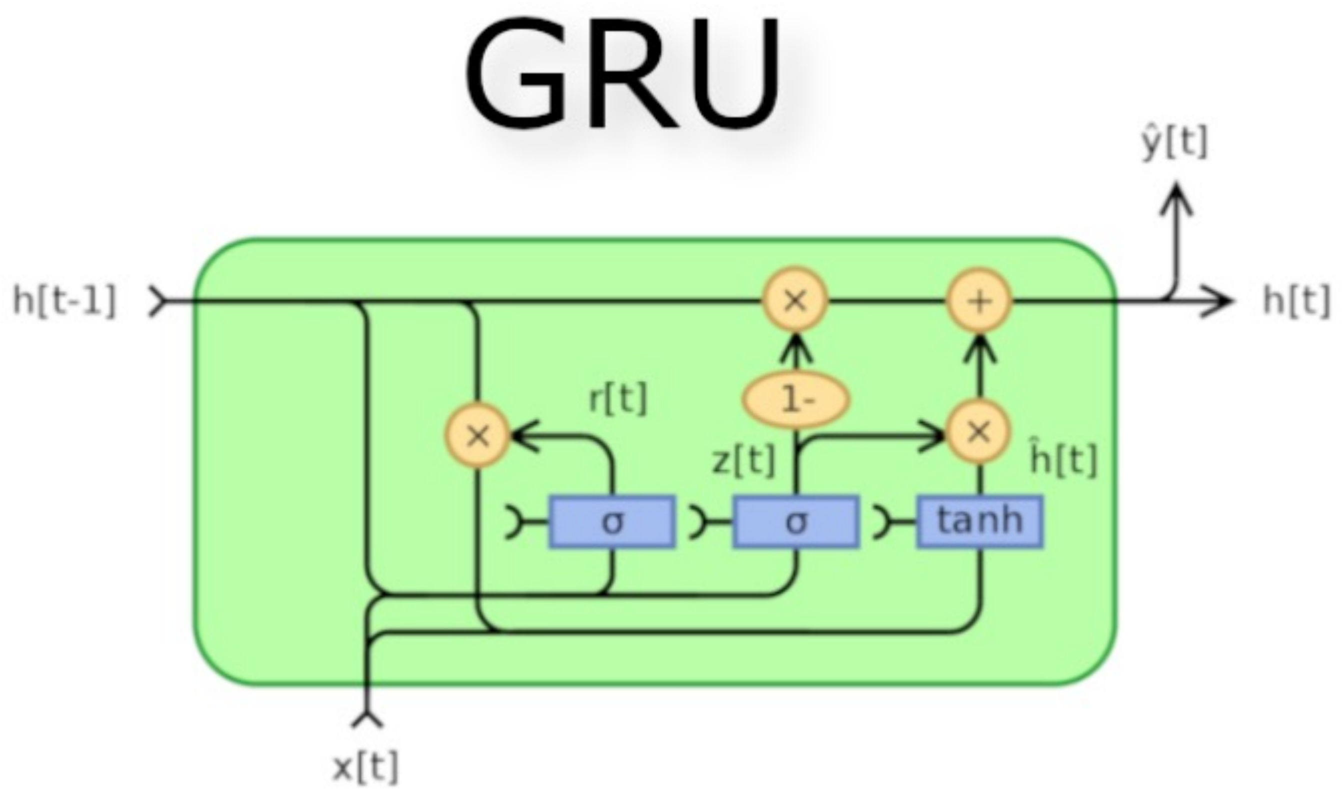


RNN: top3 models

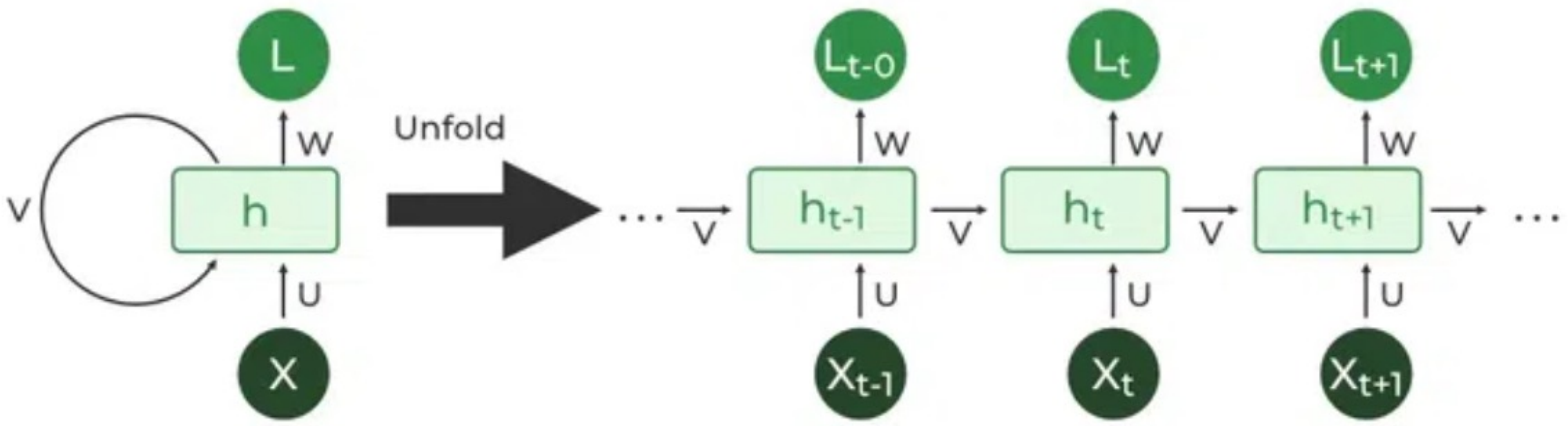
- Bidirectional RNNs
- LSTM
- GRU







Most Important Components in RNNs



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

MCQ Example (A difficult one)

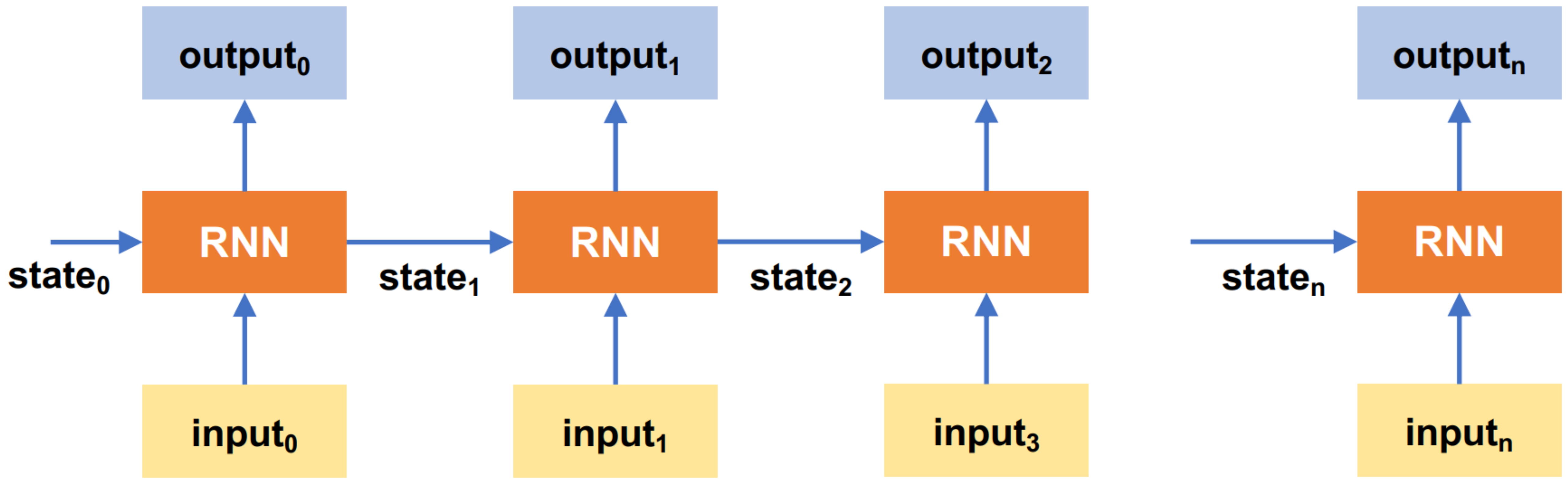
Who Invented LSTMs?

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.

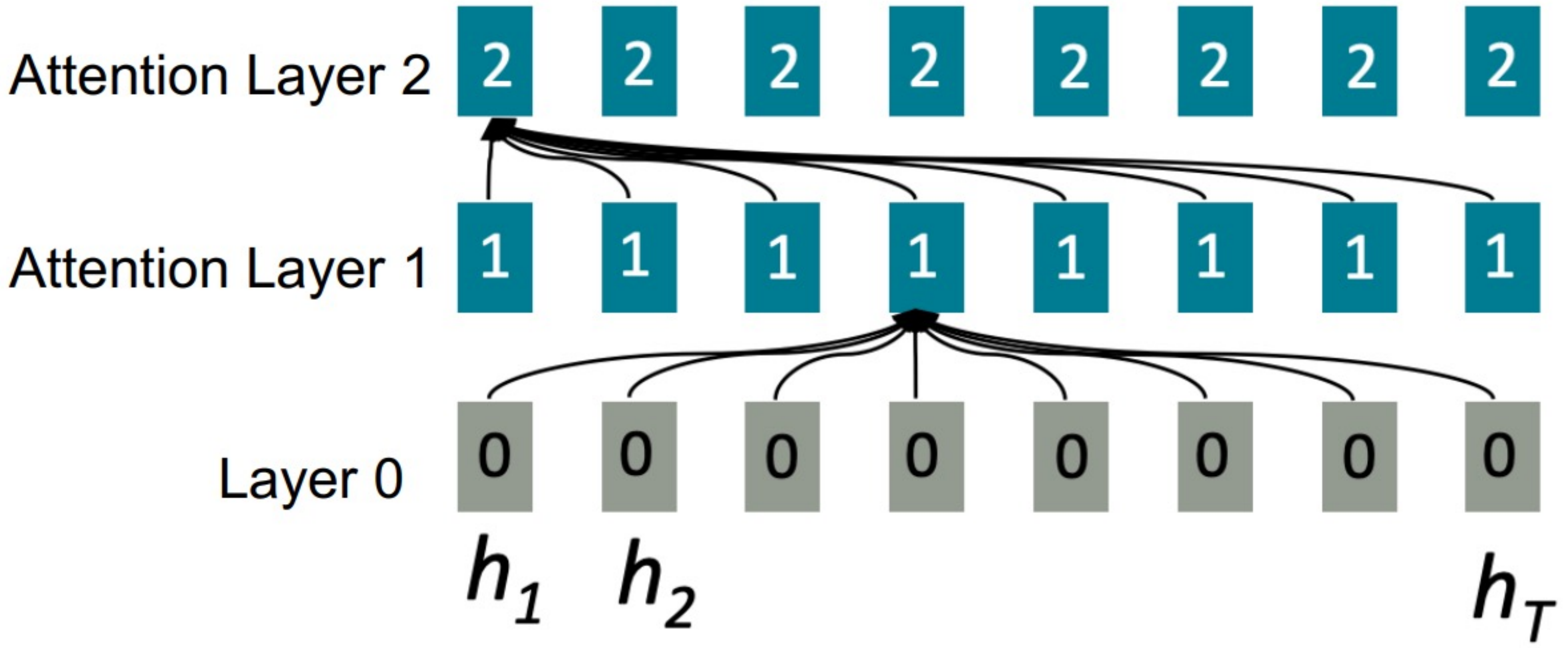
Why ChatGPT was not built using RNNs?

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



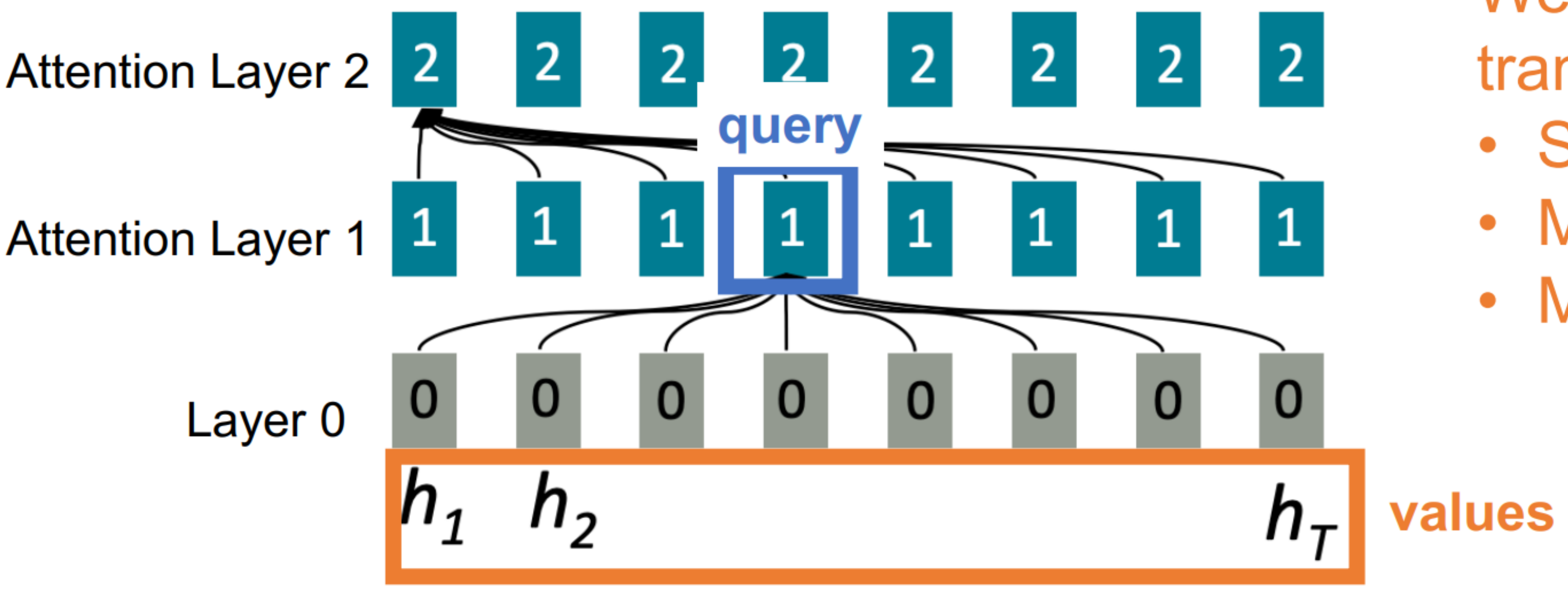
Attention: Enable parallelism

- 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



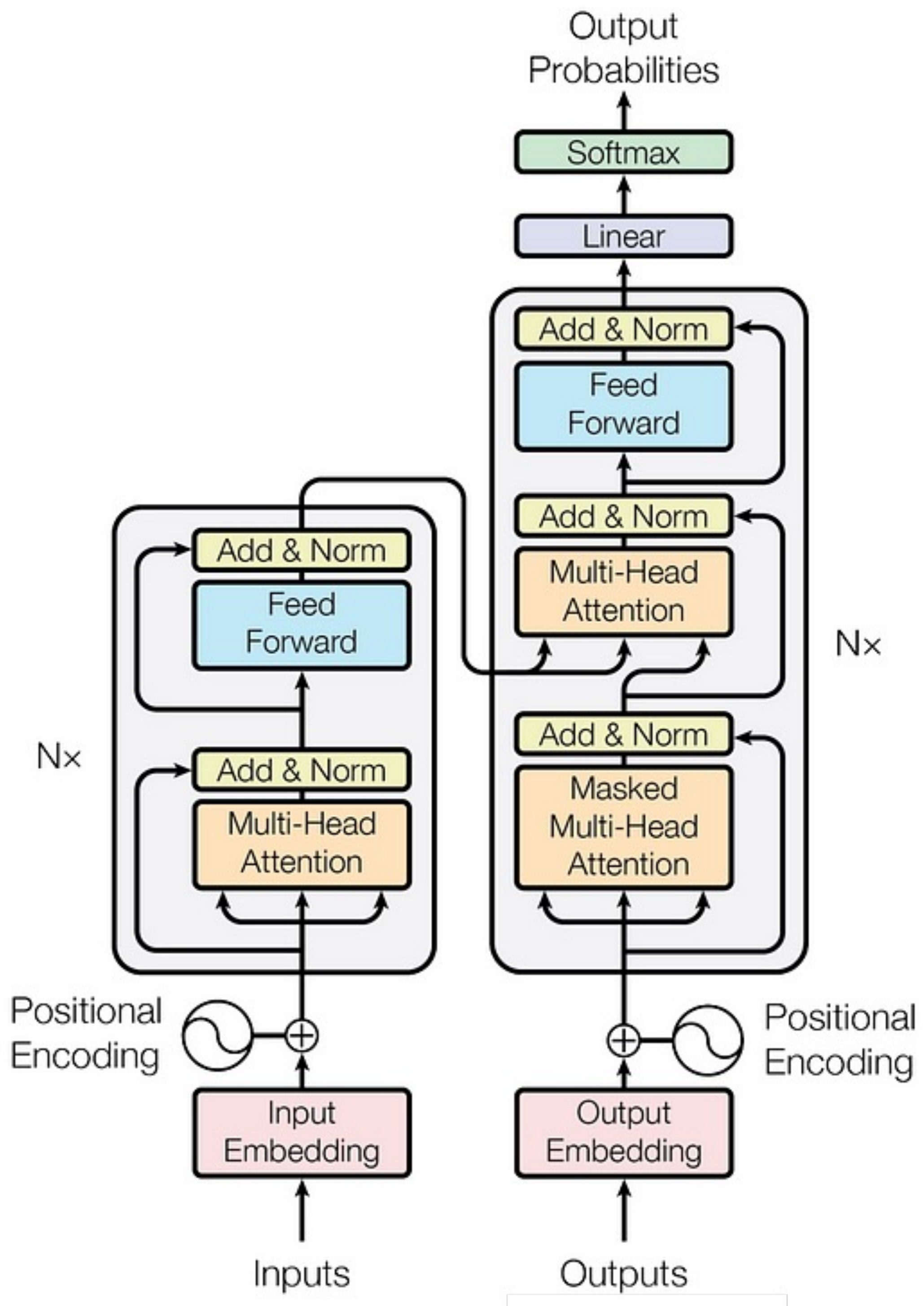
We will learn attention and transformers in depth later:

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

Transformers: Attention + MLP

BERT

Encoder

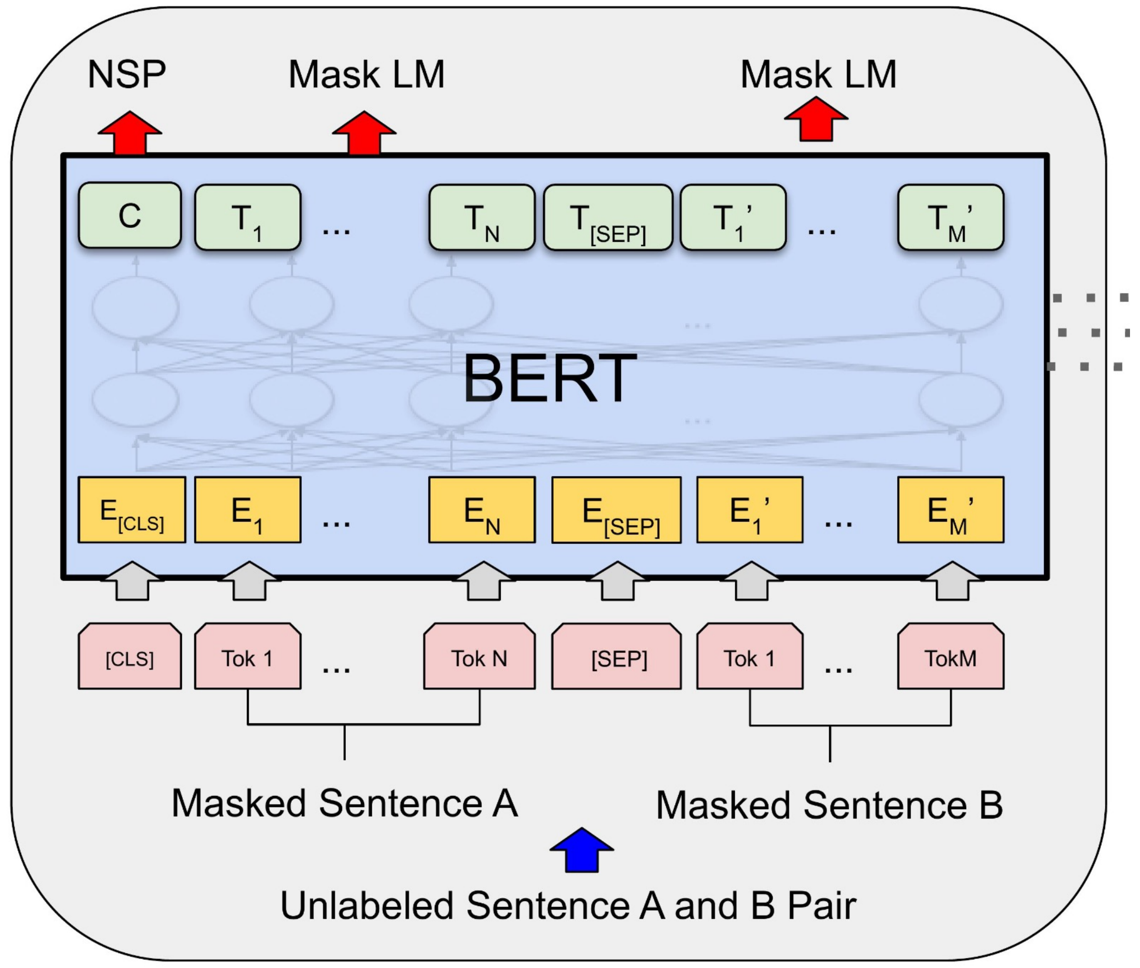


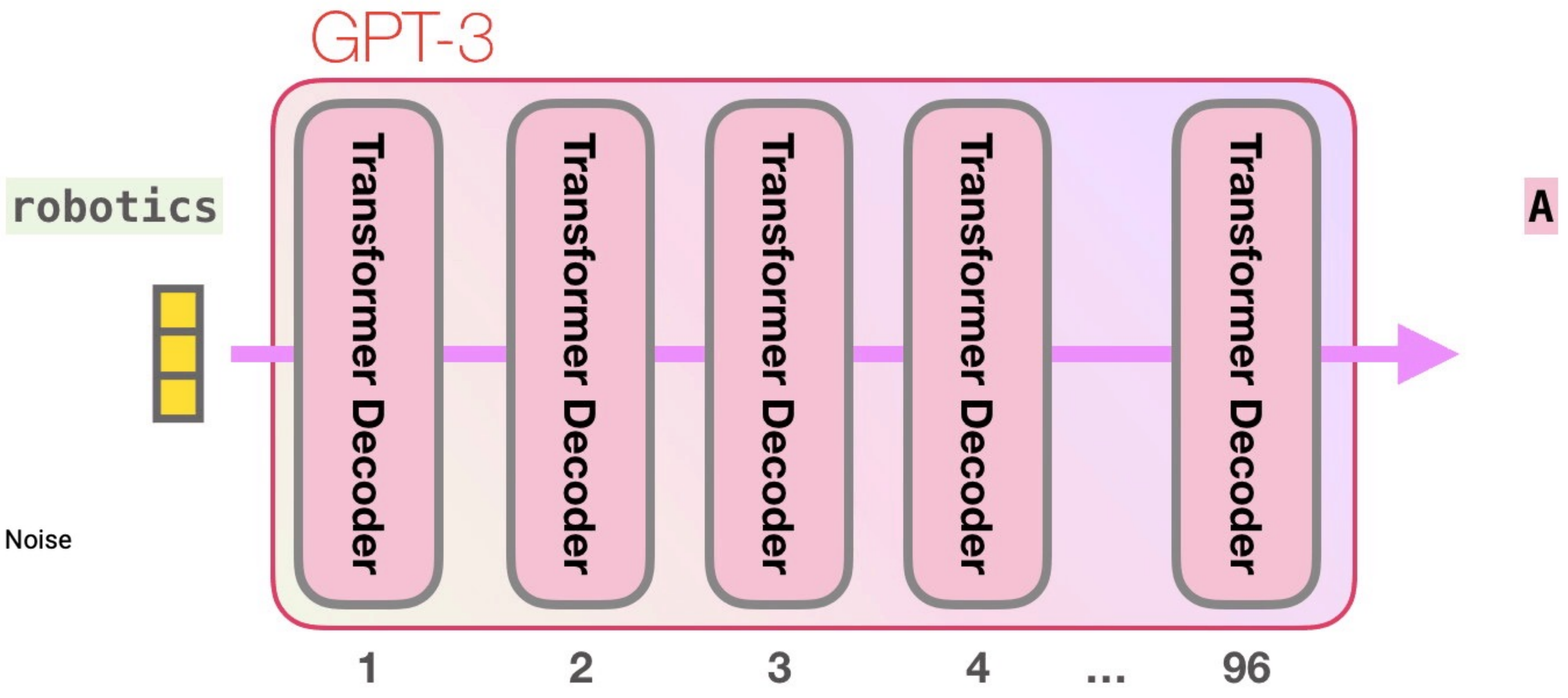
GPT

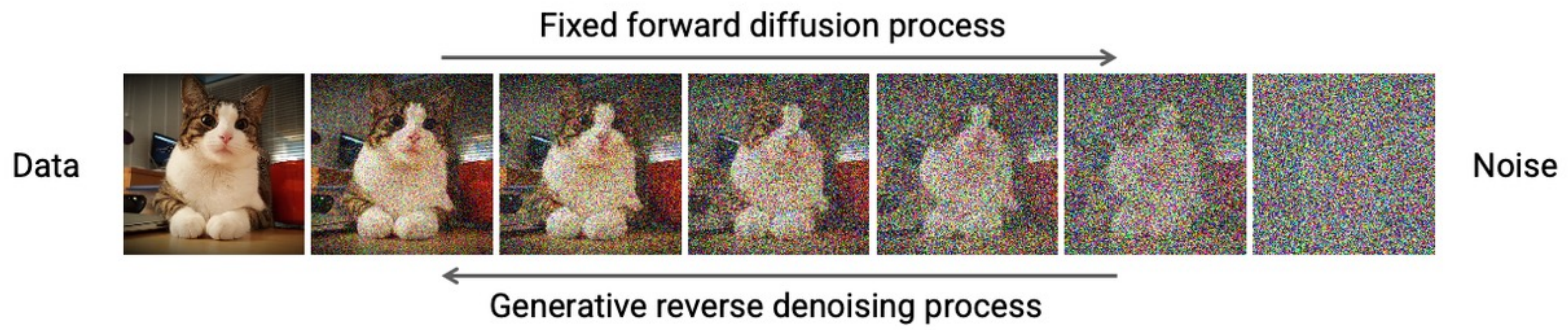
Decoder

Transformers: Top3 models?

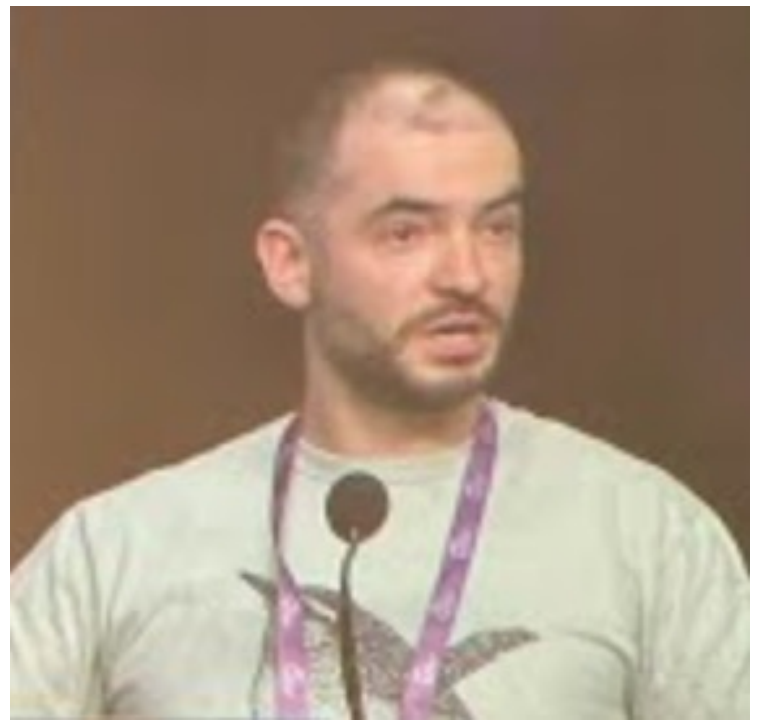
- Bert
- GPT/LLMs
- DiT: diffusion

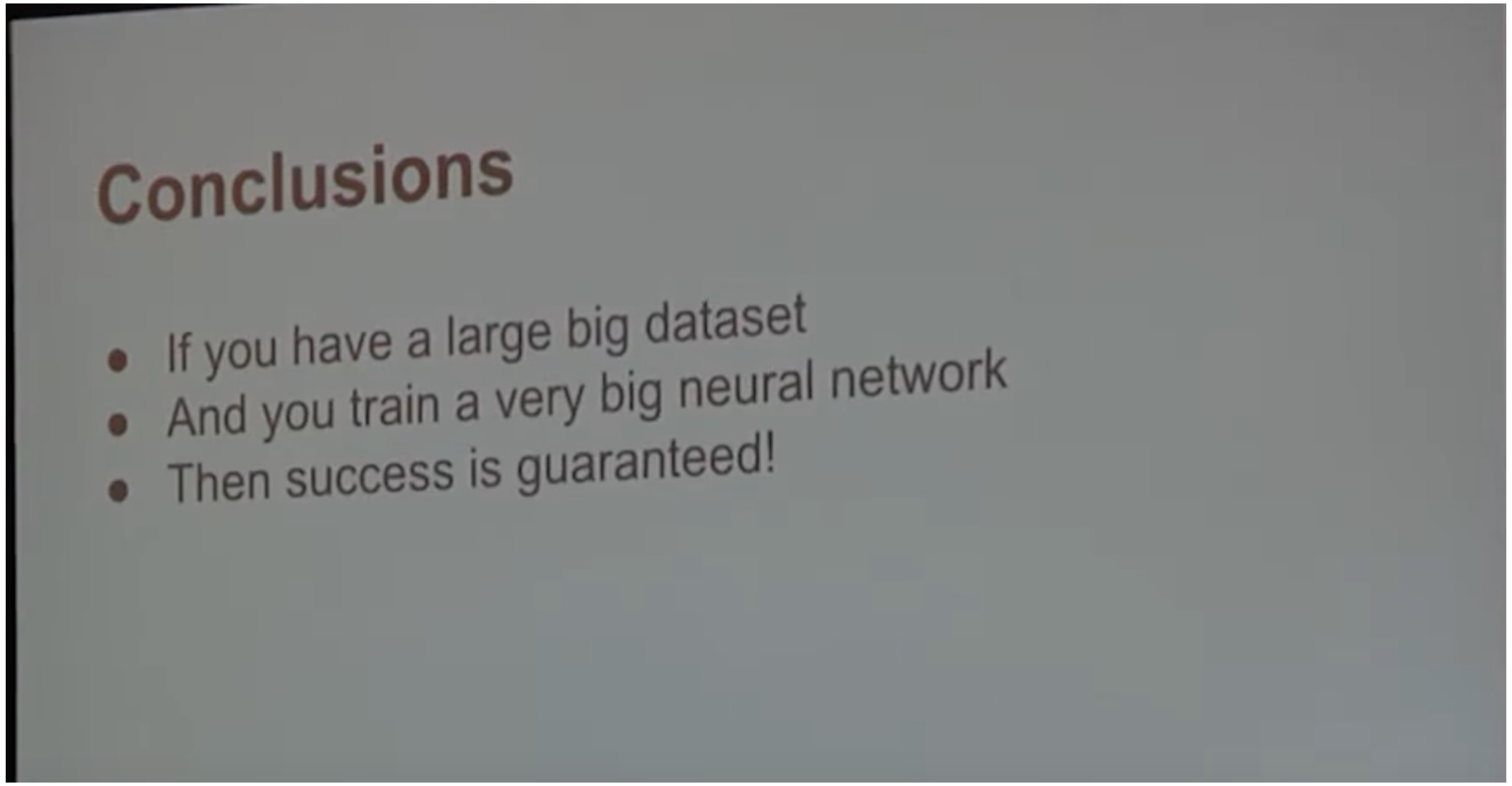






Transformer becomes the chosen one… (for a reason)





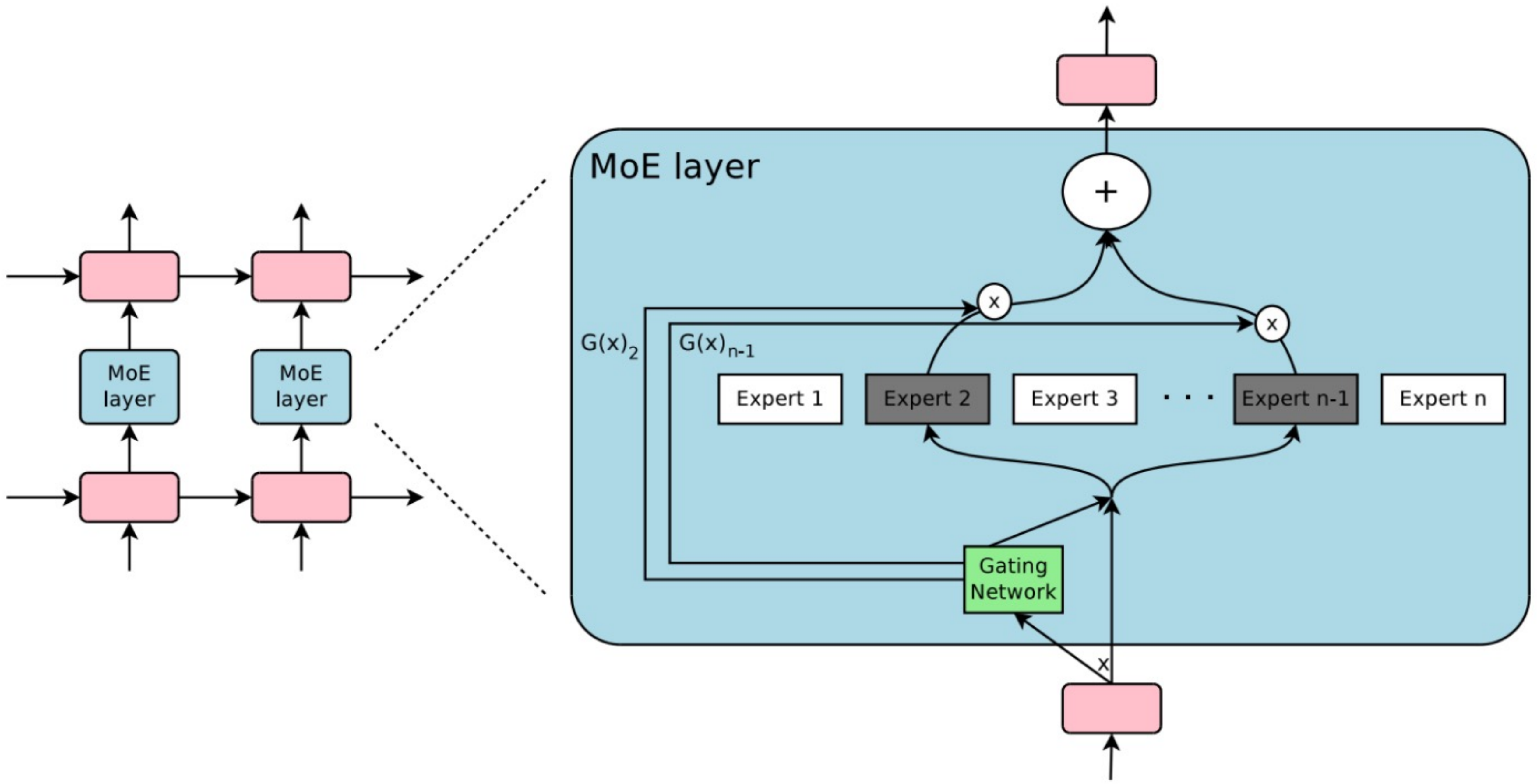
https://www.youtube.com/watch?v=9l9xGoyDhMM

Most Important Components in Transformers?

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

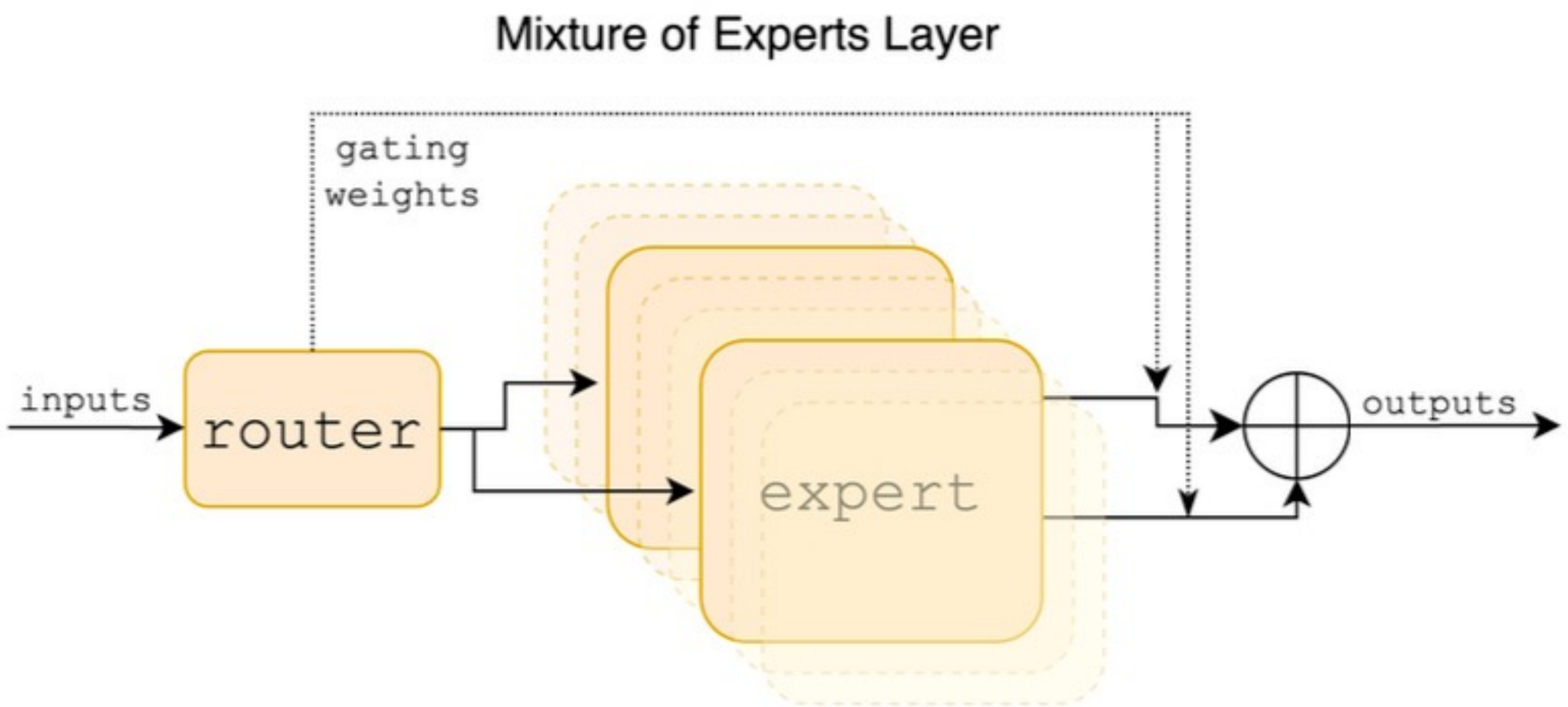
MoE: mixture of experts

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



Novel Component in MoE?

- 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 ~= Matmul Sys

High-level Picture

Data

✅ $\{x_i\}^n{}_{i=1}$

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

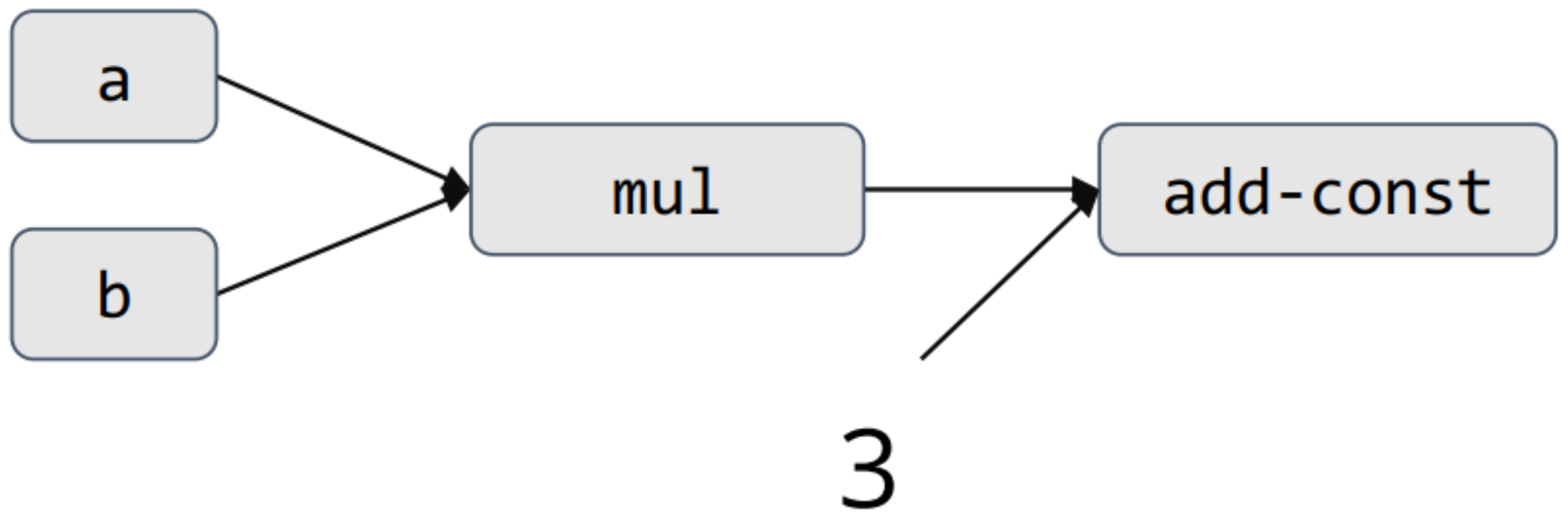
- Goal: we want to express as many as model as possible using one set of programming interface by connecting math primitives
- What constitutes a model from math primitives?
  - Model and architecture: connecting math primitives
  - Objective function
  - Optimizer
  - Data

Discussion: how we express computation in history Applications <-> System Design

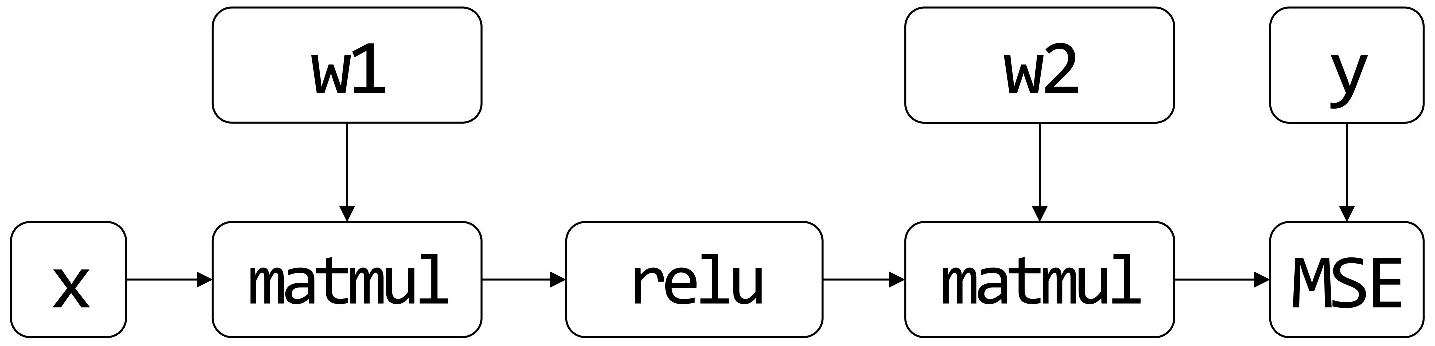
Application	Data management (OLTP)	Big data processing (OLAP)
Systems	SQL Query planner Relational database Storage	Spark/mapreduce Dataflow, lineage Data warehousing Column storage

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 x b + 3



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

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

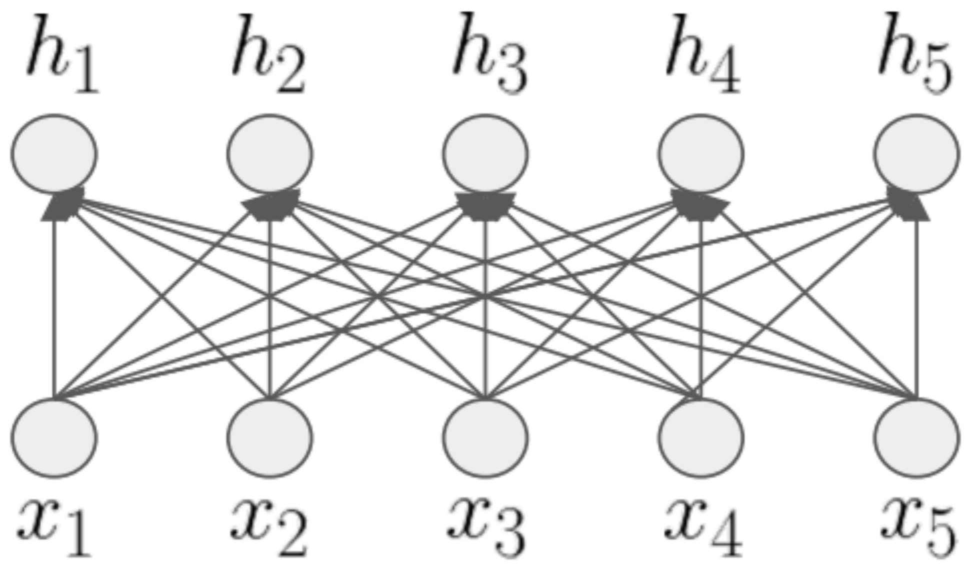
One linear NN: Logistic Regression

Input

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

One Linear Layer

$$h_k = w_k^T x_i$$



Softmax

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

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

```python
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

```python
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

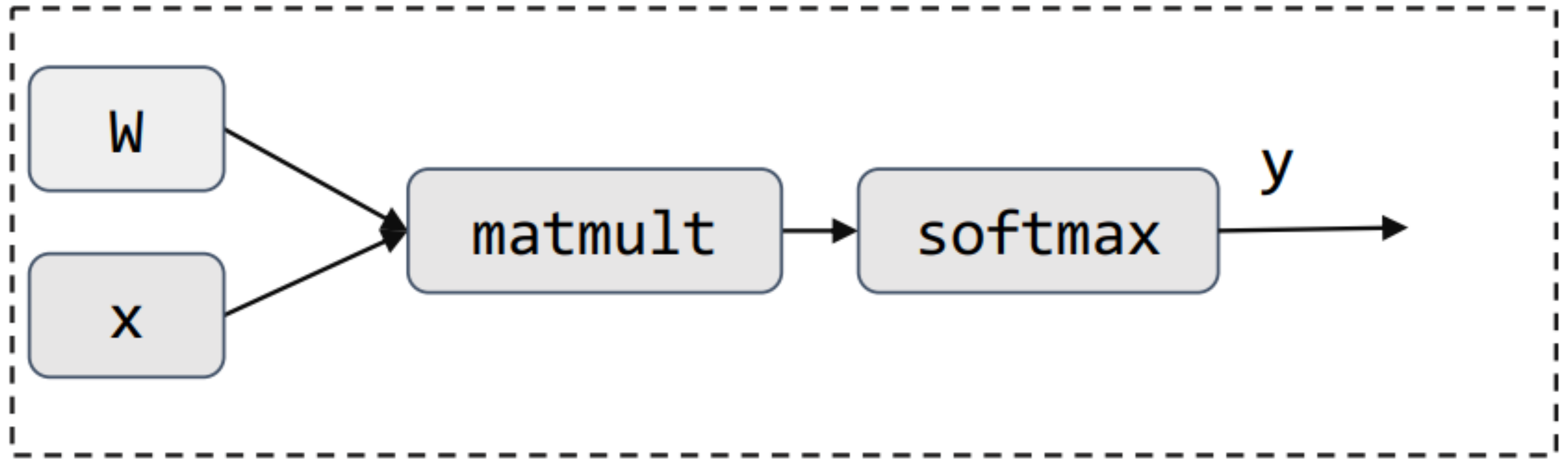
Trigger the Execution

```
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!

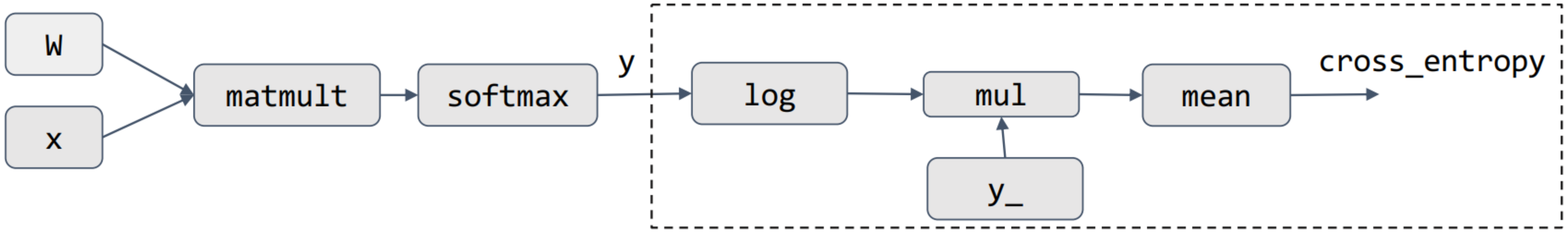
What happens behind the Scene

```
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.)

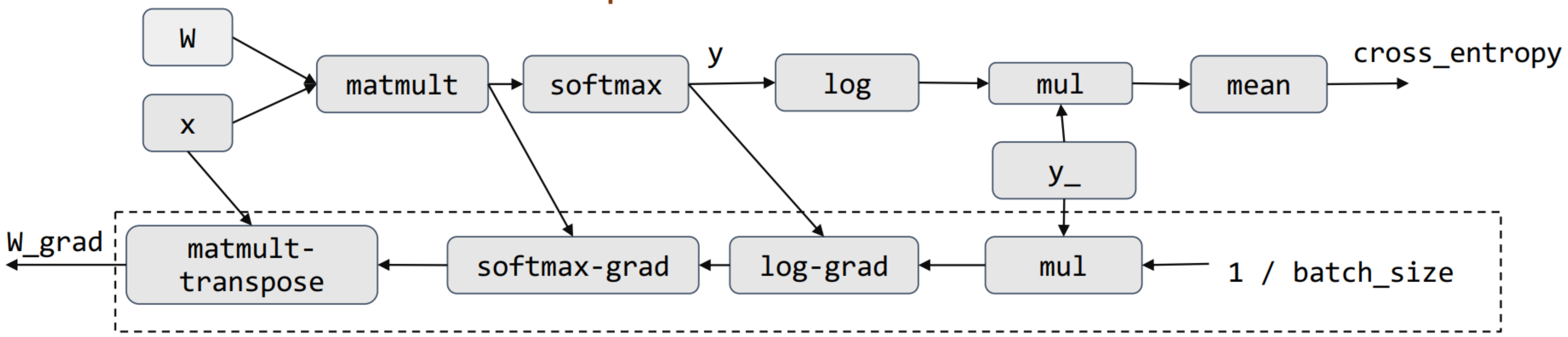
```
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
```



What happens behind the Scene (Cond.)

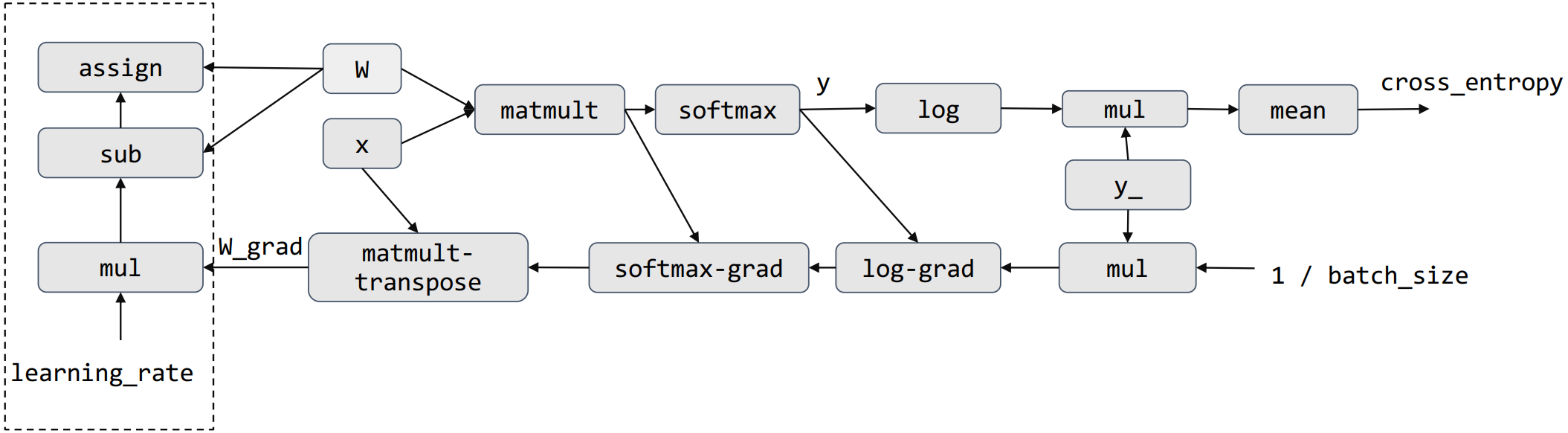
```
W_grad = tf.gradients(cross_entropy, [W])[0]
```

Automatic Differentiation, more details in follow up lectures



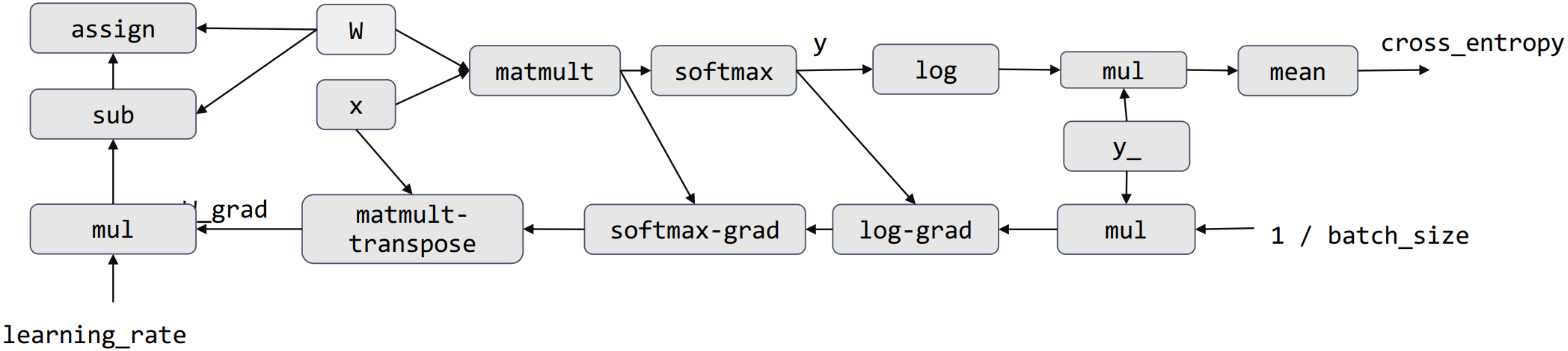
What happens behind the Scene (Cond.)

```
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

```python
# 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

```python
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

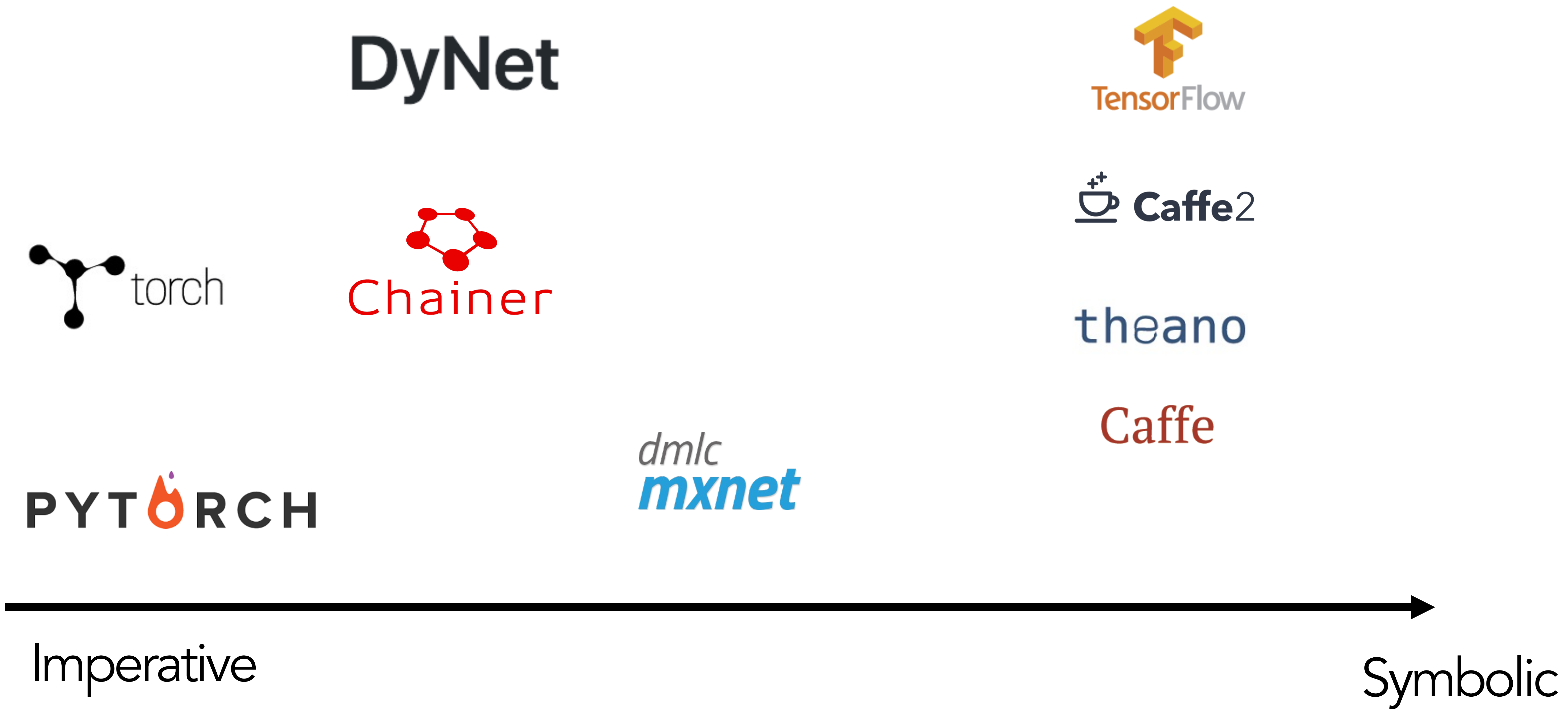
- Which category, symbolic vs. imperative, is the following PL belonging to?
  - C++
  - Python
  - SQL

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

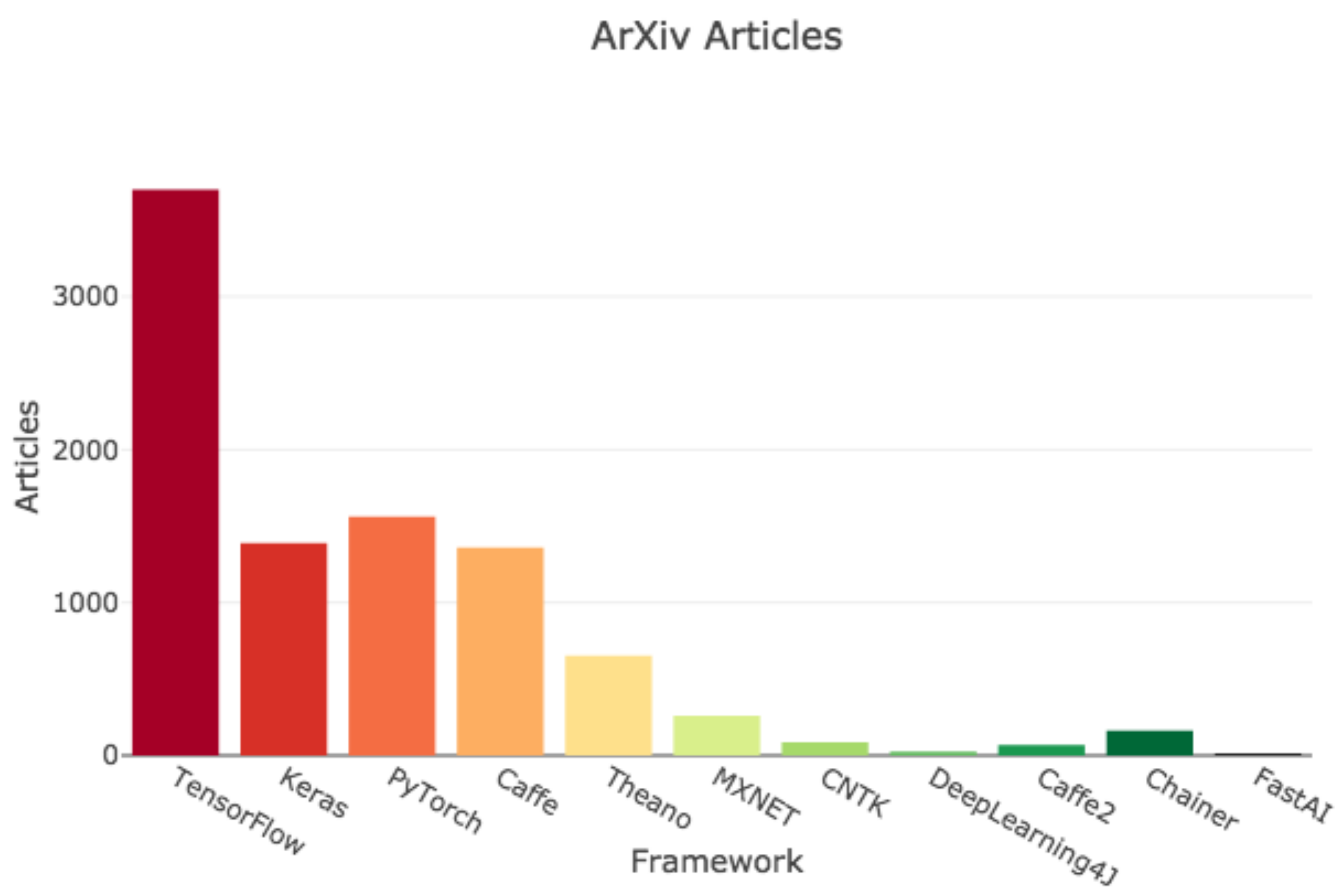
- You are indeed using a DSL built on top of Python
  - But PyTorch DSL is more *pythonic* than Tensorflow DSL.

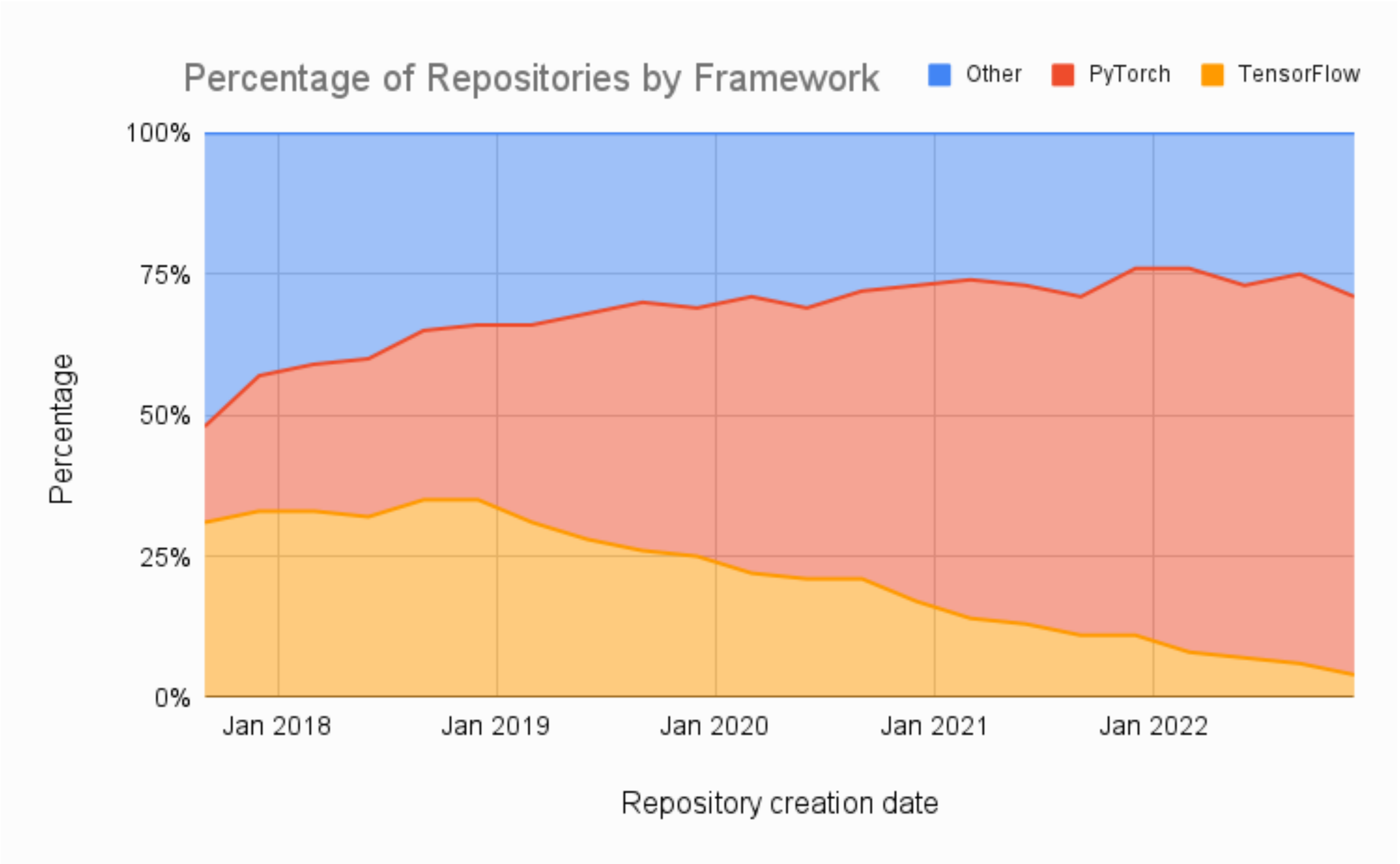
Symbolic vs. Imperative (2016)

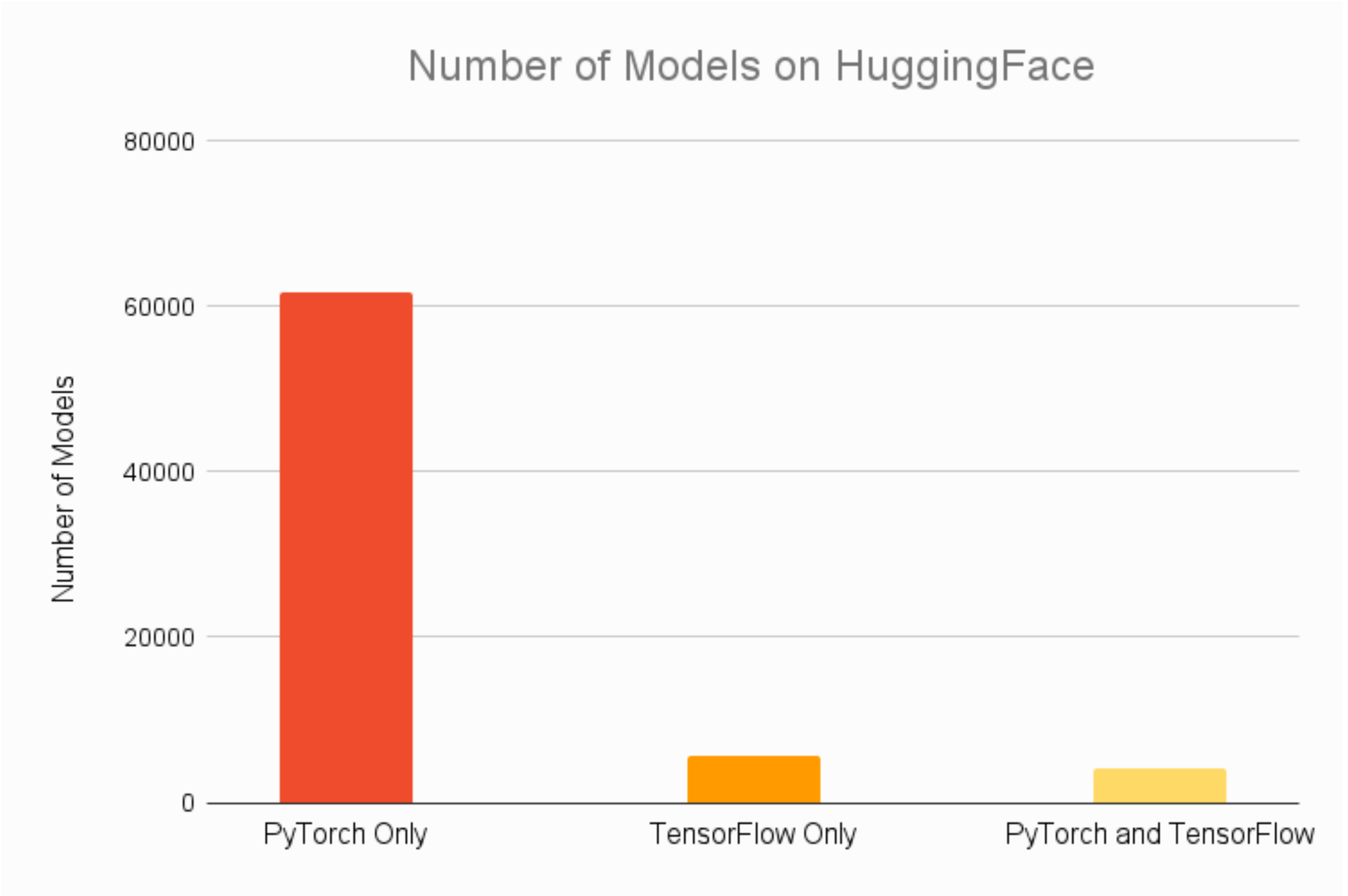


Symbolic vs. Imperative (2024)

Market size of frameworks



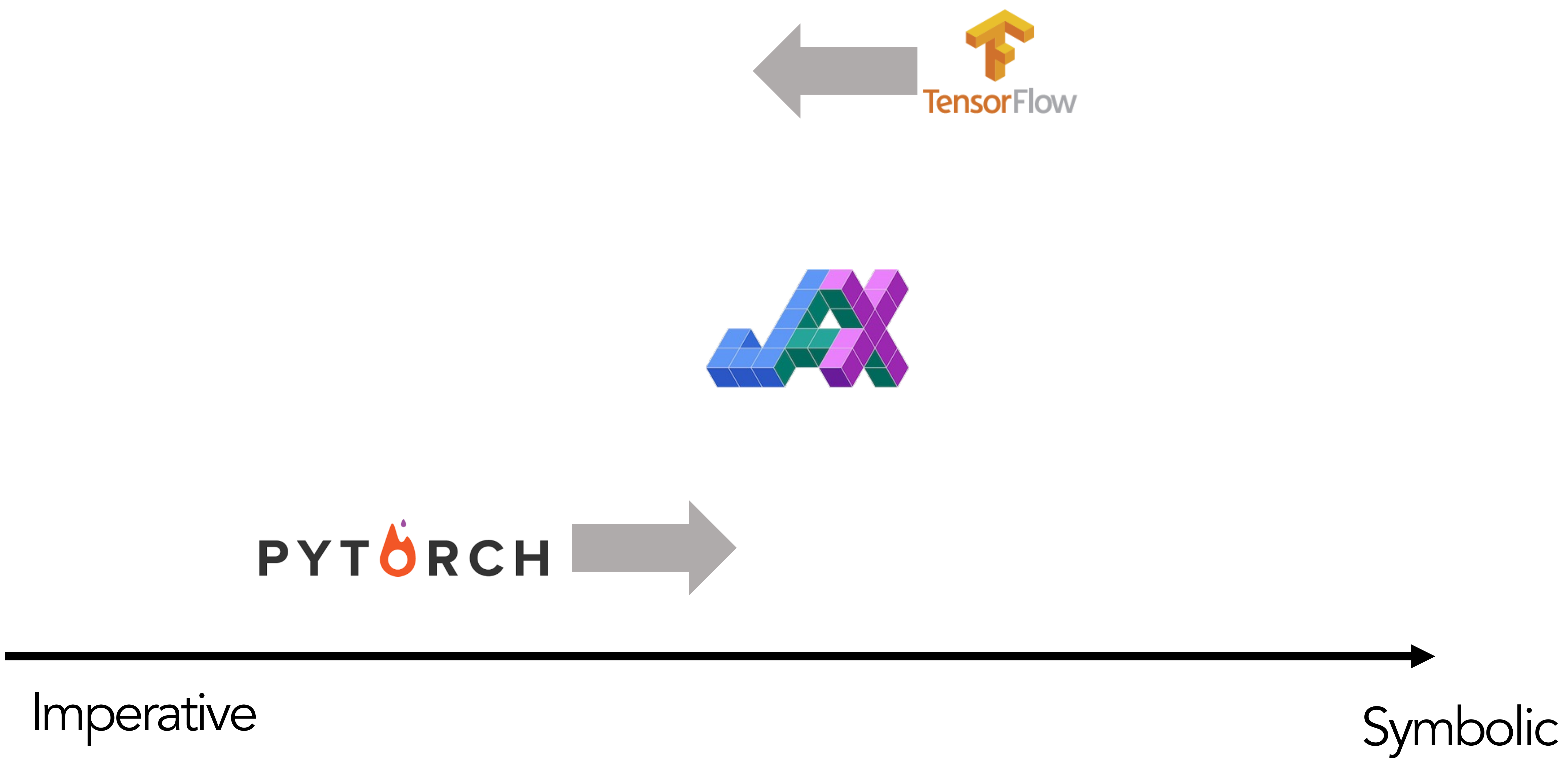




After-class Question

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?

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

Dev mode

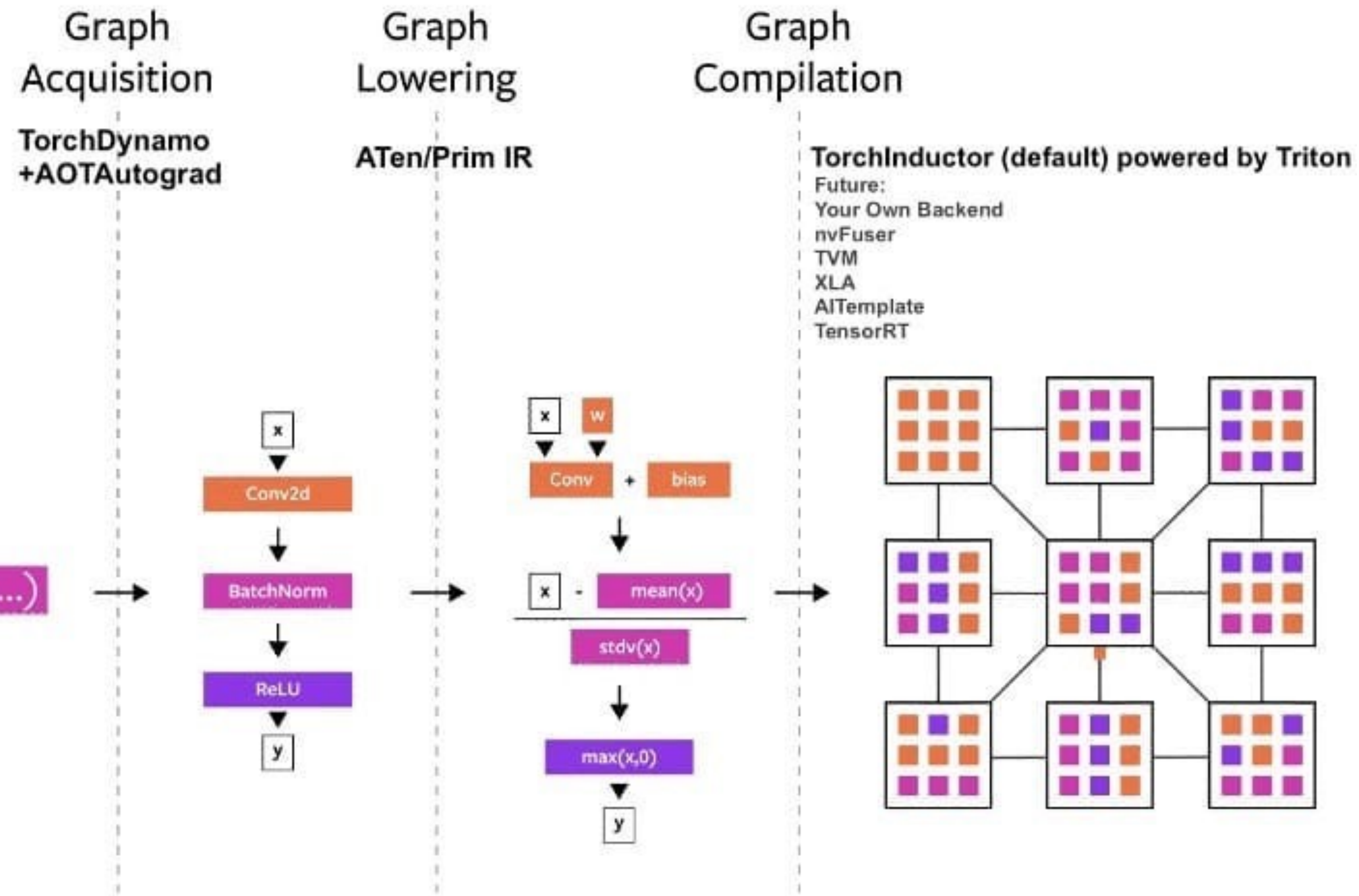
@torch.compile()

```python
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

```python
@torch.compile()
def foo(x):
    y = F.conv2d(x, ...)
    z = F.batch_norm2d(y, ...)
    return F.relu(z)
```



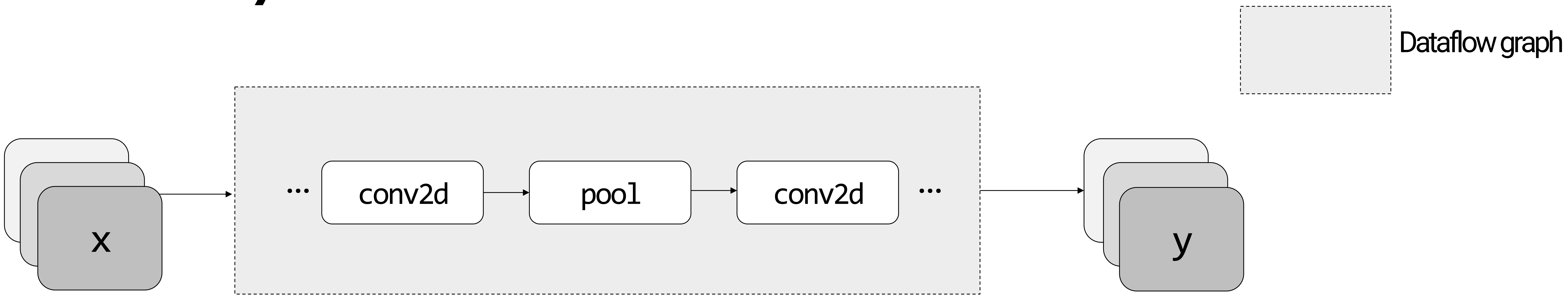
What is the problem of JIT?

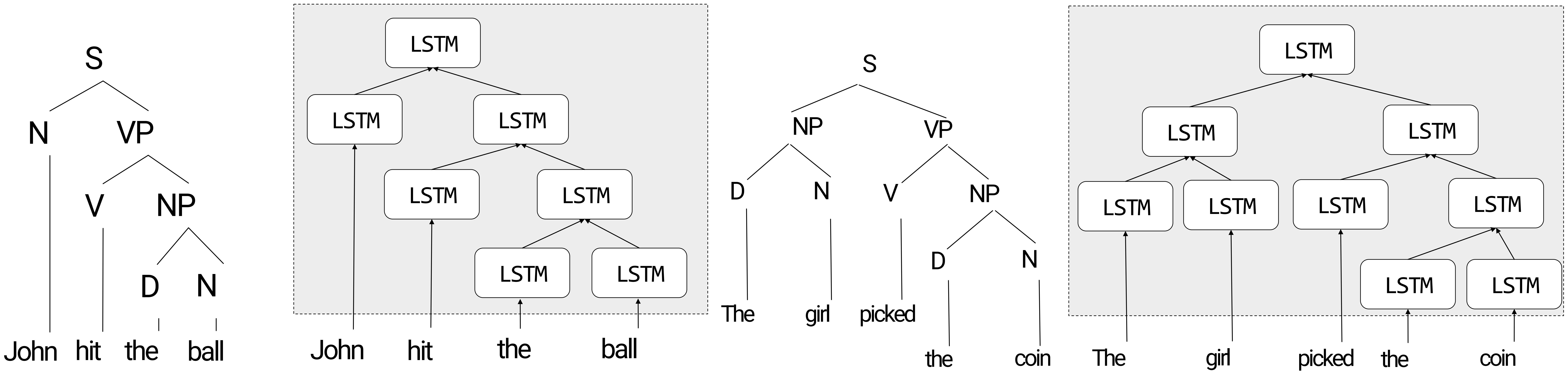
Requirements for static graphs

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

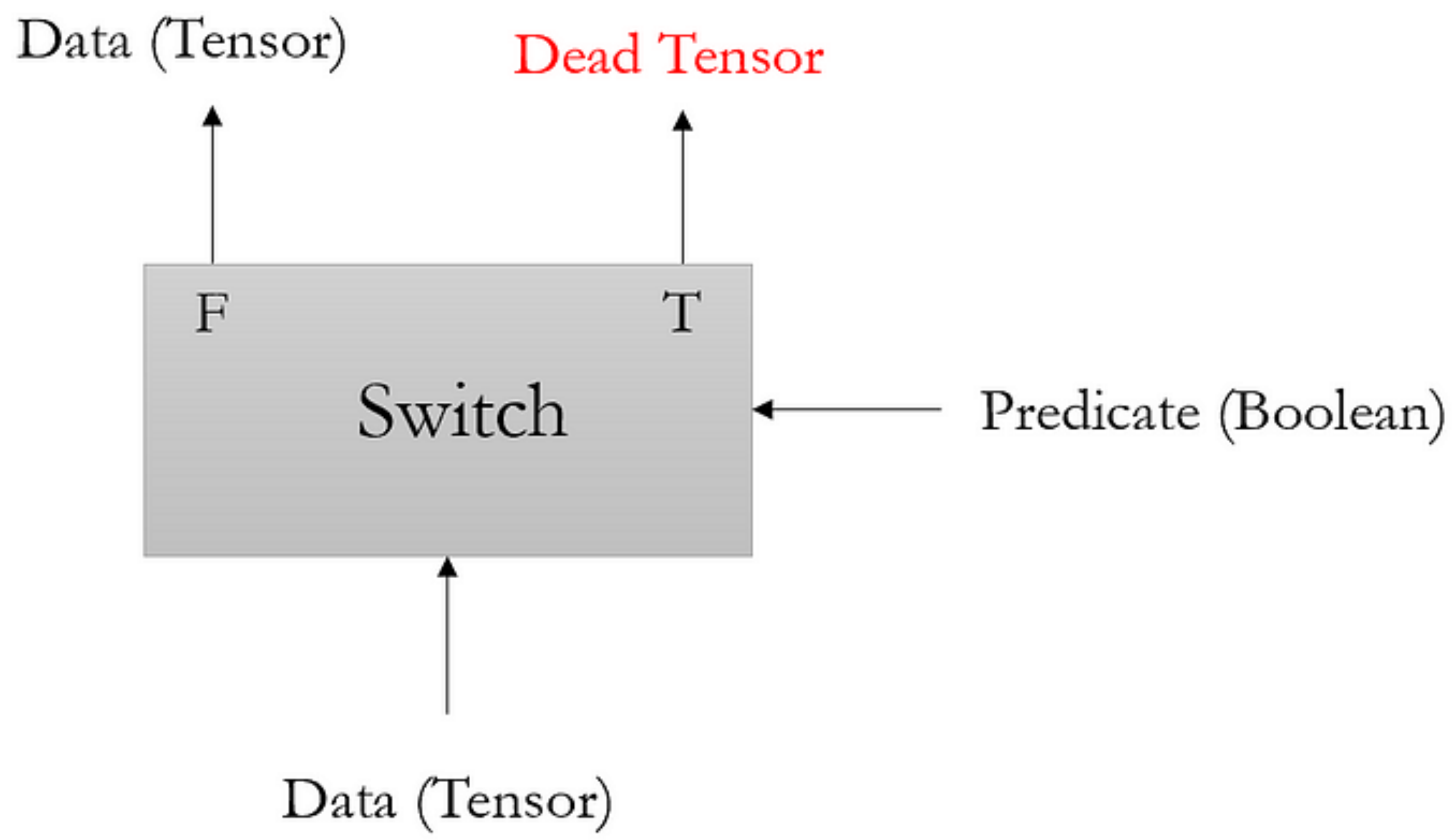
Open Research: How to Handle Dynamics?

Three ways:

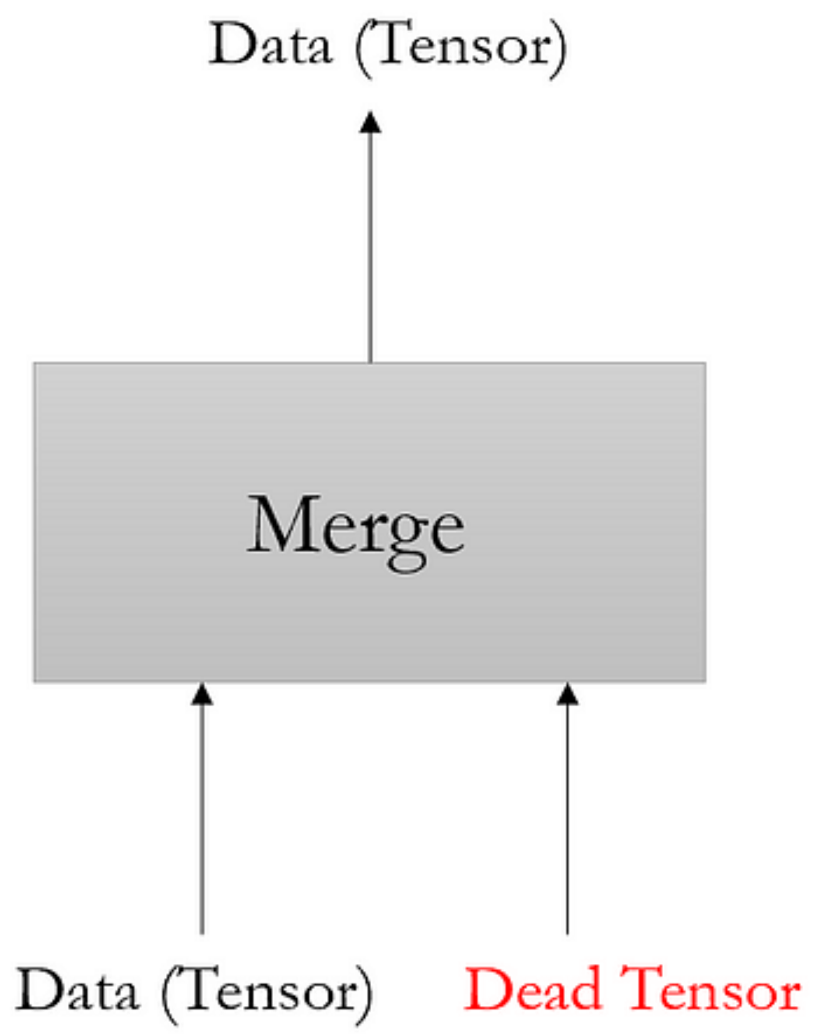
- Just do Define-and-run and forget about JIT
  - As long as you do not care about performance…
- Introduce Control flow Ops
- Piecewise compilation and guards

Control flow primitives

- Example primitive: Switch and Merge



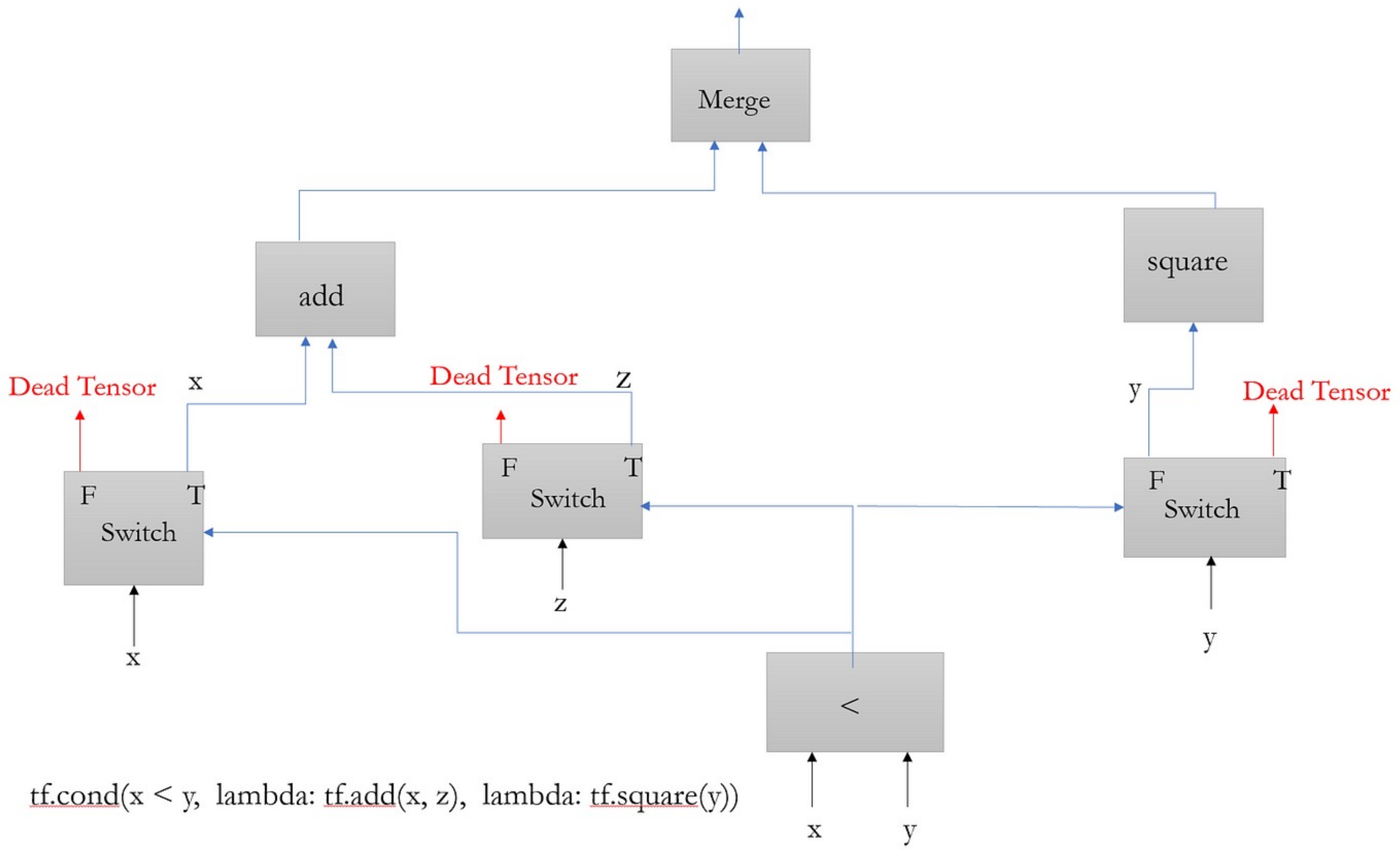
Switch receives two arguments: Data and predicate (boolean), and has two outputs: data and dead Tensor



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

Model

Compute

✅ $\{x_i\}^n{}_{i=1}$

✅ Math primitives (mostly matmul)

❓ A repr that expresses the computation using primitives

❓ 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