DLVM

DLVM

Compiler Framework for Deep Learning DSLs

DLVM

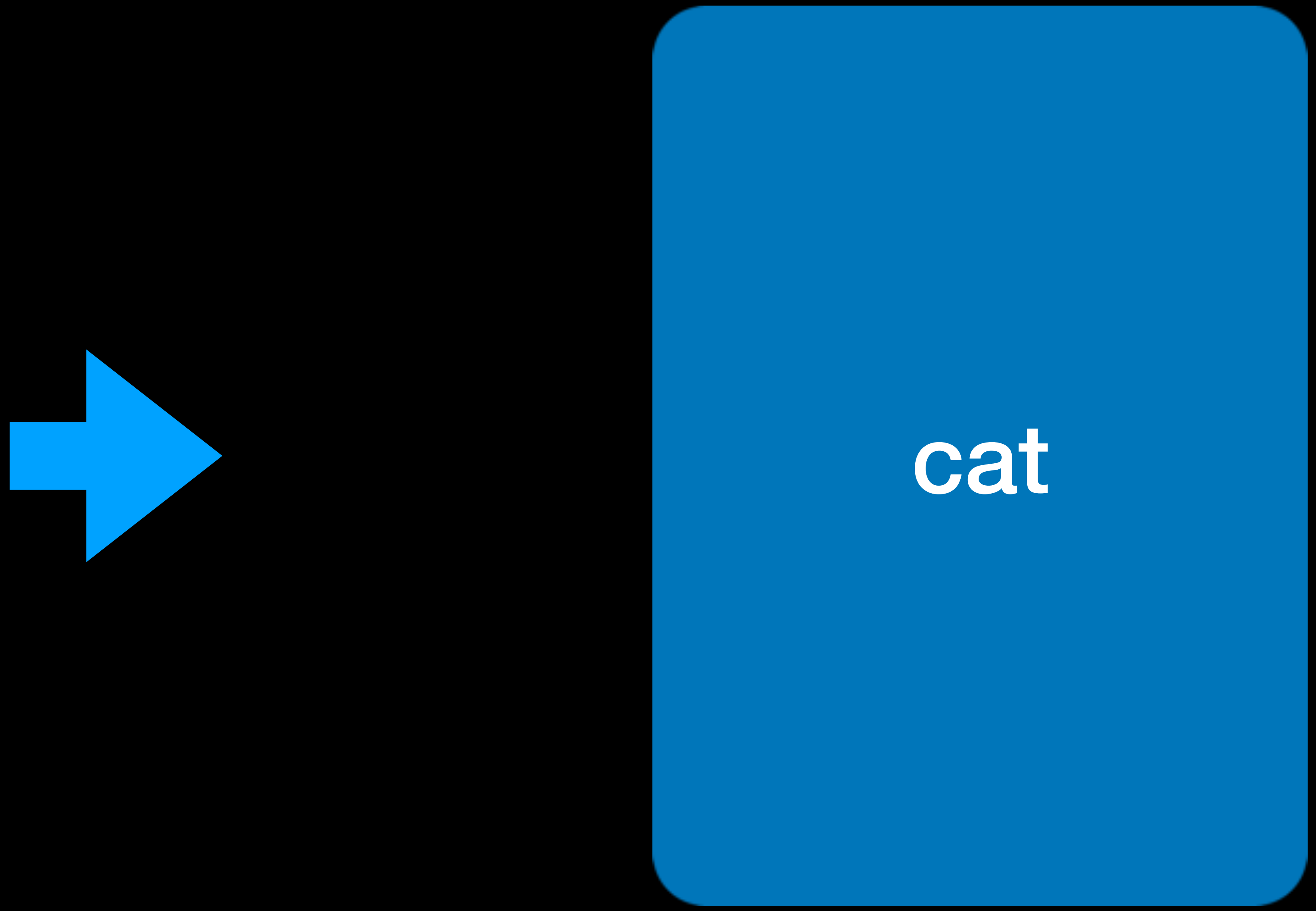
Compiler Framework for Deep Learning DSLs

Richard Wei **Vikram Adve** **Lane Schwartz**
University of Illinois at Urbana-Champaign

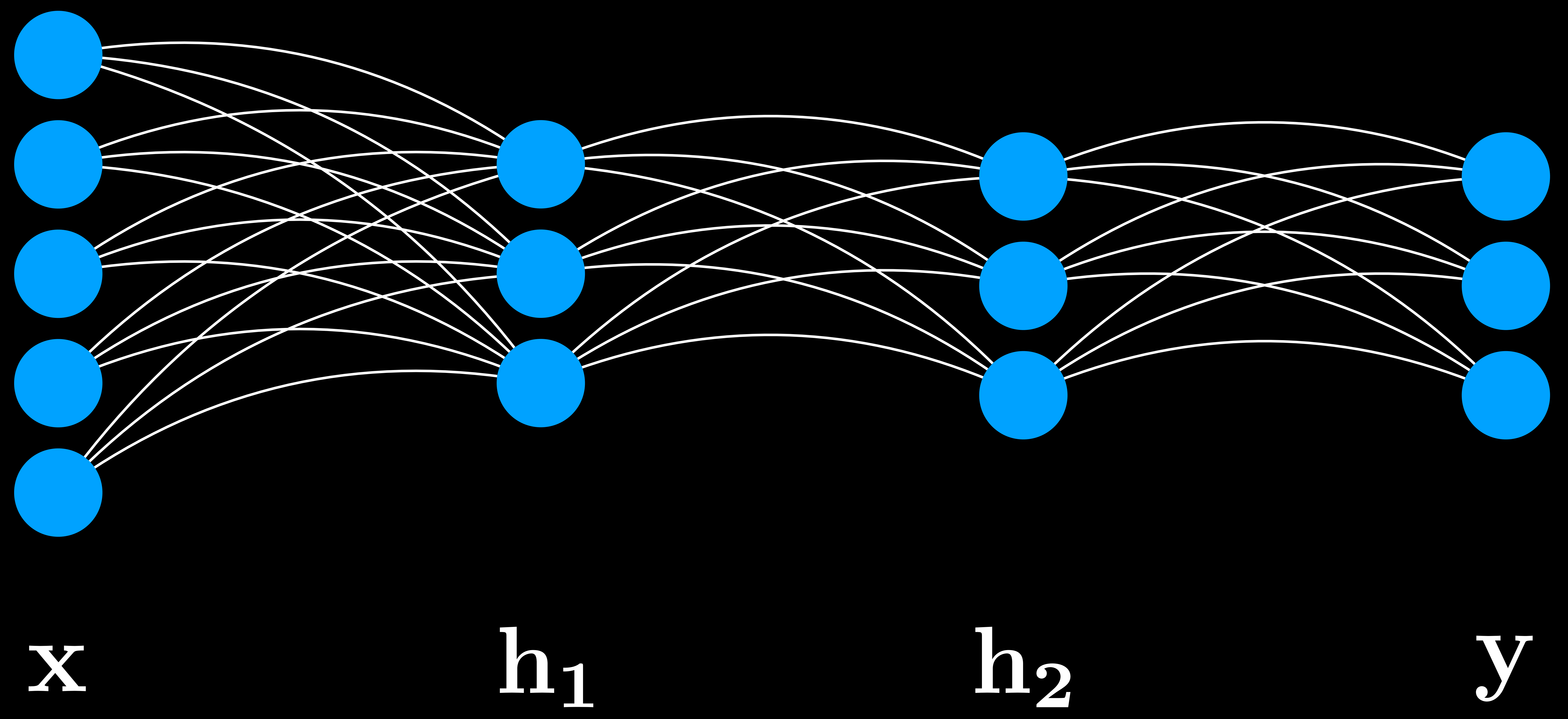
Deep Learning

Deep Learning

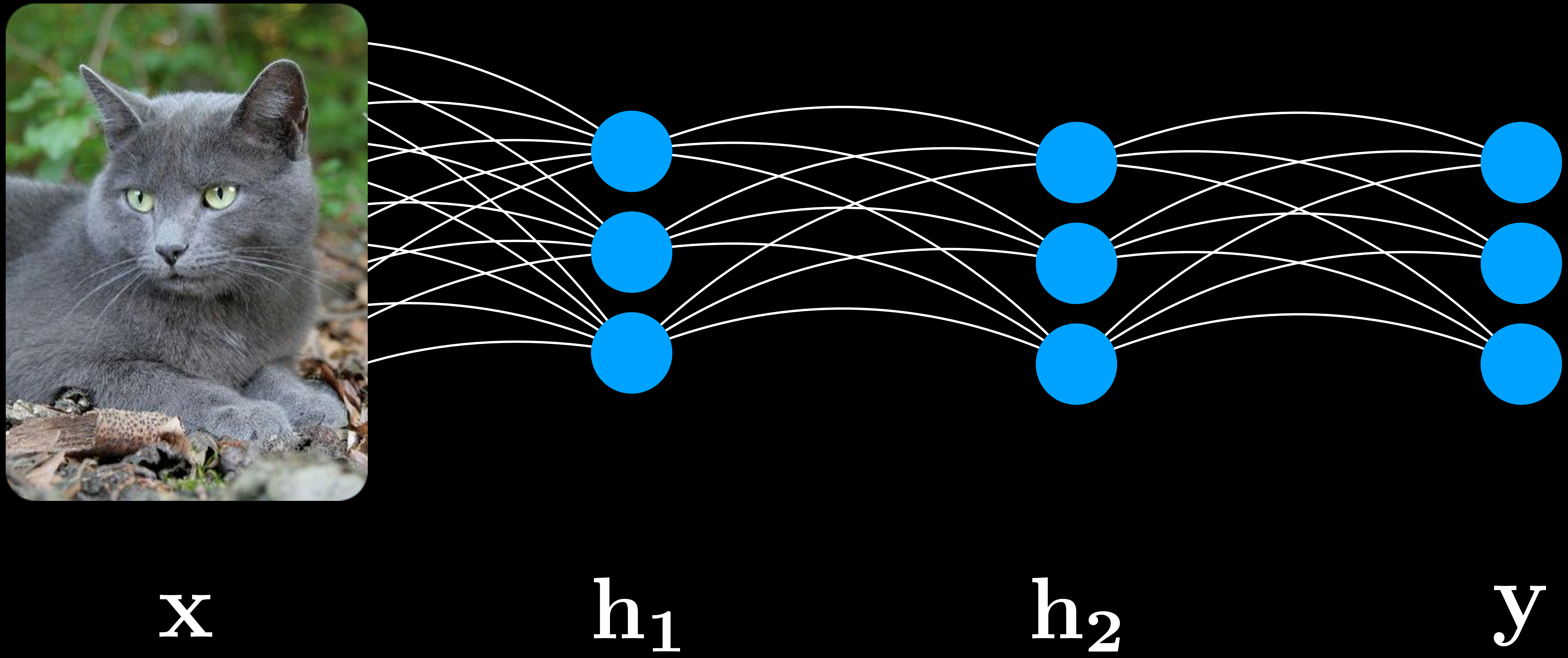
Deep Learning



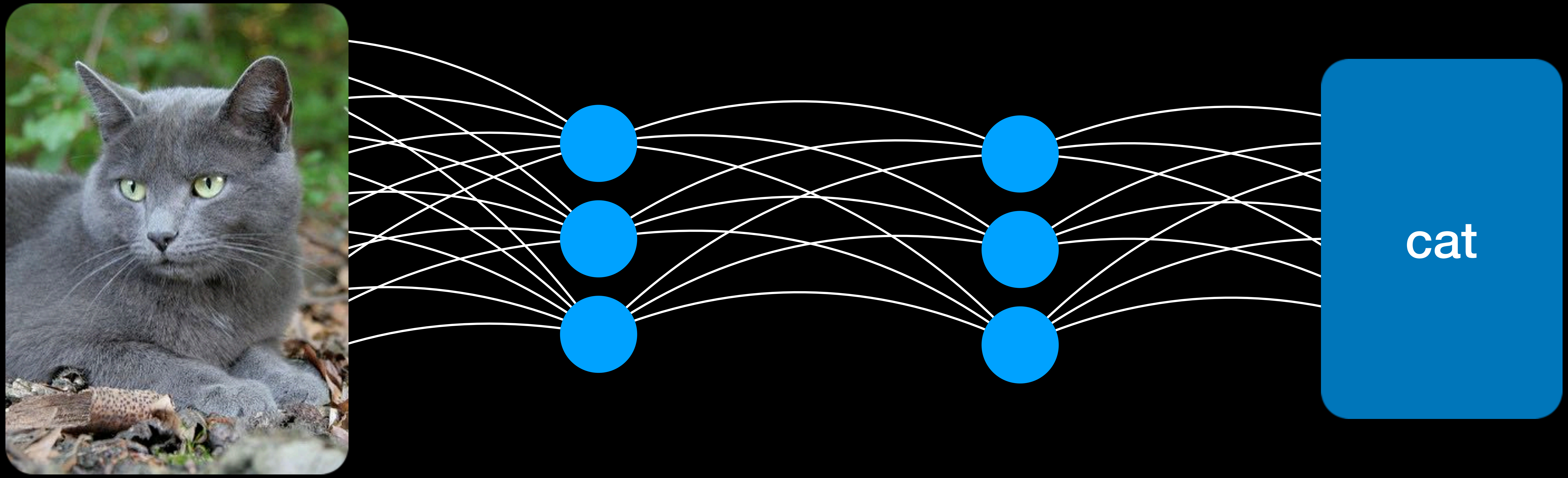
Neural Network



Neural Network

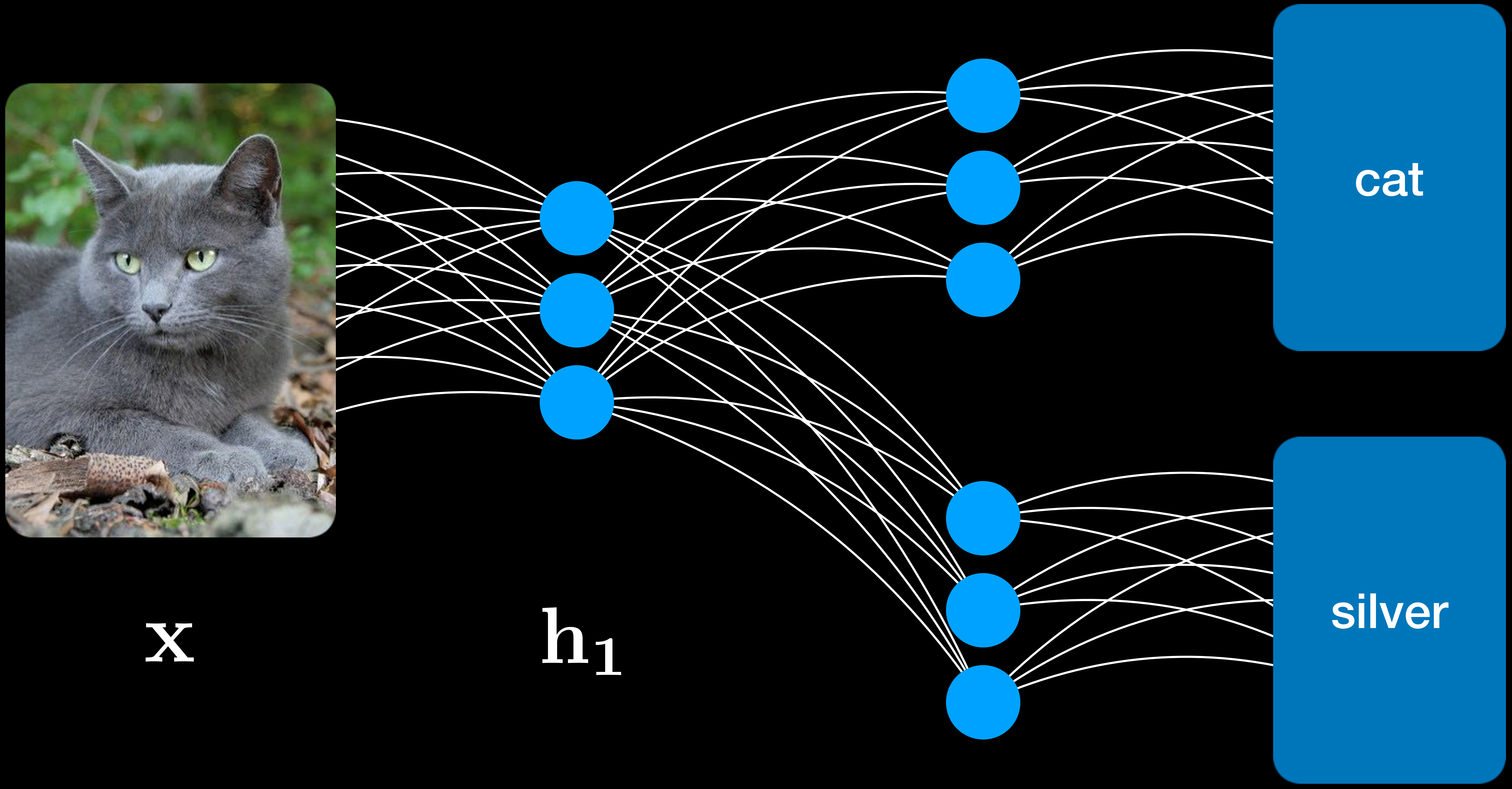


Neural Network

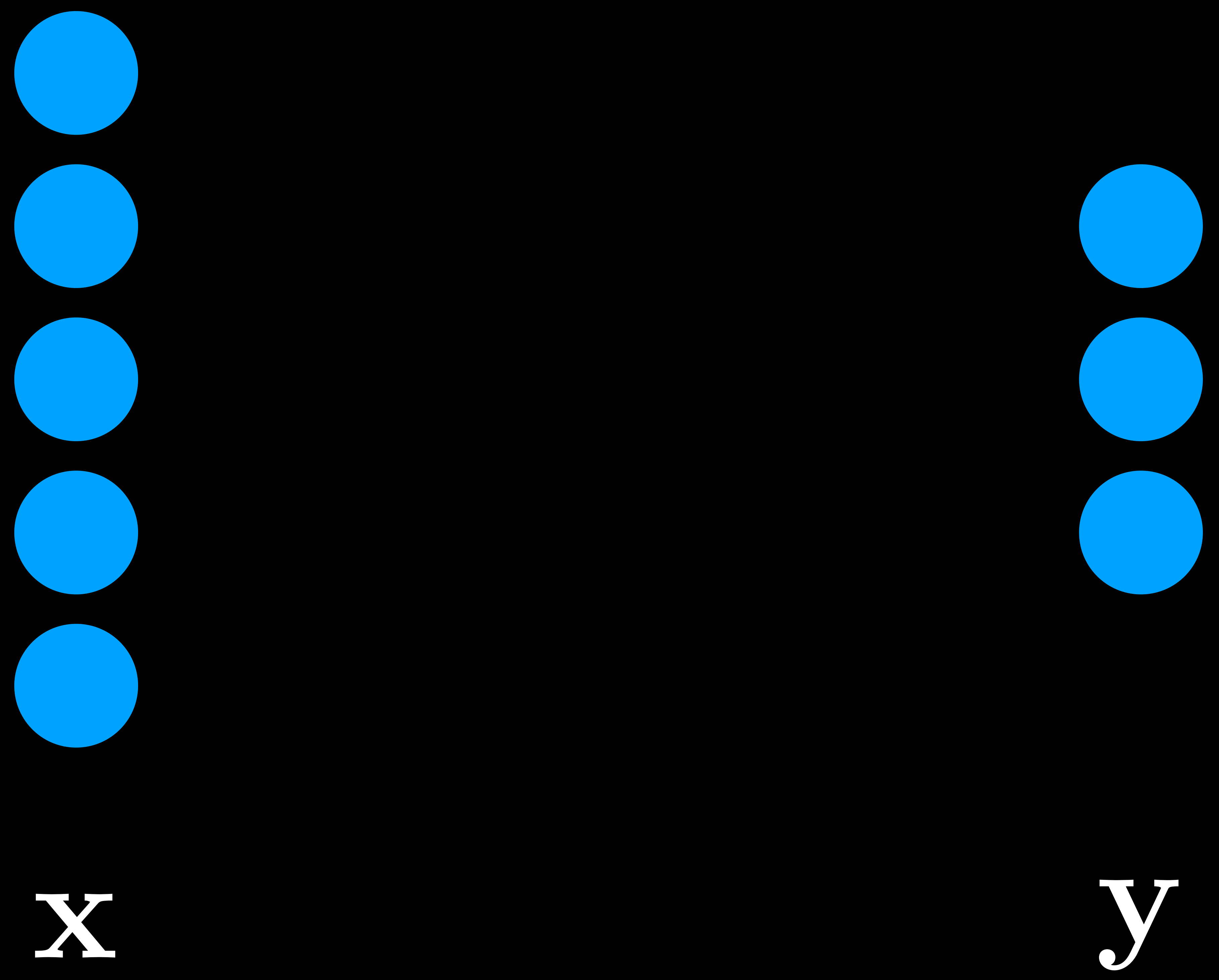


$\mathbf{x}$ $\mathbf{h}_1$ $\mathbf{h}_2$ $\mathbf{y}$

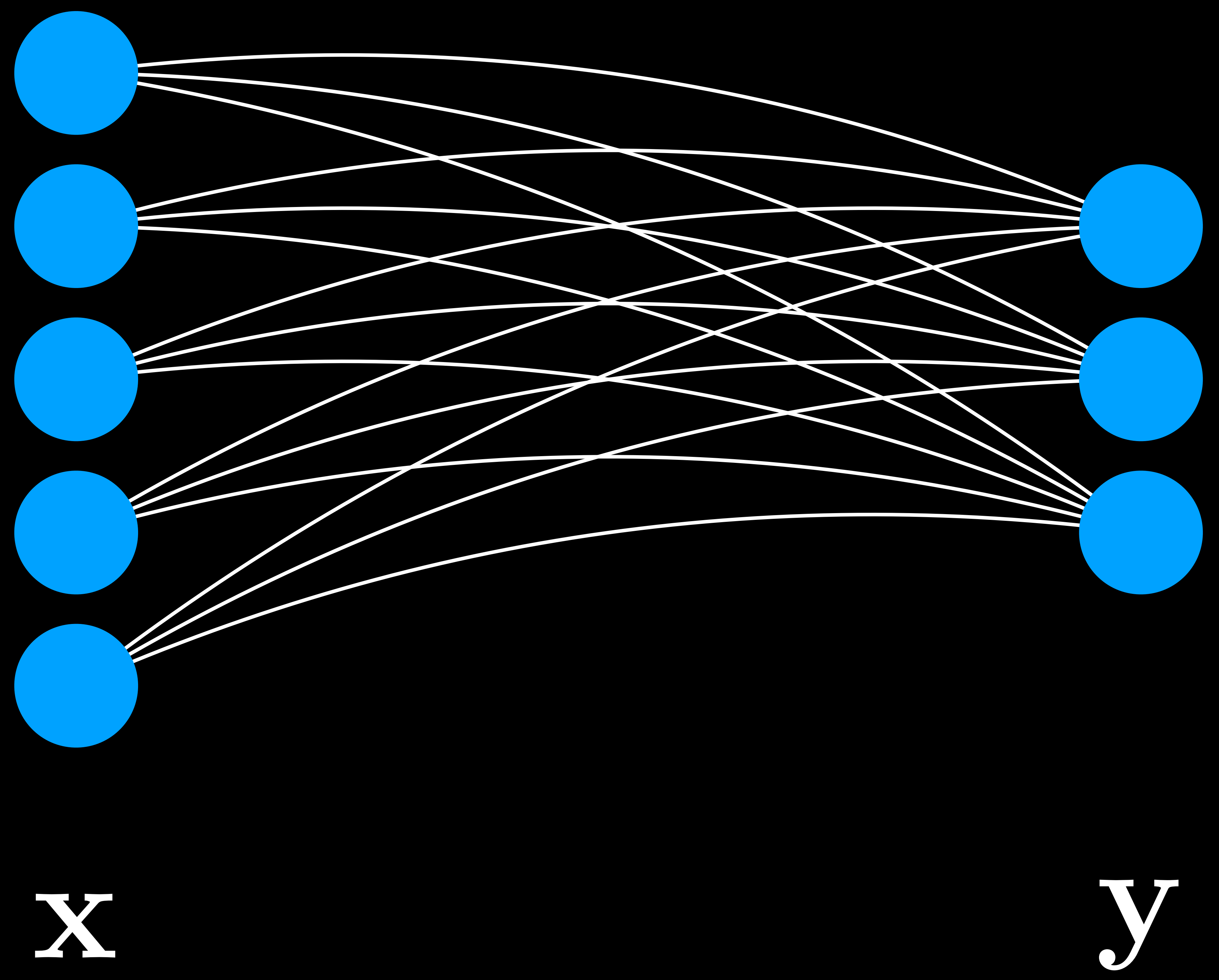
Neural Network



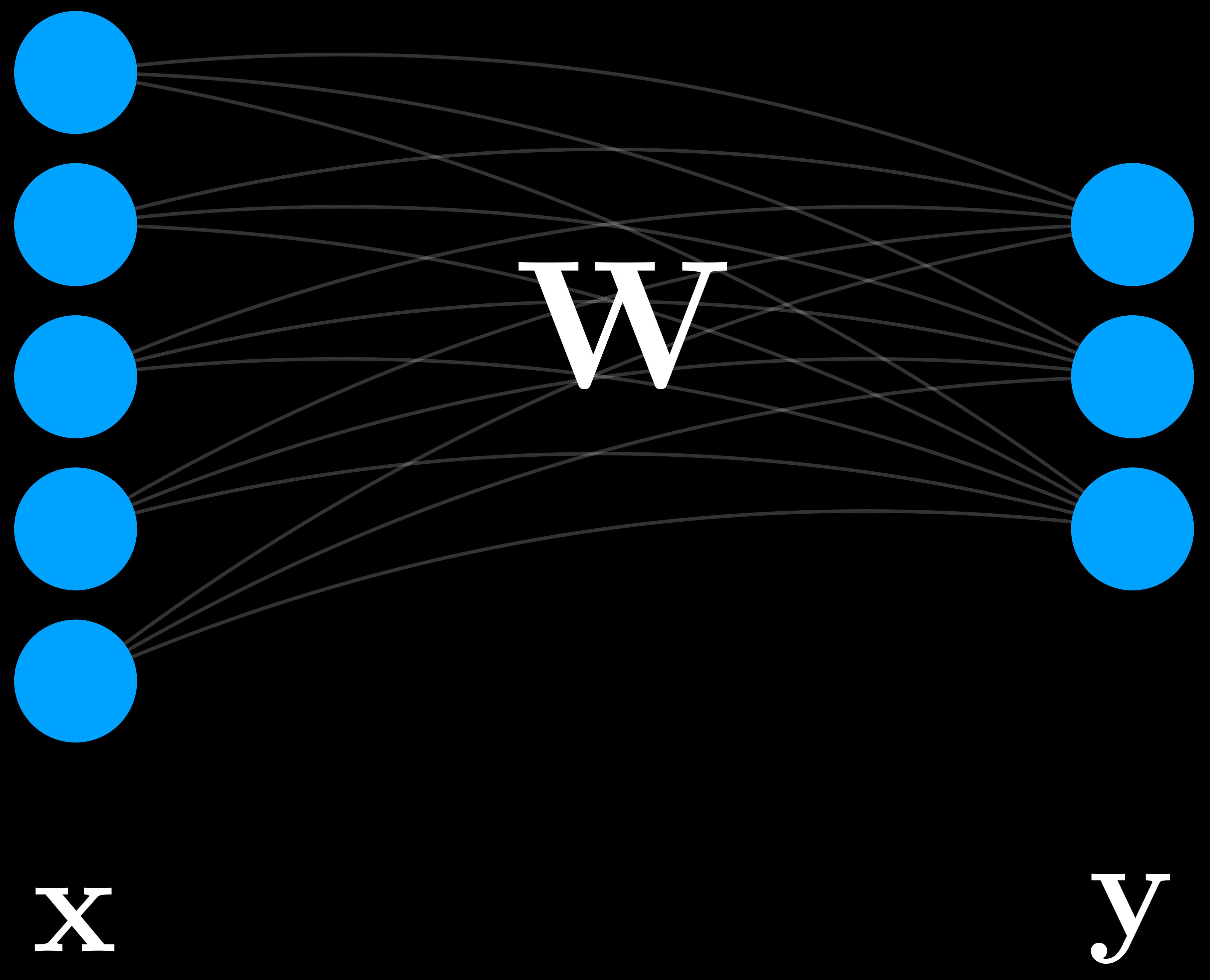
Neural Network



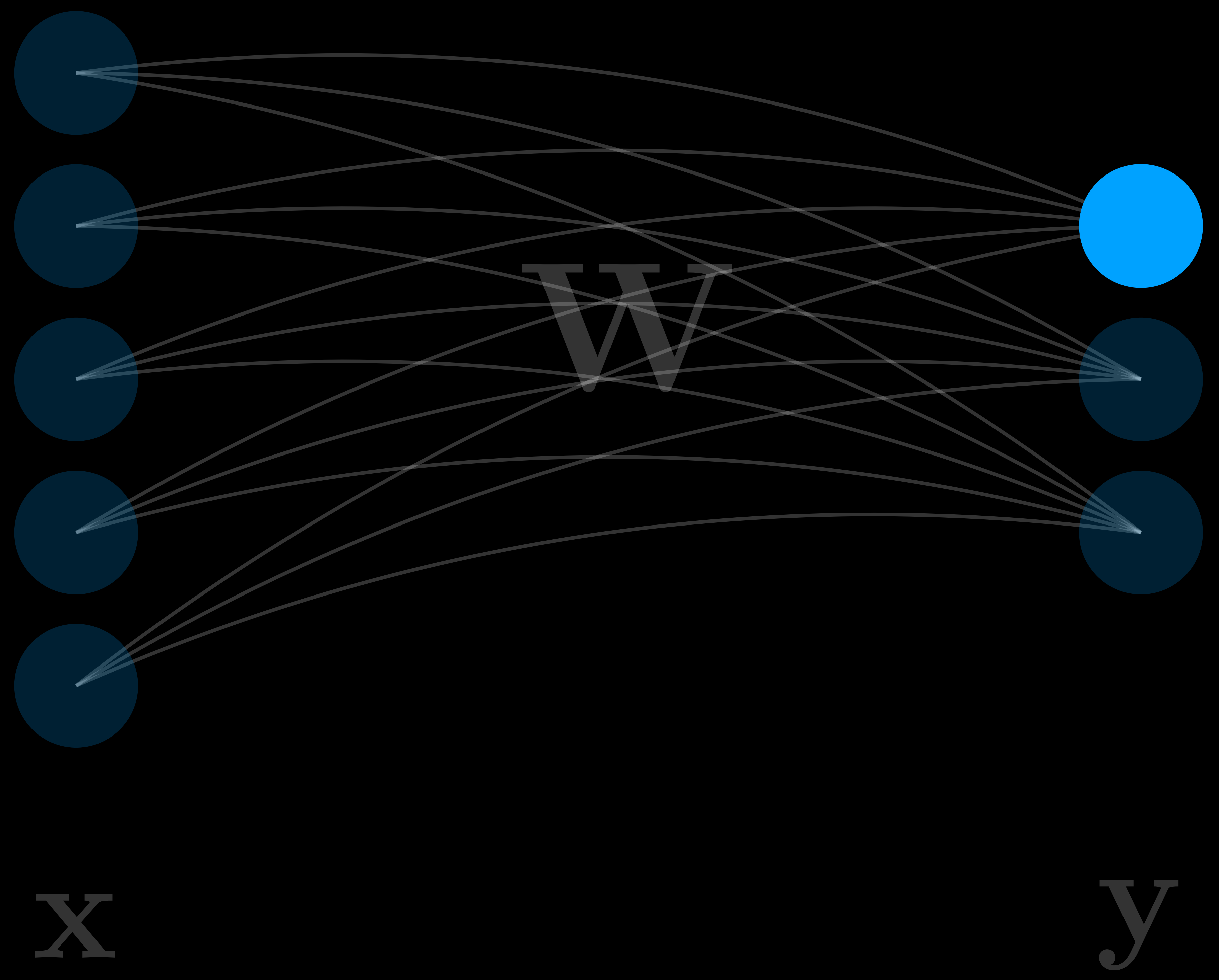
Neural Network



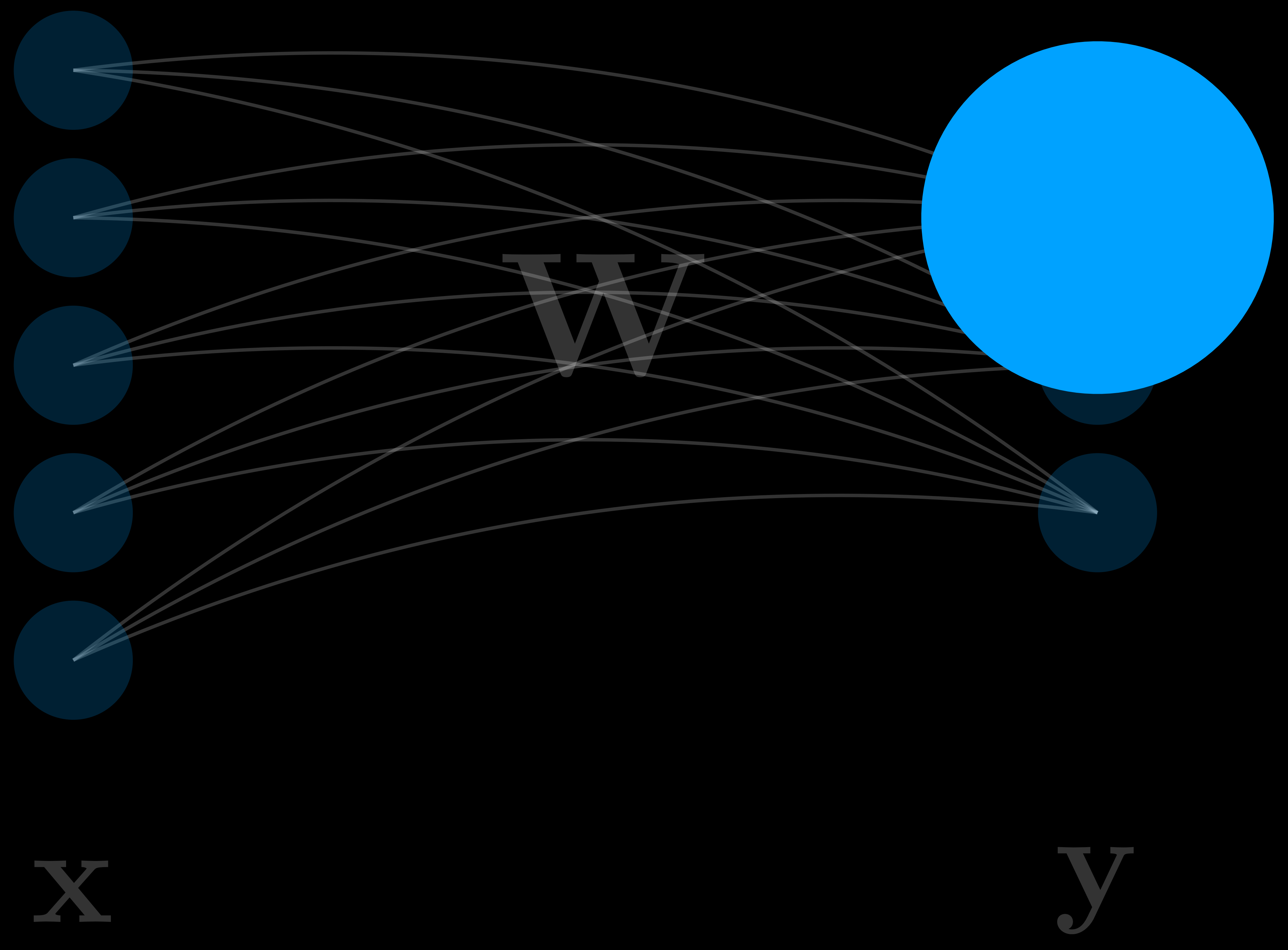
Neural Network



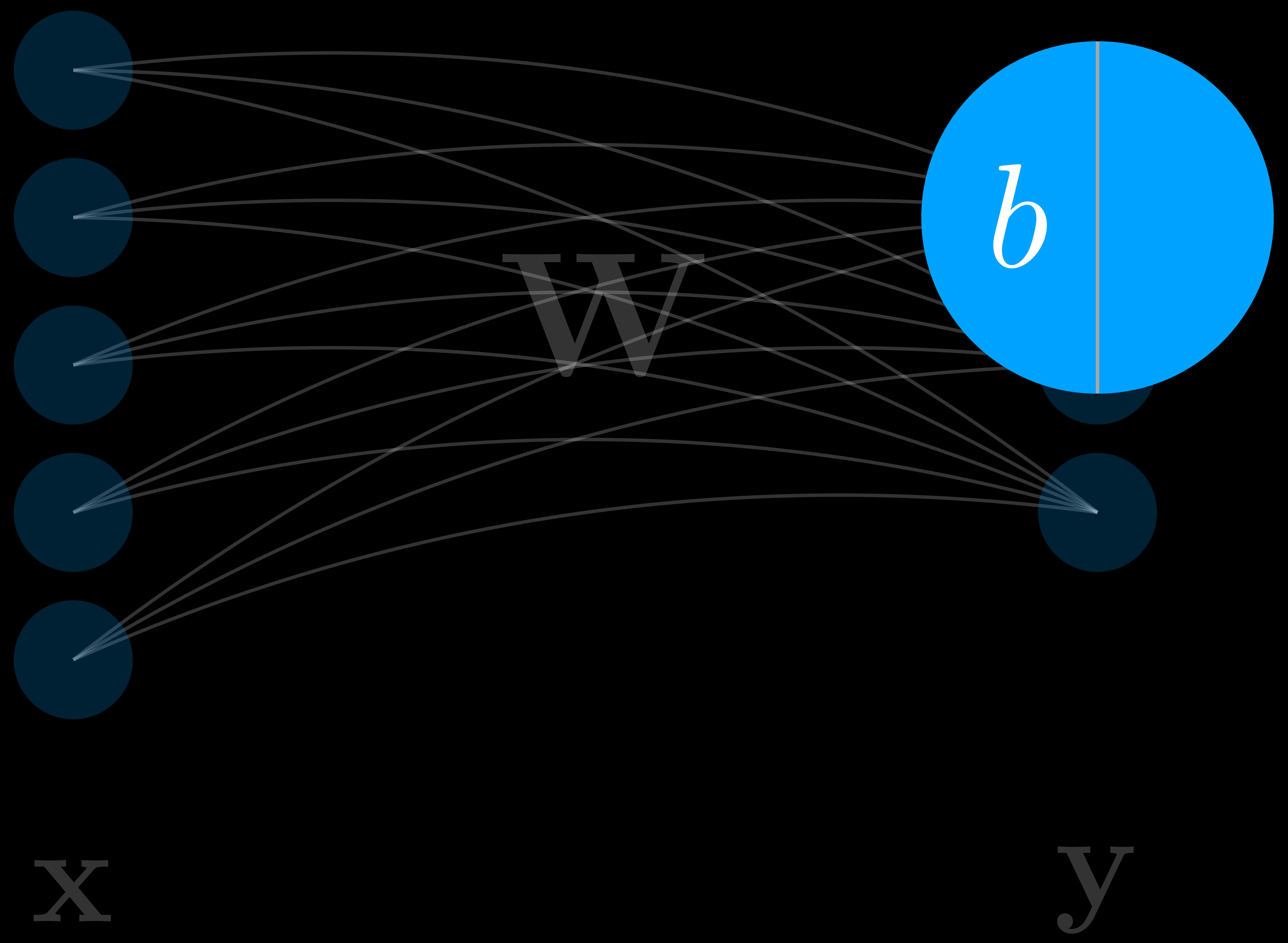
Neural Network



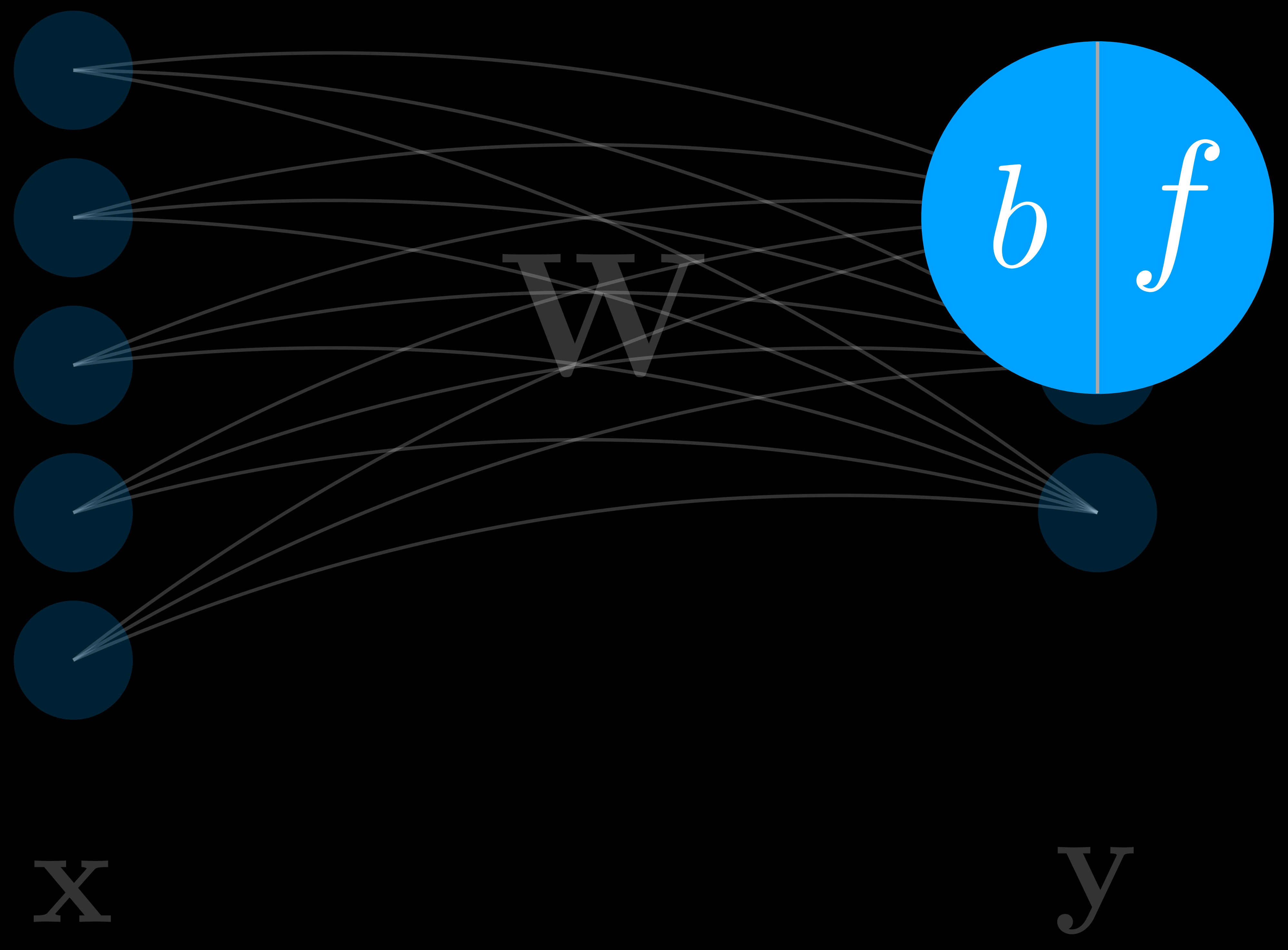
Neural Network



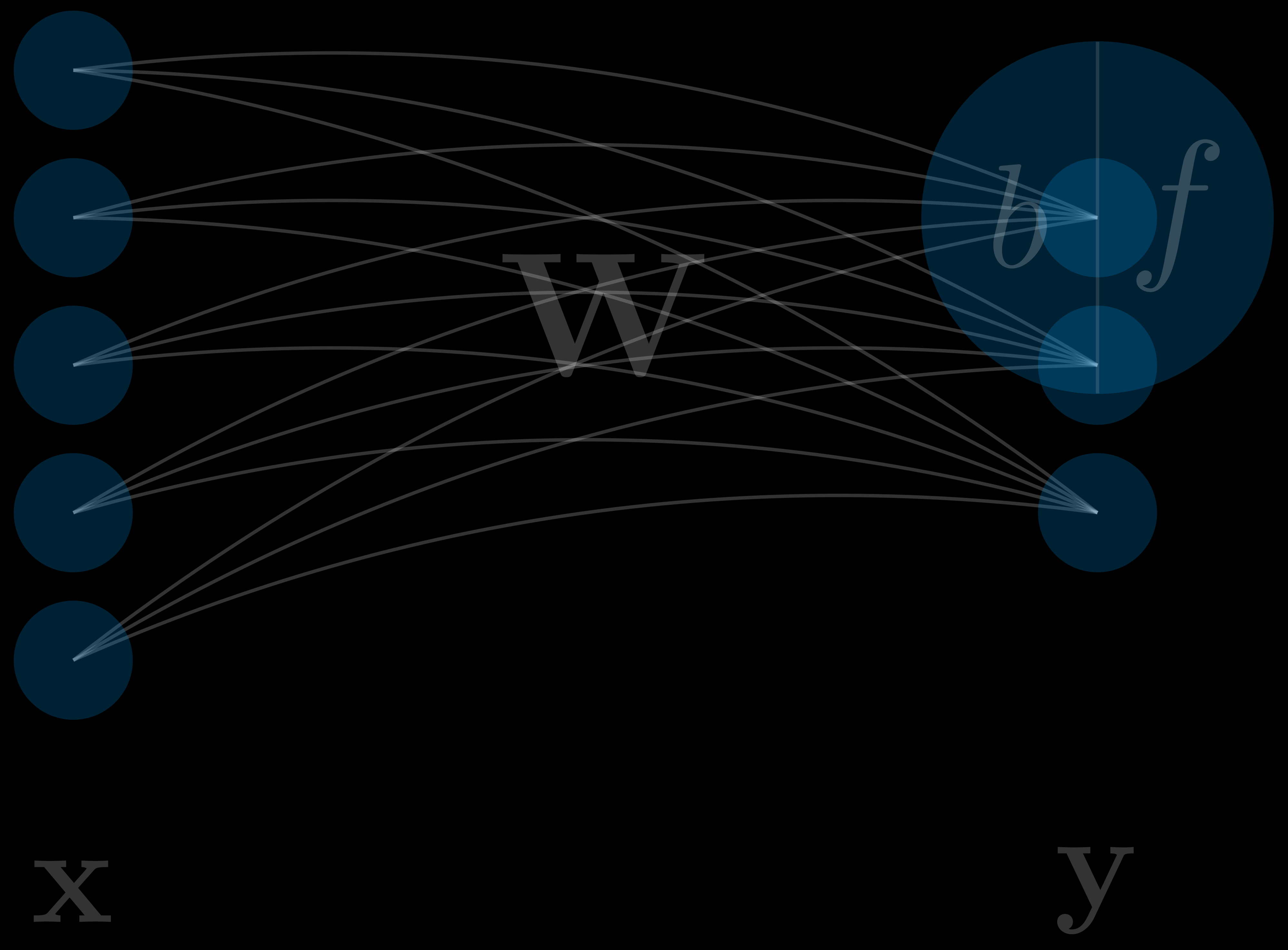
Neural Network



Neural Network

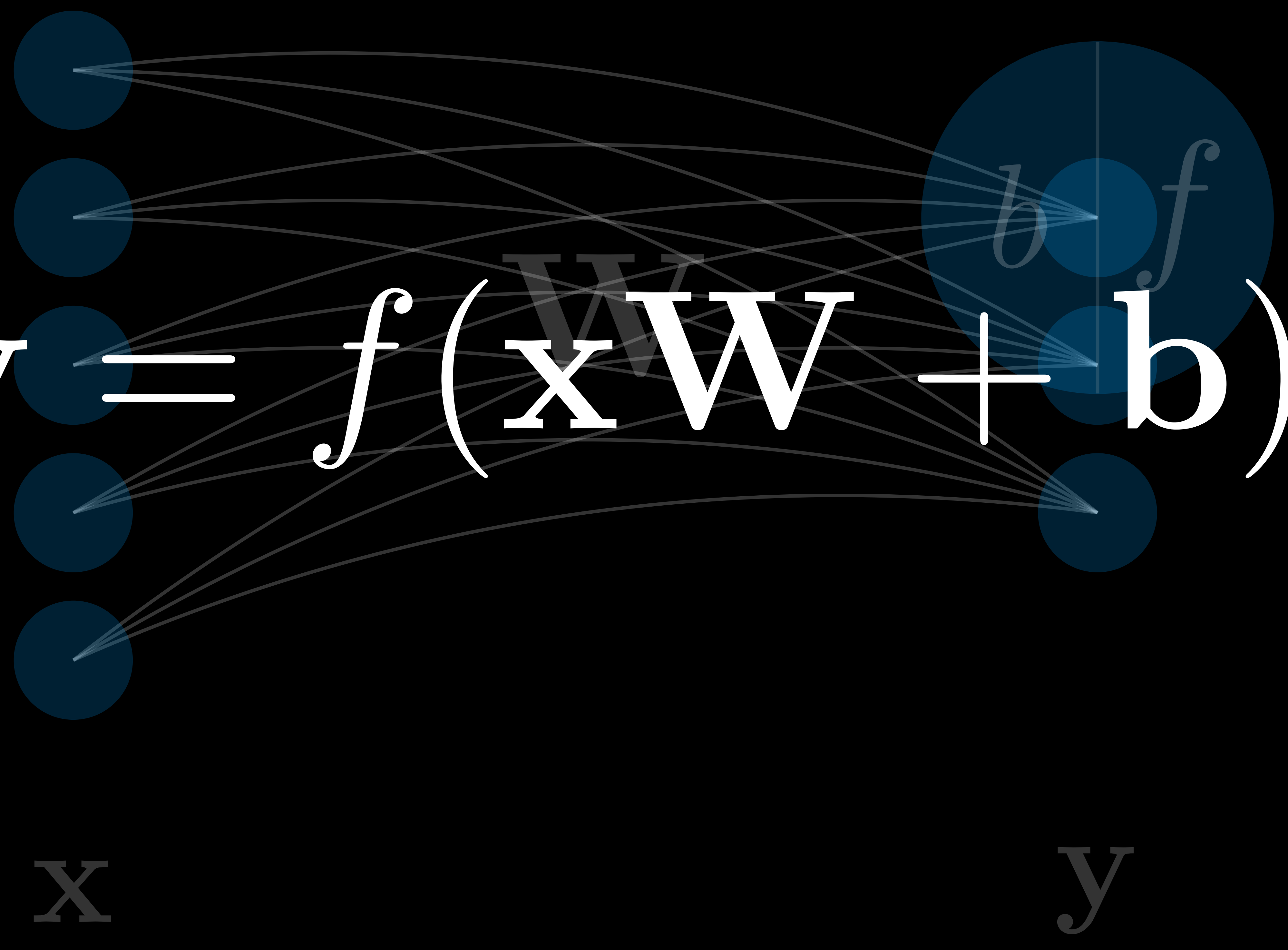


Neural Network



Neural Network

$$\mathbf{y} = f(\mathbf{x}\mathbf{W} + \mathbf{b})$$



Neural Network

$$\mathbf{y} = f(\mathbf{x}\mathbf{W} + \mathbf{b})$$

Neural Network

Input

$$\mathbf{y} = f(\mathbf{x}\mathbf{W} + \mathbf{b})$$

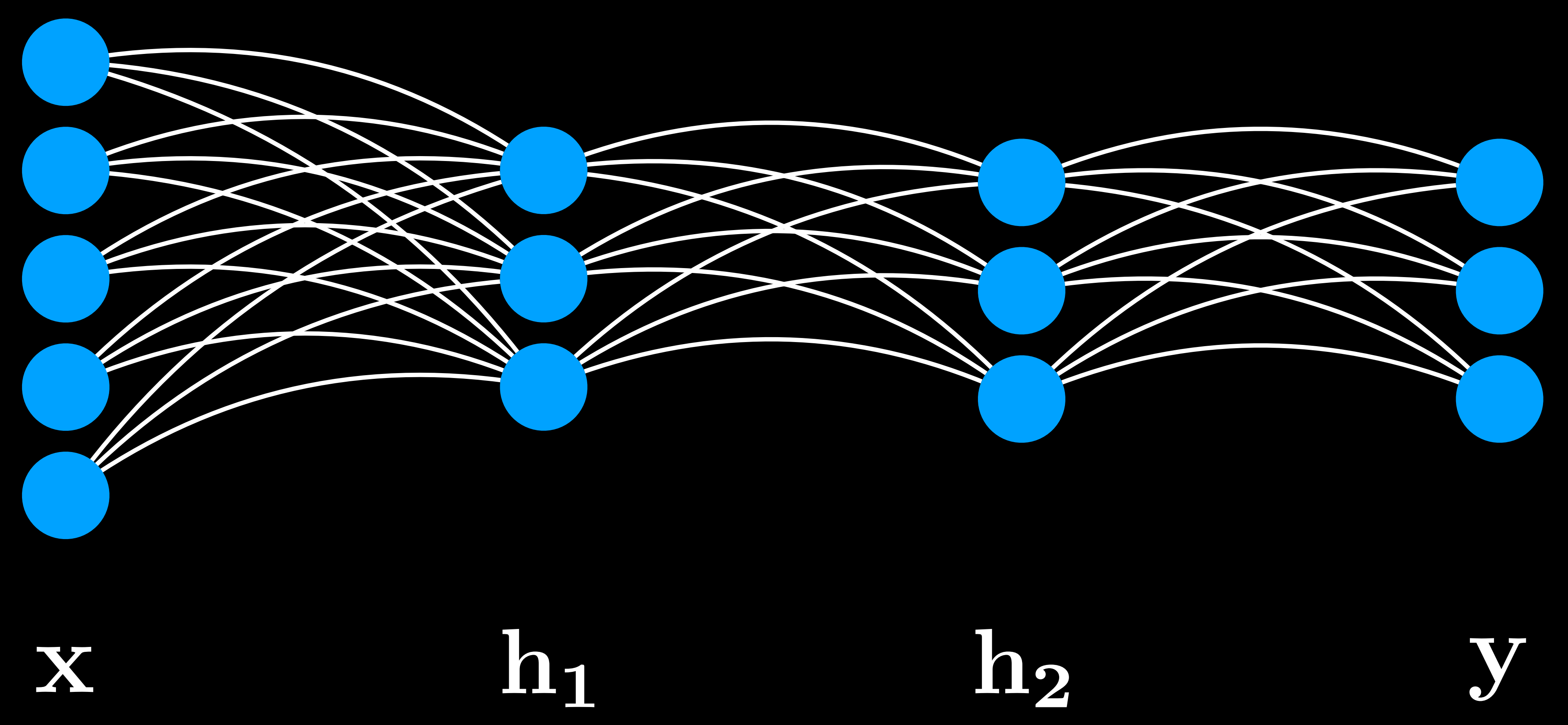
Prediction

Activation Function

Weight

Bias

Neural Network

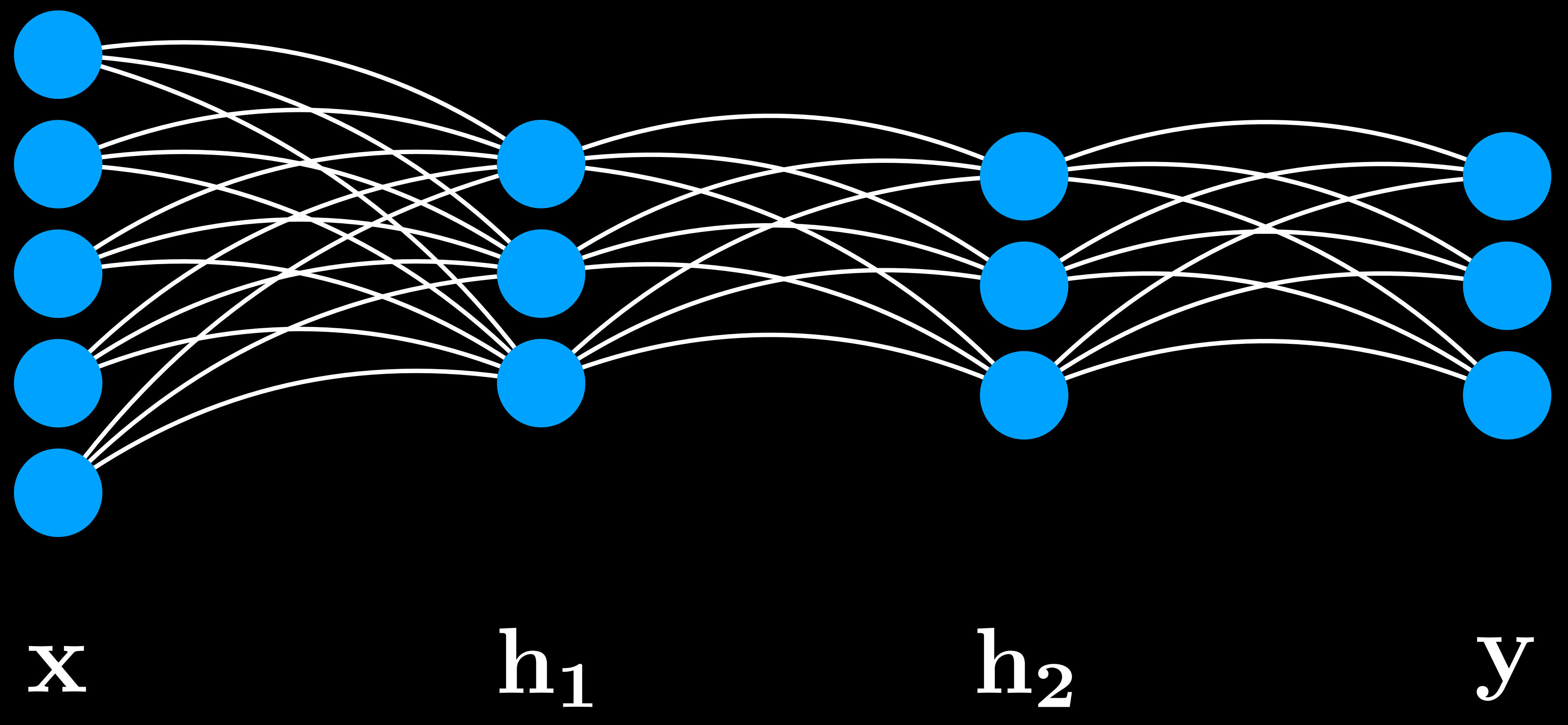


$$\mathbf{h_1} = f(\mathbf{x}\mathbf{W_1} + \mathbf{b_1})$$

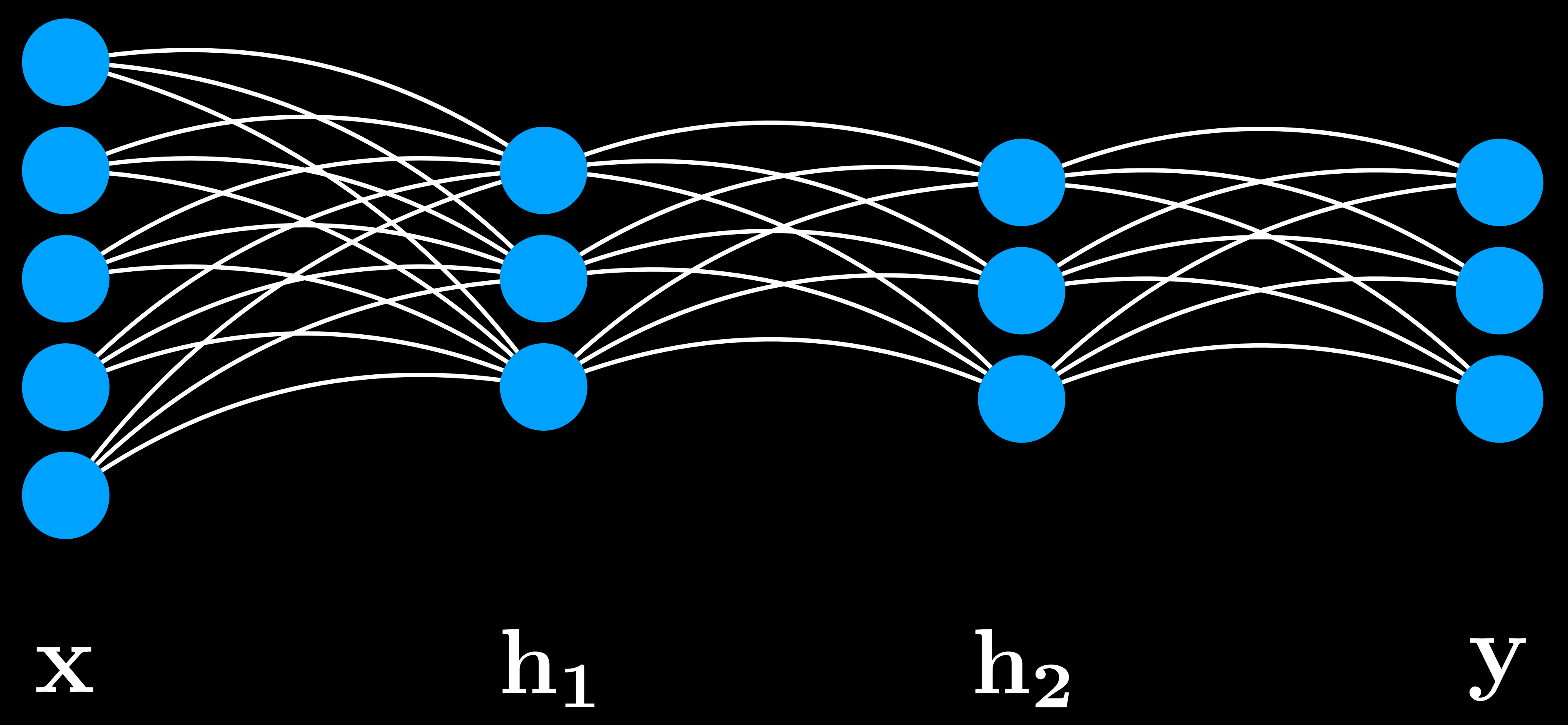
$$\mathbf{h_2} = f(\mathbf{h_1}\mathbf{W_2} + \mathbf{b_2})$$

$$\mathbf{y} = f(\mathbf{h_2}\mathbf{W_3} + \mathbf{b_3})$$

Supervised Learning



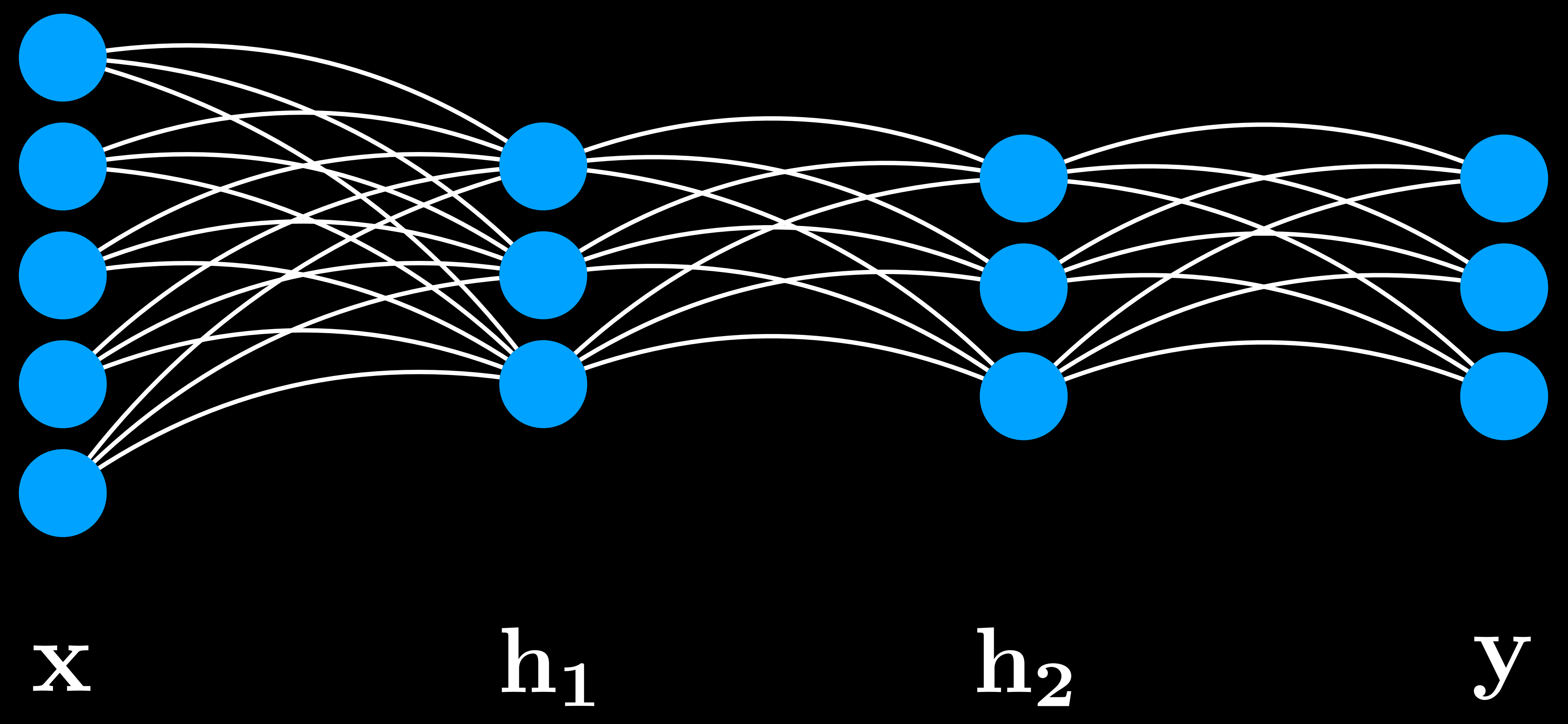
Supervised Learning



$$L = \frac{1}{n}\sum_{i}^{n}(y_i - \bar{y}_i)^2$$

Loss function

Supervised Learning

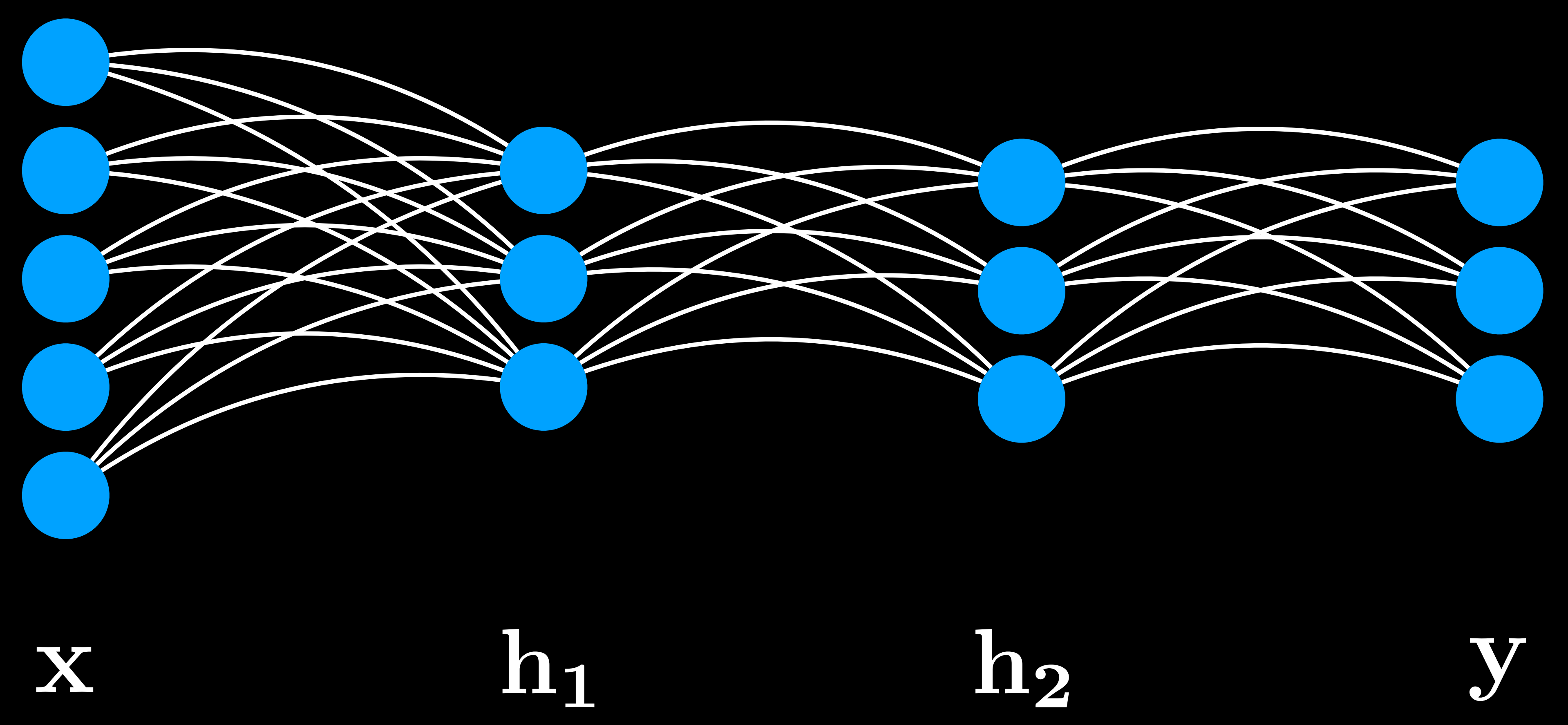


Expected output

$$L = \frac{1}{n}\sum_{i}^{n}(y_i - \bar{y}_i)^2$$

Loss function

Supervised Learning



Expected output

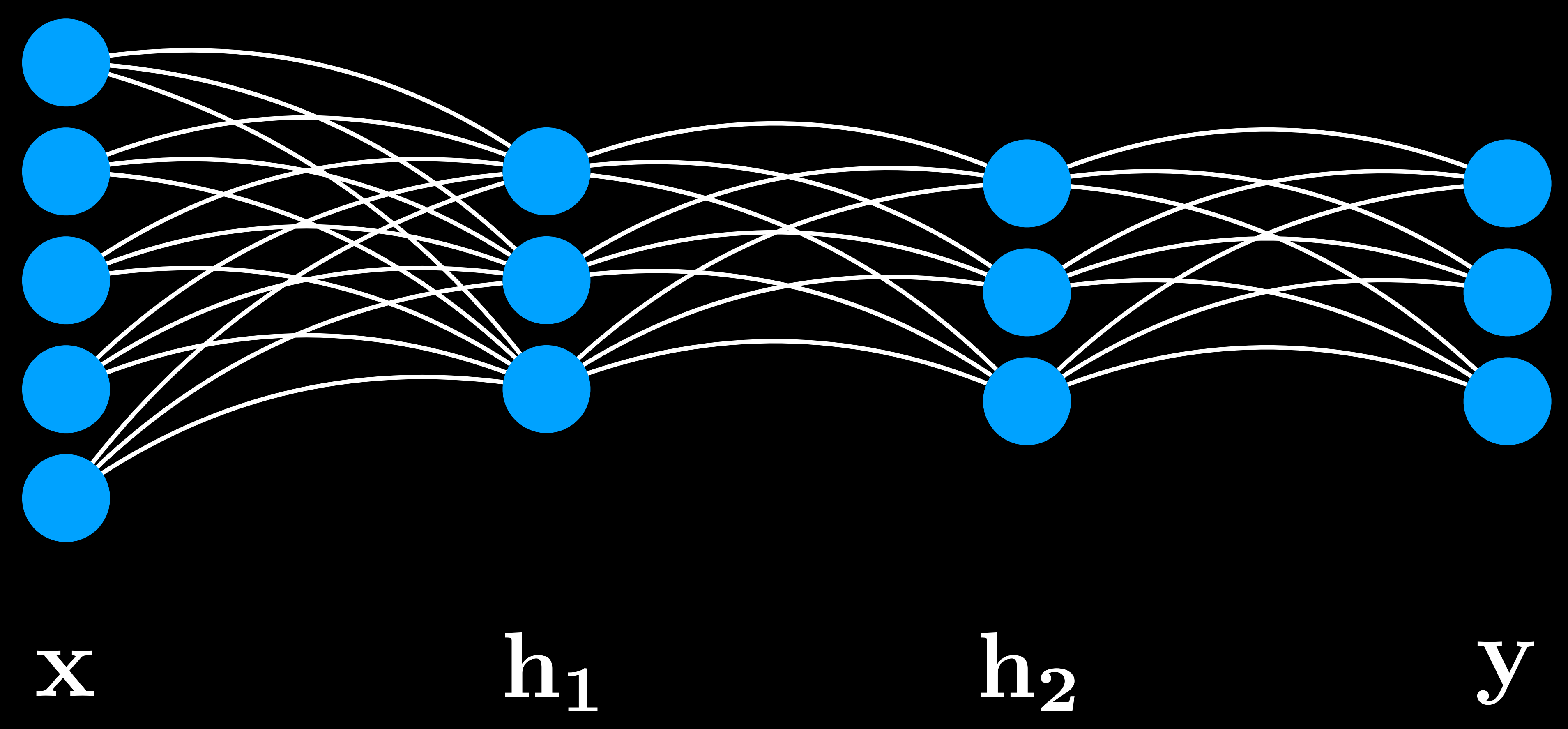
$$L = \frac{1}{n}\sum_{i}^{n}(y_i - \bar{y}_i)^2$$

Loss function

$$\hat{\theta} = \theta - \frac{\partial L}{\partial \theta}$$

Gradient descent

Supervised Learning



Expected output

$$L = \frac{1}{n}\sum_{i}^{n}(y_i - \bar{y}_i)^2$$

Loss function

Backpropagation

$$\hat{\theta} = \theta - \frac{\partial L}{\partial \theta}$$

Gradient descent

Computing Gradients

$$\frac{\partial L}{\partial \theta}$$

Computing Gradients

$$\frac{\partial L}{\partial \theta}$$

Automatic Differentiation

Symbolic Differentiation

Computing Gradients

$$\frac{\partial L}{\partial \theta}$$

Automatic Differentiation

Symbolic Differentiation

Chain rule

Reuses AST nodes

Computing Gradients

$$\frac{\partial L}{\partial \theta}$$

Automatic Differentiation	Symbolic Differentiation
Chain rule	Pen-and-paper transformation
Reuses AST nodes	Heavy common subexpressions

Computing Gradients

$$\frac{\partial L}{\partial \theta}$$

Automatic Differentiation

Chain rule

Reuses AST nodes

Symbolic Differentiation

Pen-and-paper transformation

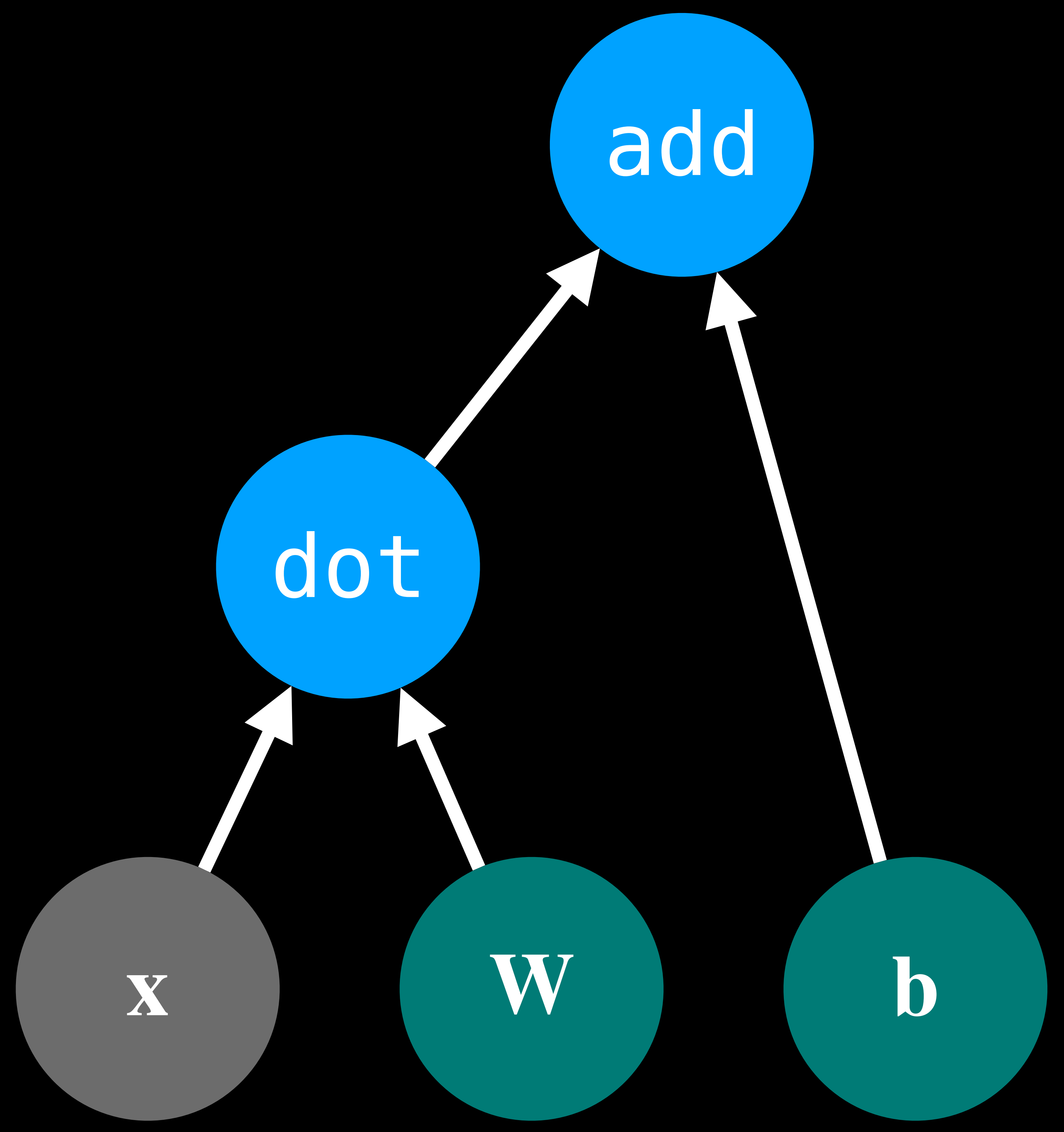
Heavy common subexpressions

Automatic Differentiation

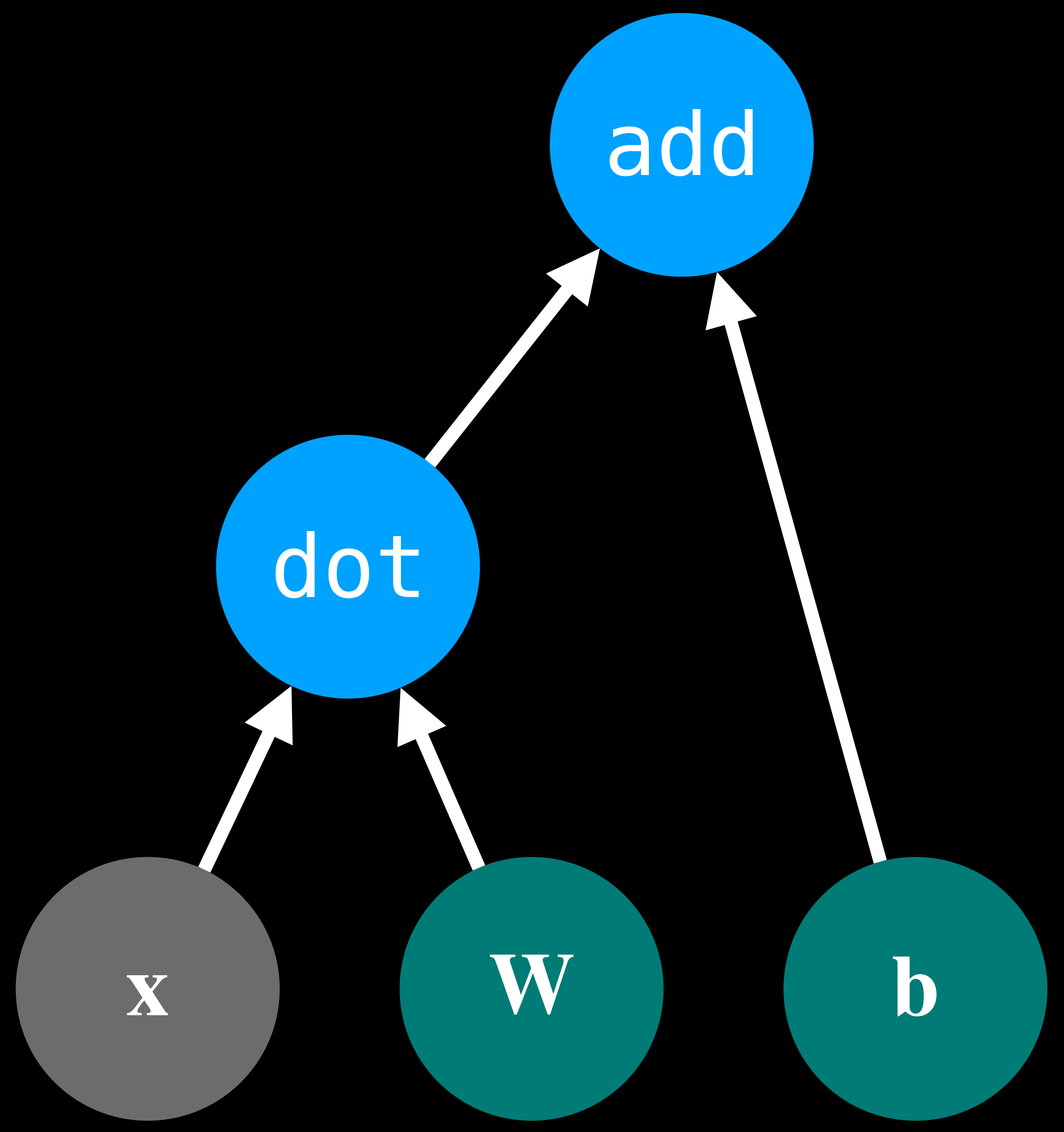
$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial x} = \left(\frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial w_2}\right)\frac{\partial w_2}{\partial x} = \left(\left(\frac{\partial y}{\partial w_1}\frac{\partial w_1}{\partial w_2}\right)\frac{\partial w_2}{\partial w_3}\right)\frac{\partial w_3}{\partial x} = \dots$$

Reverse mode AutoDiff

Automatic Differentiation



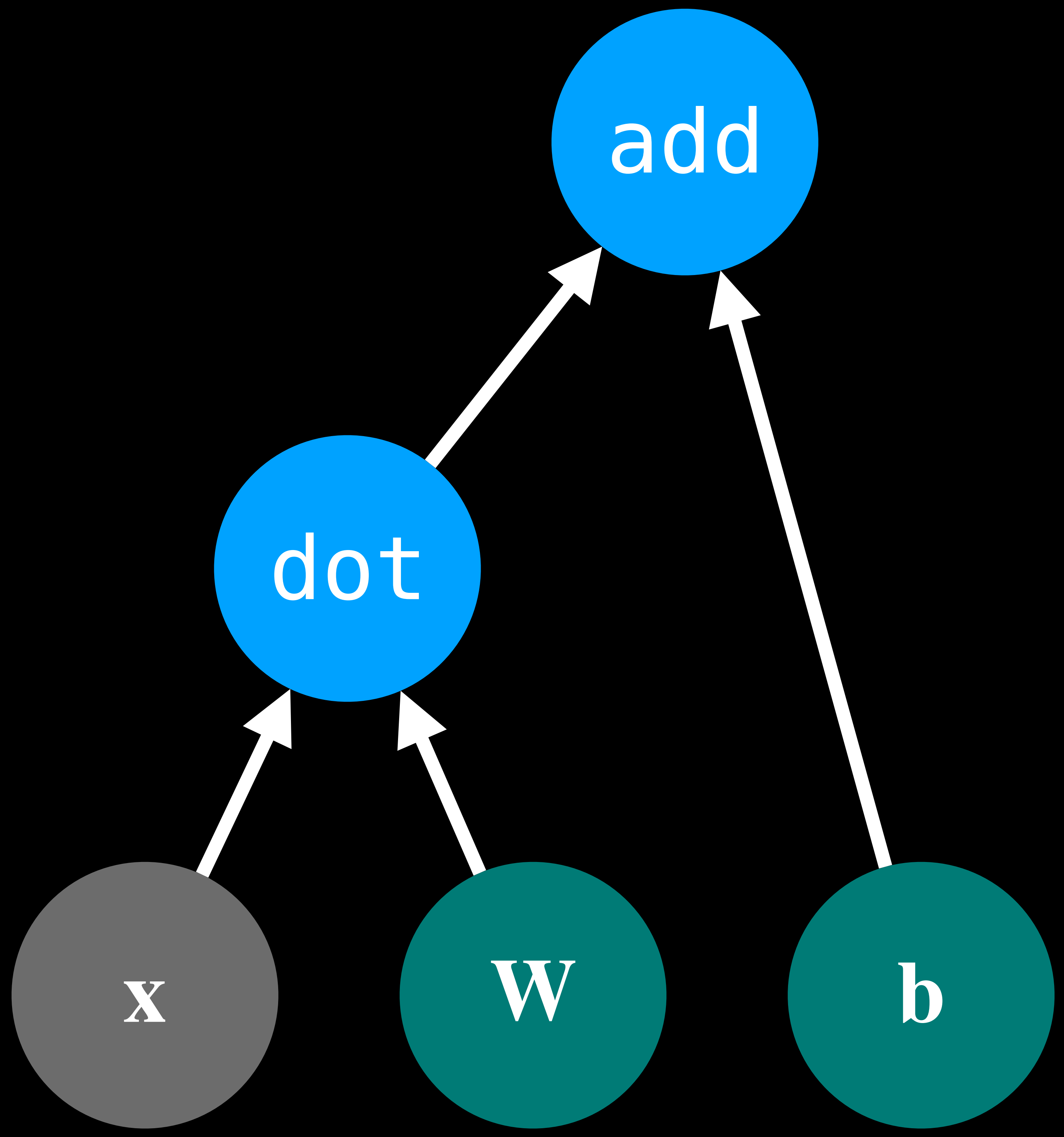
Automatic Differentiation




```
struct Node {
    var forward: Tensor
    var backward: Tensor
}
```

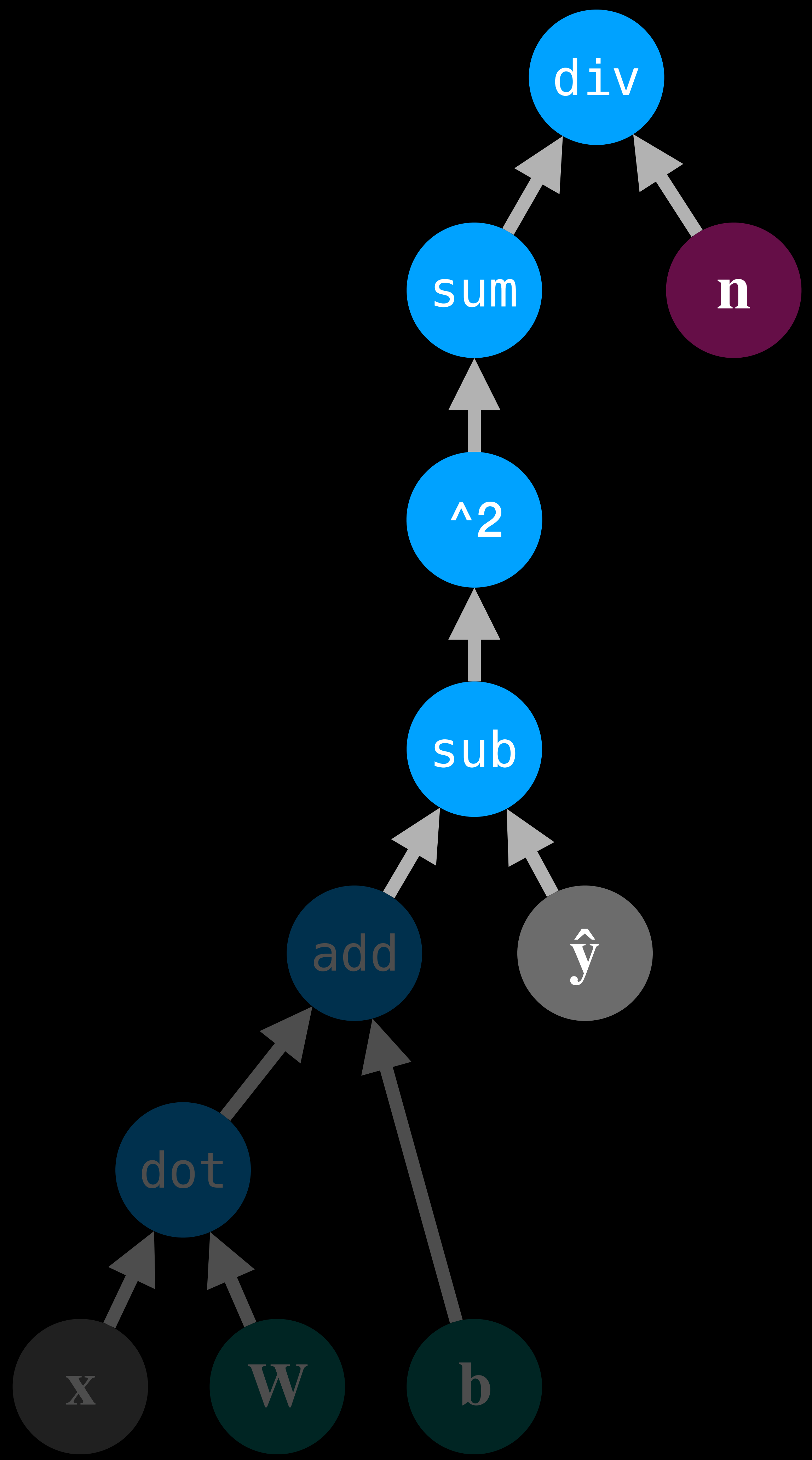
Interpretation approach

Automatic Differentiation



- Learn W and b from example inputs & outputs

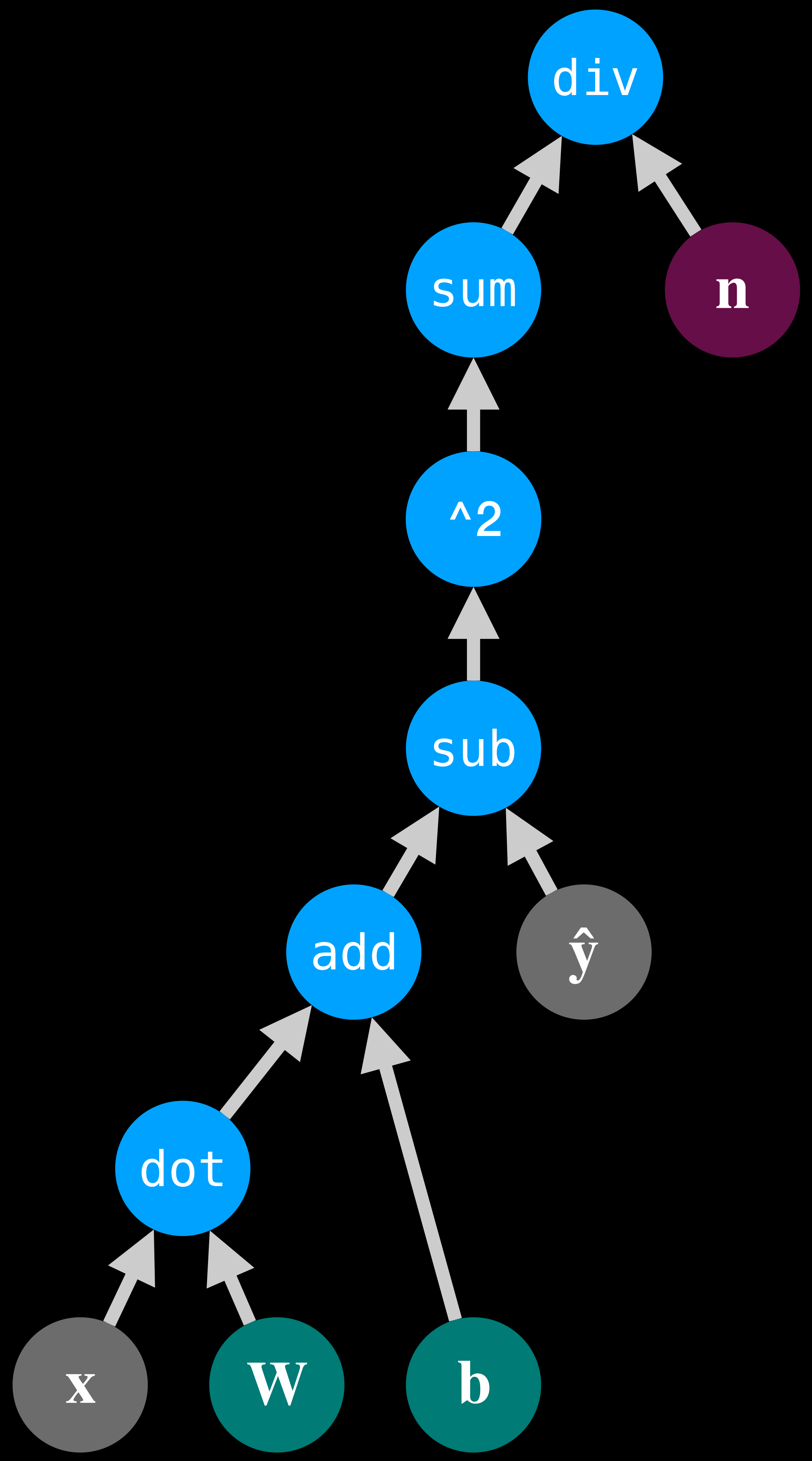
Automatic Differentiation



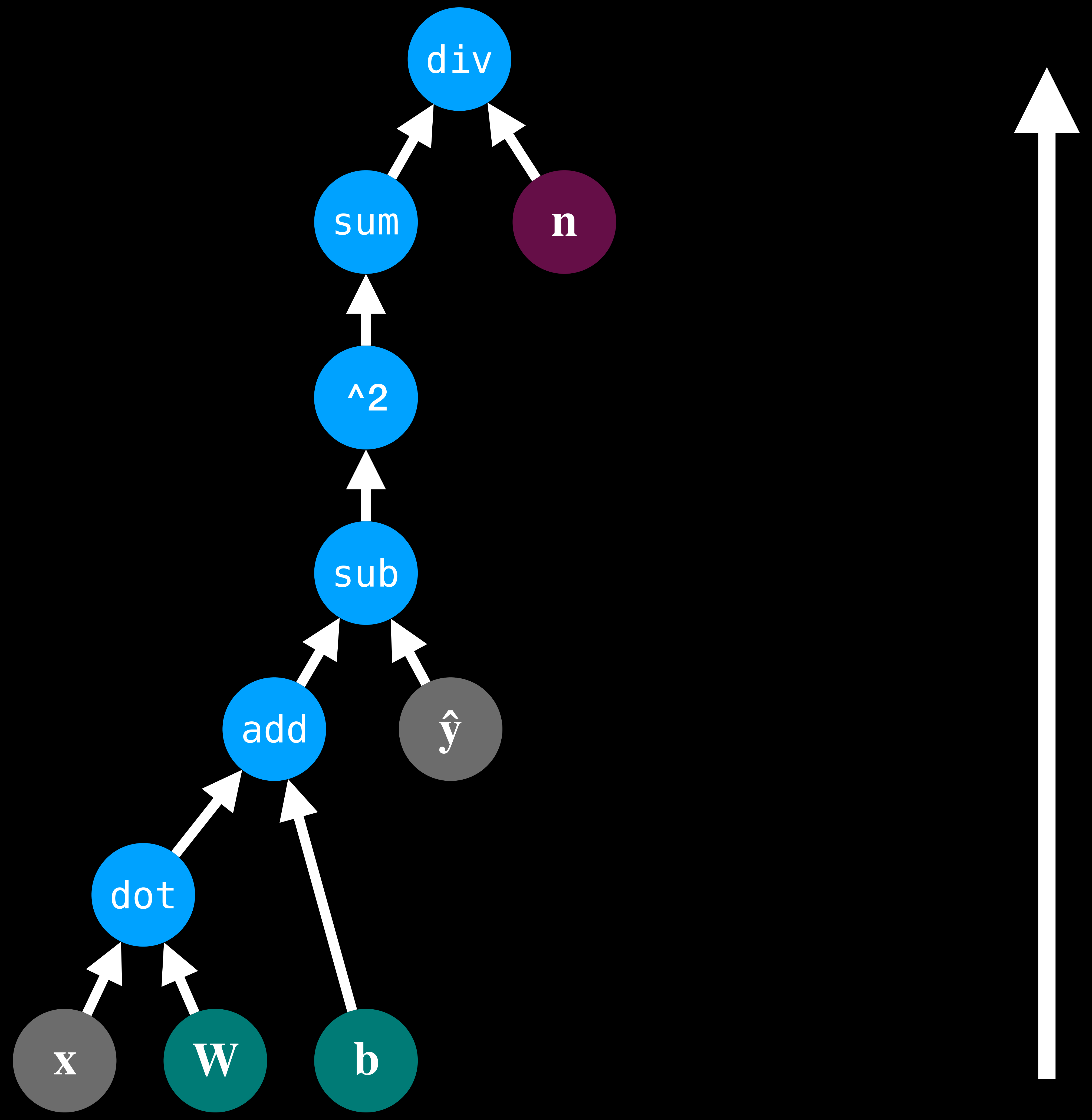
$$L = \frac{1}{n} \sum_{i}^{n} (y_i - \bar{y}_i)^2$$

Loss function

Automatic Differentiation

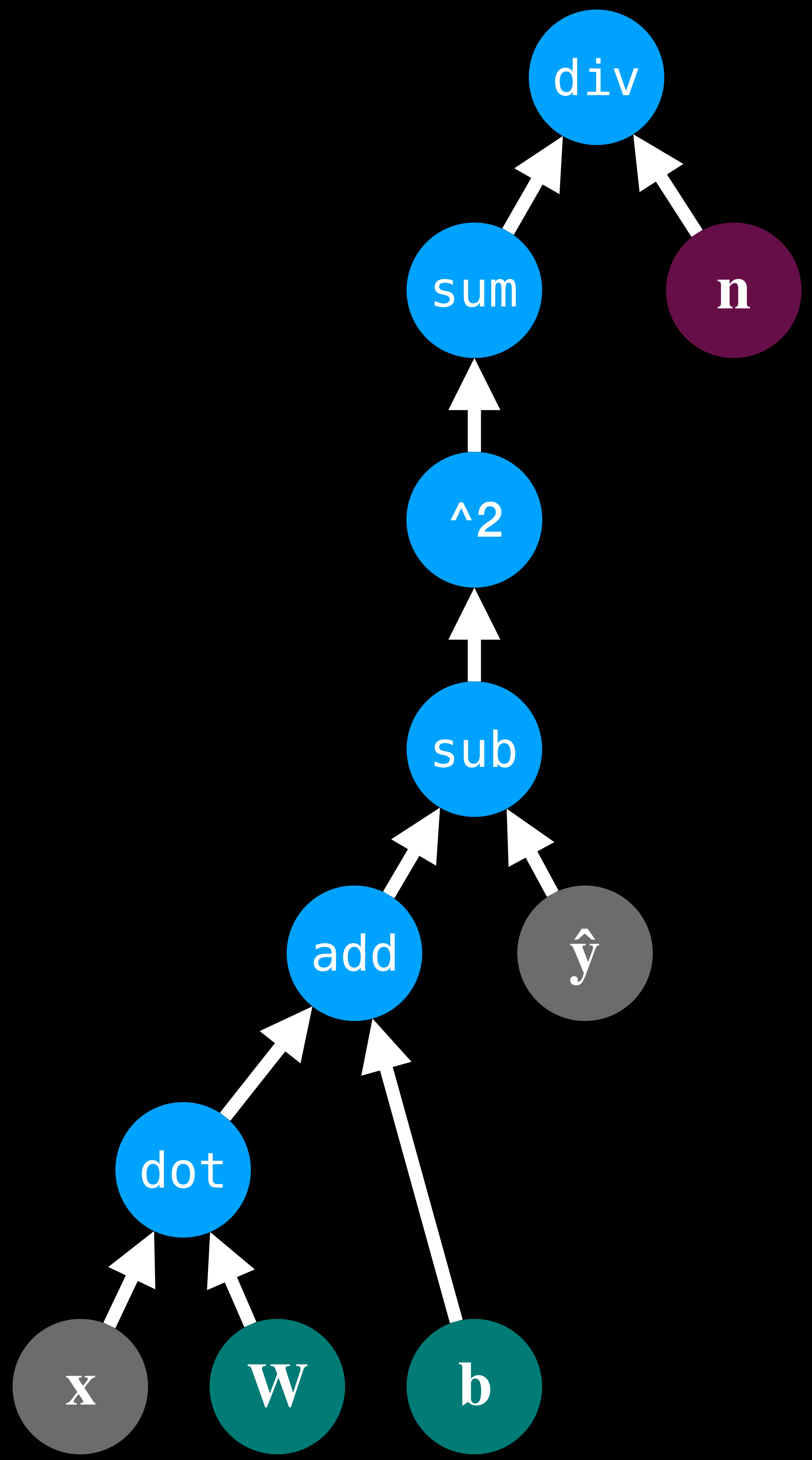


Automatic Differentiation

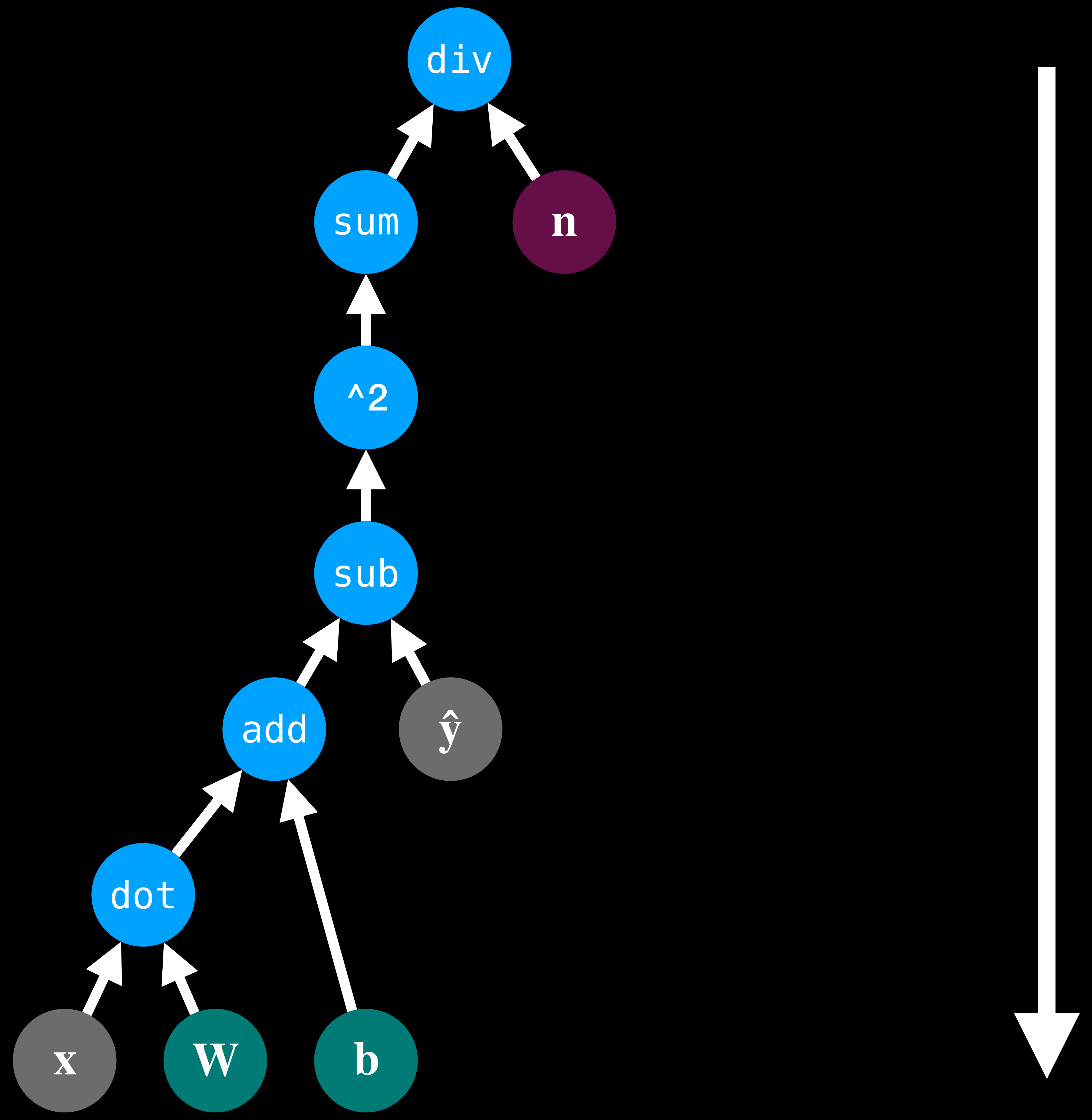


Forward pass computes loss L

Automatic Differentiation



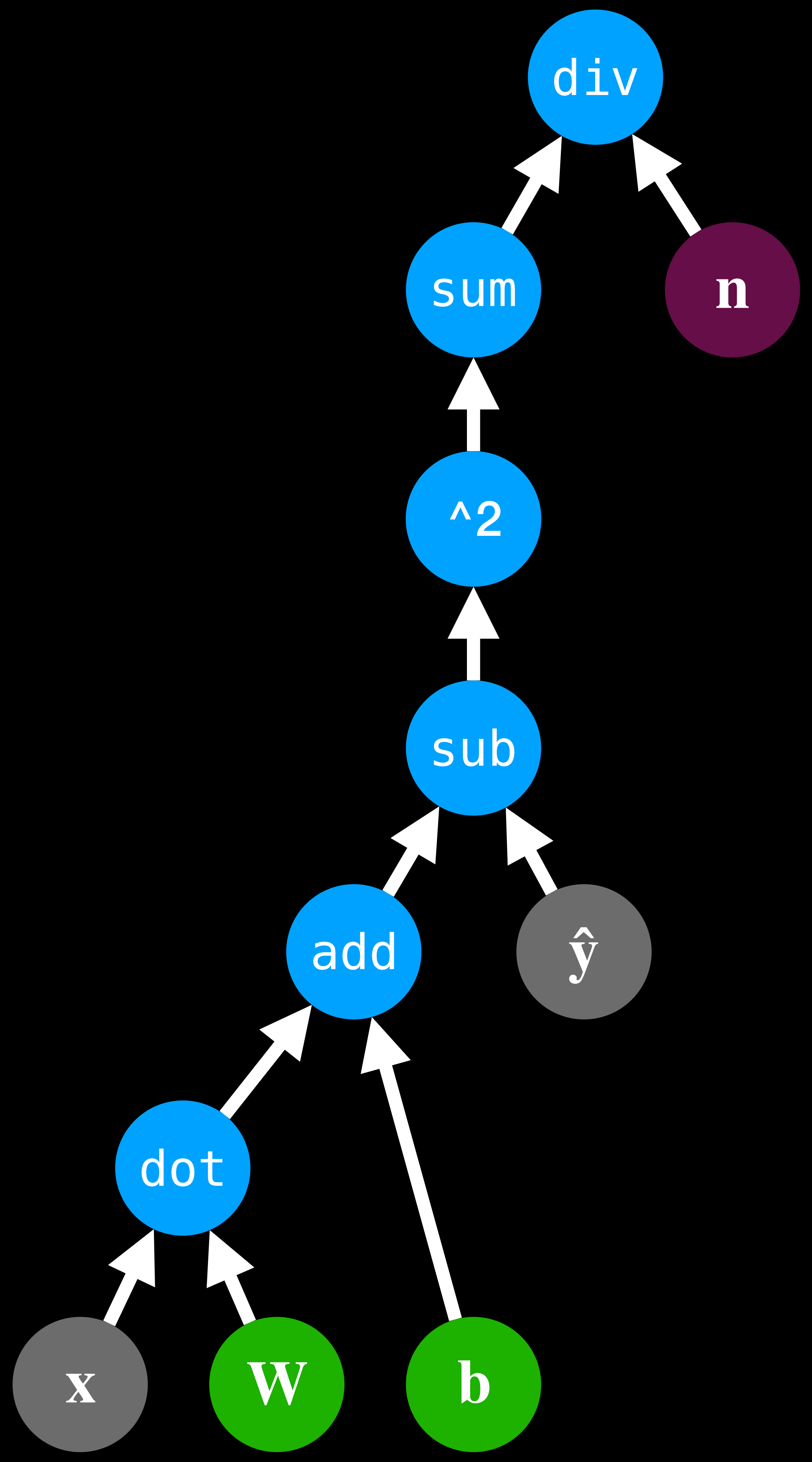
Automatic Differentiation



Backward pass computes gradients

$\partial L/\partial \mathbf{W}$, $\partial L/\partial \mathbf{b}$

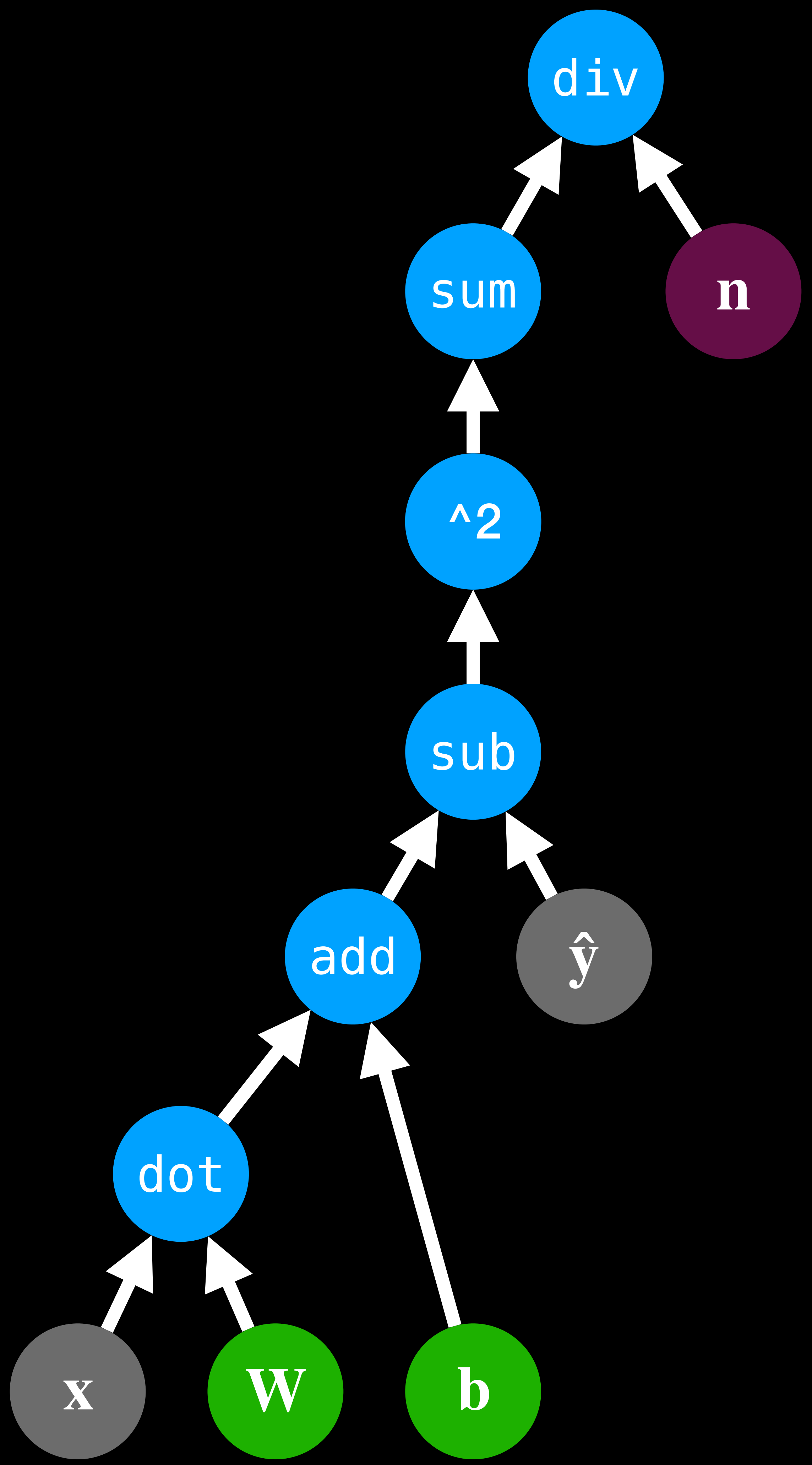
Automatic Differentiation



$$\hat{\theta} = \theta - \frac{\partial L}{\partial \theta}$$

Gradient descent

Automatic Differentiation

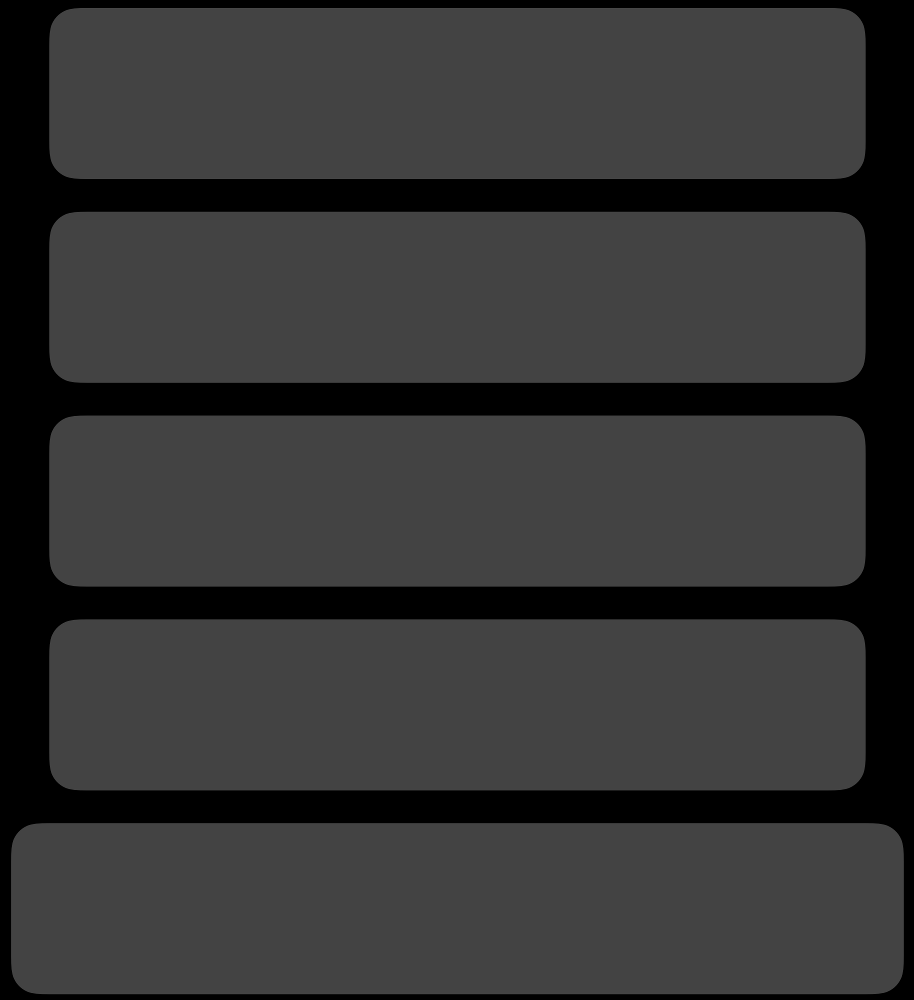


$$\hat{\theta} = \theta - \frac{\partial L}{\partial \theta}$$

Gradient descent

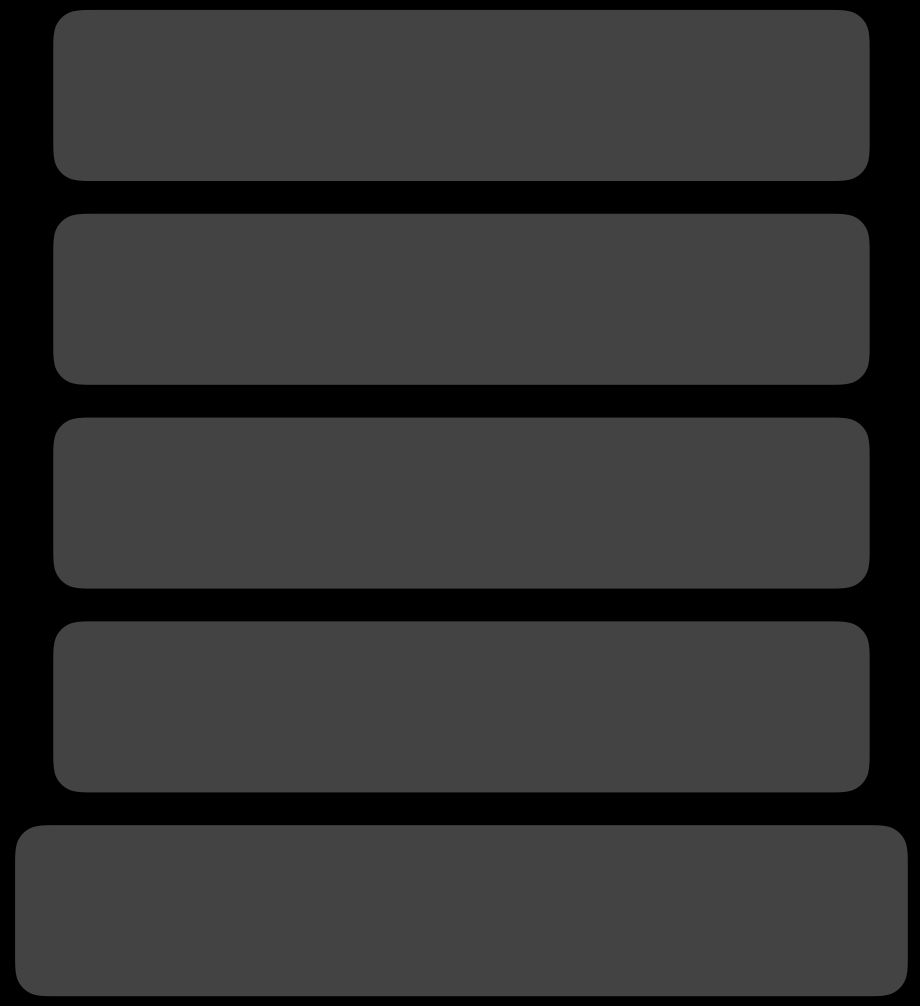
Existing Tools

Existing Tools



Layer-oriented library

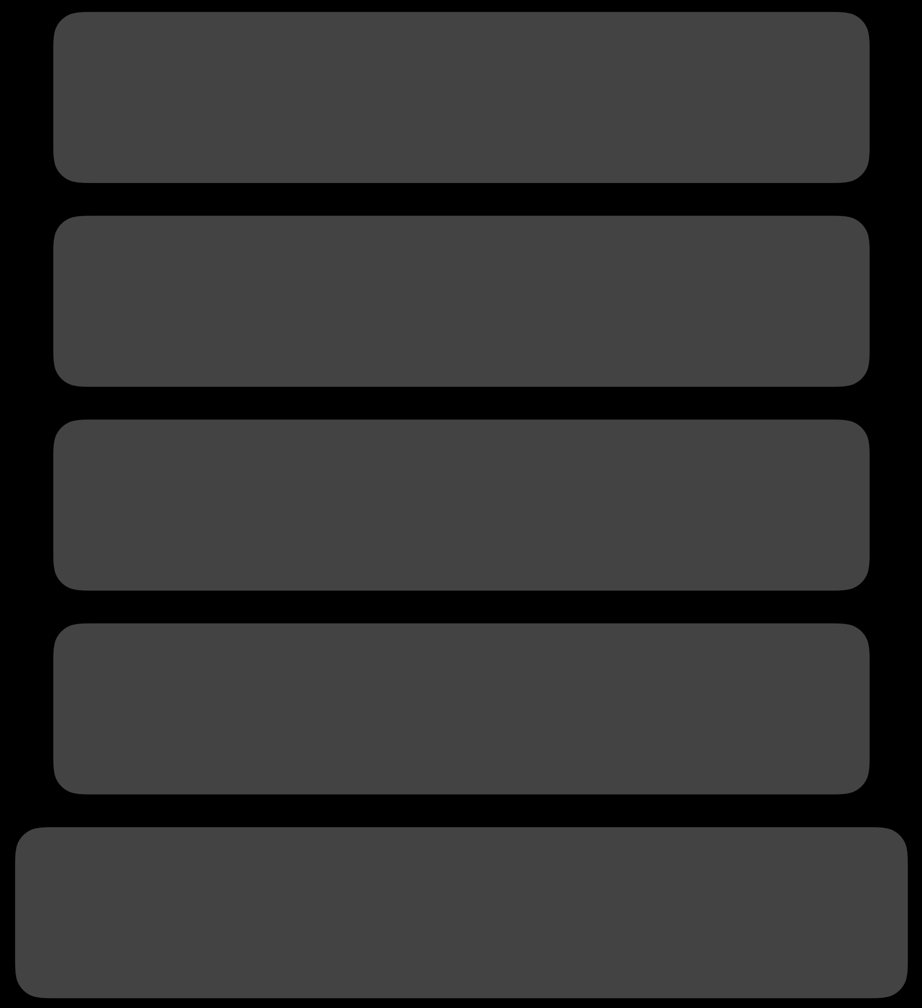
Existing Tools



Layer-oriented library

- Caffe, etc

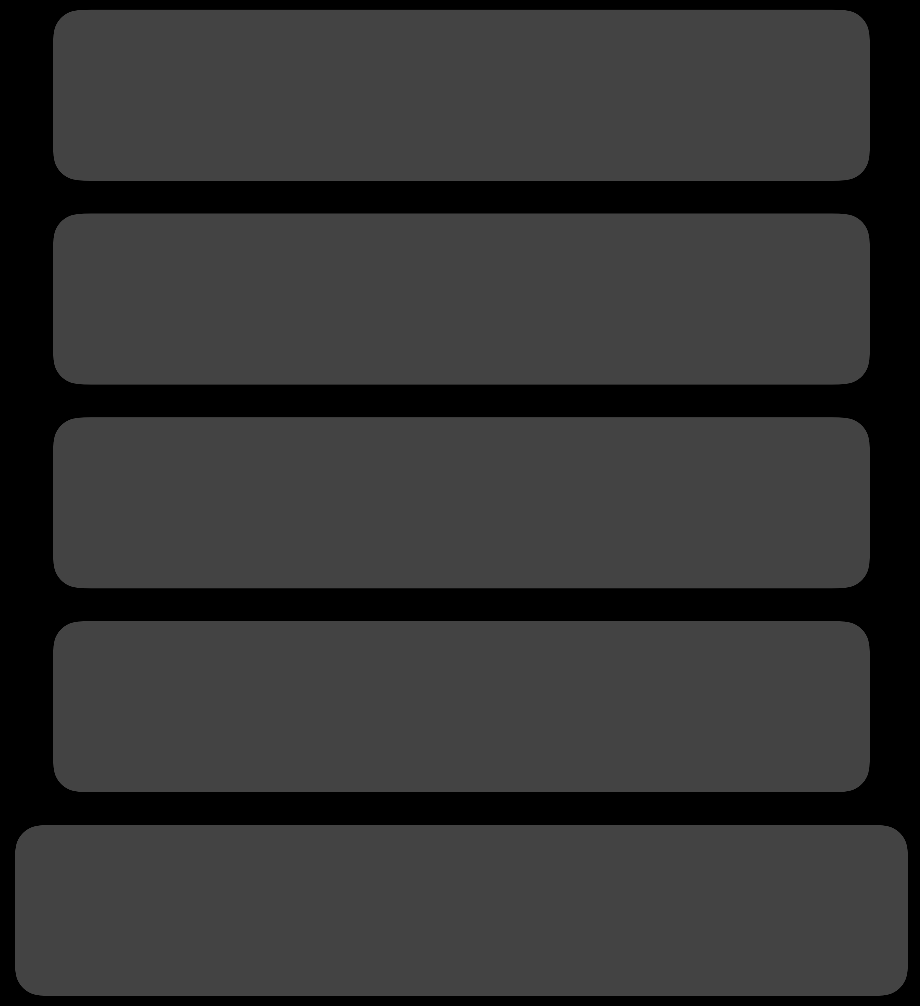
Existing Tools



Layer-oriented library

- Caffe, etc
    - Limited layer description language

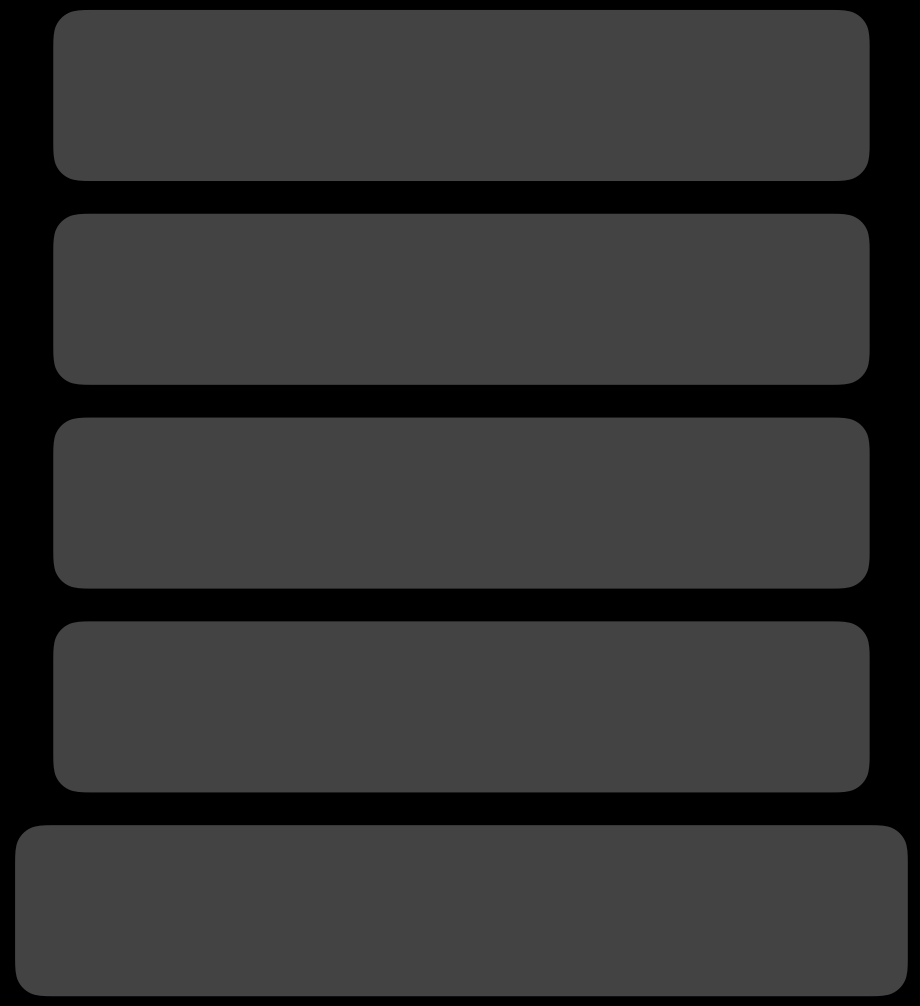
Existing Tools



Layer-oriented library

- Caffe, etc
  - Limited layer description language
  - Hard-coded gradients per layer

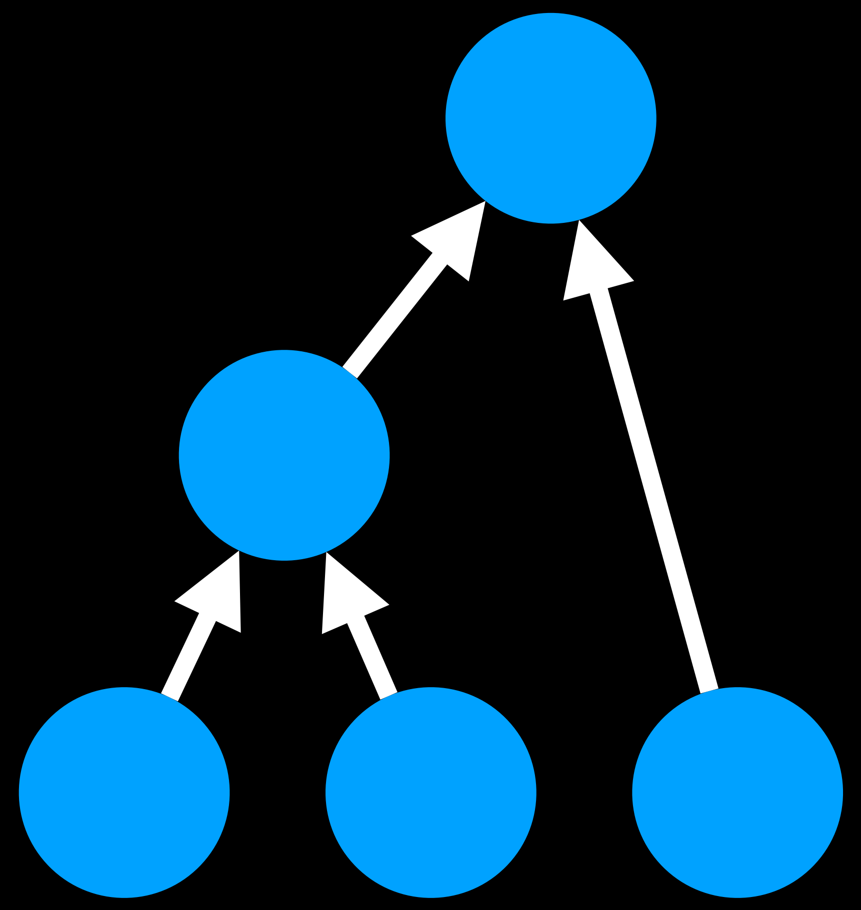
Existing Tools



Layer-oriented library

- Caffe, etc
  - Limited layer description language
  - Hard-coded gradients per layer
  - Cannot easily define custom computation

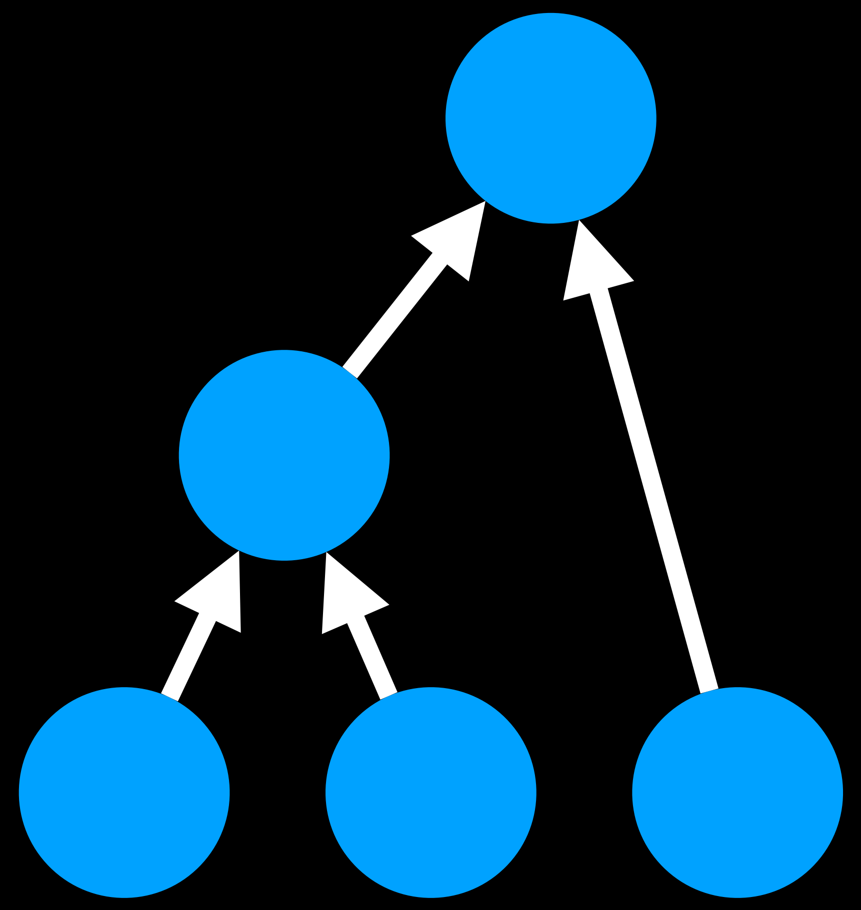
Existing Tools



Computation graph interpreter

* some but not all of the tools mentioned

Existing Tools

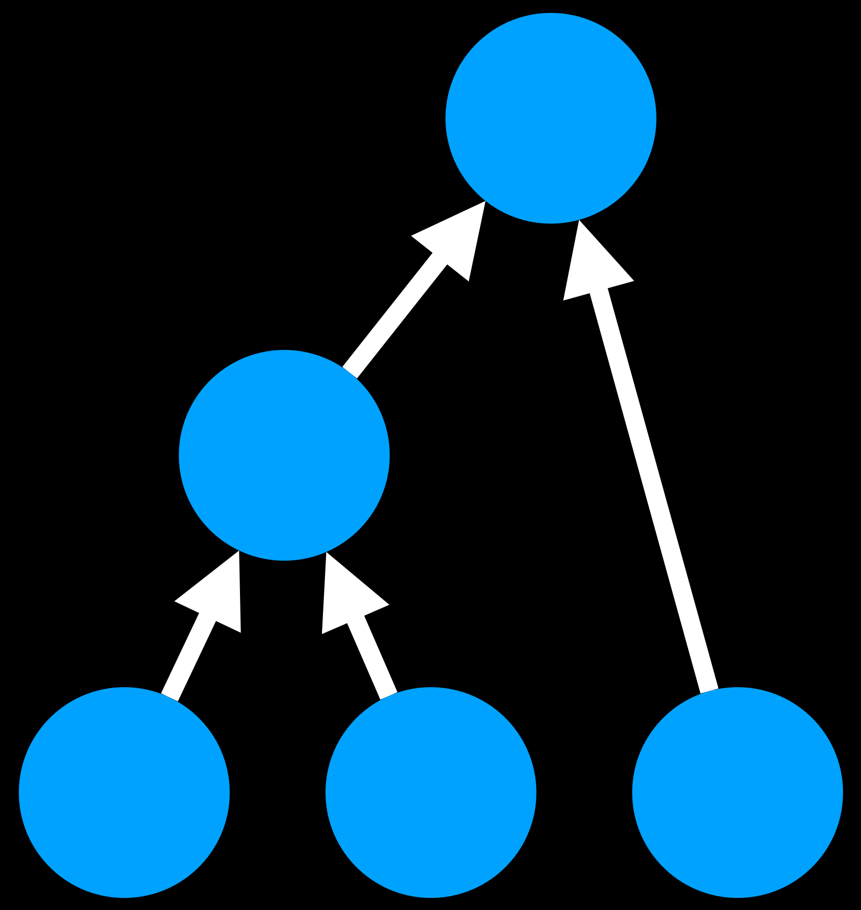


Computation graph interpreter

- Theano, TensorFlow, Torch, PyTorch, MXNet

* some but not all of the tools mentioned

Existing Tools

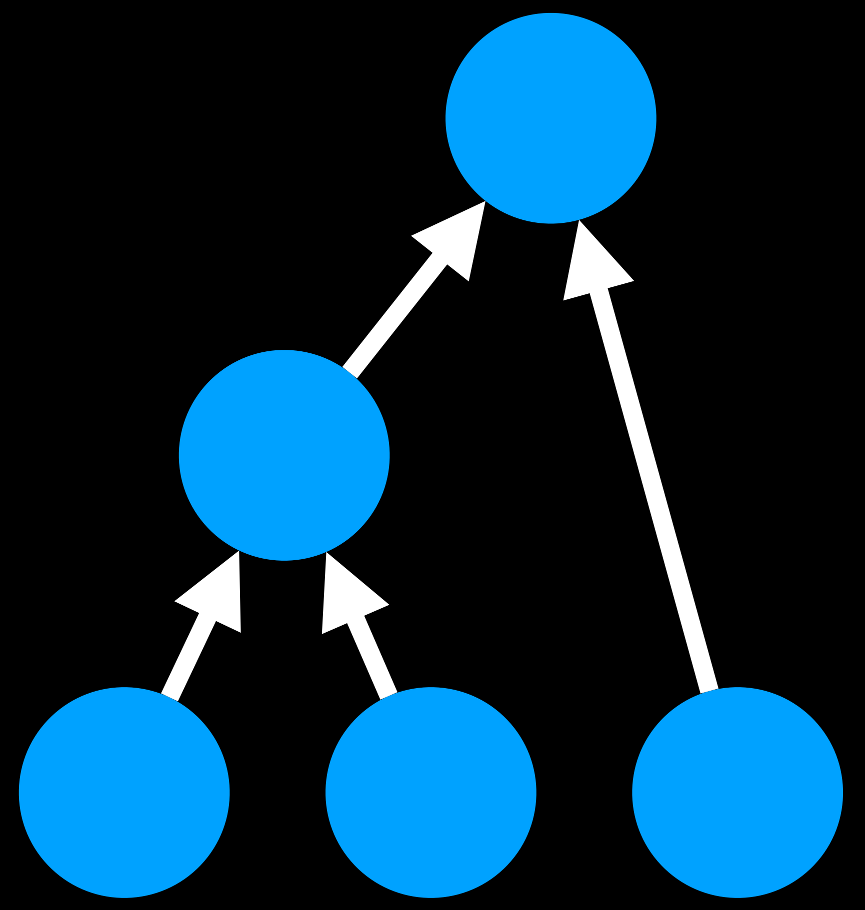


Computation graph interpreter

- Theano, TensorFlow, Torch, PyTorch, MXNet
  - One graph for everything

* some but not all of the tools mentioned

Existing Tools

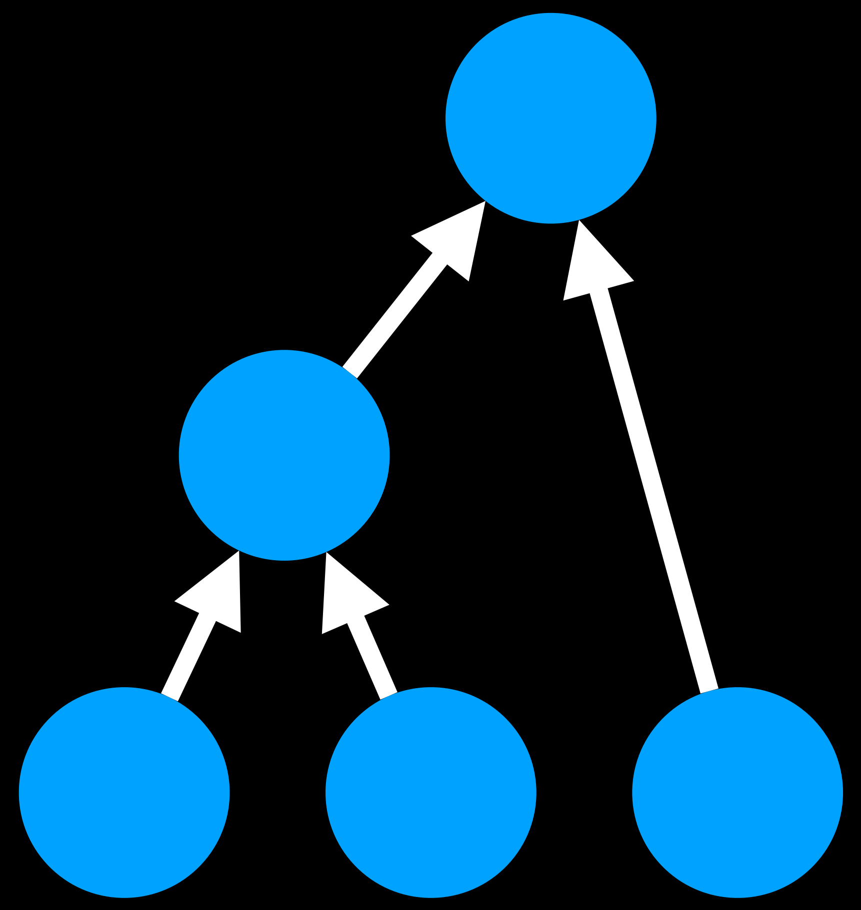


Computation graph interpreter

- Theano, TensorFlow, Torch, PyTorch, MXNet
  - One graph for everything
  - Embedded DSL in Python

* some but not all of the tools mentioned

Existing Tools

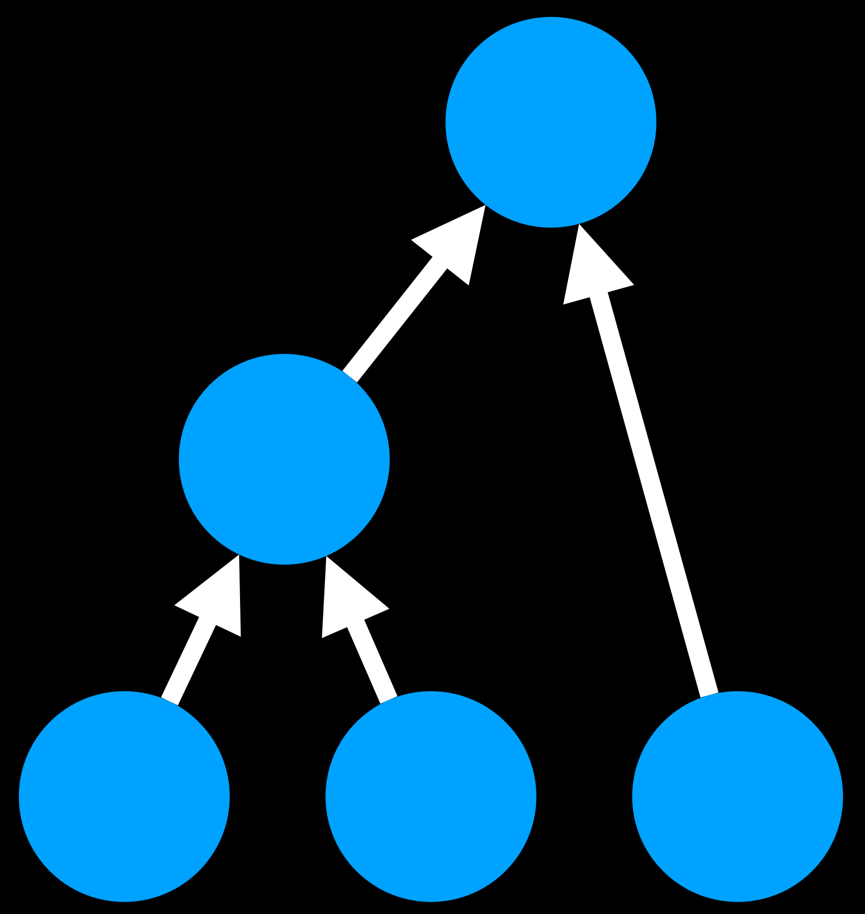


Computation graph interpreter

- Theano, TensorFlow, Torch, PyTorch, MXNet
  - One graph for everything
  - Embedded DSL in Python
  - Each node stores forward & backward results*

* some but not all of the tools mentioned

Existing Tools



Computation graph interpreter

- Theano, TensorFlow, Torch, PyTorch, MXNet
  - One graph for everything
  - Embedded DSL in Python
  - Each node stores forward & backward results*
  - One kernel for each node*

* some but not all of the tools mentioned

Programming Model

```
x = tf.placeholder(tf.float32, shape=[None, 200])
W = tf.Variable(tf.zeros([100, 50]))
b = tf.Variable(tf.zeros([50]))
y = tf.matmul(x, W) + b
```

Programming Model

```
x = tf.placeholder(tf.float32, shape=[None, 200])
W = tf.Variable(tf.zeros([100, 50]))
b = tf.Variable(tf.zeros([50]))
y = tf.matmul(x, W) + b
```

```
ValueError: Dimensions must be equal, but are
200 and 100 for 'MatMul' (op: 'MatMul') with
input shapes: [?,200], [100,50].
```

Programming Model

```
x = tf.placeholder(tf.float32, shape=[None, 200])
W = tf.Variable(tf.zeros([100, 50]))
b = tf.Variable(tf.zeros([50]))
y = tf.matmul(x, W) + b
```

```
ValueError: Dimensions must be equal, but are
200 and 100 for 'MatMul' (op: 'MatMul') with
input shapes: [?,200], [100,50].
```

Lack of safety

Programming Model

```
 fuse = edge_pool.apply(run_caffe, [src])
    fuse = fuse[border:-border, border:-border]
    with tempfile.File(suffix=".png") as png_file,
tempfile.NamedTemporaryFile(suffix=".mat") as mat_file:
        scipy.io.savemat(mat_file.name, {"input": fuse})
        octave_code = r"""
E = 1-load(input_path).input;
E = imresize(E, [image_width,image_width]);
E = 1 - E;
E = single(E);
[Ox, Oy] = gradient(convTri(E, 4), 1);
[Oxx, ~] = gradient(Ox, 1);
[Oxy, Oyy] = gradient(Oy, 1);
O = mod(atan(Oyy .* sign(-Oxy) ./ (Oxx + 1e-5)), pi);
E = edgesNmsMex(E, O, 1, 5, 1.01, 1);
...
```

Production code

Programming Model

```
 fuse = edge_pool.apply(run_caffe, [src])
    fuse = fuse[border:-border, border:-borde
    with tempfile.File(suffix=".png") as pn
tempfile.NamedTemporaryFile(suffix=".mat") a
        scipy.io.savemat(mat_file.name, {"inp
        octave_code = r"""
E = 1-load(input_path).input;
E = imresize(E, [image_width,image_width]);
E = 1 - E;
E = single(E);
[Ox, Oy] = gradient(convTri(E, 4), 1);
[Oxx, ~] = gradient(Ox, 1);
[Oxy, Oyy] = gradient(Oy, 1);
O = mod(atan(Oyy .* sign(-Oxy) ./ (Oxx + 1e-5)), pi);
E = edgesNmsMex(E, O, 1, 5, 1.01, 1);
...
```

GNU Octave

Production code

Programming Model

```
 fuse = edge_pool.apply(run_caffe, [src])
    fuse = fuse[border:-border, border:-borde
    with tempfile.File(suffix=".png") as pn
tempfile.NamedTemporaryFile(suffix=".mat") a
        scipy.io.savemat(mat_file.name, {"inp
        octave_code = r"""
E = 1-load(input_path).input;
E = imresize(E, [image_width,image_width]);
E = 1 - E;
E = single(E);
[Ox, Oy] = gradient(convTri(E, 4), 1);
[Oxx, ~] = gradient(Ox, 1);
[Oxy, Oyy] = gradient(Oy, 1);
O = mod(atan(Oyy .* sign(-Oxy) ./ (Oxx + 1e-5)), pi);
E = edgesNmsMex(E, O, 1, 5, 1.01, 1);
...
```

GNU Octave

Production code

- Developers ignore safety

Programming Model

```
 fuse = edge_pool.apply(run_caffe, [src])
    fuse = fuse[border:-border, border:-borde
    with tempfile.File(suffix=".png") as pn
tempfile.NamedTemporaryFile(suffix=".mat") a
        scipy.io.savemat(mat_file.name, {"inp
        octave_code = r"""
E = 1-load(input_path).input;
E = imresize(E, [image_width,image_width]);
E = 1 - E;
E = single(E);
[Ox, Oy] = gradient(convTri(E, 4), 1);
[Oxx, ~] = gradient(Ox, 1);
[Oxy, Oyy] = gradient(Oy, 1);
O = mod(atan(Oyy .* sign(-Oxy) ./ (Oxx + 1e-5)), pi);
E = edgesNmsMex(E, O, 1, 5, 1.01, 1);
...
```

GNU Octave

Production code

- Developers ignore safety
- Failures are impenetrable

Programming Model

```
 fuse = edge_pool.apply(run_caffe, [src])
    fuse = fuse[border:-borde
    with tempfile.File(suffix=".png") as pn
tempfile.NamedTemporaryFile(suffix=".mat") a
        scipy.io.savemat(mat_file.name, {"inp
        octave_code = r"""
E = 1-load(input_path).input;
E = imresize(E, [image_width,image_width]);
E = 1 - E;
E = single(E);
[Ox, Oy] = gradient(convTri(E, 4), 1);
[Oxx, ~] = gradient(Ox, 1);
[Oxy, Oyy] = gradient(Oy, 1);
O = mod(atan(Oyy .* sign(-Oxy) ./ (Oxx + 1e-5)), pi);
E = edgesNmsMex(E, O, 1, 5, 1.01, 1);
...
```

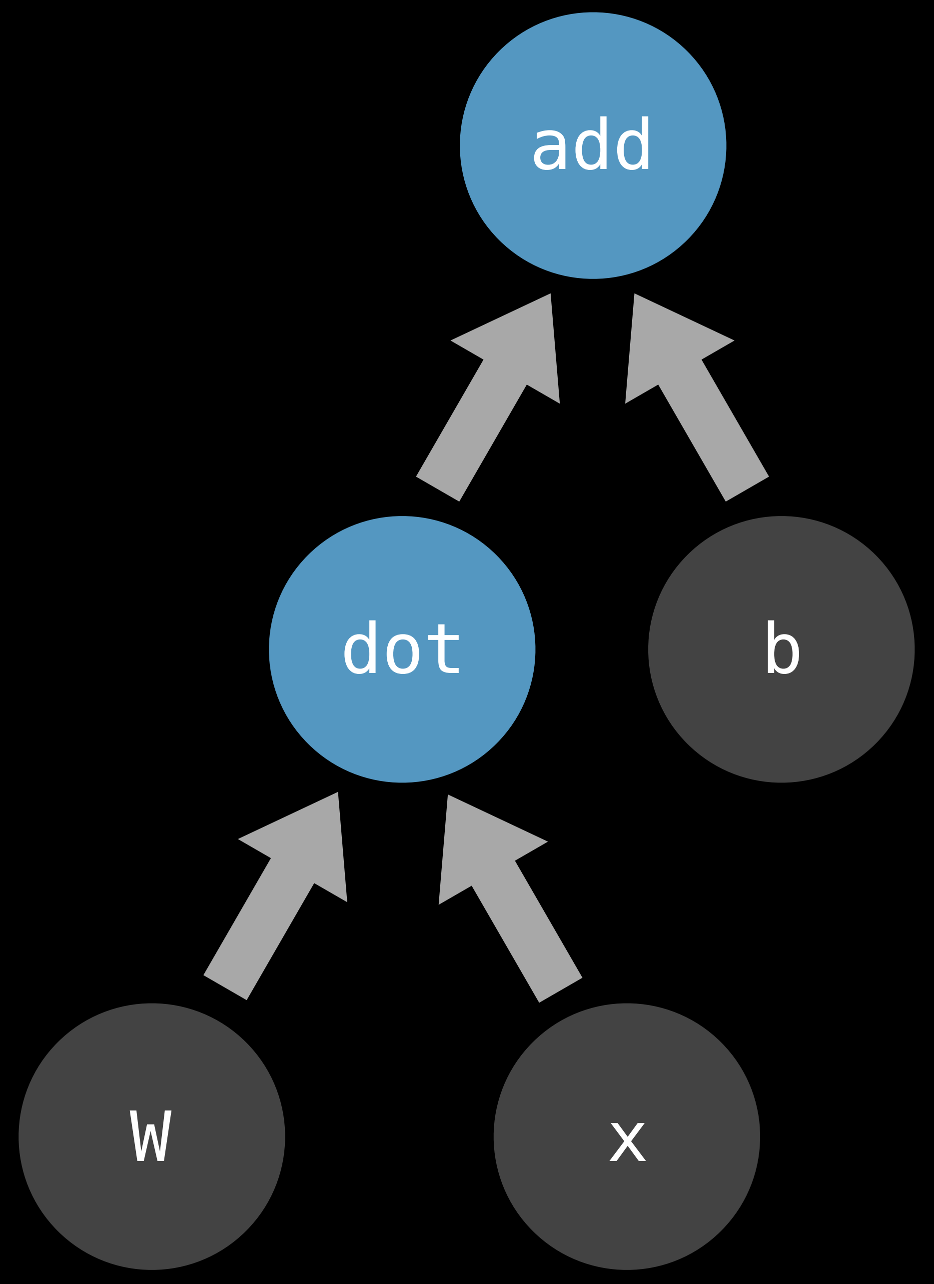
GNU Octave

Production code

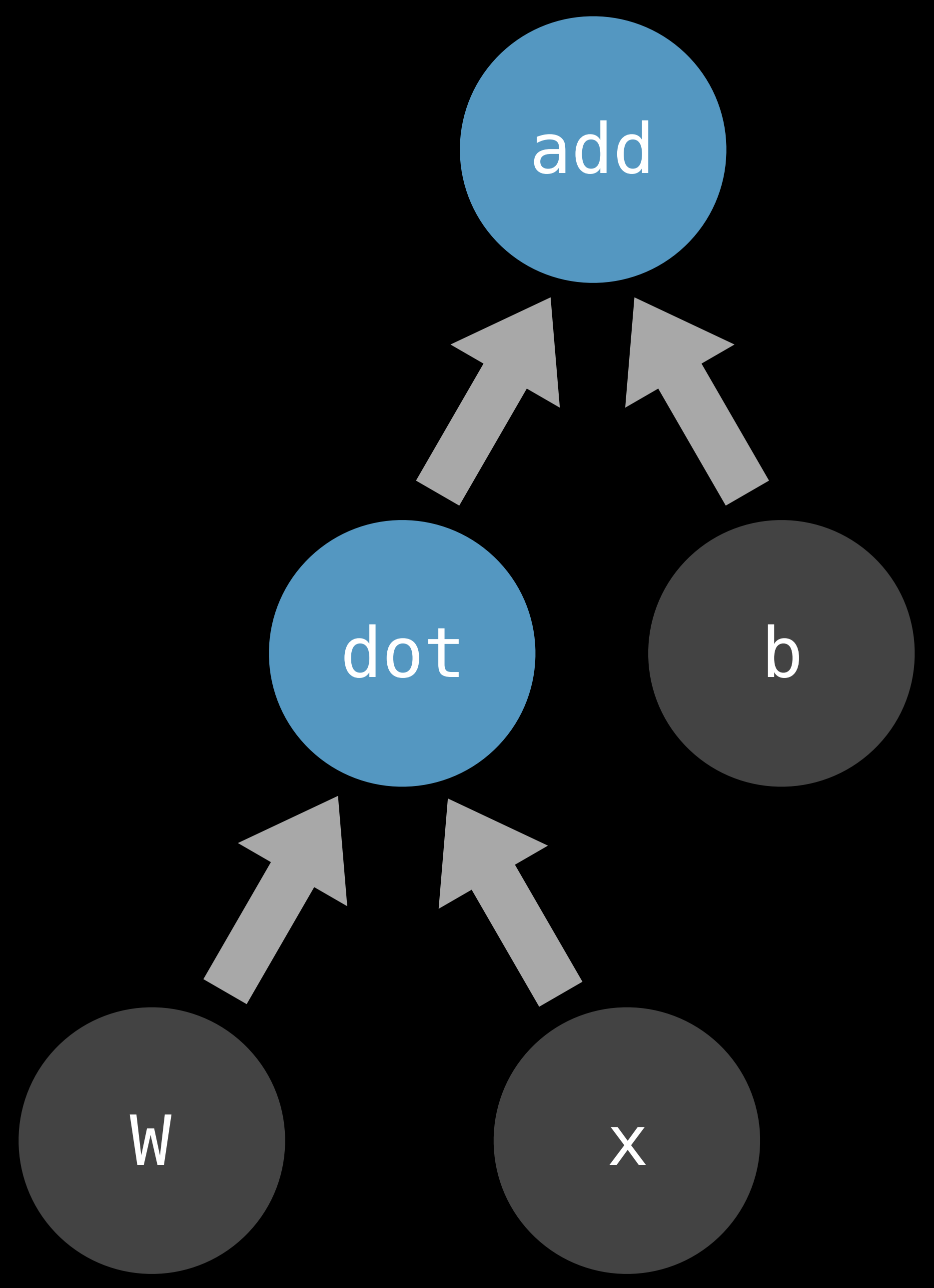
- Developers ignore safety
- Failures are impenetrable
- Software engineering is gone!

Automatic Differentiation

Automatic Differentiation

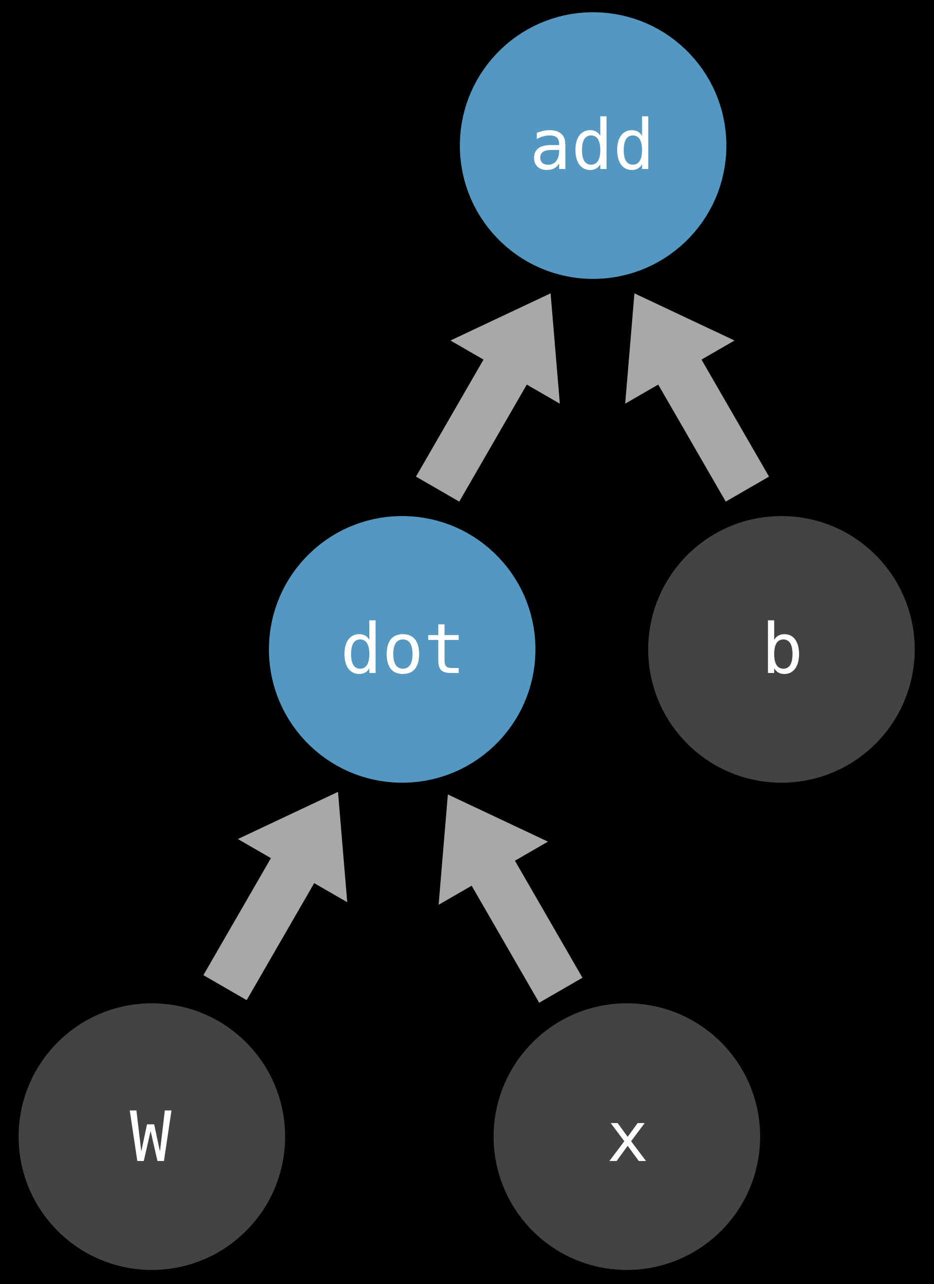


Automatic Differentiation



$$f(x, W, b) = Wx + b$$

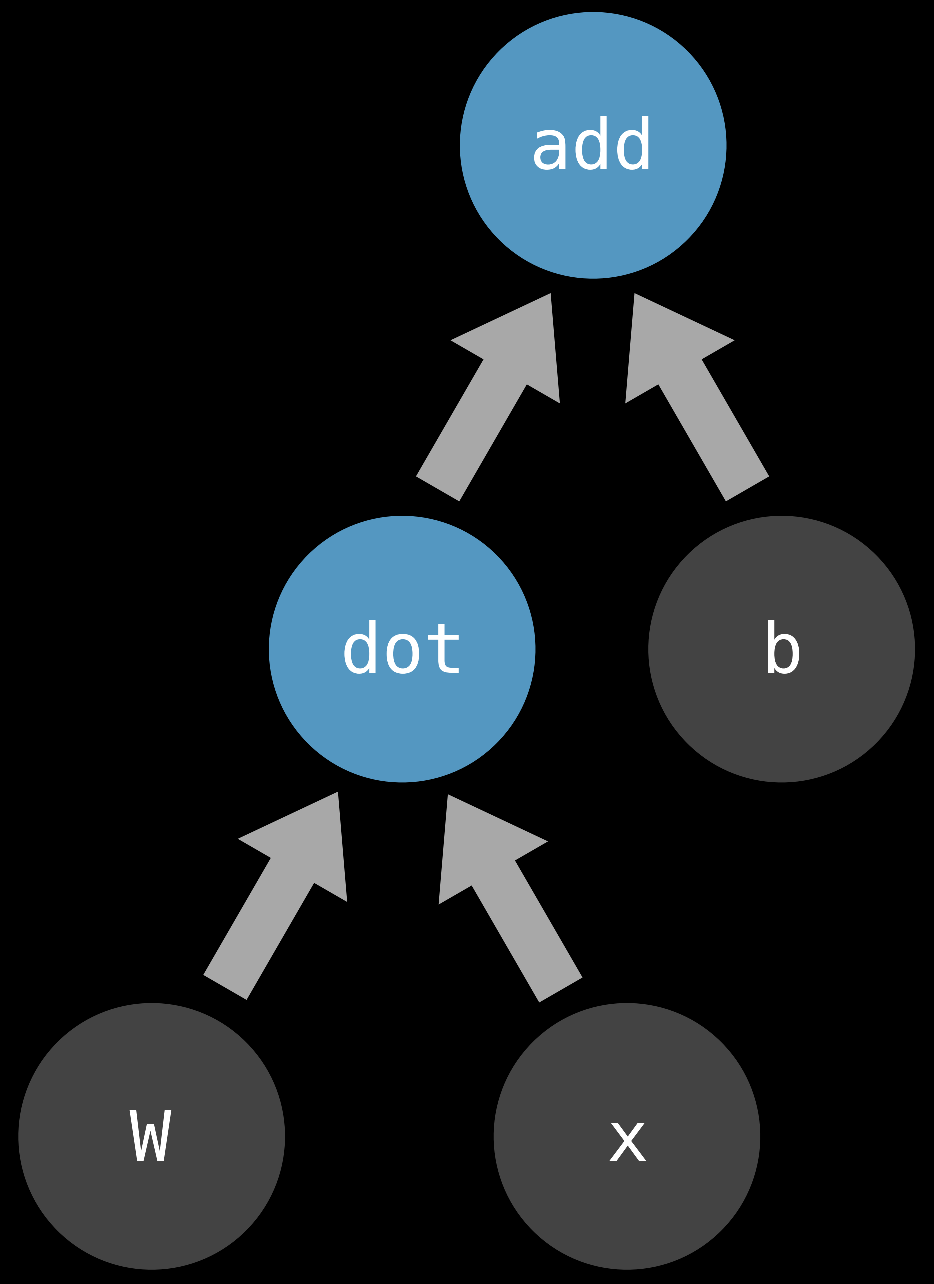
Automatic Differentiation



$$f(x, W, b) = Wx + b$$

$$\frac{\partial f}{\partial W} = x^T 1$$

Automatic Differentiation

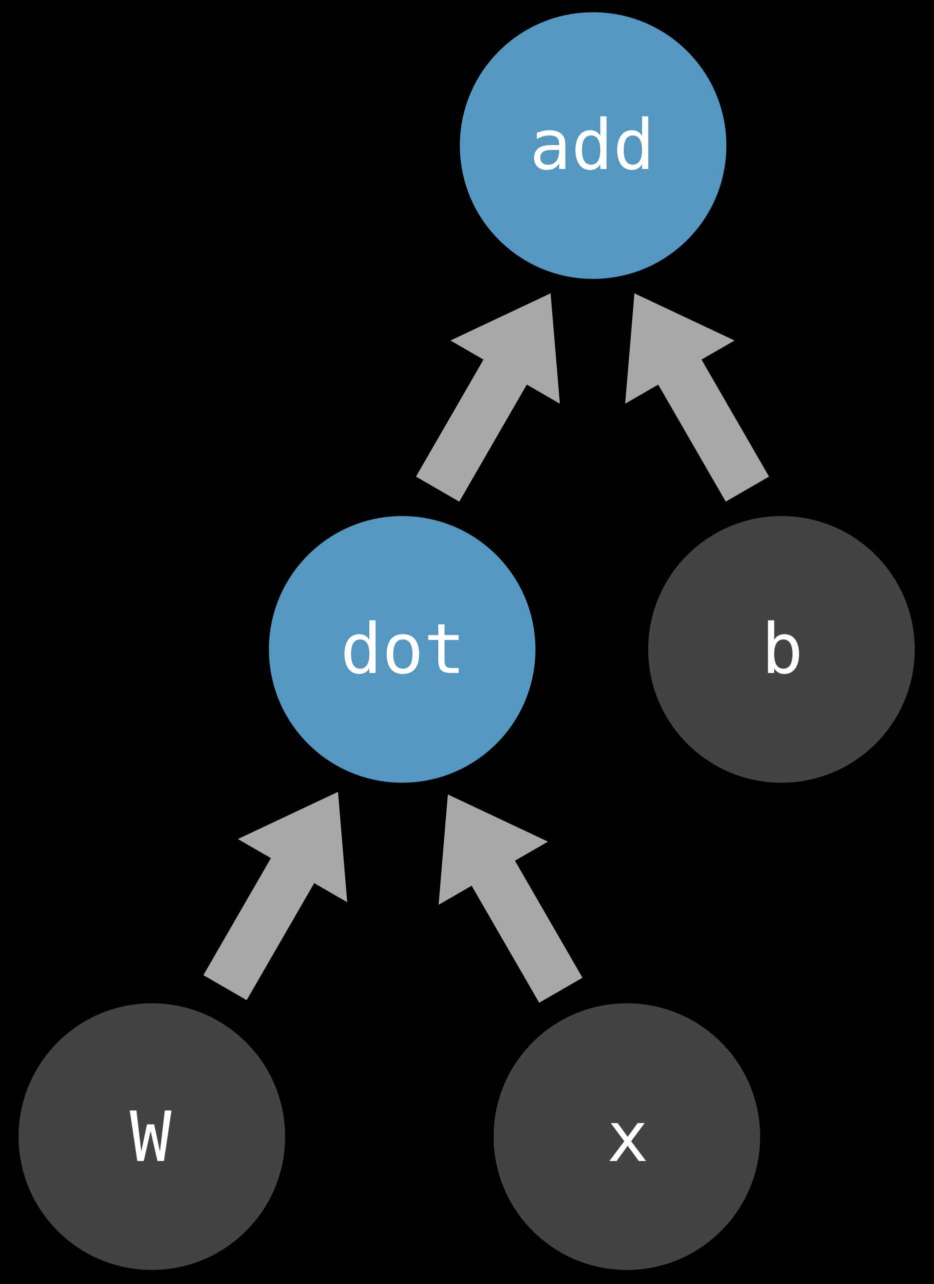


$$f(x, W, b) = Wx + b$$

$$\frac{\partial f}{\partial W} = x^T 1$$

$$\frac{\partial f}{\partial b} = 1$$

Automatic Differentiation



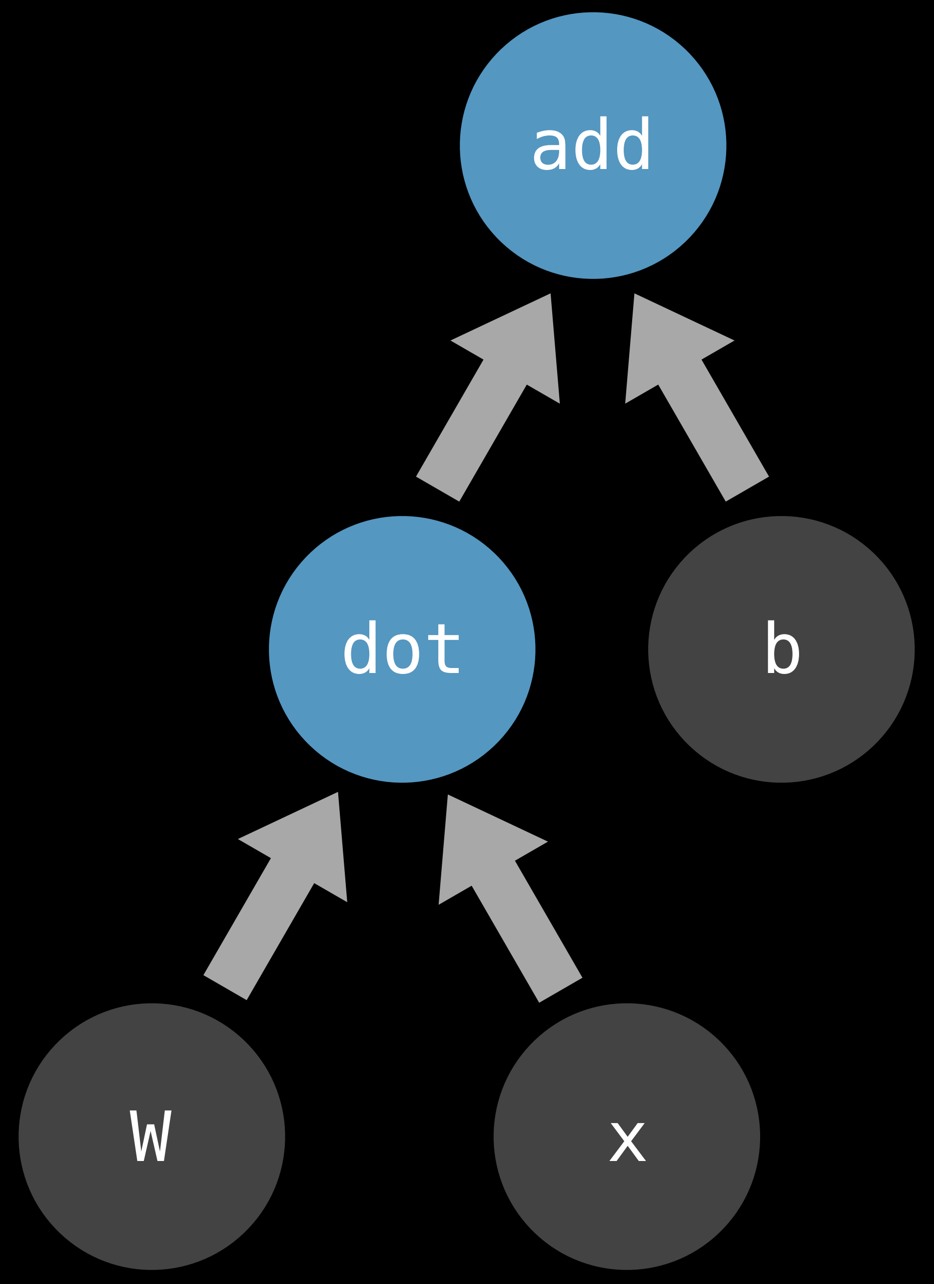
$$f(x, W, b) = Wx + b$$

$$\frac{\partial f}{\partial W} = x^T 1$$

$$\frac{\partial f}{\partial b} = 1$$

- Redundant computation

Automatic Differentiation



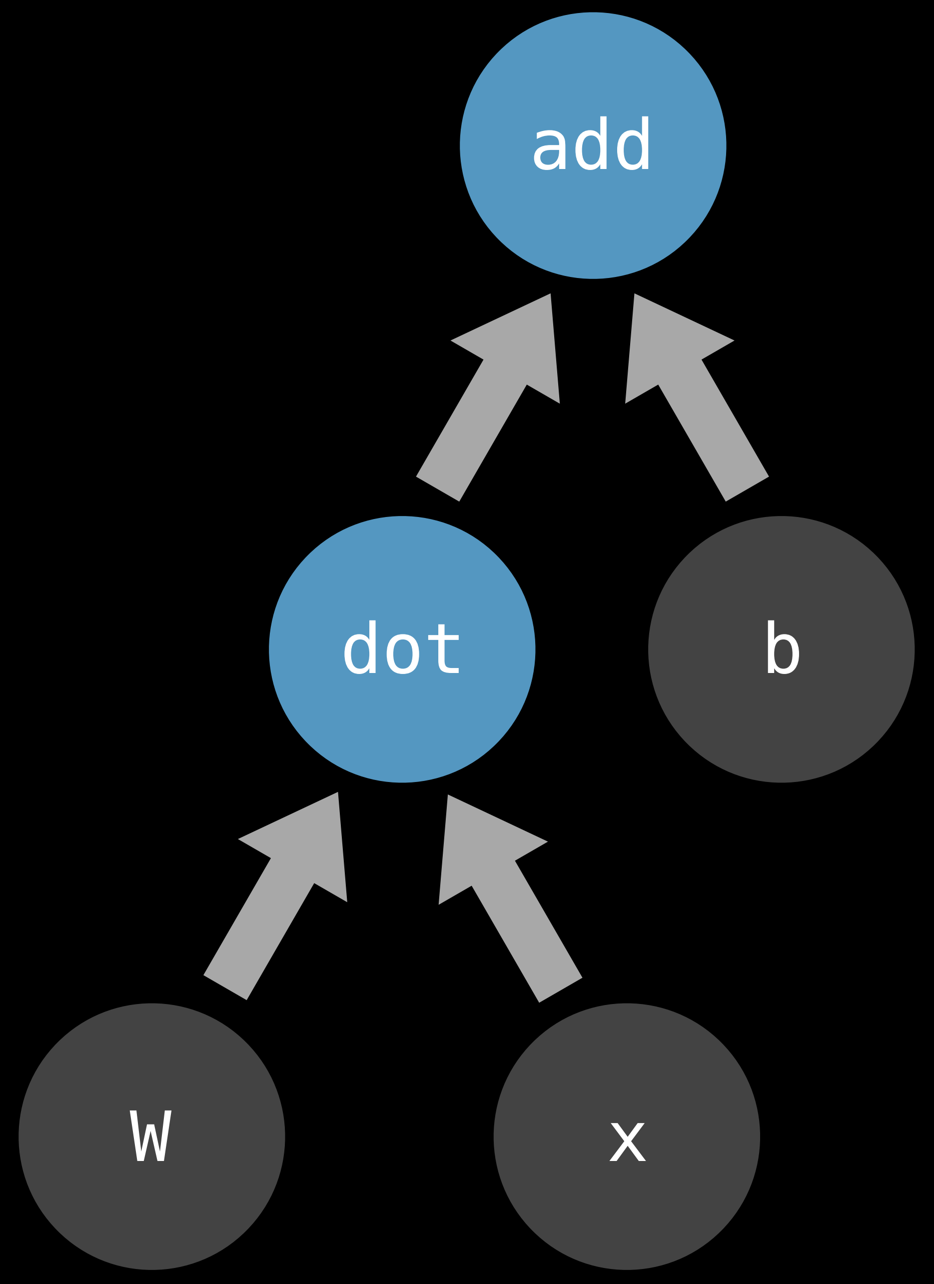
$$f(x, W, b) = Wx + b$$

$$\frac{\partial f}{\partial W} = x^T 1$$

$$\frac{\partial f}{\partial b} = 1$$

- Redundant computation
- Occupies large memory

Automatic Differentiation



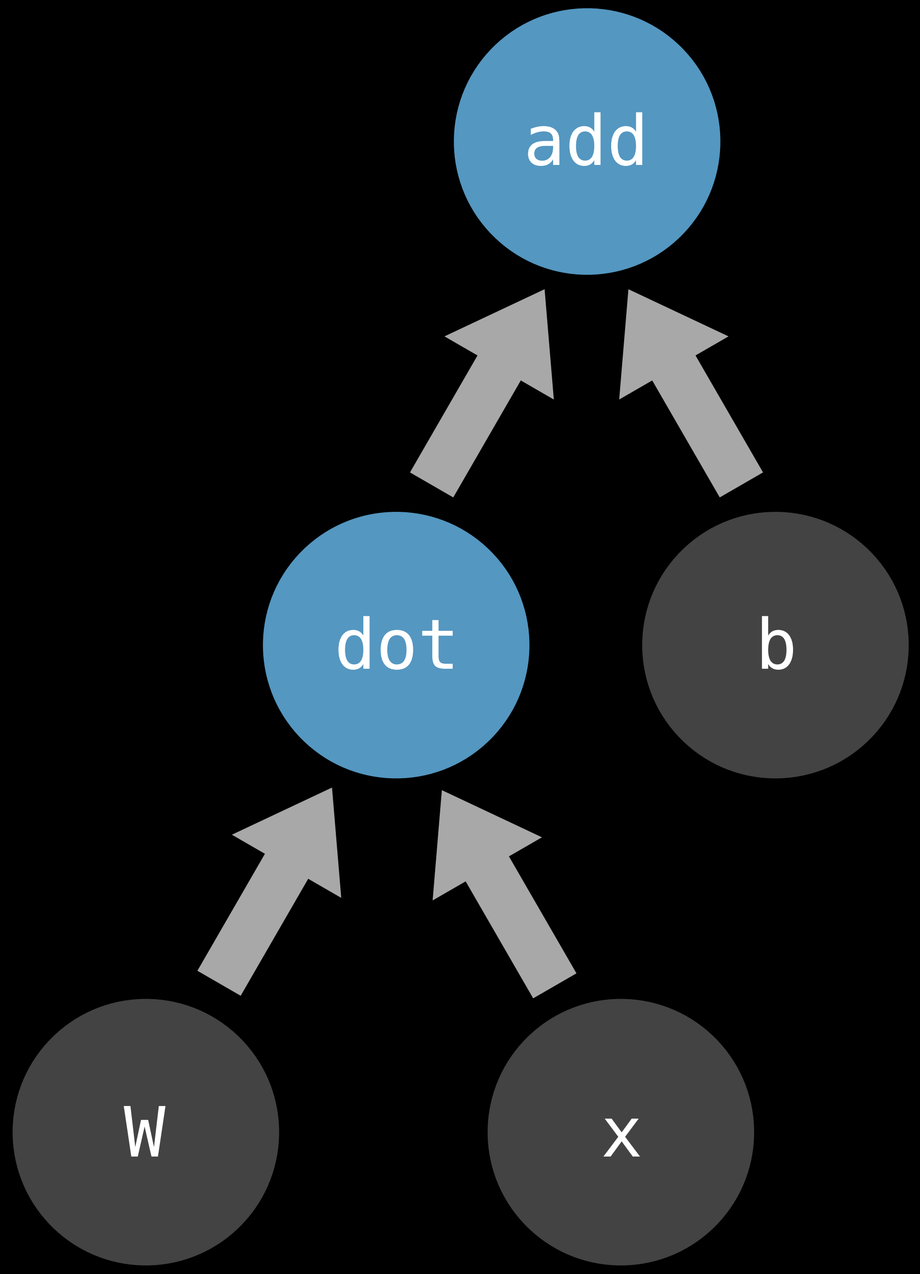
$$f(x, W, b) = Wx + b$$

$$\frac{\partial f}{\partial W} = x^T 1$$

$$\frac{\partial f}{\partial b} = 1$$

- Redundant computation
- Occupies large memory
- Hard to compute Hessian, or higher-order derivatives

Automatic Differentiation



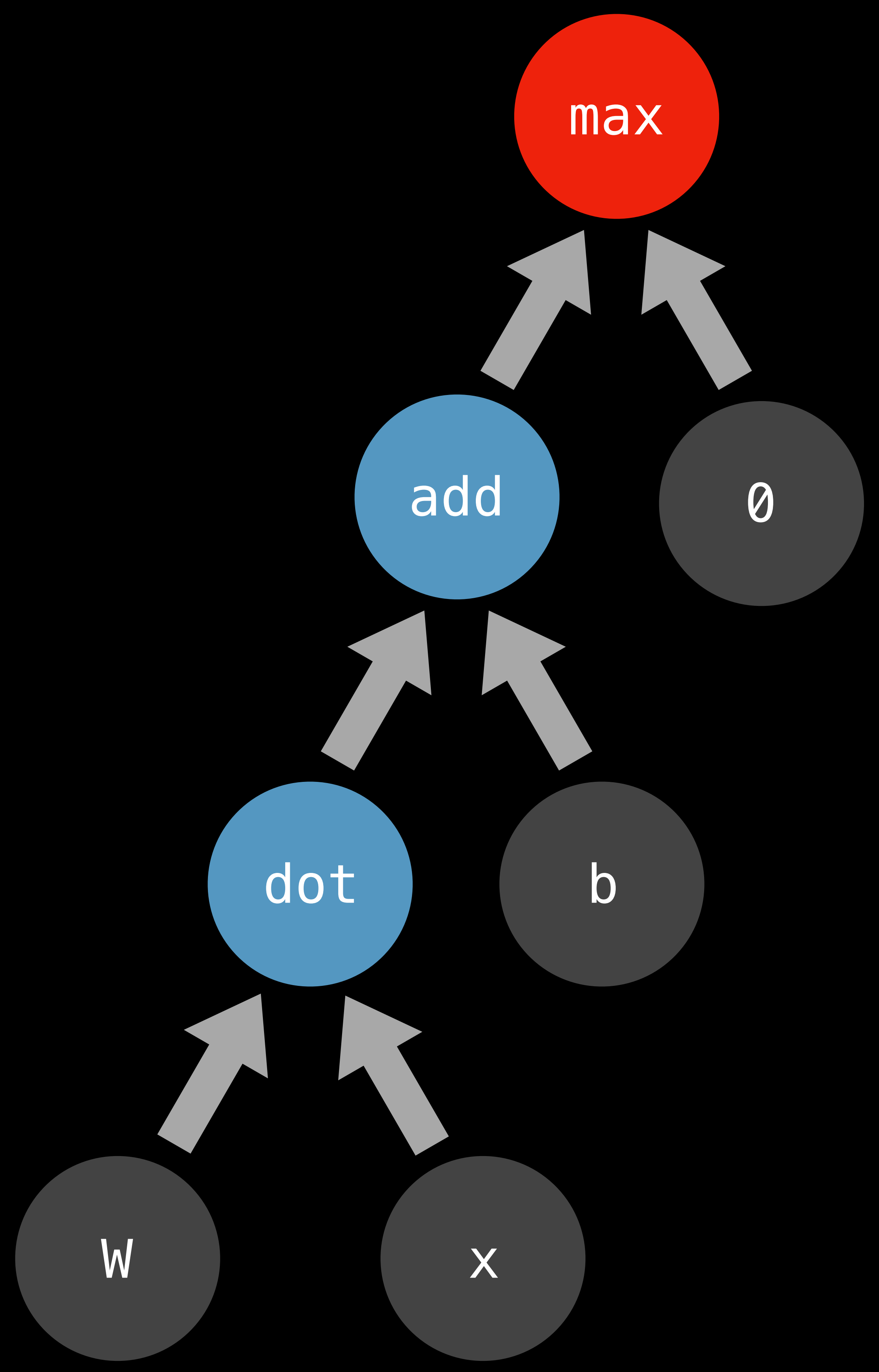
$$f(x, W, b) = Wx + b$$

$$\frac{\partial f}{\partial W} = x^T 1$$

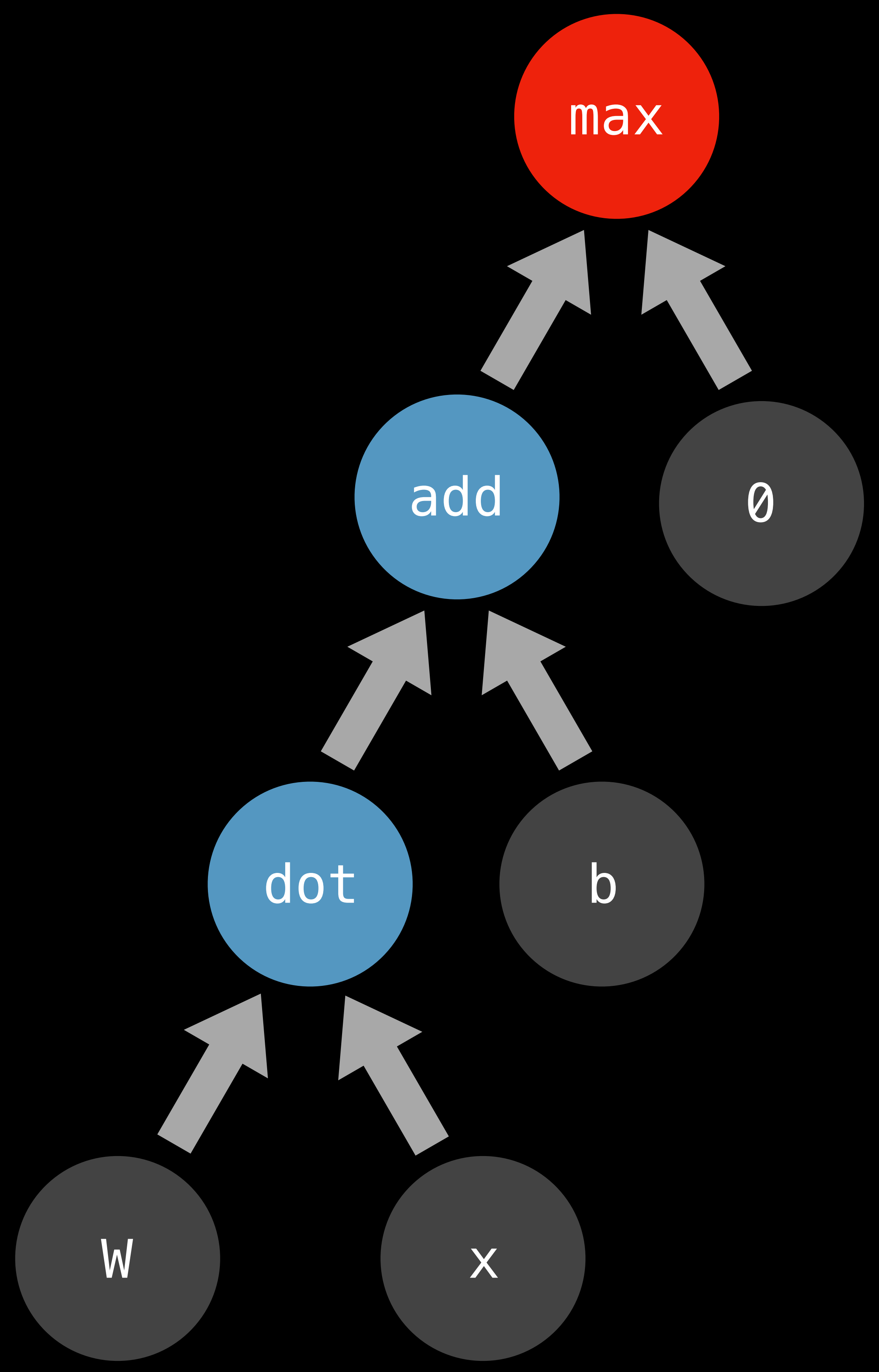
$$\frac{\partial f}{\partial b} = 1$$

- Redundant computation
- Occupies large memory
- Hard to compute Hessian, or higher-order derivatives
- Graph optimizations don't apply to backward computation

Automatic Differentiation

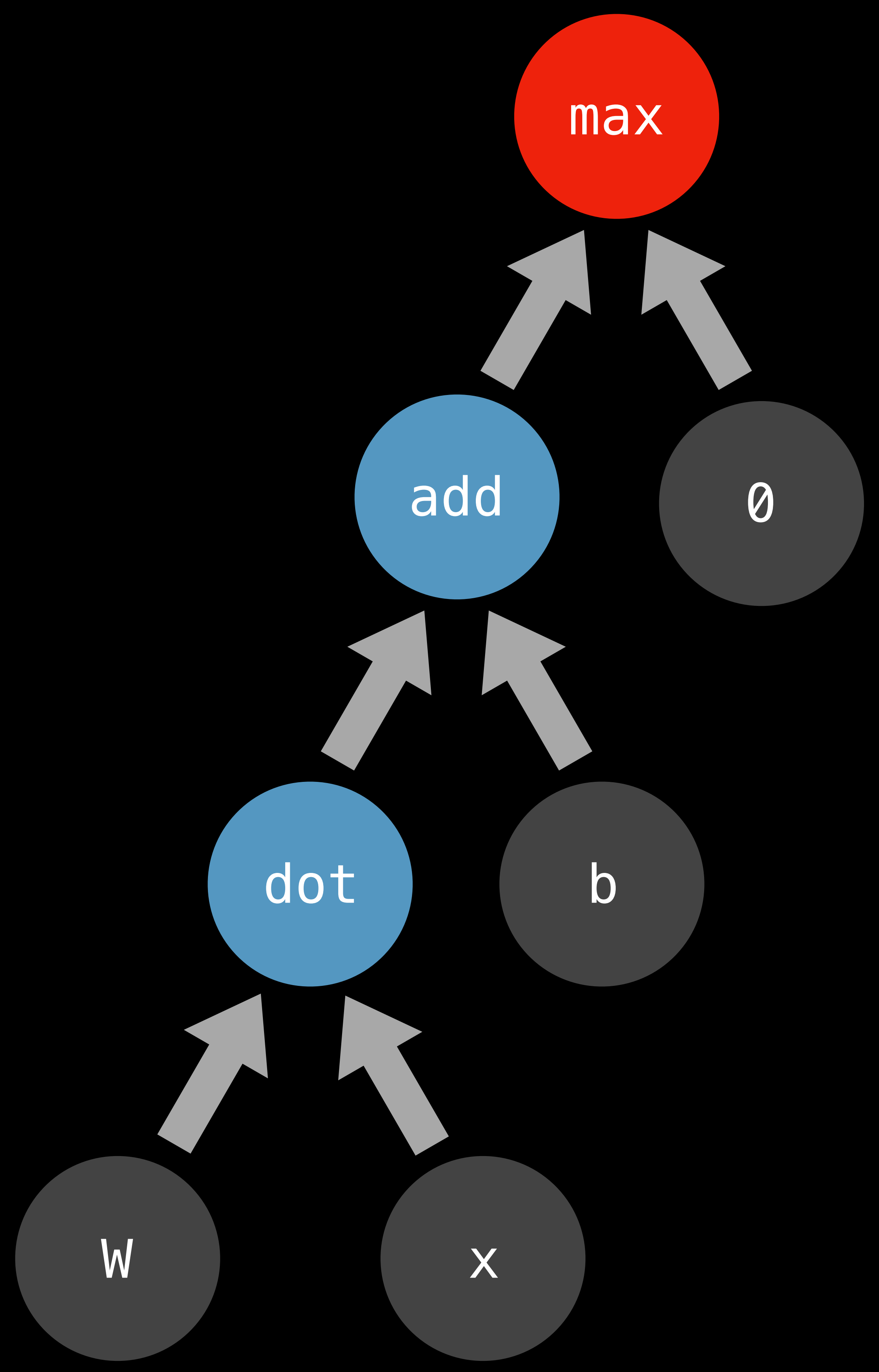


Automatic Differentiation



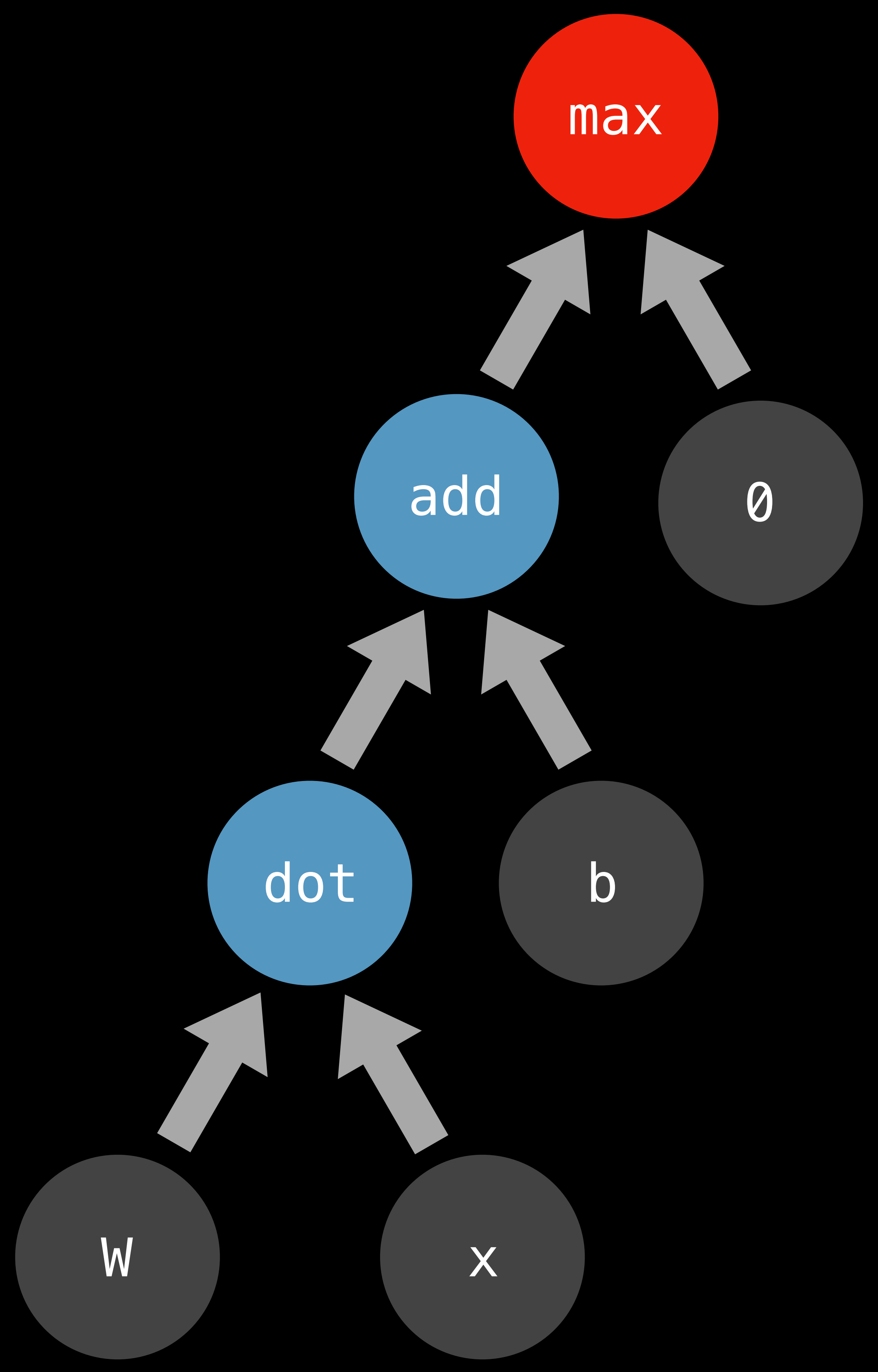
- Sea-of-nodes representation

Automatic Differentiation



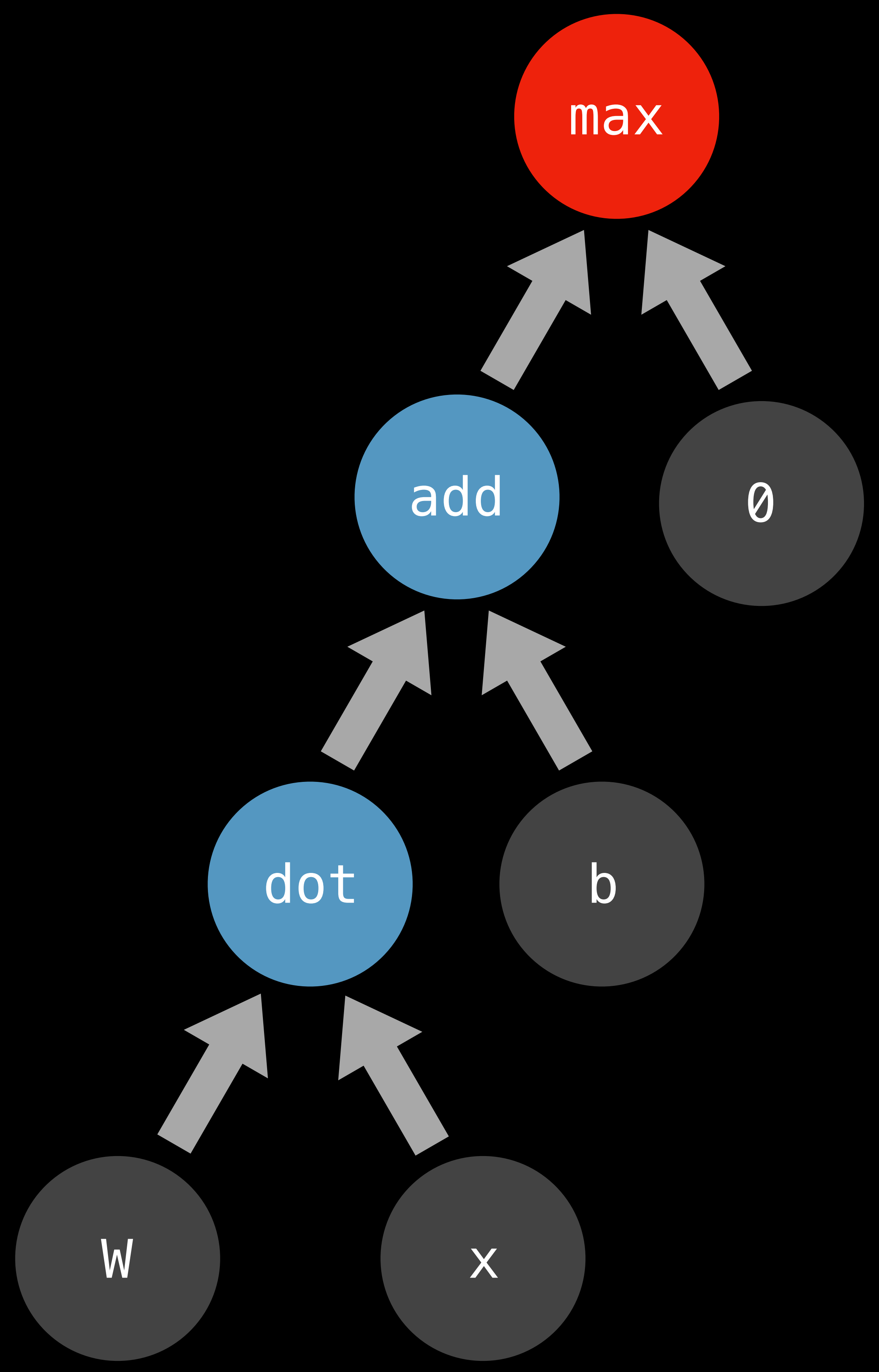
- Sea-of-nodes representation
- No control flow graph

Automatic Differentiation



- Sea-of-nodes representation
- No control flow graph
- Composite functions

Automatic Differentiation



- Sea-of-nodes representation
- No control flow graph
- Composite functions
- Runtime “tape” to define evaluation order

Backend

```
__global__ void tanh_float(const float *in, float *out, int count)
{
    int idx = blockIdx.x + blockDim.x + threadIdx.x;
    if (idx < count)
        out[idx] = tanh(in[idx]);
}
```

Backend

```
__global__ void tanh_float(const float *in, float *out, int count)
{
    int idx = blockIdx.x + blockDim.x + threadIdx.x;
    if (idx < count)
        out[idx] = tanh(in[idx]);
}
```

Boundary check → `if (idx < count)`

Backend

```
__global__ void tanh_float(const float *in, float *out, int count)
{
    int idx = blockIdx.x + blockDim.x + threadIdx.x;
    if (idx < count)
        out[idx] = tanh(in[idx]);
}
```

Boundary check → `if (idx < count)`

Launched with `tanh_float<<<GRID_SIZE, BLOCK_SIZE>>>`

Backend

```
__global__ void tanh_float(const float *in, float *out, int count)
{
    int idx = blockIdx.x + blockDim.x + threadIdx.x;
    if (idx < count)
        out[idx] = tanh(in[idx]);
}
```

Boundary check → `if (idx < count)`

Launched with `tanh_float<<<GRID_SIZE, BLOCK_SIZE>>>`

Boundary check is redundant when `GRID_SIZE * BLOCK_SIZE == count`

Backend

```
__global__ void tanh_float(const float *in, float *out, int count)
{
    int idx = blockIdx.x + blockDim.x + threadIdx.x;
    if (idx < count)
        out[idx] = tanh(in[idx]);
}

__global__ void tanh_double(const double *in, double *out, int count)
{
    int idx = blockIdx.x + blockDim.x + threadIdx.x;
    if (idx < count)
        out[idx] = tanh(in[idx]);
}
```

Need to precompile everything at install time

Existing Toolkits

TensorFlow

PyTorch

torch

Existing Toolkits

TensorFlow

PyTorch

torch

- Unsafe

Existing Toolkits

TensorFlow

PyTorch

torch

- Unsafe
- Differentiation by interpretation

Existing Toolkits

TensorFlow

PyTorch

torch

- Unsafe
- Differentiation by interpretation
- Hard-coded GPU kernels

Existing Toolkits

TensorFlow

PyTorch

torch

- Unsafe
- Differentiation by interpretation
- Hard-coded GPU kernels
- Mostly CUDA only

Existing Toolkits

TensorFlow

PyTorch

torch

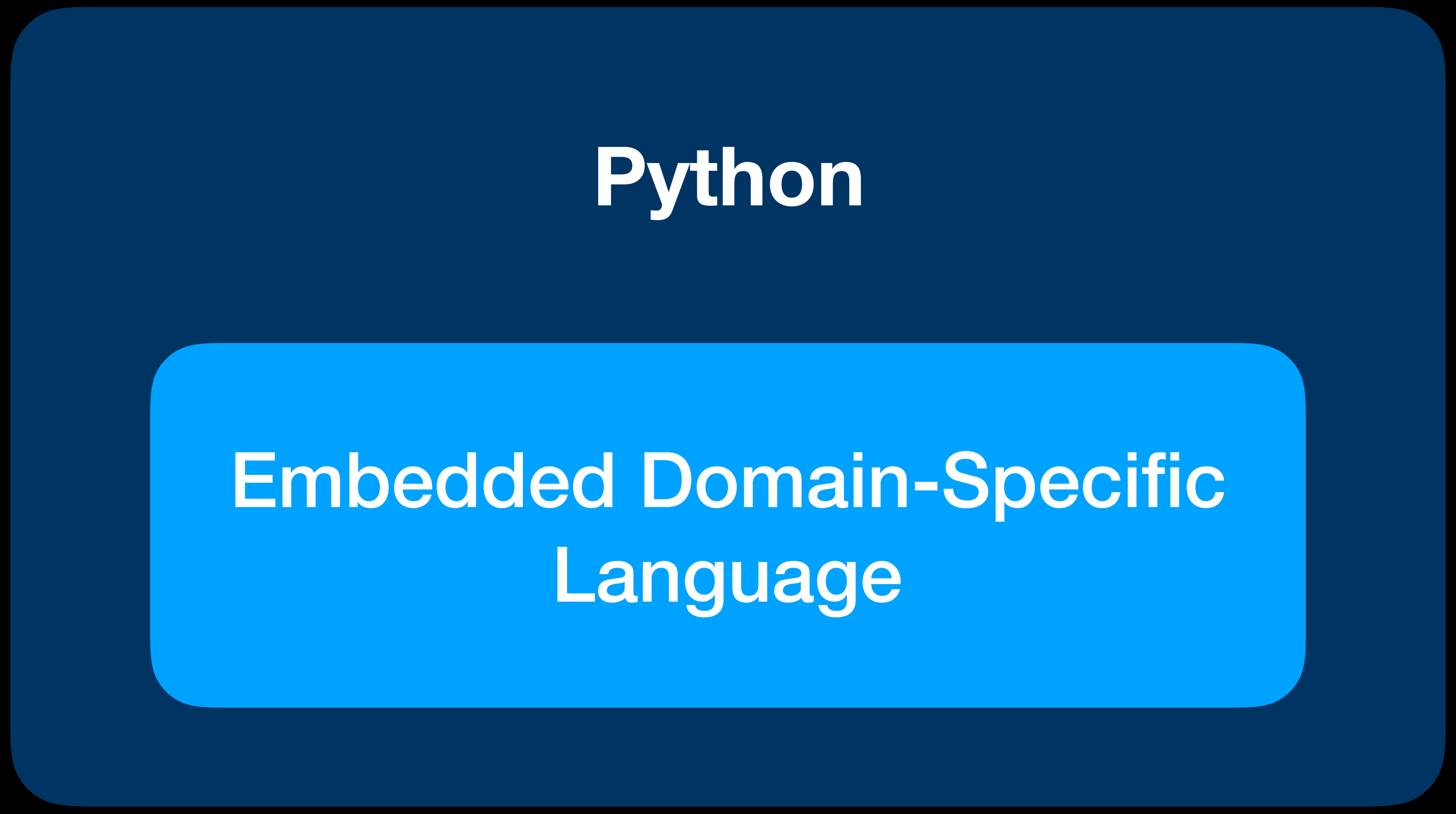
- Unsafe
- Differentiation by interpretation
- Hard-coded GPU kernels
- Mostly CUDA only
- Lack of software engineering!

Existing Toolkits

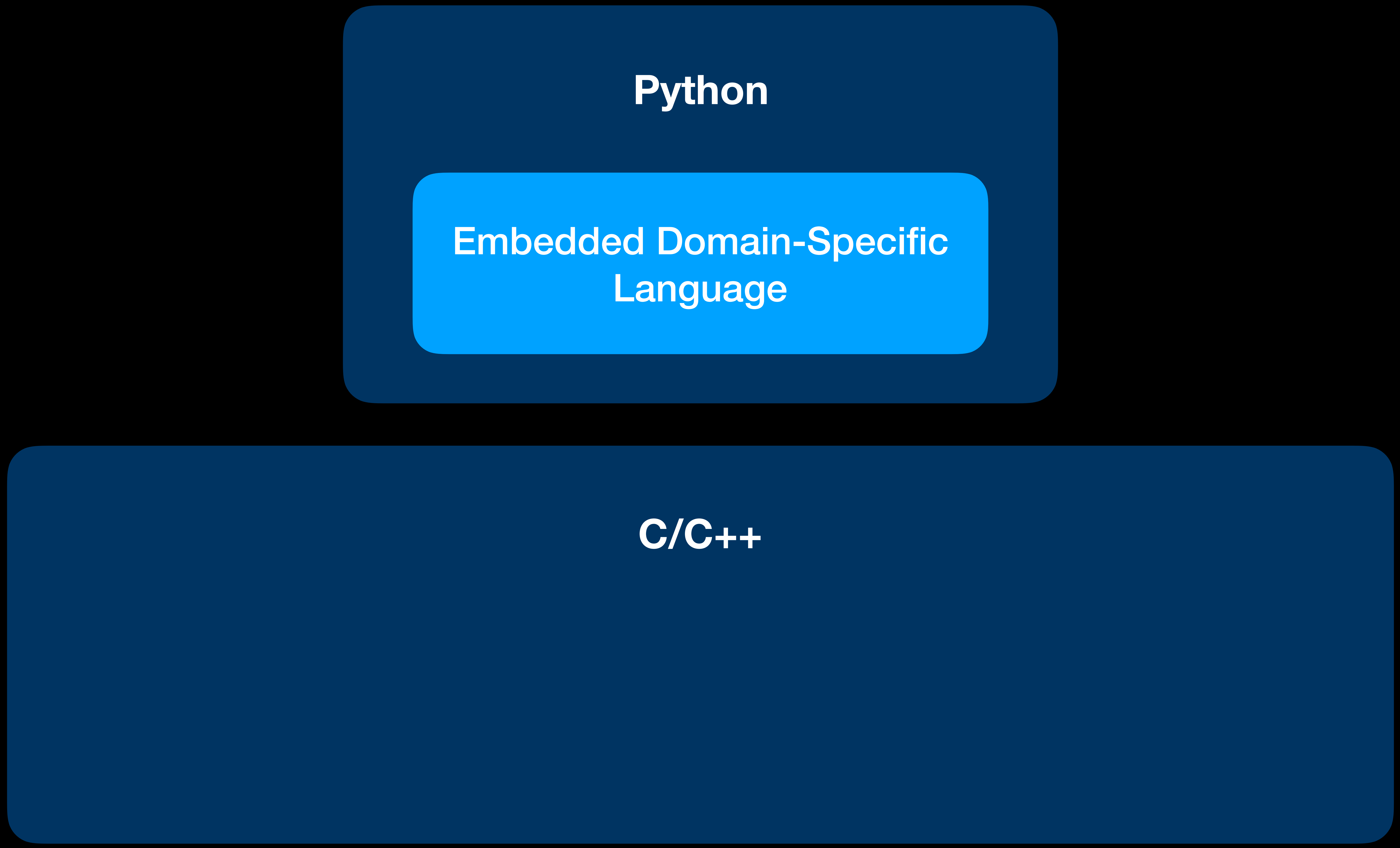
Existing Toolkits

Python

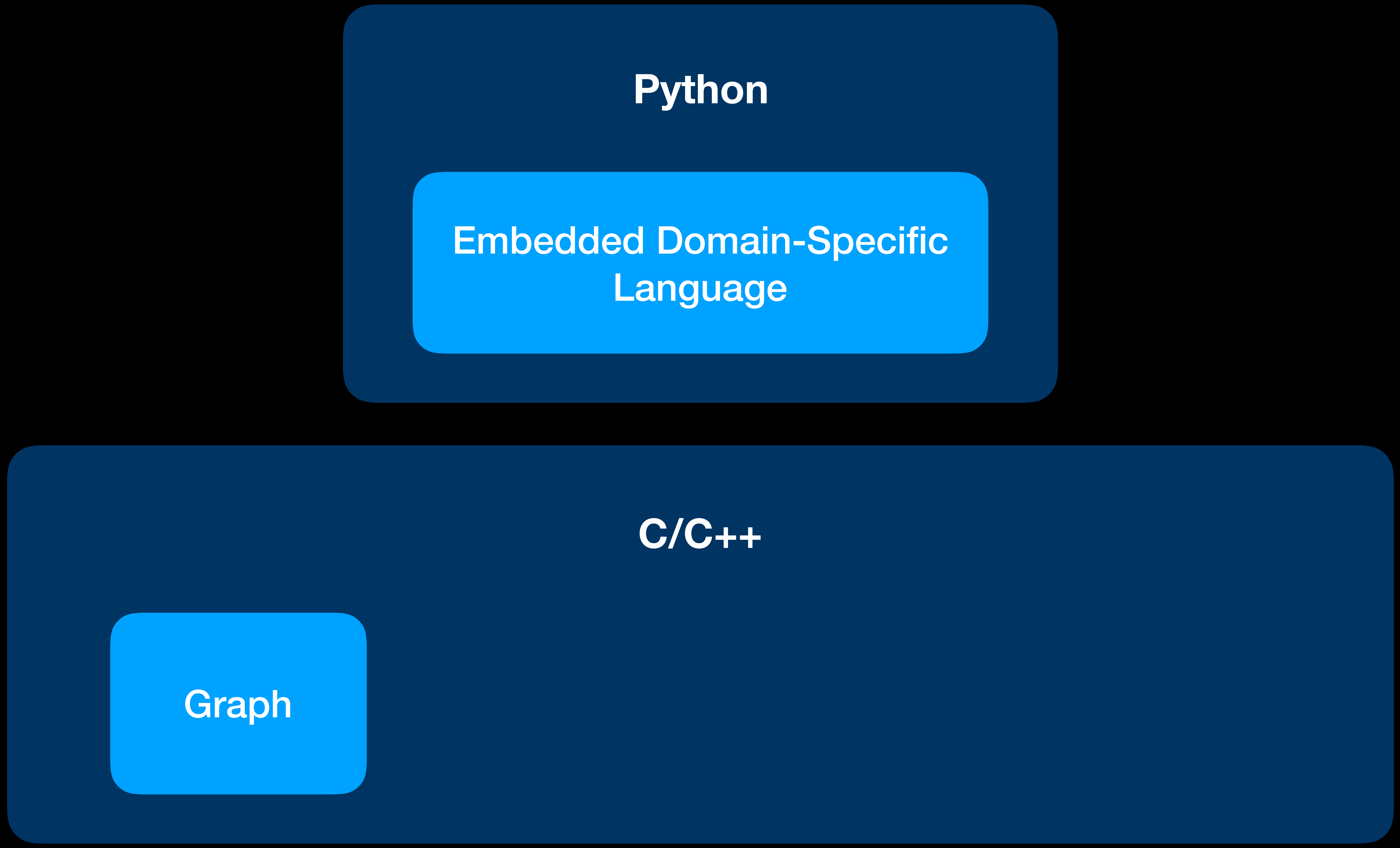
Existing Toolkits



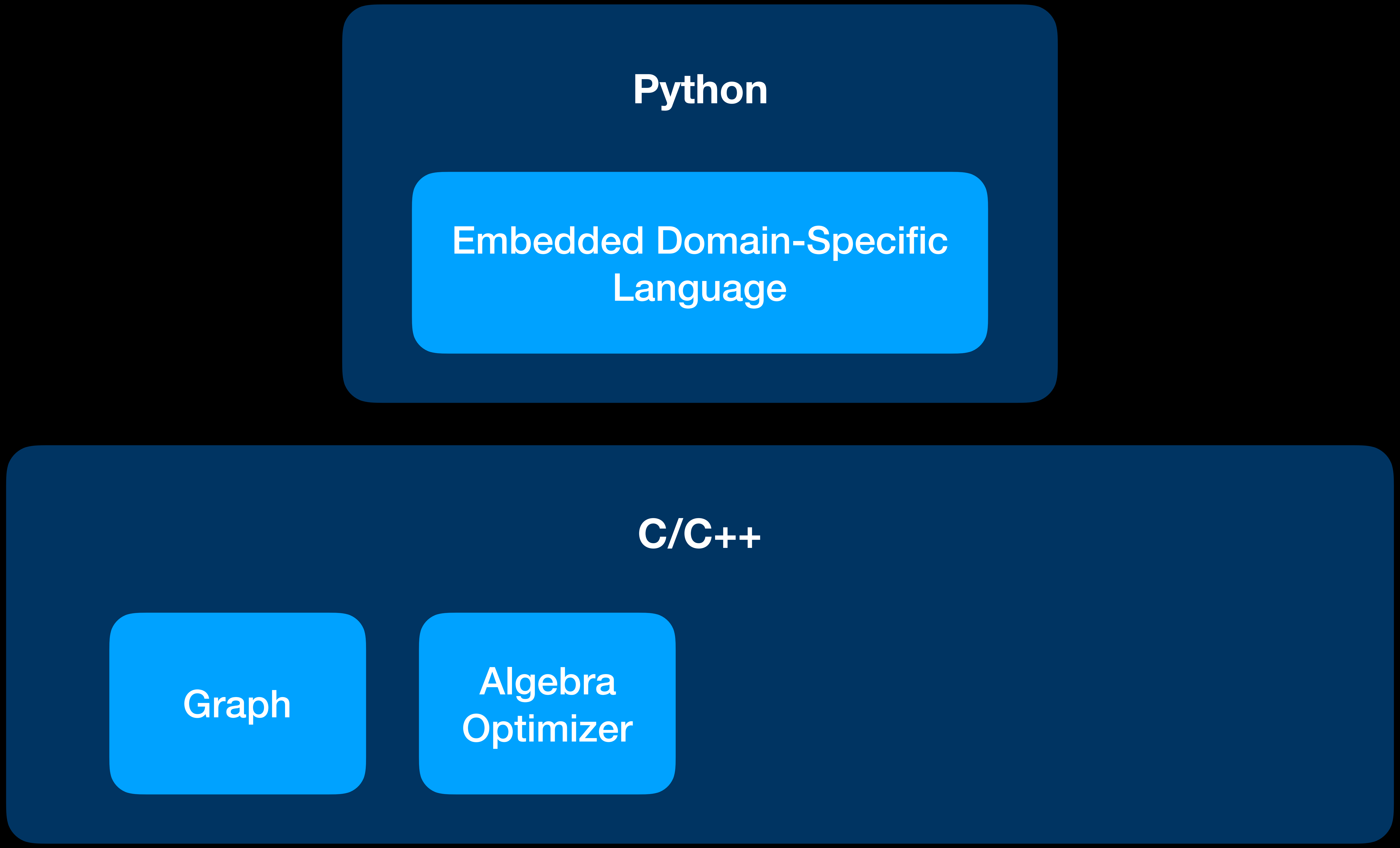
Existing Toolkits



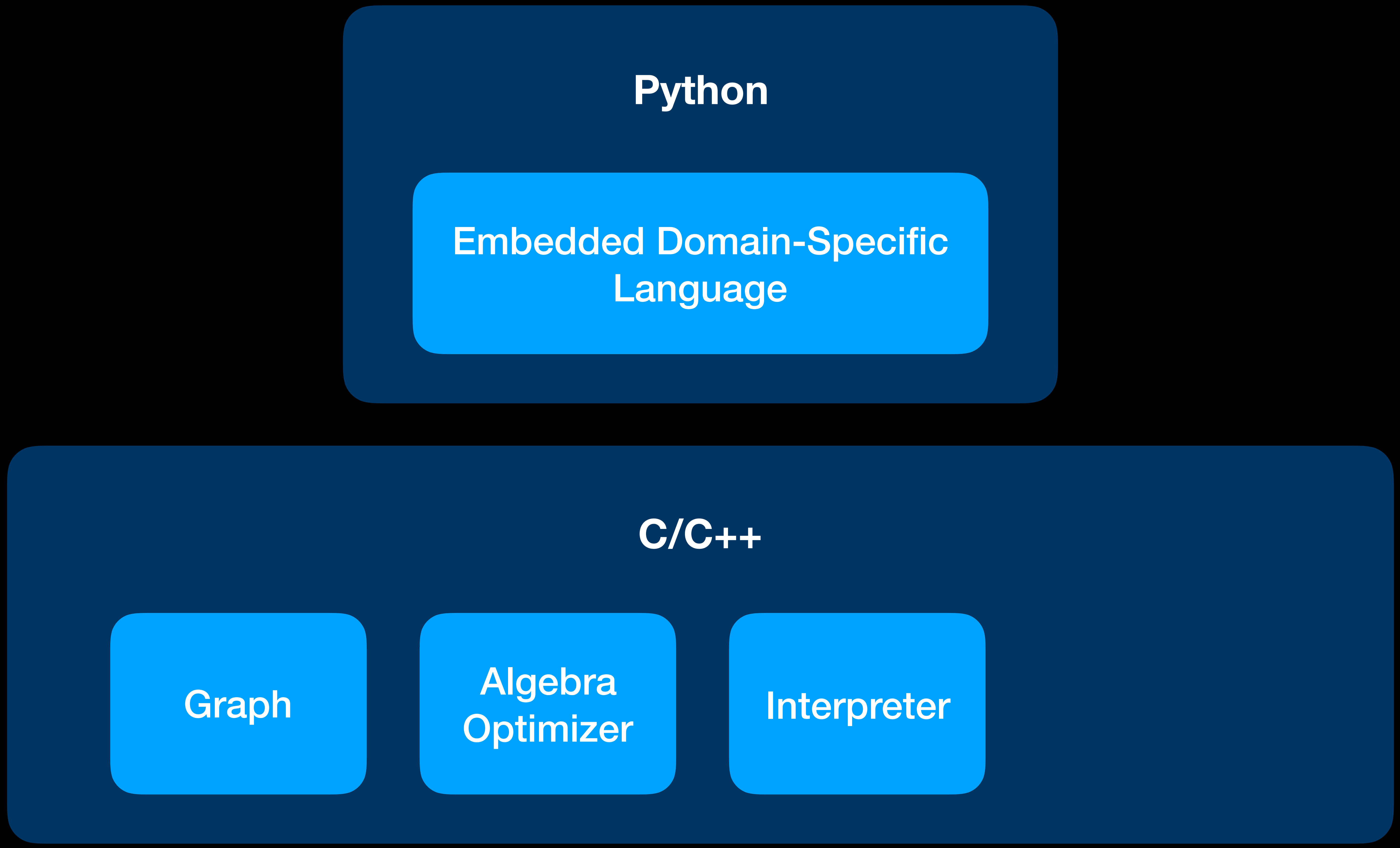
Existing Toolkits



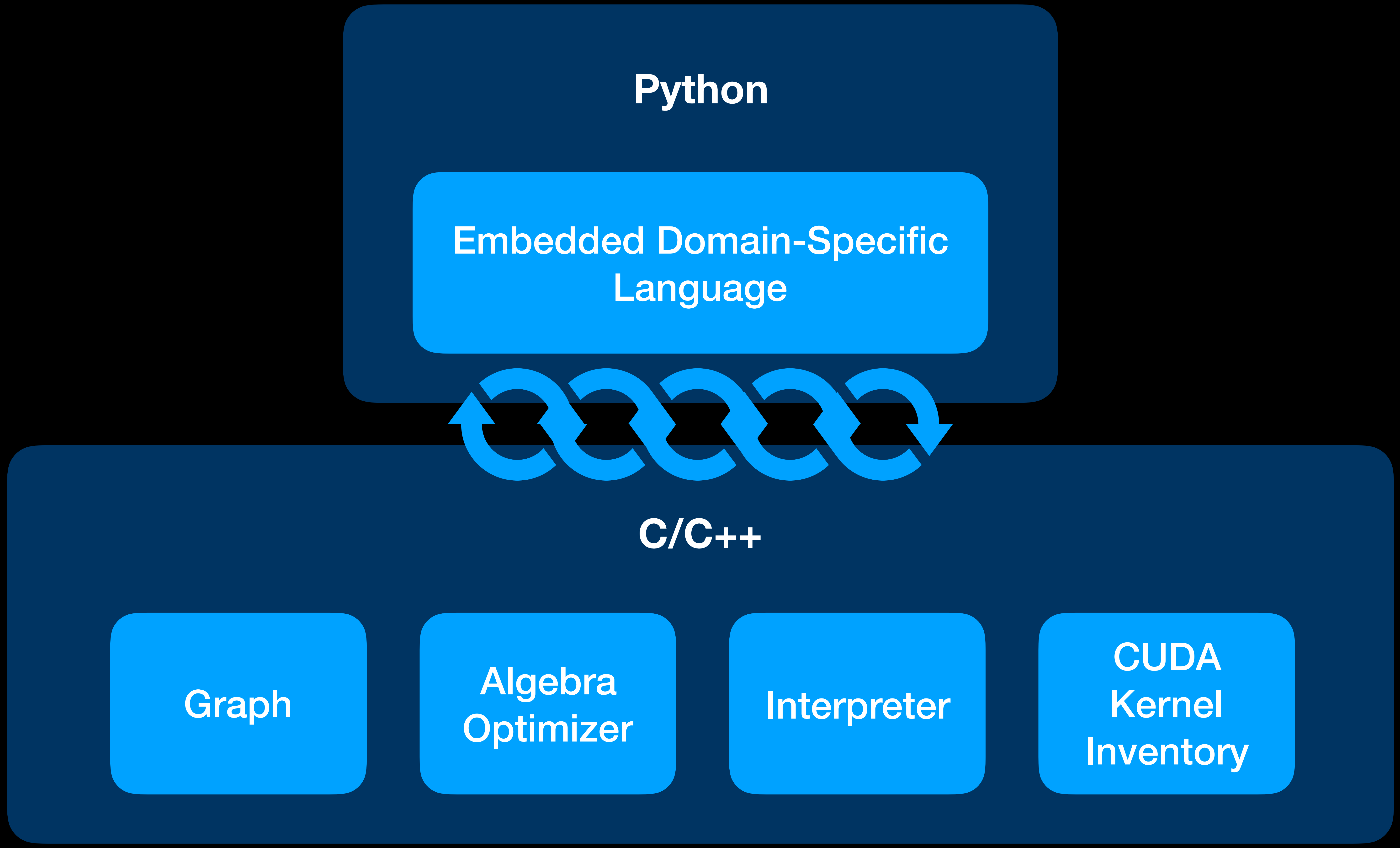
Existing Toolkits



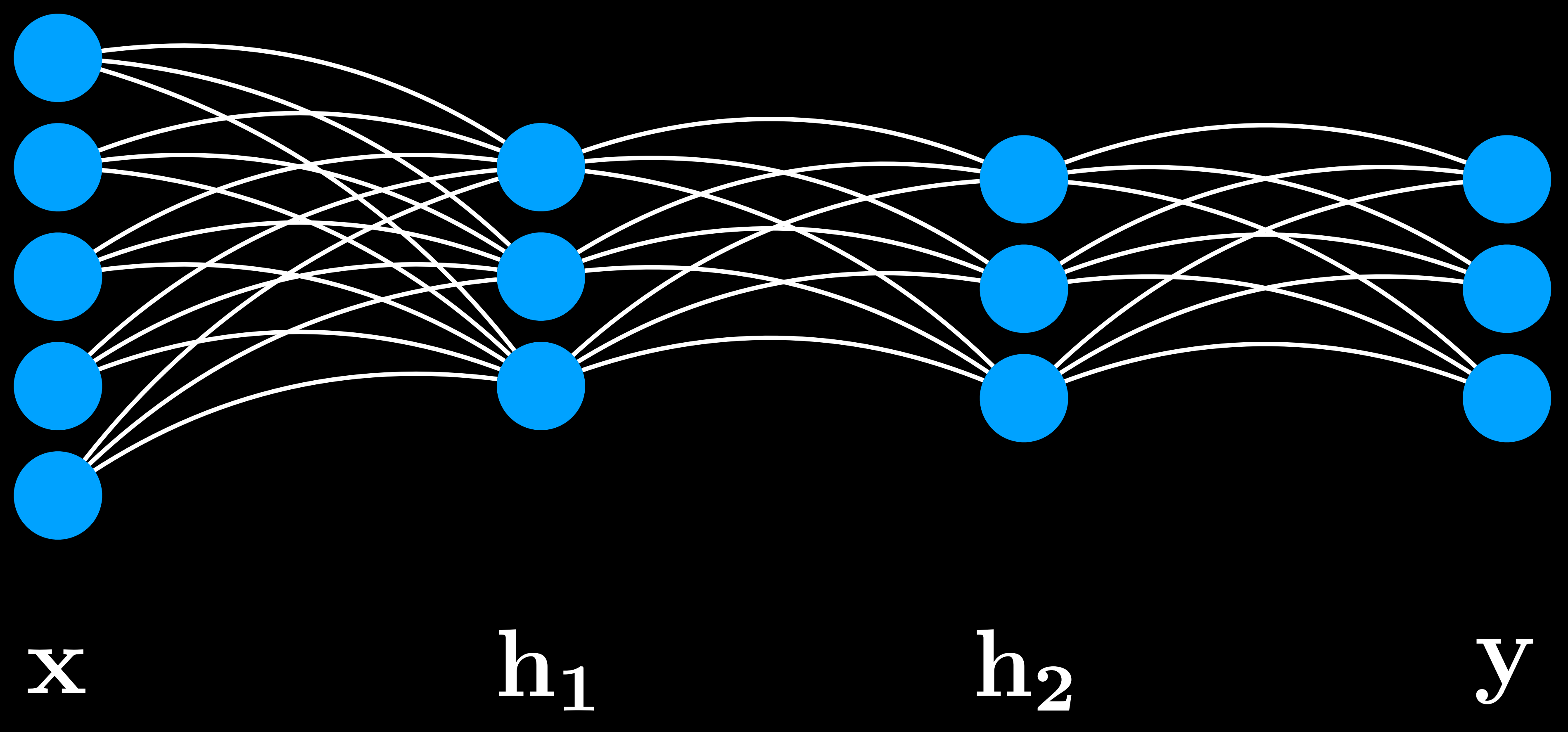
Existing Toolkits



Existing Toolkits



Rethink



$$\mathbf{h_1} = f(\mathbf{x}\mathbf{W_1} + \mathbf{b_1})$$

$$\mathbf{h_2} = f(\mathbf{h_1}\mathbf{W_2} + \mathbf{b_2})$$

$$\mathbf{y} = f(\mathbf{h_2}\mathbf{W_3} + \mathbf{b_3})$$

Neural networks are programs!

Compute

Optimizations

Auto Vectorization

Intermediate Representation

Neural networks are programs!

Control Flow

Automatic Differentiation

Static Analysis

A New Compiler Problem

A New Compiler Problem

- Neural networks as functions, without single-graph restrictions

A New Compiler Problem

- Neural networks as functions, without single-graph restrictions
- Efficient AutoDiff

A New Compiler Problem

- Neural networks as functions, without single-graph restrictions
- Efficient AutoDiff
- High-order differentiation

A New Compiler Problem

- Neural networks as functions, without single-graph restrictions
- Efficient AutoDiff
- High-order differentiation
- Cross-platform: GPUs and ML accelerators

A New Compiler Problem

- Neural networks as functions, without single-graph restrictions
- Efficient AutoDiff
- High-order differentiation
- Cross-platform: GPUs and ML accelerators
- Lightweight installation

A New Compiler Problem

- Neural networks as functions, without single-graph restrictions
- Efficient AutoDiff
- High-order differentiation
- Cross-platform: GPUs and ML accelerators
- Lightweight installation
- Just-in-time & ahead-of-time compilation

DLVM

DLVM

DLVM

- Intermediate representation for neural networks

DLVM

- Intermediate representation for neural networks
- Framework for building DSLs

- Intermediate representation for neural networks
- Framework for building DSLs
- Automatic backpropagator

DLVM

DLVM

- Intermediate representation for neural networks
- Framework for building DSLs
- Automatic backpropagator
- High-level optimizer

DLVM

- Intermediate representation for neural networks
- Framework for building DSLs
- Automatic backpropagator
- High-level optimizer
- LLVM-based compiler targeting CPU and GPU

DLVM

- DLVM-based Deep Learning Toolkits

DLVM

- DLVM-based Deep Learning Toolkits
  - Lightweight installation—no precompiled kernels

DLVM

DLVM

- DLVM-based Deep Learning Toolkits
  - Lightweight installation—no precompiled kernels
  - Multiple compute architectures

DLVM

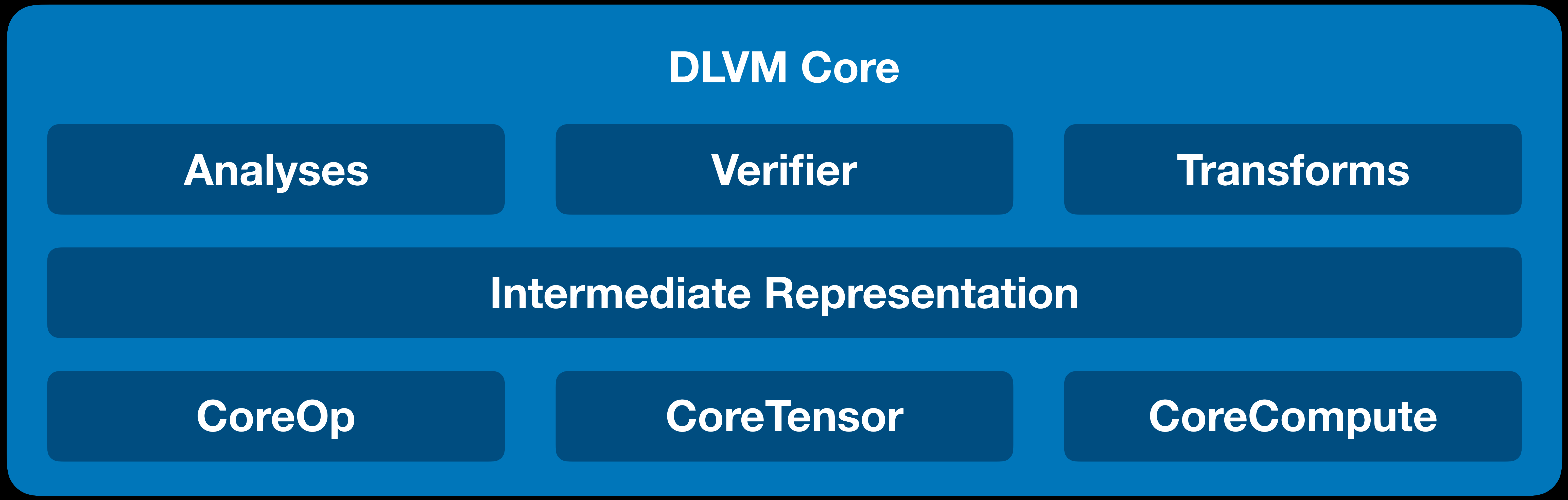
- DLVM-based Deep Learning Toolkits
  - Lightweight installation—no precompiled kernels
  - Multiple compute architectures
  - High performance

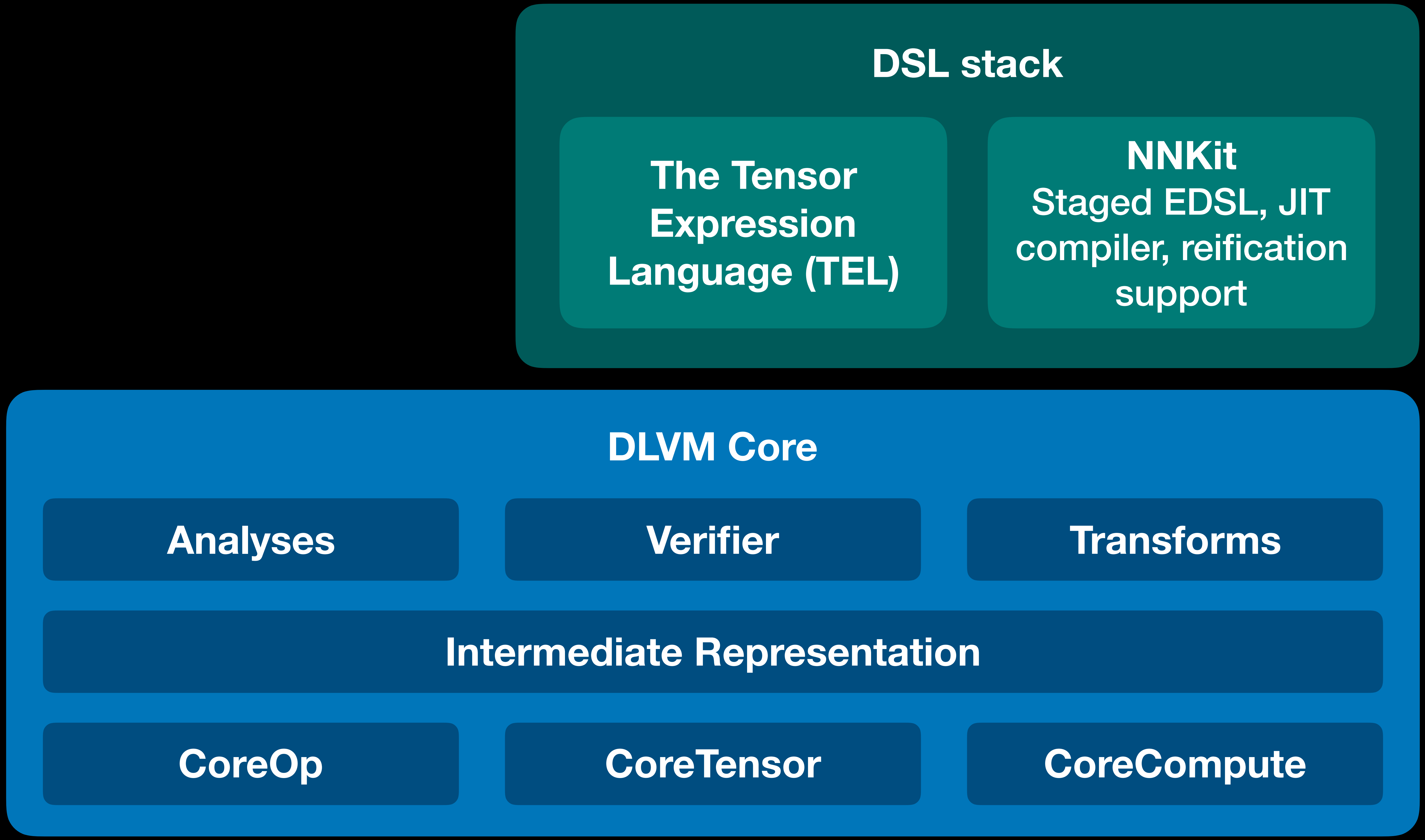
DLVM

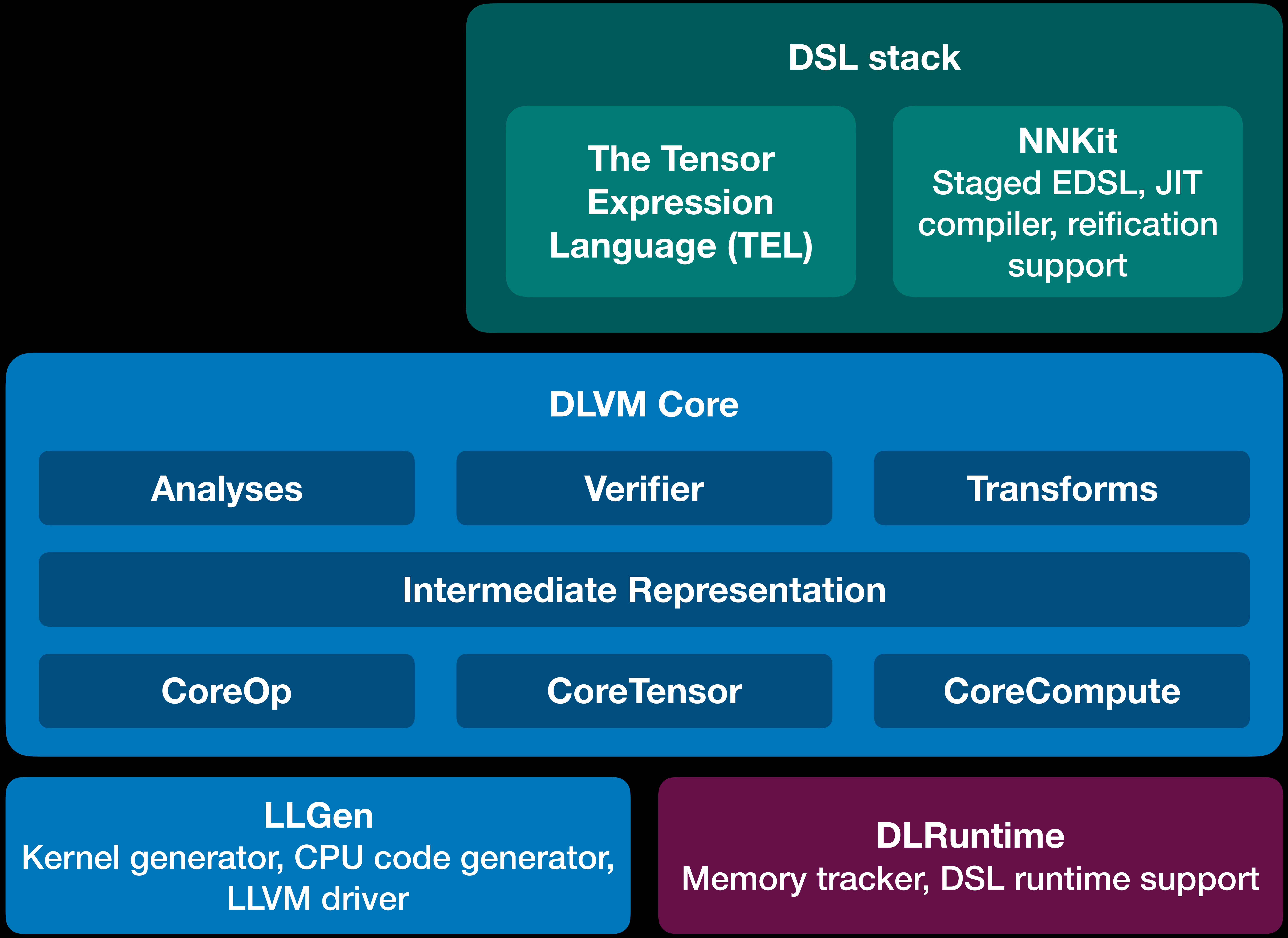
- DLVM-based Deep Learning Toolkits
  - Lightweight installation—no precompiled kernels
  - Multiple compute architectures
  - High performance
  - Optimized for low-memory devices

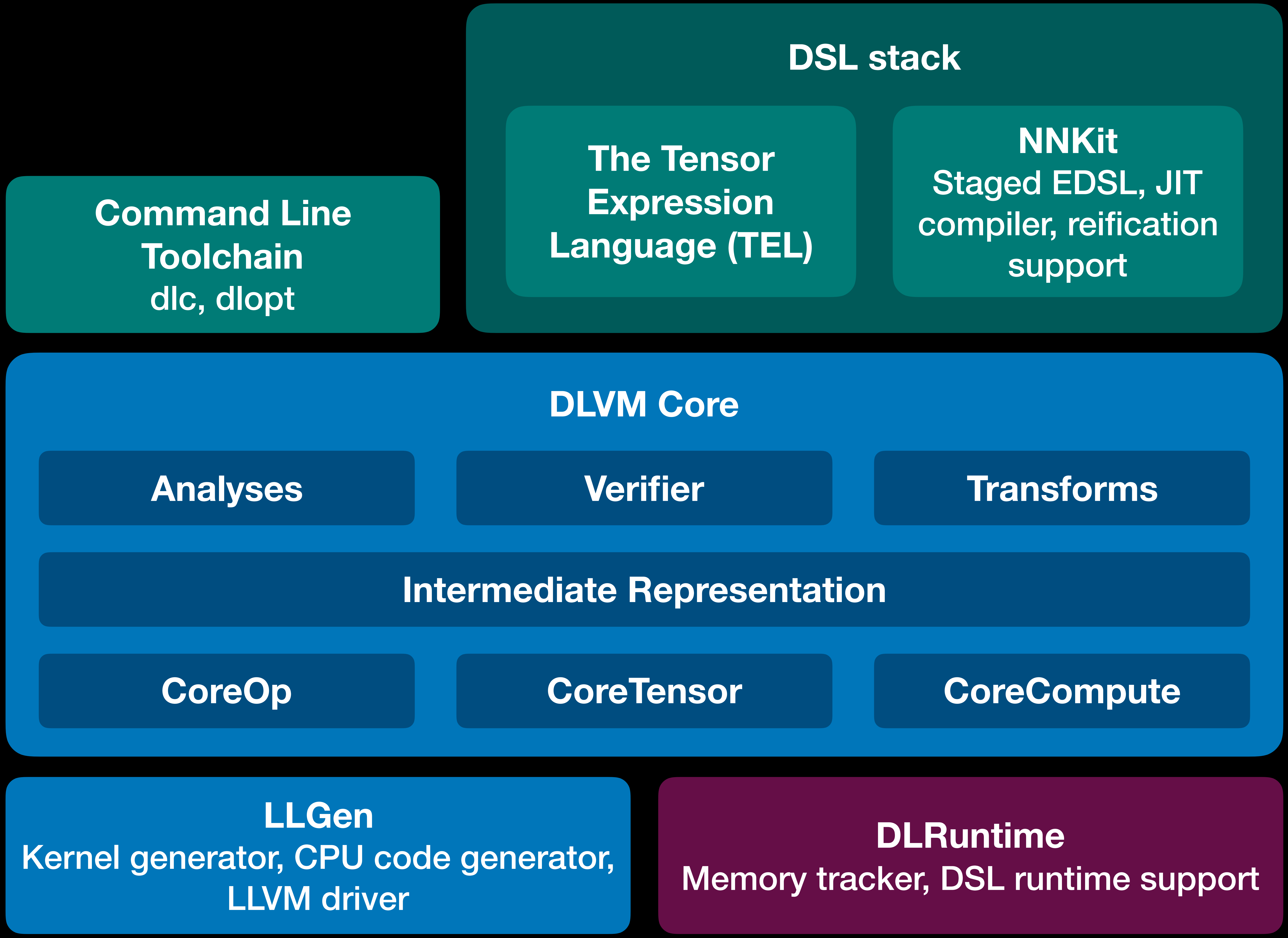
DLVM

- DLVM-based Deep Learning Toolkits
  - Lightweight installation — no precompiled kernels
  - Multiple compute architectures
  - High performance
  - Optimized for low-memory devices
  - Bridging the gap between prototyping and production


DLVM Core
Analyses
Verifier
Transforms
Intermediate Representation
CoreOp
CoreTensor
CoreCompute


DSL stack
The Tensor Expression Language (TEL)
NNKit
Staged EDSL, JIT compiler, reification support
DLVM Core
Analyses
Verifier
Transforms
Intermediate Representation
CoreOp
CoreTensor
CoreCompute


DSL stack
The Tensor Expression Language (TEL)
NNKit
Staged EDSL, JIT compiler, reification support
DLVM Core
Analyses
Verifier
Transforms
Intermediate Representation
CoreOp
CoreTensor
CoreCompute
LLGen
Kernel generator, CPU code generator, LLVM driver
DLRuntime
Memory tracker, DSL runtime support


DSL stack
The Tensor Expression Language (TEL)
NNKit
Staged EDSL, JIT compiler, reification support
Command Line Toolchain
dlc, dlopt
DLVM Core
Analyses
Verifier
Transforms
Intermediate Representation
CoreOp
CoreTensor
CoreCompute
LLGen
Kernel generator, CPU code generator, LLVM driver
DLRuntime
Memory tracker, DSL runtime support

DLVM Core

Intermediate Representation

Intermediate Representation

- SSA form
  - Perfect for handling control flow in AutoDiff

Intermediate Representation

- SSA form
  - Perfect for handling control flow in AutoDiff
- Basic blocks with arguments (like SIL)

Intermediate Representation

- SSA form
  - Perfect for handling control flow in AutoDiff
- Basic blocks with arguments (like SIL)
- Textual format & in-memory format
  - Built-in parser and verifier
  - FileCheck for robust unit testing

Tensor Type

Rank	Notation	Descripton
0	`i64`	64-bit integer
1	`<100 x f32>`	float vector of size 100
2	`<100 x 300 x f64>`	double matrix of size 100x300
n	`<100 x 300 x ... x bool>`	rank-n tensor

First-class tensors

High Level Instructions

Kind	Example
Element-wise unary	tanh %a: <10 x f32>
Element-wise binary	power %a: <10 x f32>, %b: 2: f32
Dot	dot %a: <10 x 20 x f32>, %b: <20 x 2 x f32>
Concatenate	concatenate %a: <10 x f32>, %b: <20 x f32> along 0
Reduce	reduce %a: <10 x 30 x f32> by add along 1
Transpose	transpose %m: <2 x 3 x 4 x 5 x i32>
Convolution	convolve %a: <…> kernel %b: <…> stride %c: <…> …
Slice	slice %a: <10 x 20 x i32> from 1 unto 5
Random	random 768 x 10 from 0.0: f32 unto 1.0: f32
Select	select %x: <10 x f64>, %y: <10 x f64> by %flags: <10 x bool>
Compare	greaterThan %a: <10 x 20 x bool>, %b: <1 x 20 x bool>
Data type cast	dataTypeCast %x: <10 x i32> to f64

General Purpose Instructions

Kind	Example
Function Application	`apply %foo(%x: f32, %y: f32): (f32, f32) -> <10 x 10 x f32>`
Branch	`branch 'block_name(%a: i32, %b: i32)`
Conditional (if-then-else)	`conditional %cond: bool then 'then_block() else 'else_block()`
Shape cast	`shapeCast %a: <1 x 40 x f32> to 2 x 20`
Extract	`extract #x from %pt: $Point`
Insert	`insert 10: f32 to %pt: $Point at #x`
Allocate stack	`allocateStack $Point count 1`
Allocate heap	`allocateHeap $MNIST count 1`
Deallocate	`deallocate %x: *<10 x f32>`
Load	`load %ptr: *<10 x i32>`
Store	`store %x: <10 x i32> to %ptr: *<10 x i32>`
Copy	`copy from %src: *<10 x f16> to %dst: *<10 x f16> count 1: i64`

Example IR

```
module "mnist" // Module declaration
stage raw      // Raw stage IR in the compilation phase

struct $MNIST {
    #w: <784 x 10 x f32>,
    #b: <1 x 10 x f32>,
}

type $MyMnist = $MNIST
```

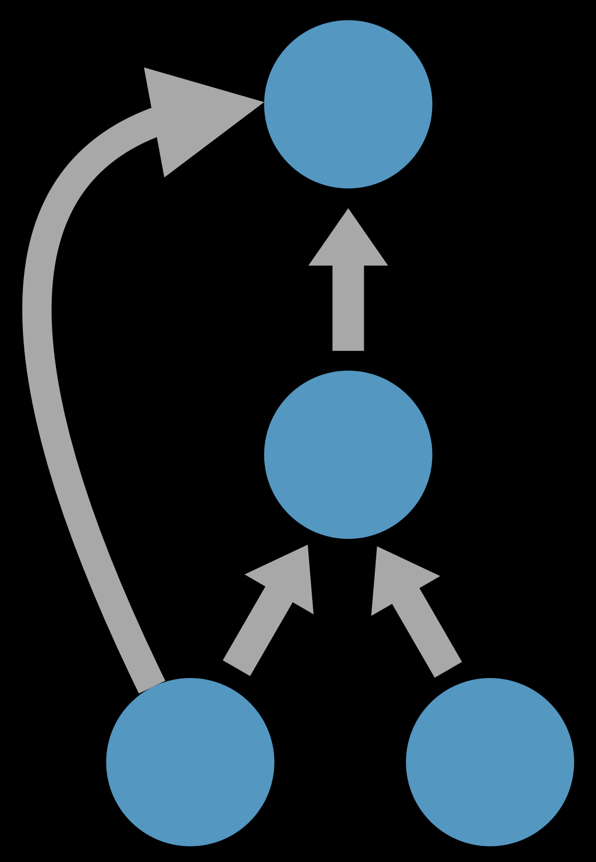
Example IR

```
module "mnist"
stage raw

struct $MNIST {
    #w: <784 x 10 x f32>,
    #b: <1 x 10 x f32>,
}

type $MyMnist = $MNIST

func @inference: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>) -> <1 x 10 x f32> {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    return %0.1: <1 x 10 x f32>
}
```

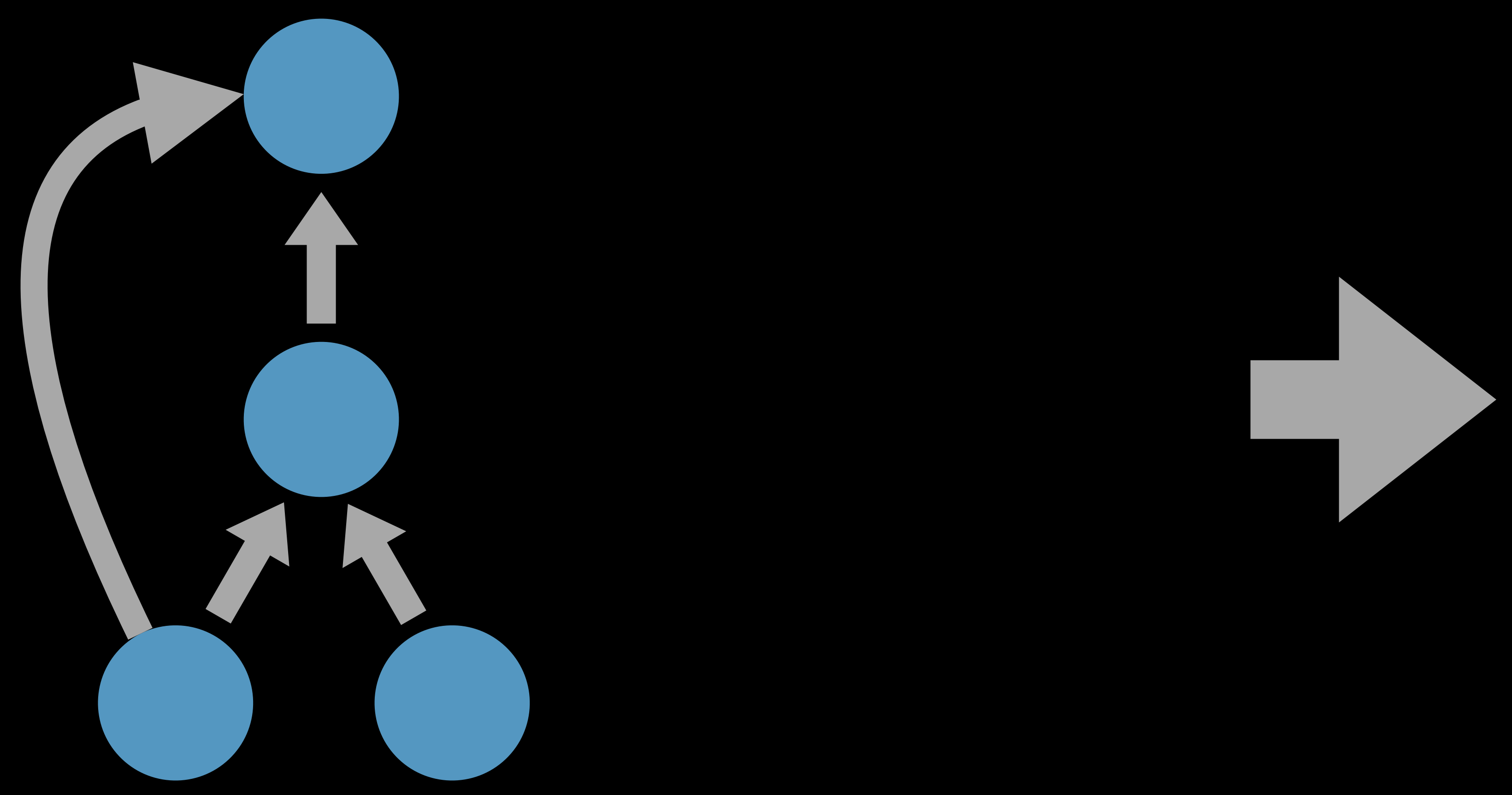
Transformations:
Differentiation & Optimizations

Automatic Differentiation

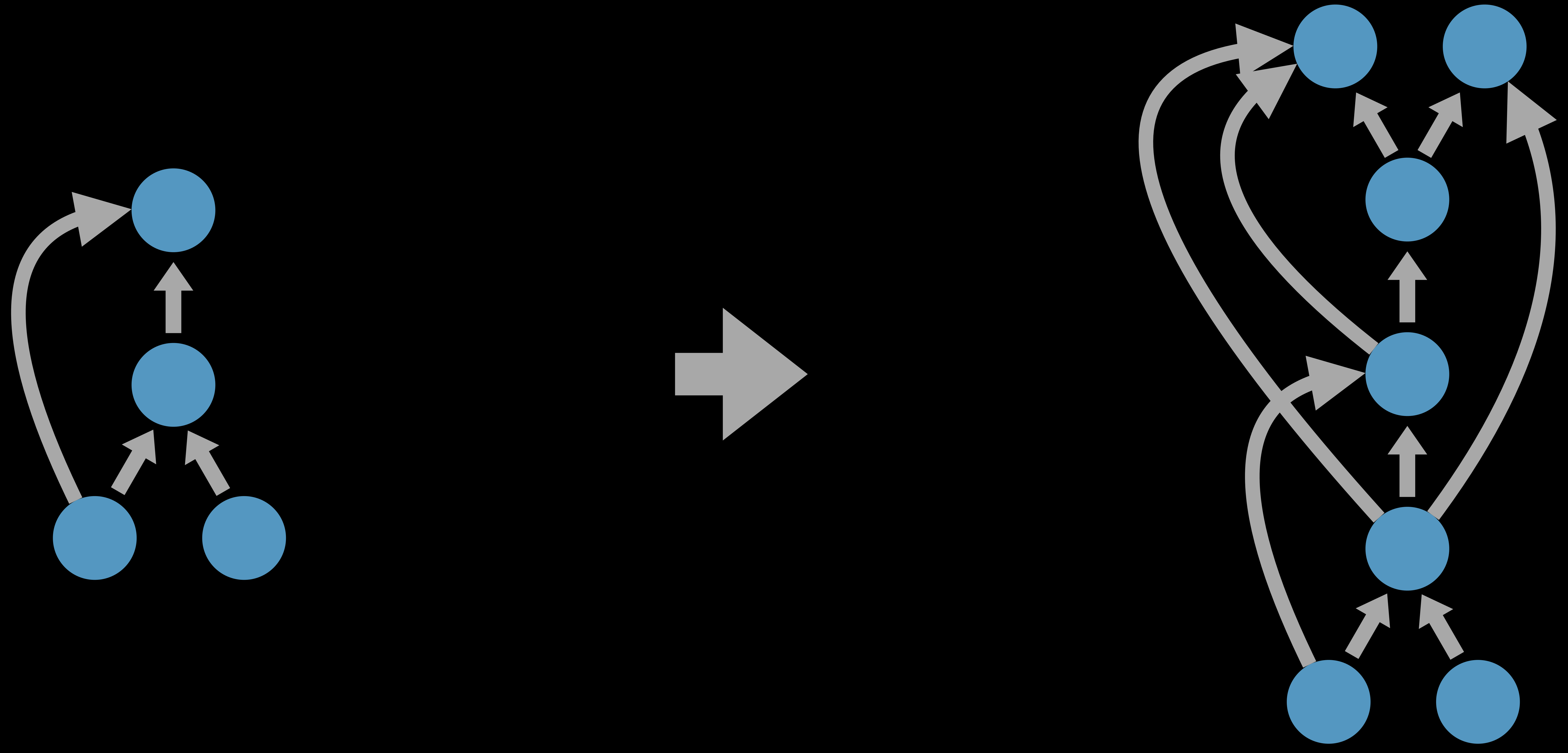
Automatic Differentiation



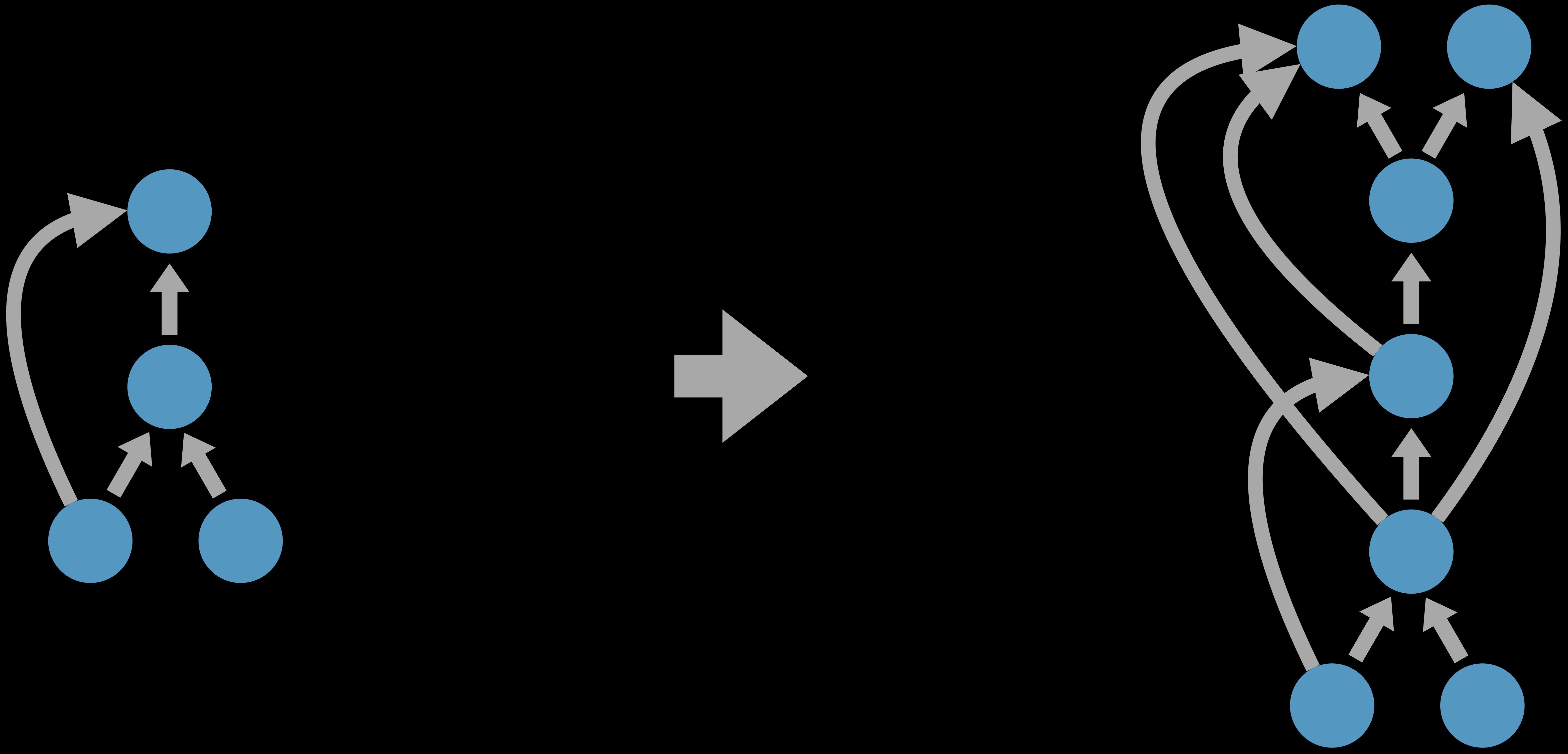
Automatic Differentiation



Automatic Differentiation

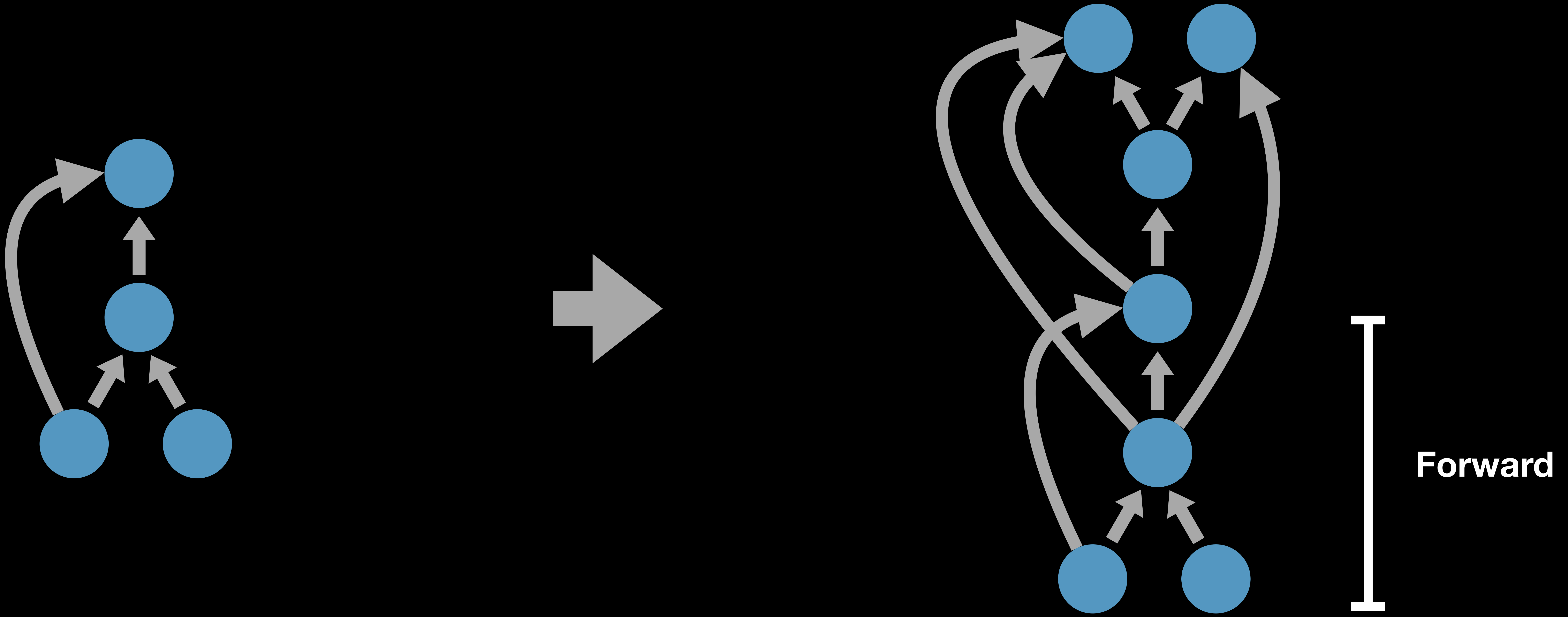


Automatic Differentiation



❌ Operator overloading (interpretation)
✅ Source code transformation

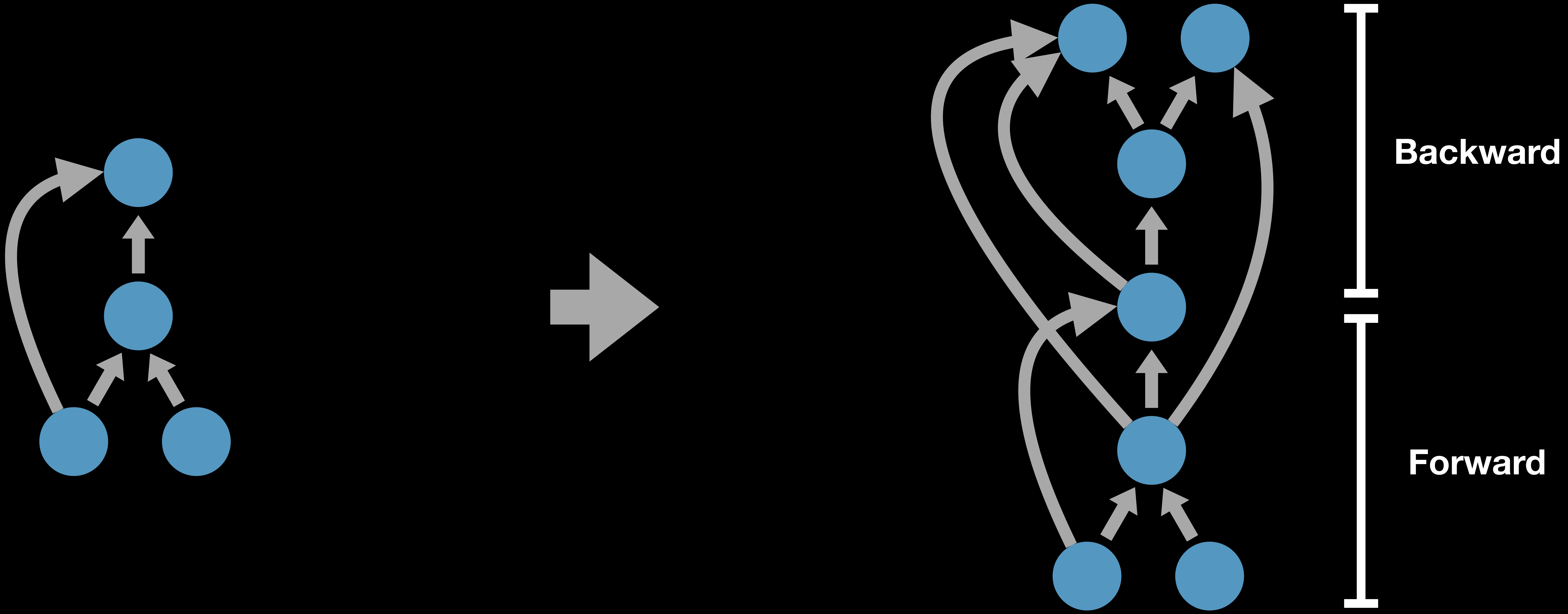
Automatic Differentiation



❌ Operator overloading (interpretation)

✅ Source code transformation

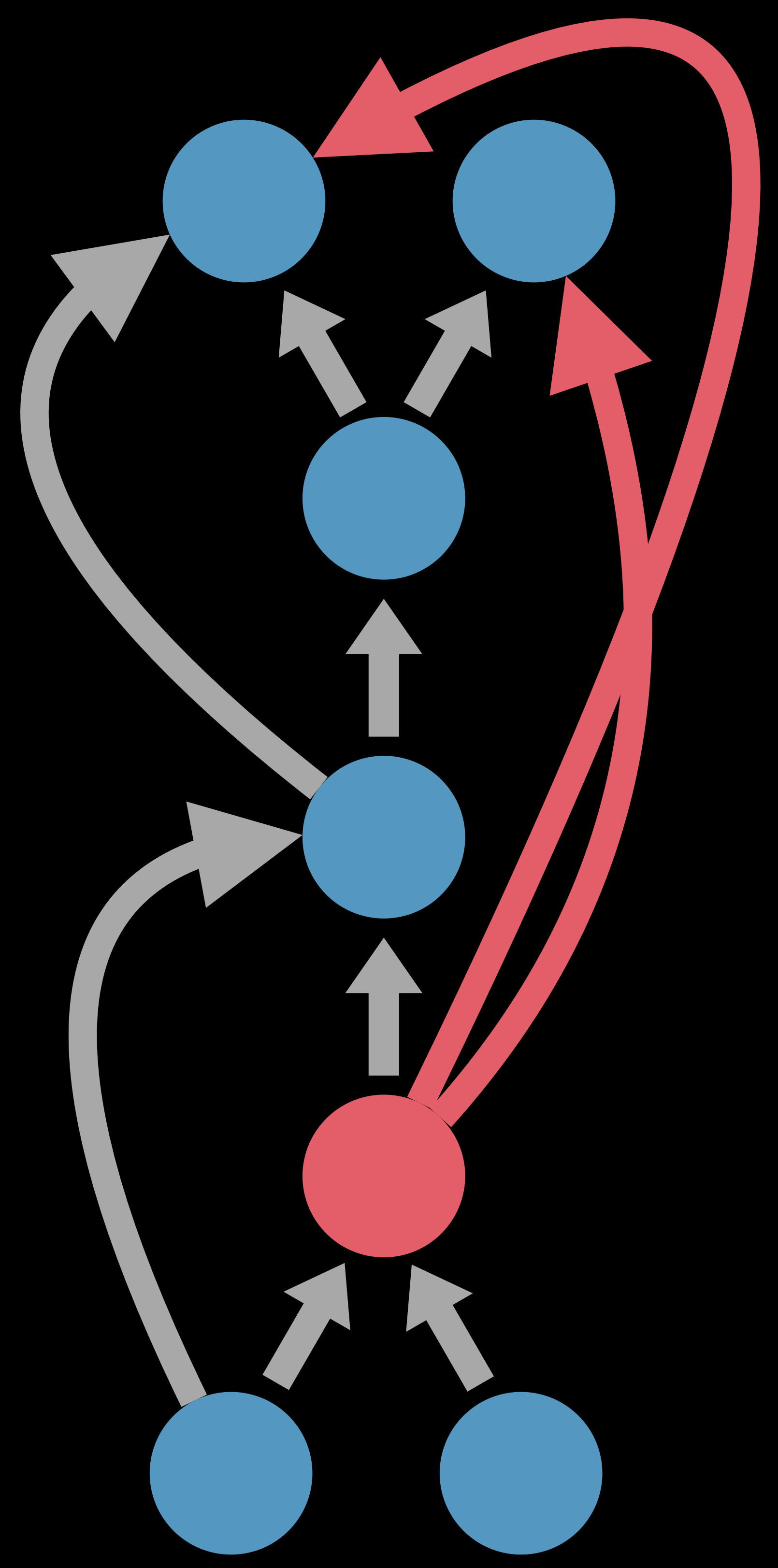
Automatic Differentiation



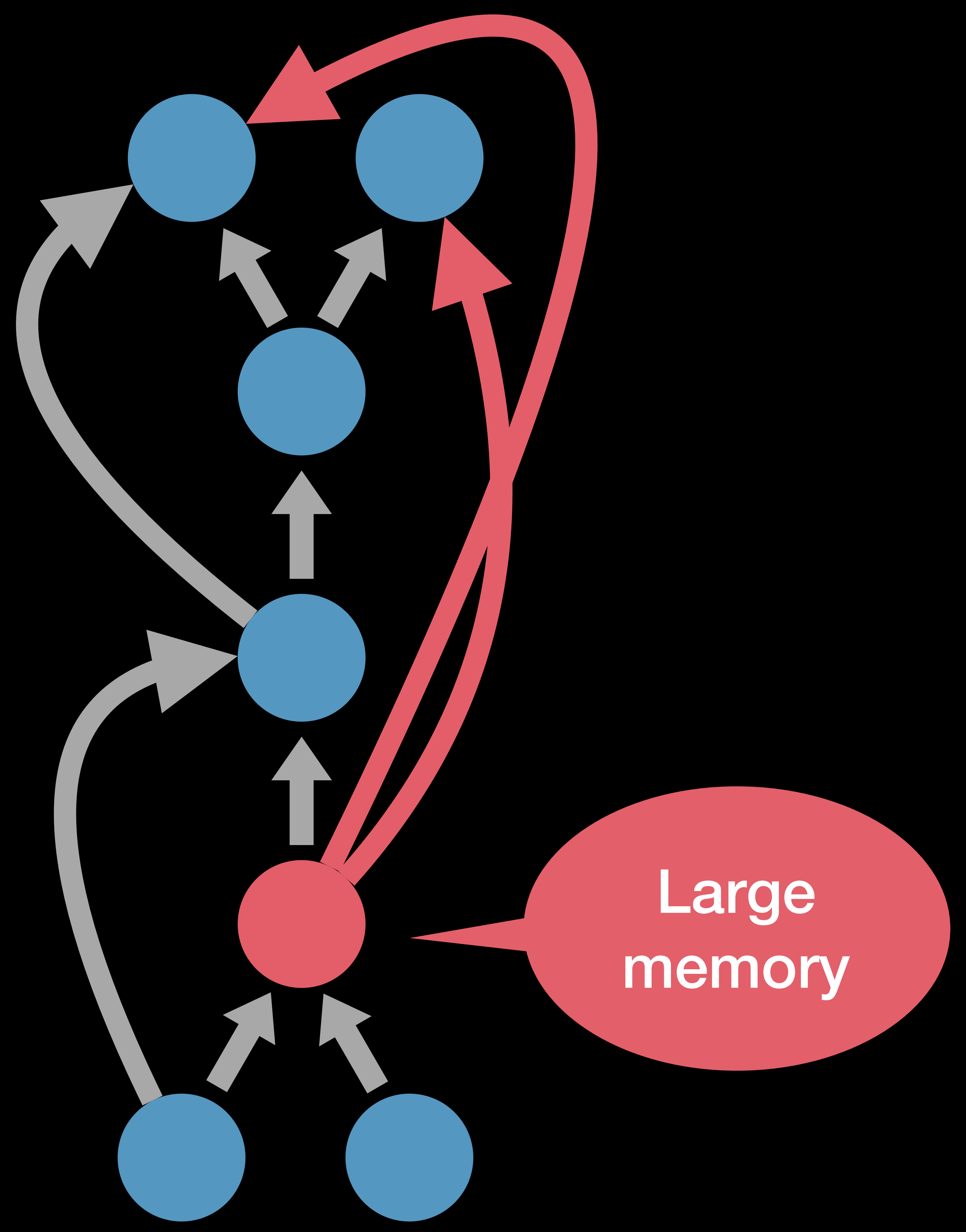
❌ Operator overloading (interpretation)
✅ Source code transformation

Differentiation Optimization

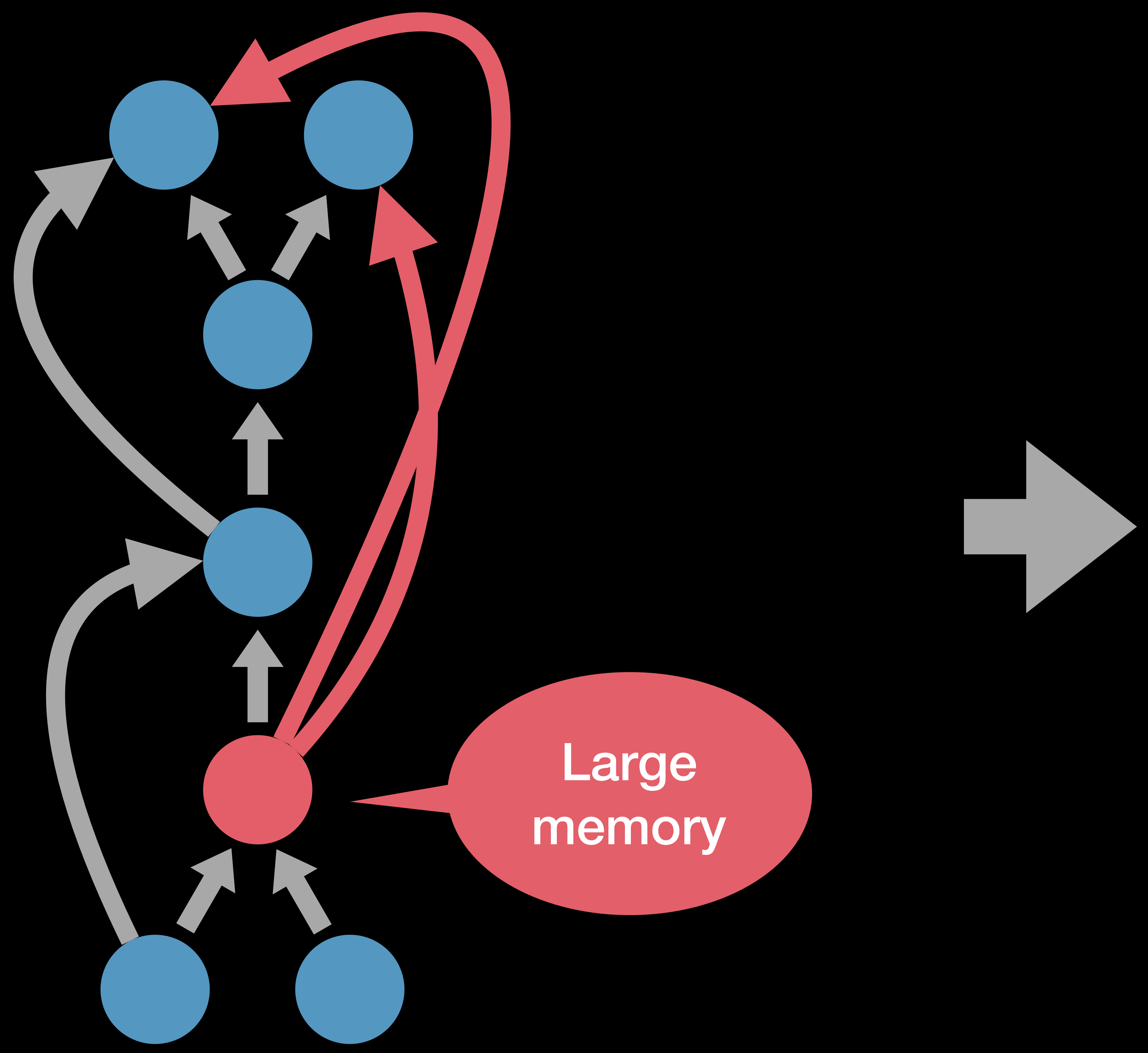
Differentiation Optimization



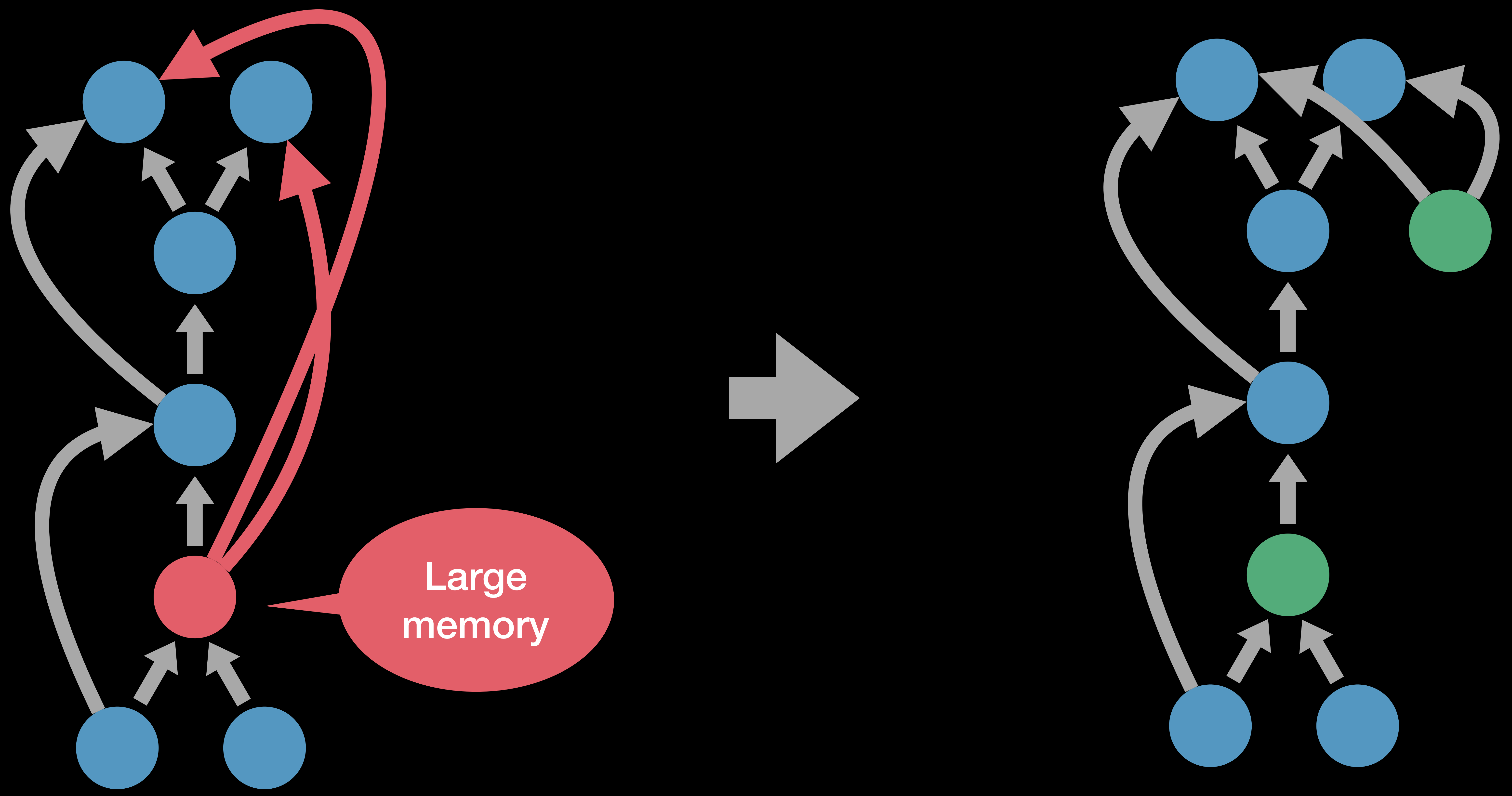
Differentiation Optimization



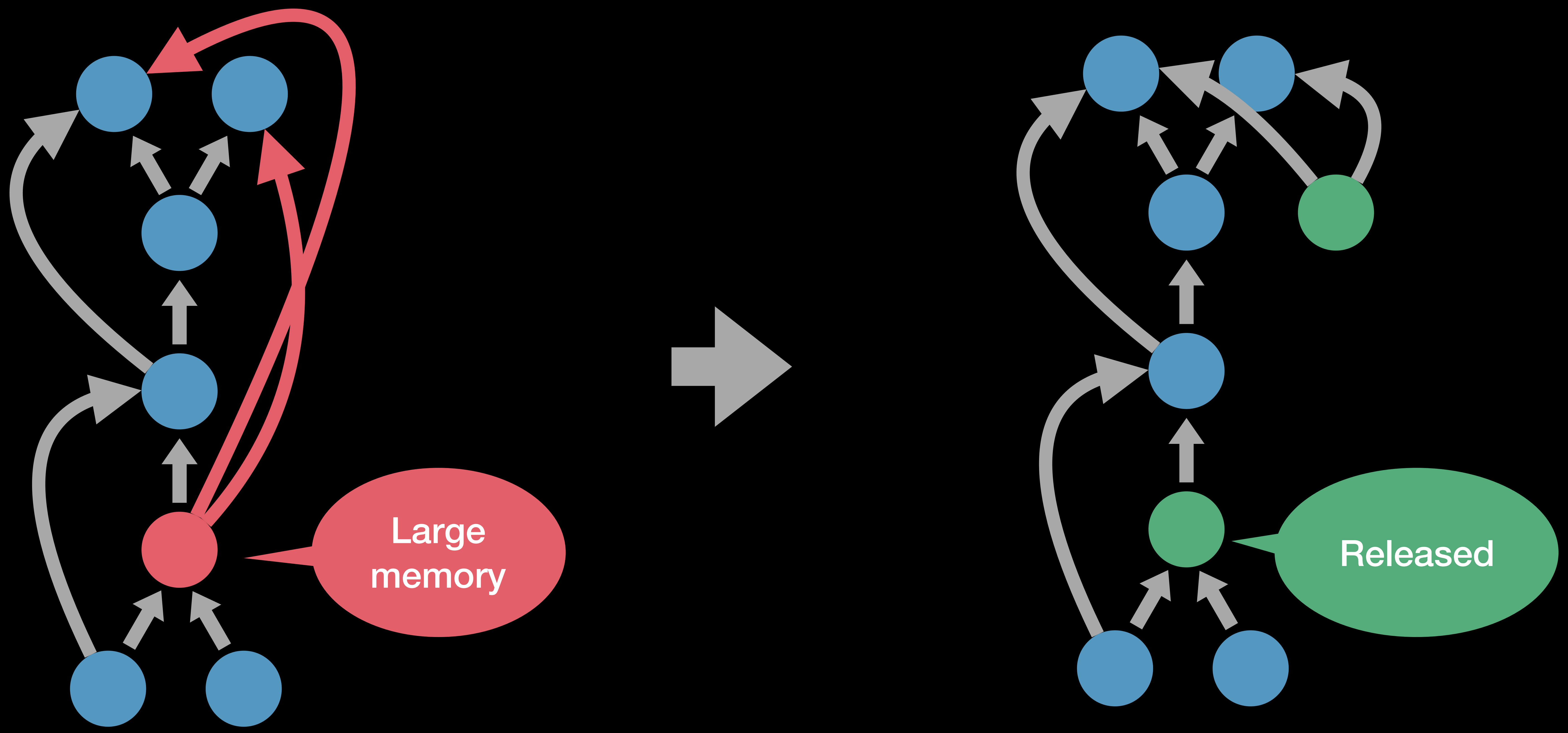
Differentiation Optimization



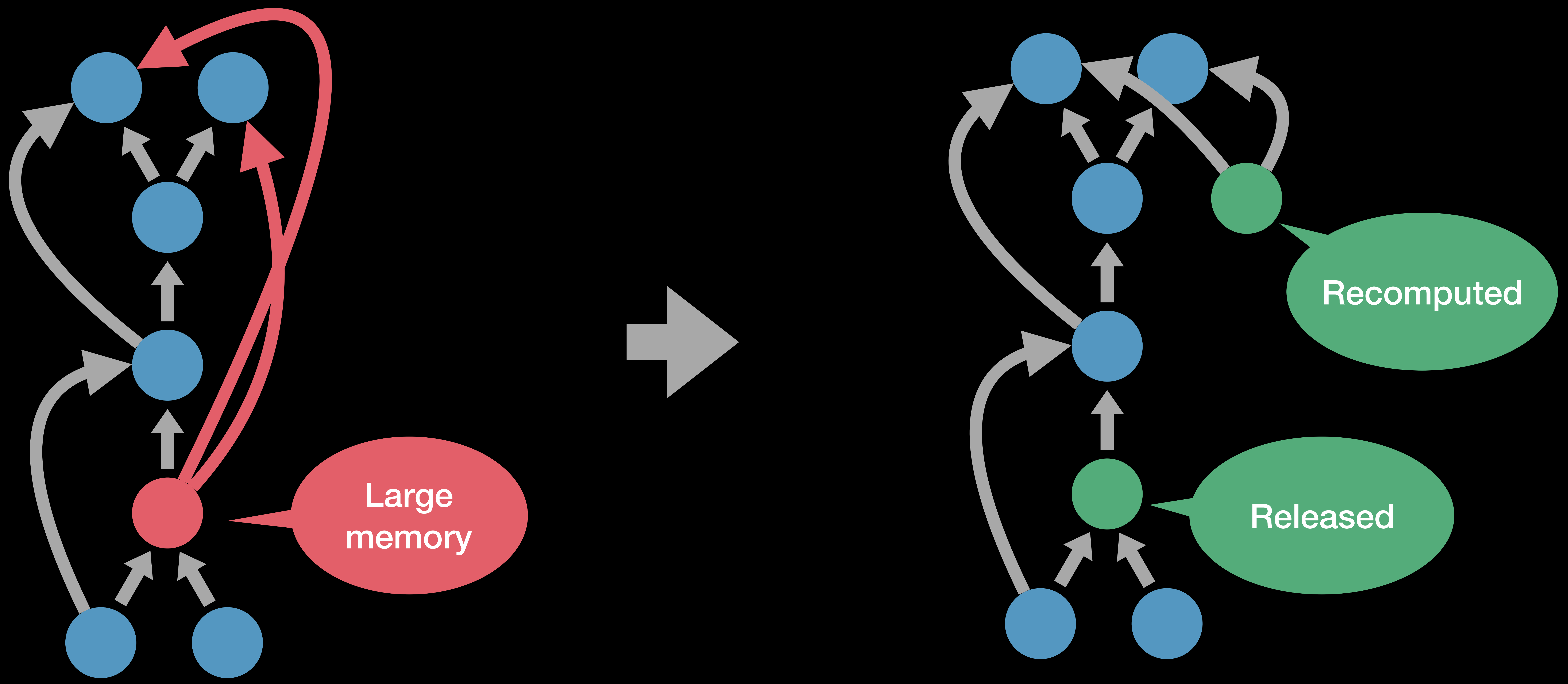
Differentiation Optimization



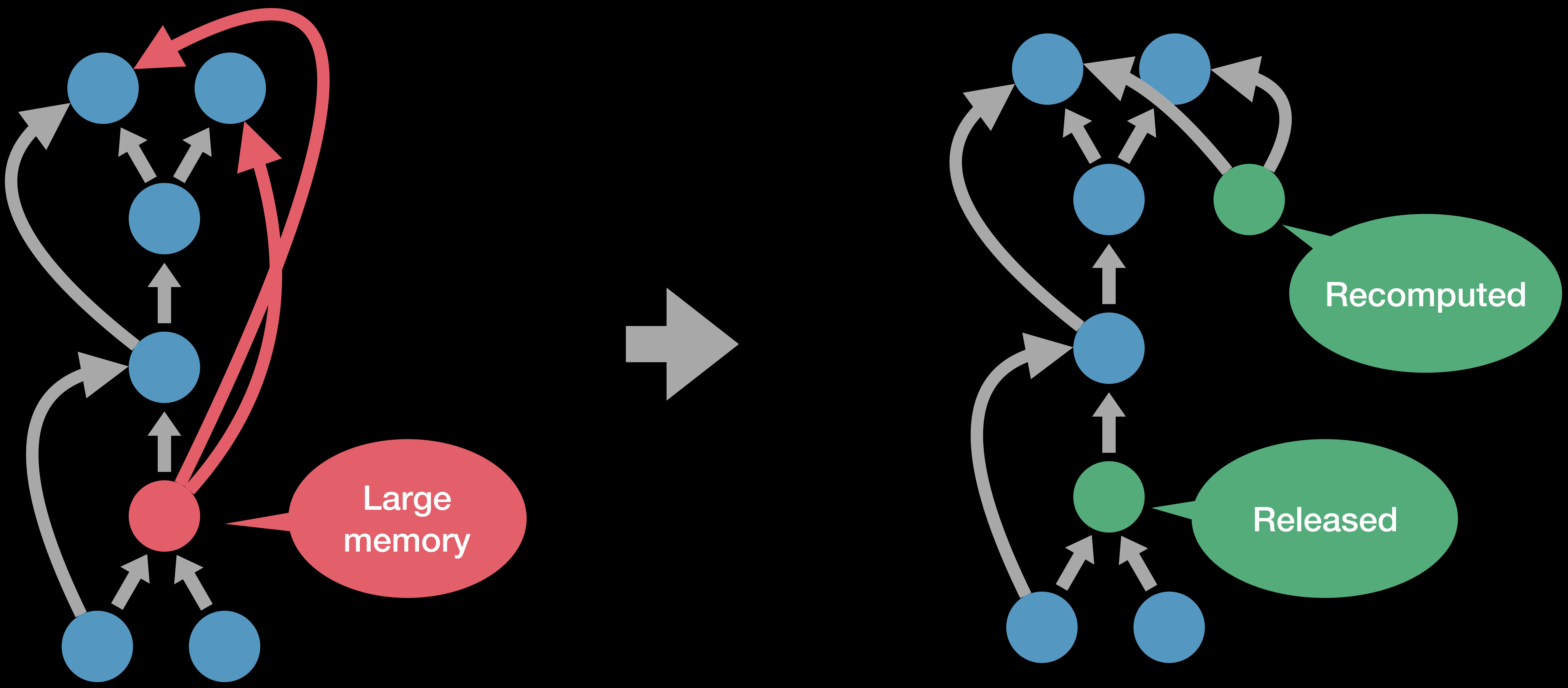
Differentiation Optimization



Differentiation Optimization



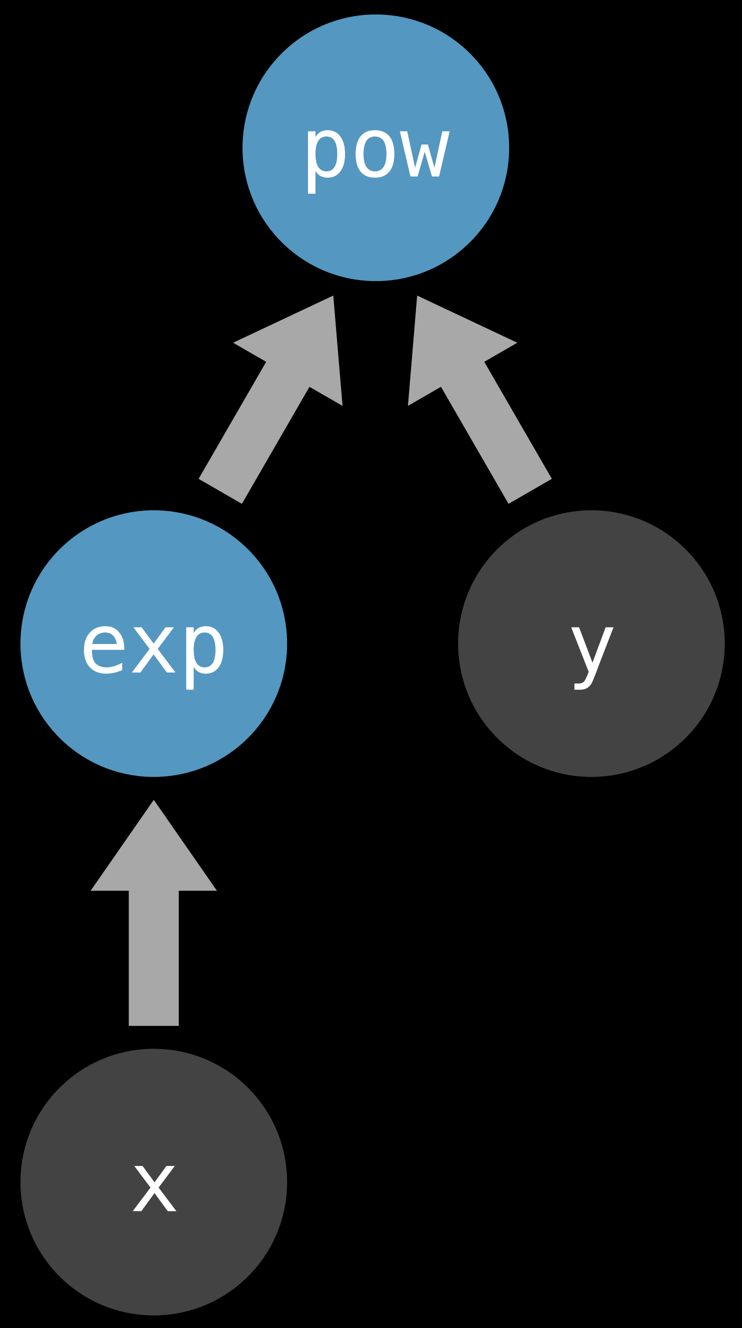
Differentiation Optimization



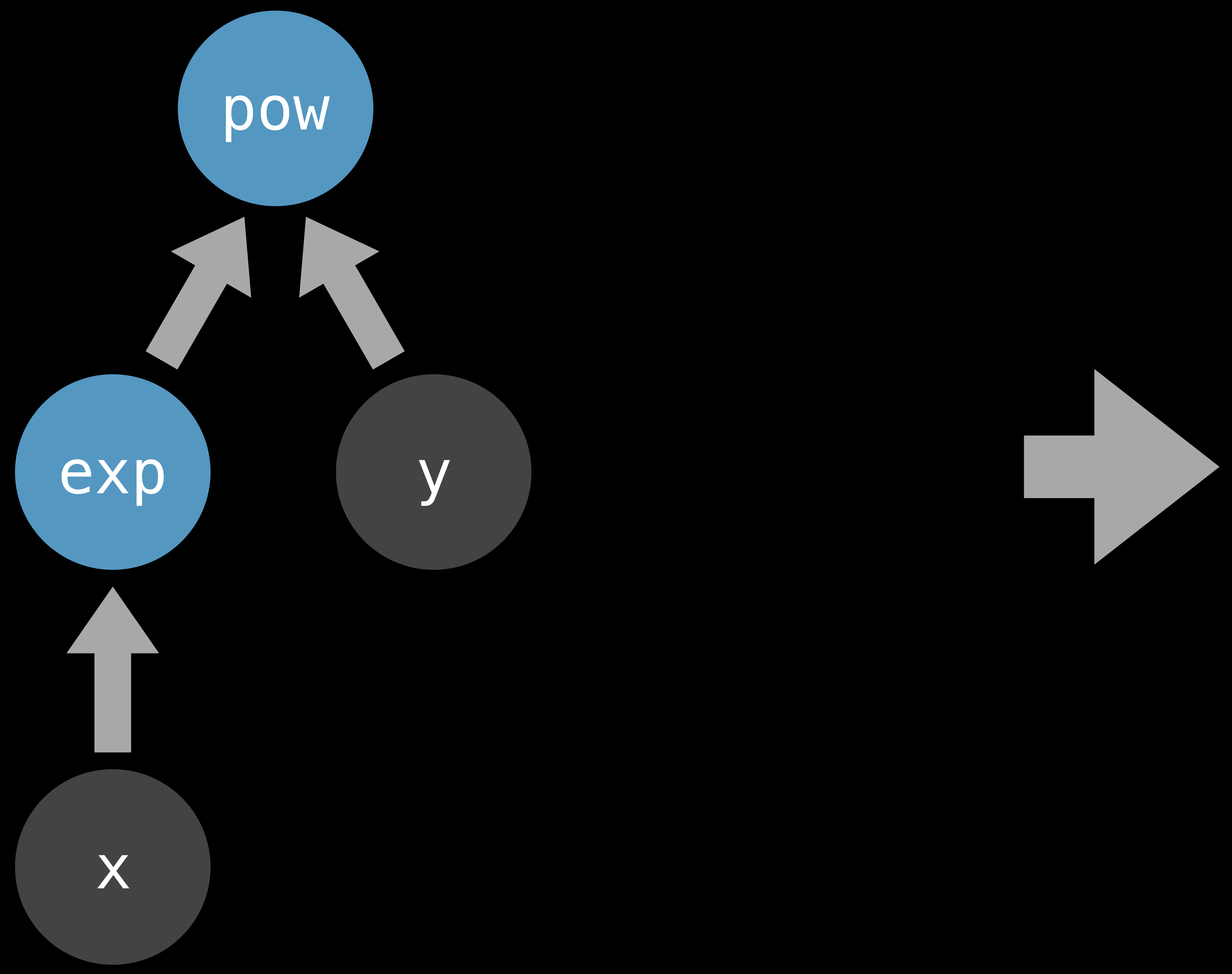
Checkpointing for reverse-mode AutoDiff

Algebra Simplification

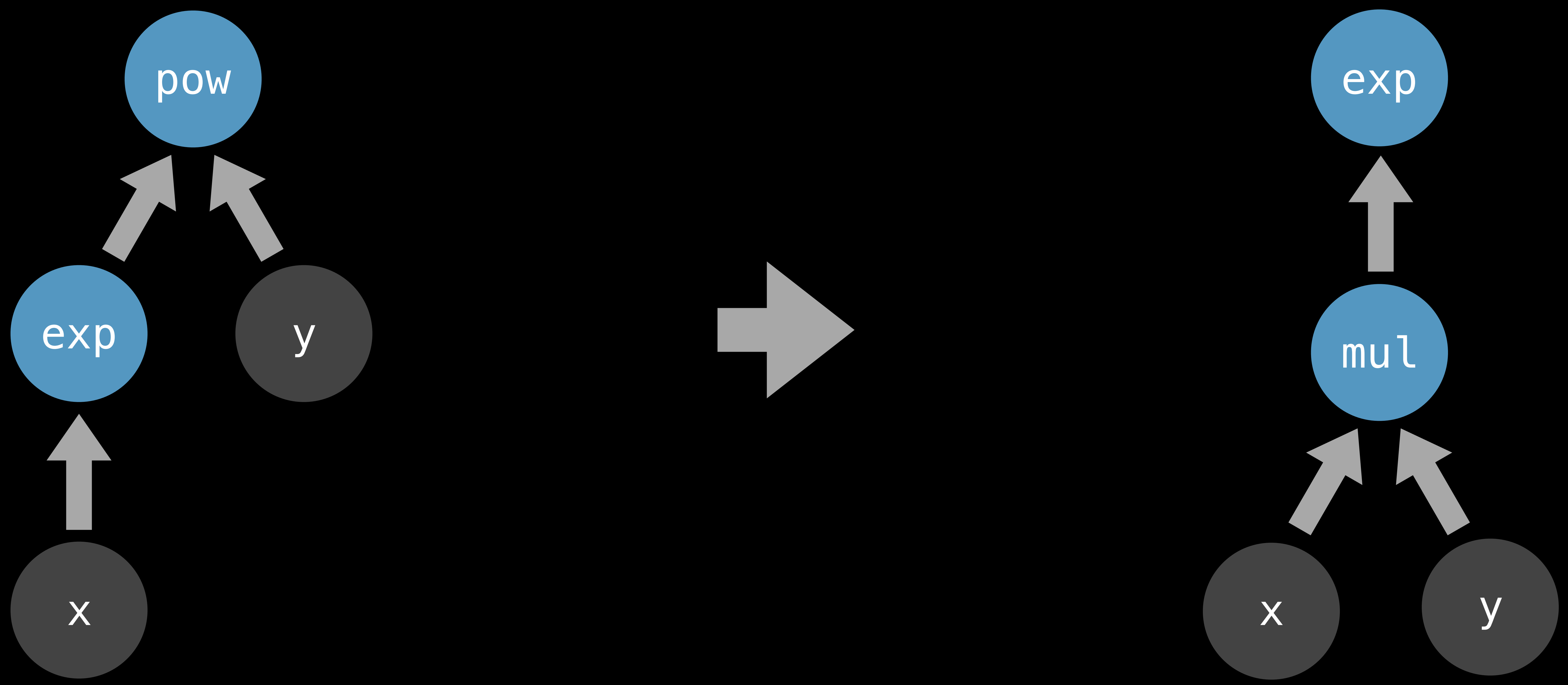
Algebra Simplification



Algebra Simplification

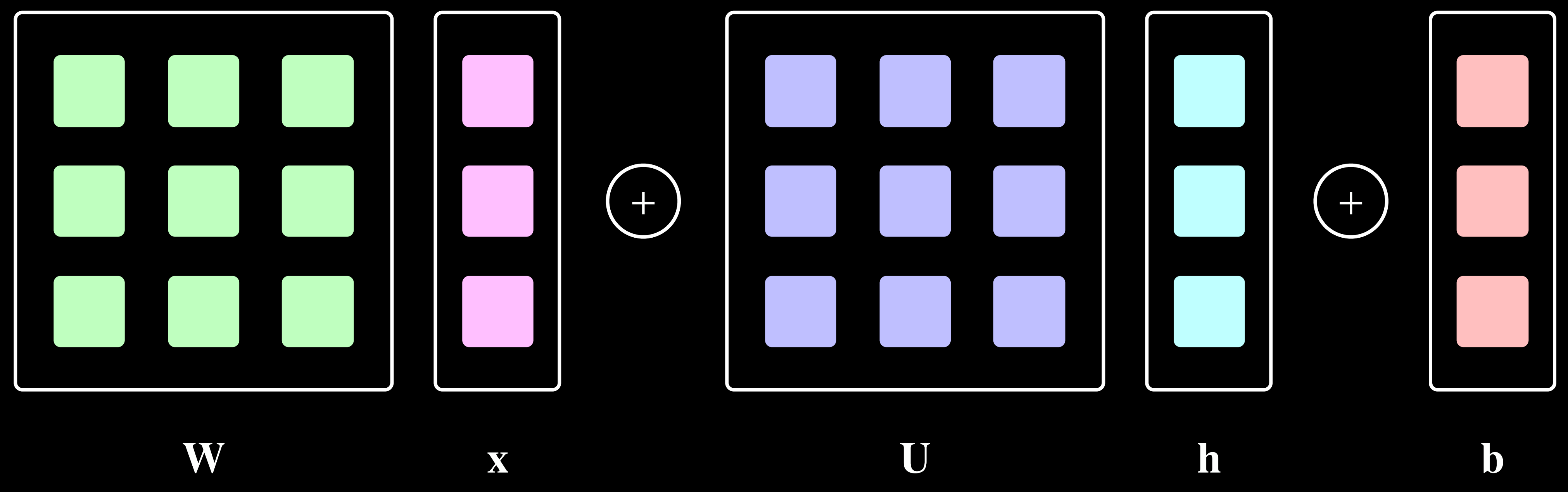


Algebra Simplification

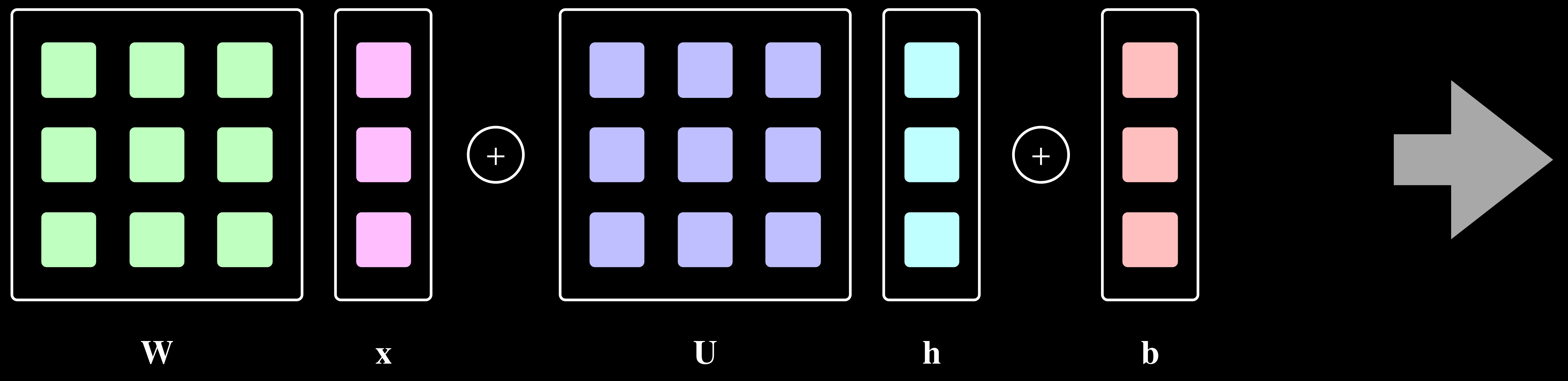


Linear Algebra Fusion

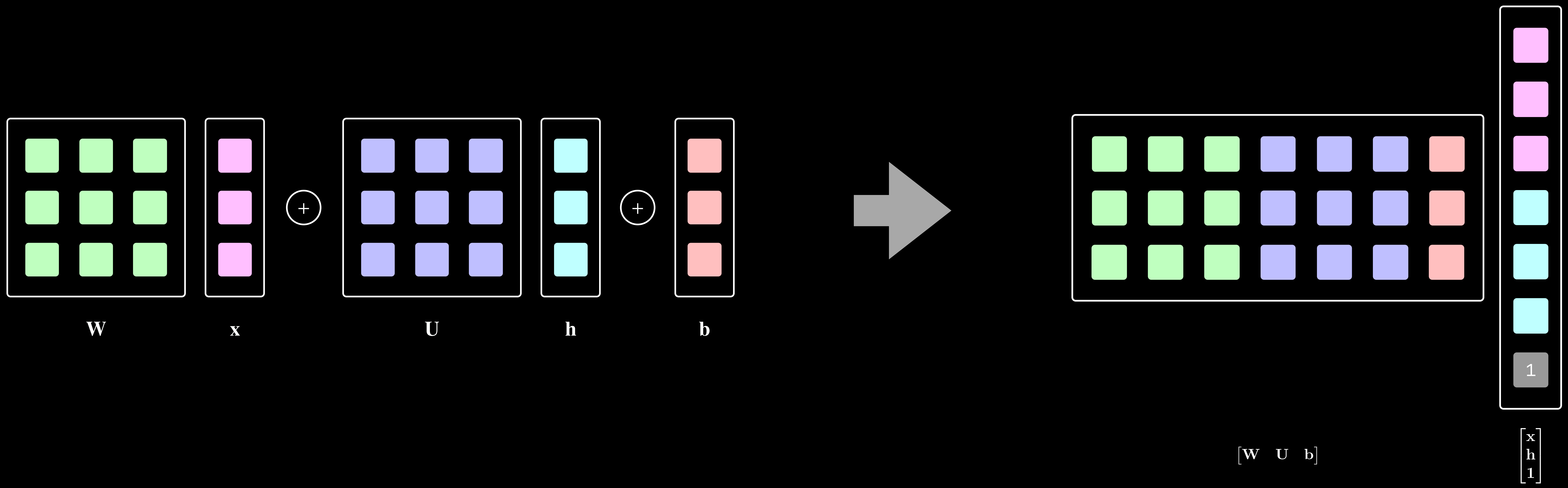
Linear Algebra Fusion



Linear Algebra Fusion

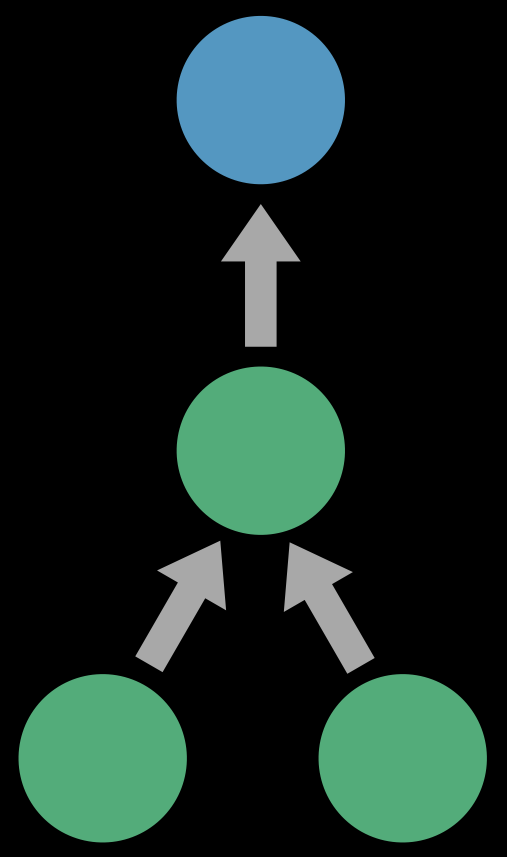


Linear Algebra Fusion



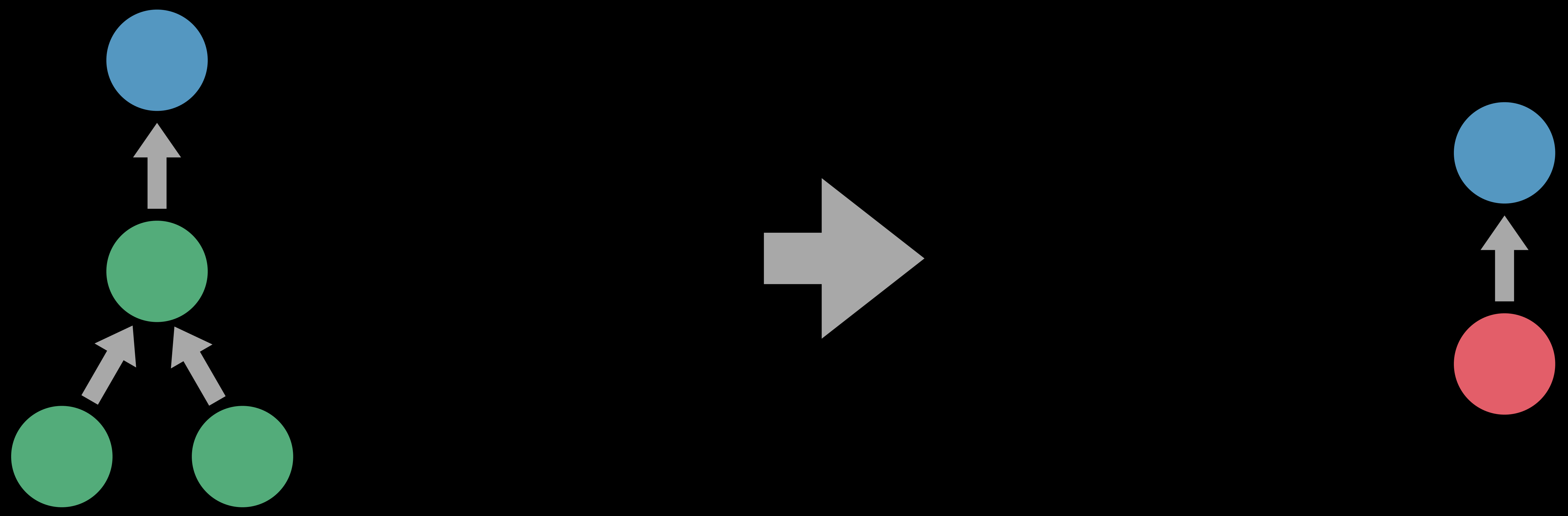
Compute Kernel Fusion

Compute Kernel Fusion



Compute Kernel Fusion

Compute Kernel Fusion



```
func @inference: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>) -> <1 x 10 x f32> {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    return %0.1: <1 x 10 x f32>
}
```

```
func @inference: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>) -> <1 x 10 x f32> {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    return %0.1: <1 x 10 x f32>
}

[gradient @inference wrt 1, 2]
func @inference_grad: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>)
                    -> (<784 x 10 x f32>, <1 x 10 x f32>)
```

```
func @inference: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>) -> <1 x 10 x f32> {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    return %0.1: <1 x 10 x f32>
}

[gradient @inference wrt 1, 2]
func @inference_grad: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>)
                    -> (<784 x 10 x f32>, <1 x 10 x f32>)
```

Differentiation Pass

Canonicalizes every gradient function declaration in an IR module

```
func @inference: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>) -> <1 x 10 x f32> {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    return %0.1: <1 x 10 x f32>
}


func @inference_grad: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>)
                    -> (<784 x 10 x f32>, <1 x 10 x f32>) {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):






}
```

```
func @inference: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>) -> <1 x 10 x f32> {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    return %0.1: <1 x 10 x f32>
}
```

```
func @inference_grad: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>)
                    -> (<784 x 10 x f32>, <1 x 10 x f32>) {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
```

Copy instructions from forward pass

```
}
```

```
func @inference: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>) -> <1 x 10 x f32> {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    return %0.1: <1 x 10 x f32>
}


func @inference_grad: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>)
                    -> (<784 x 10 x f32>, <1 x 10 x f32>) {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>




}
```

```
func @inference: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>) -> <1 x 10 x f32> {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    return %0.1: <1 x 10 x f32>
}


func @inference_grad: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>)
                    -> (<784 x 10 x f32>, <1 x 10 x f32>) {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>

       Generate backward computation

}
```

```
func @inference: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>) -> <1 x 10 x f32> {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    return %0.1: <1 x 10 x f32>
}


func @inference_grad: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>)
                    -> (<784 x 10 x f32>, <1 x 10 x f32>) {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    %0.2 = transpose %x: <1 x 784 x f32>
    %0.3 = multiply %0.2: <1 x 784 x f32>, 1: f32
    return (%0.3: <1 x 10 x f32>, 1: f32): (<1 x 10 x f32>, f32)
}
```

```
func @inference: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>) -> <1 x 10 x f32> {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    return %0.1: <1 x 10 x f32>
}
```

```
func @inference_grad: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>)
                    -> (<784 x 10 x f32>, <1 x 10 x f32>) {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    %0.2 = transpose %x: <1 x 784 x f32>
    %0.3 = multiply %0.2: <1 x 784 x f32>, 1: f32
    return (%0.3: <1 x 10 x f32>, 1: f32): (<1 x 10 x f32>, f32)
}
```

```
func @inference: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>) -> <1 x 10 x f32> {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    return %0.1: <1 x 10 x f32>
}
```

```
func @inference_grad: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>)
                    -> (<784 x 10 x f32>, <1 x 10 x f32>) {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    %0.2 = transpose %x: <1 x 784 x f32>
    %0.3 = multiply %0.2: <1 x 784 x f32>, 1: f32
    return (%0.3: <1 x 10 x f32>, 1: f32): (<1 x 10 x f32>, f32)
}
```

Algebra Simplification Pass

```
func @inference: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>) -> <1 x 10 x f32> {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    return %0.1: <1 x 10 x f32>
}


func @inference_grad: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>)
                    -> (<784 x 10 x f32>, <1 x 10 x f32>) {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    %0.2 = transpose %x: <1 x 784 x f32>
    return (%0.2: <1 x 10 x f32>, 1: f32): (<1 x 10 x f32>, f32)
}
```

```
func @inference: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>) -> <1 x 10 x f32> {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    return %0.1: <1 x 10 x f32>
}
```

```
func @inference_grad: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>)
                    -> (<784 x 10 x f32>, <1 x 10 x f32>) {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    %0.2 = transpose %x: <1 x 784 x f32>
    return (%0.2: <1 x 10 x f32>, 1: f32): (<1 x 10 x f32>, f32)
}
```

```
func @inference: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>) -> <1 x 10 x f32> {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    return %0.1: <1 x 10 x f32>
}
```

```
func @inference_grad: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>)
                    -> (<784 x 10 x f32>, <1 x 10 x f32>) {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    %0.2 = transpose %x: <1 x 784 x f32>
    return (%0.2: <1 x 10 x f32>, 1: f32): (<1 x 10 x f32>, f32)
}
```

Dead Code Elimination Pass

```
func @inference: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>) -> <1 x 10 x f32> {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = dot %x: <1 x 784 x f32>, %w: <784 x 10 x f32>
    %0.1 = add %0.0: <1 x 10 x f32>, %b: <1 x 10 x f32>
    return %0.1: <1 x 10 x f32>
}


func @inference_grad: (<1 x 784 x f32>, <784 x 10 x f32>, <1 x 10 x f32>)
                    -> (<784 x 10 x f32>, <1 x 10 x f32>) {
'entry(%x: <1 x 784 x f32>, %w: <784 x 10 x f32>, %b: <1 x 10 x f32>):
    %0.0 = transpose %x: <1 x 784 x f32>
    return (%0.0: <1 x 10 x f32>, 1: f32): (<1 x 10 x f32>, f32)
}
```

Compilation Phases

Compilation Phases

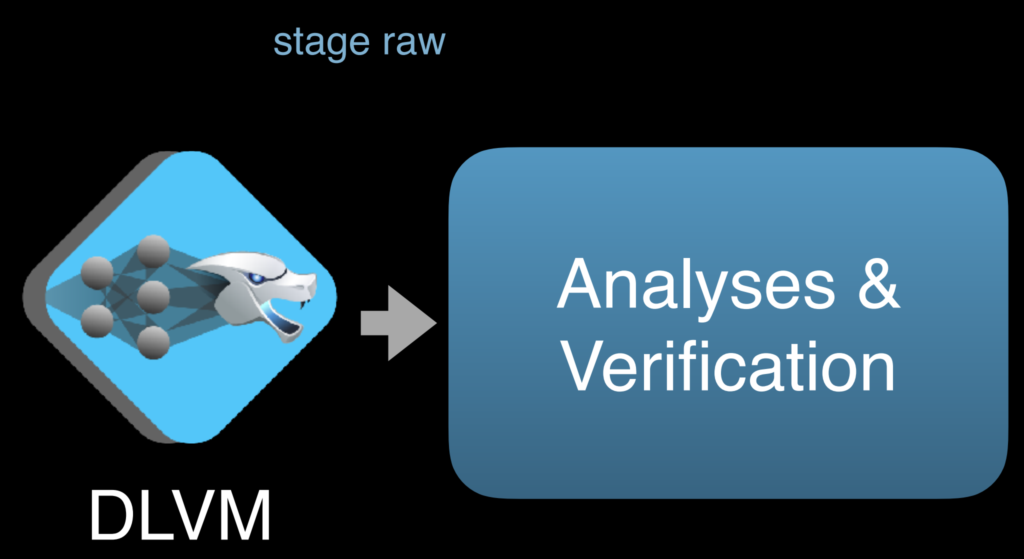
DLVM

Compilation Phases

stage raw

DLVM

Compilation Phases



Compilation Phases

stage raw

DLVM

Analyses & Verification

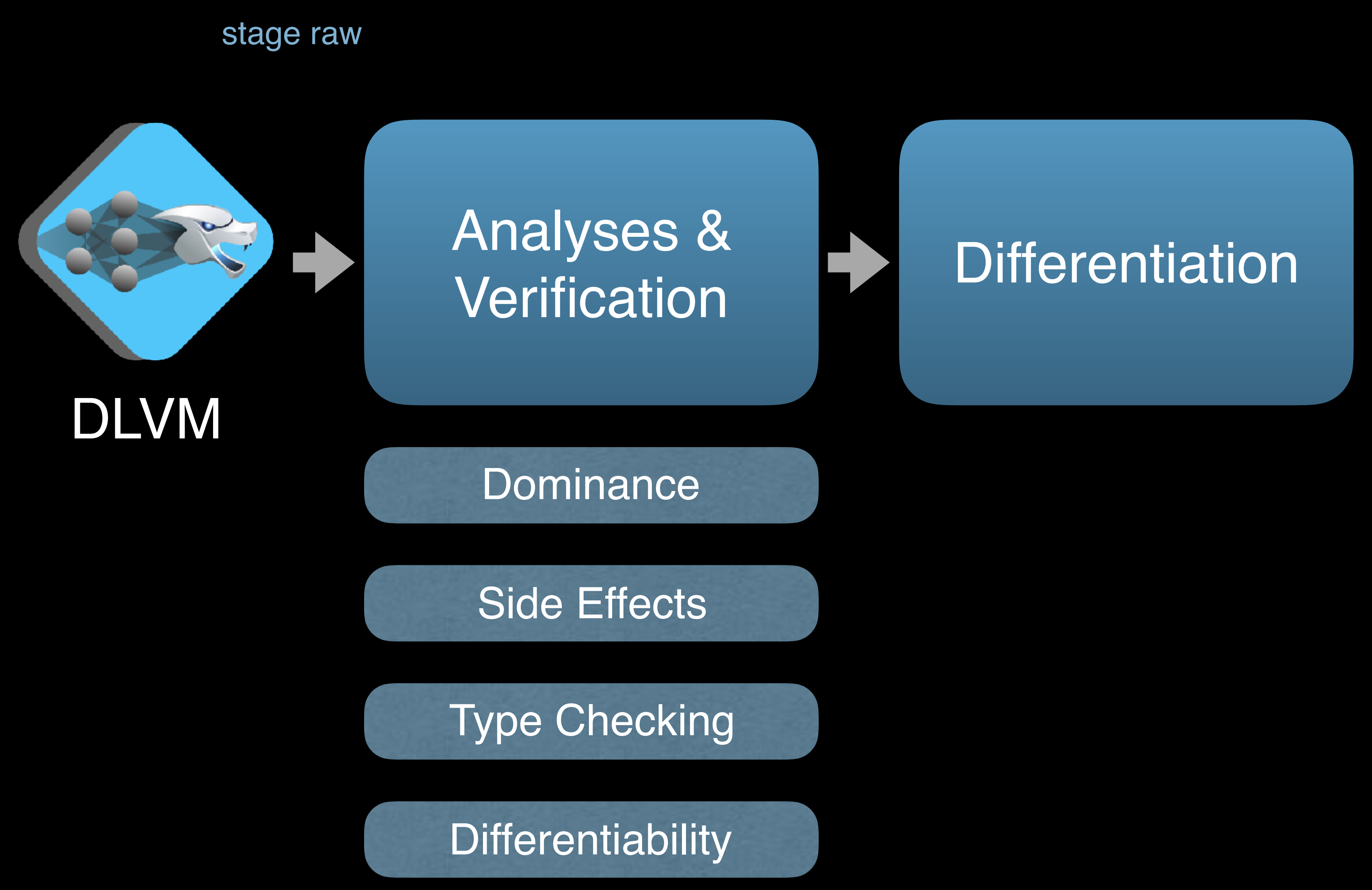
Dominance

Side Effects

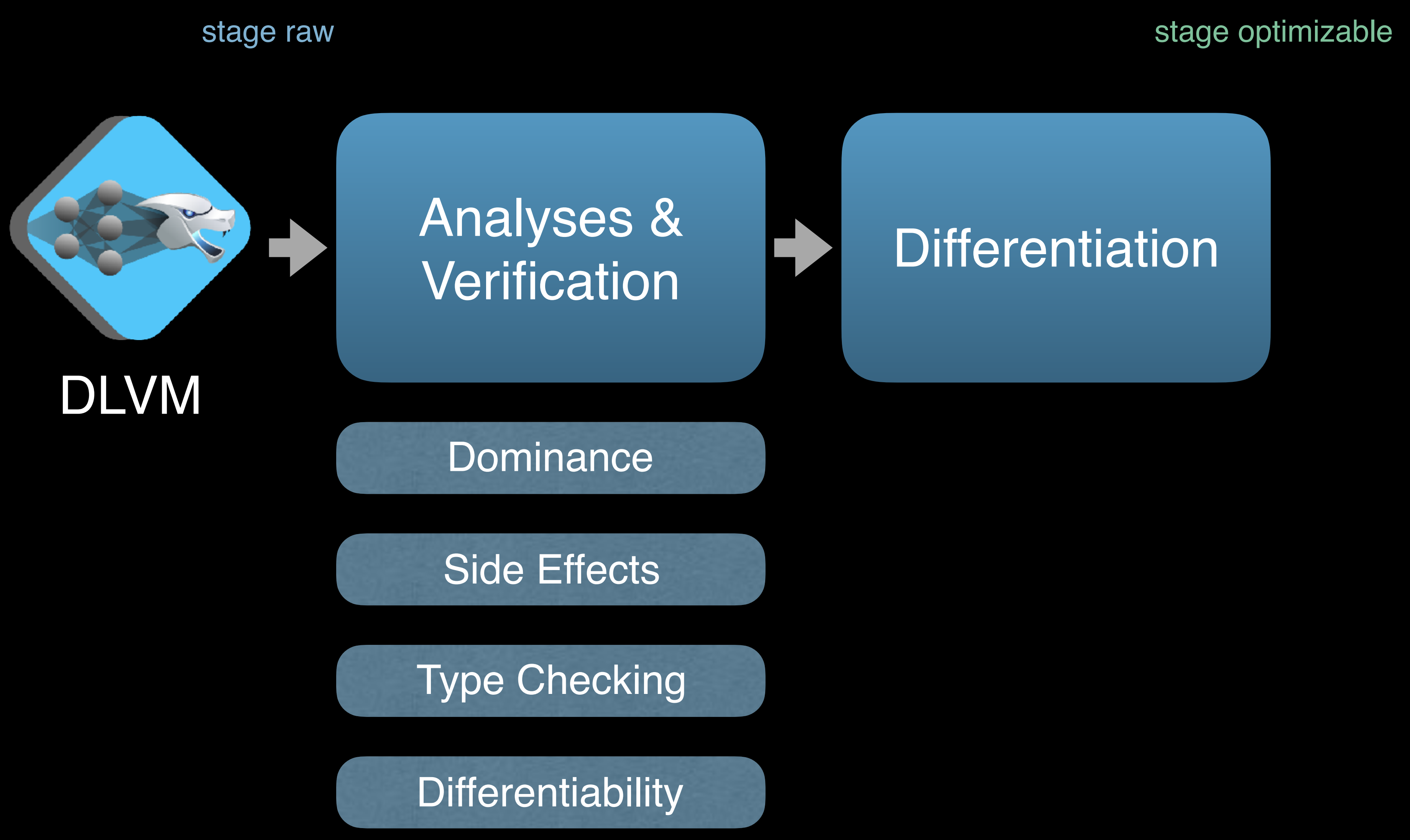
Type Checking

Differentiability

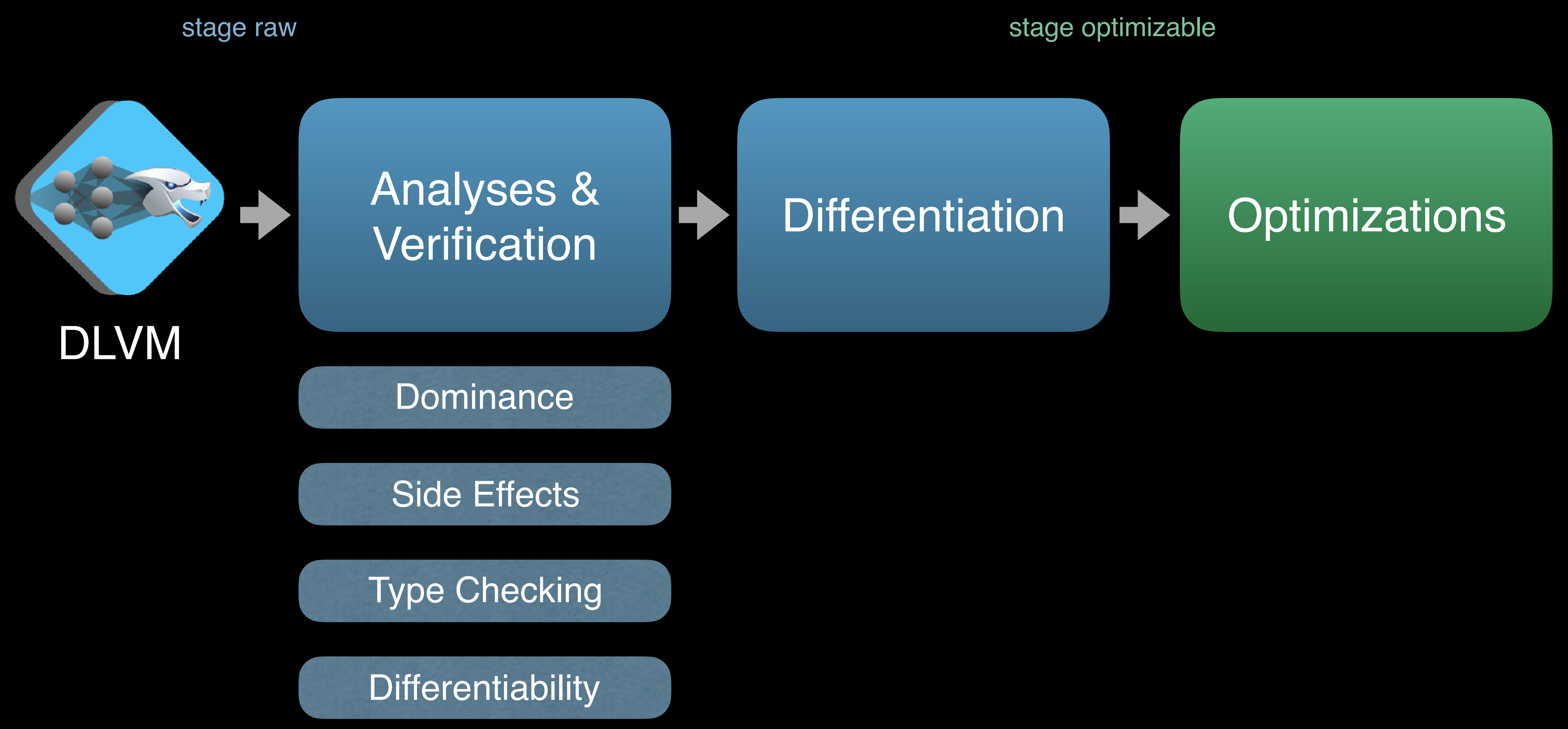
Compilation Phases



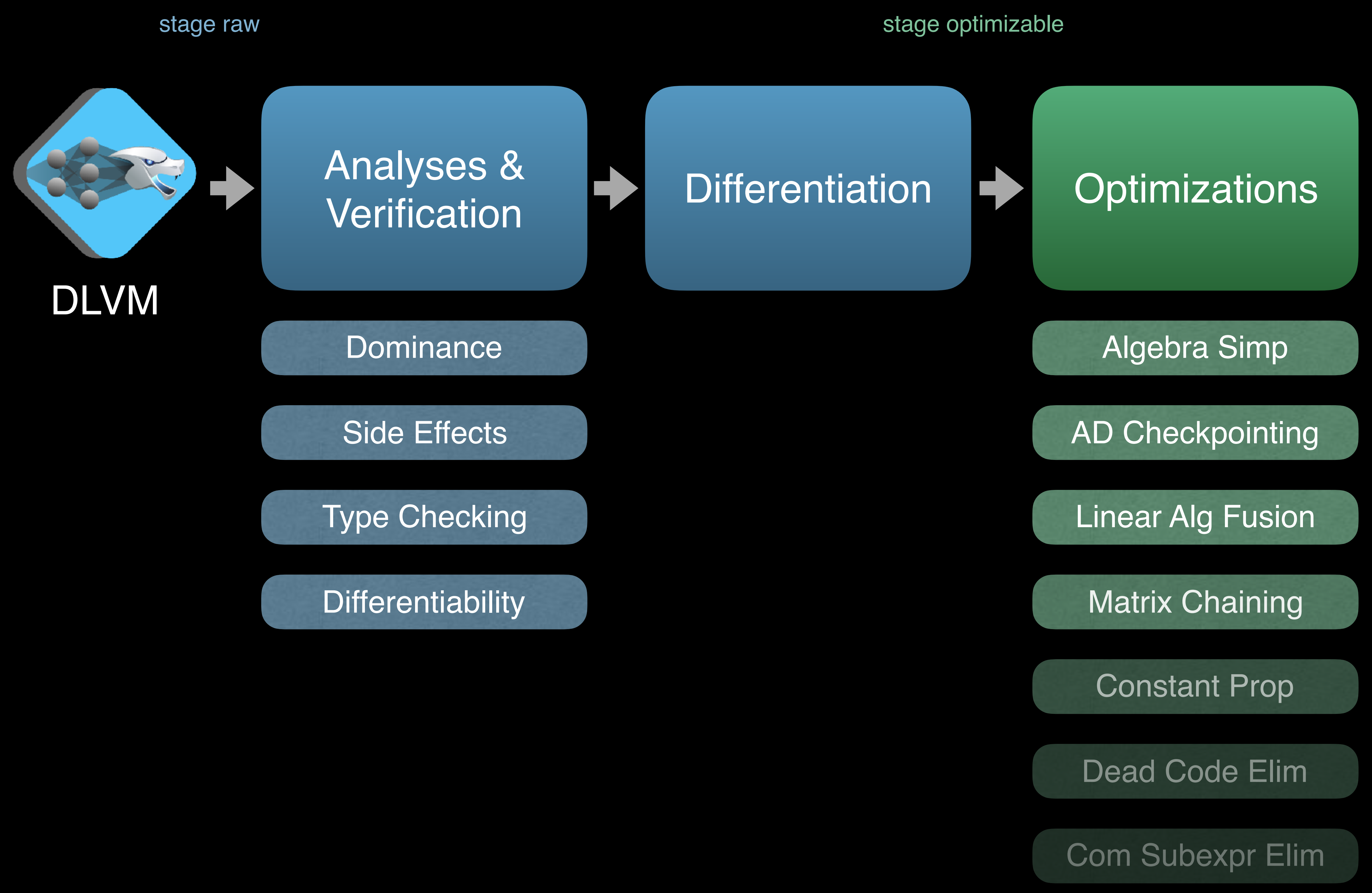
Compilation Phases



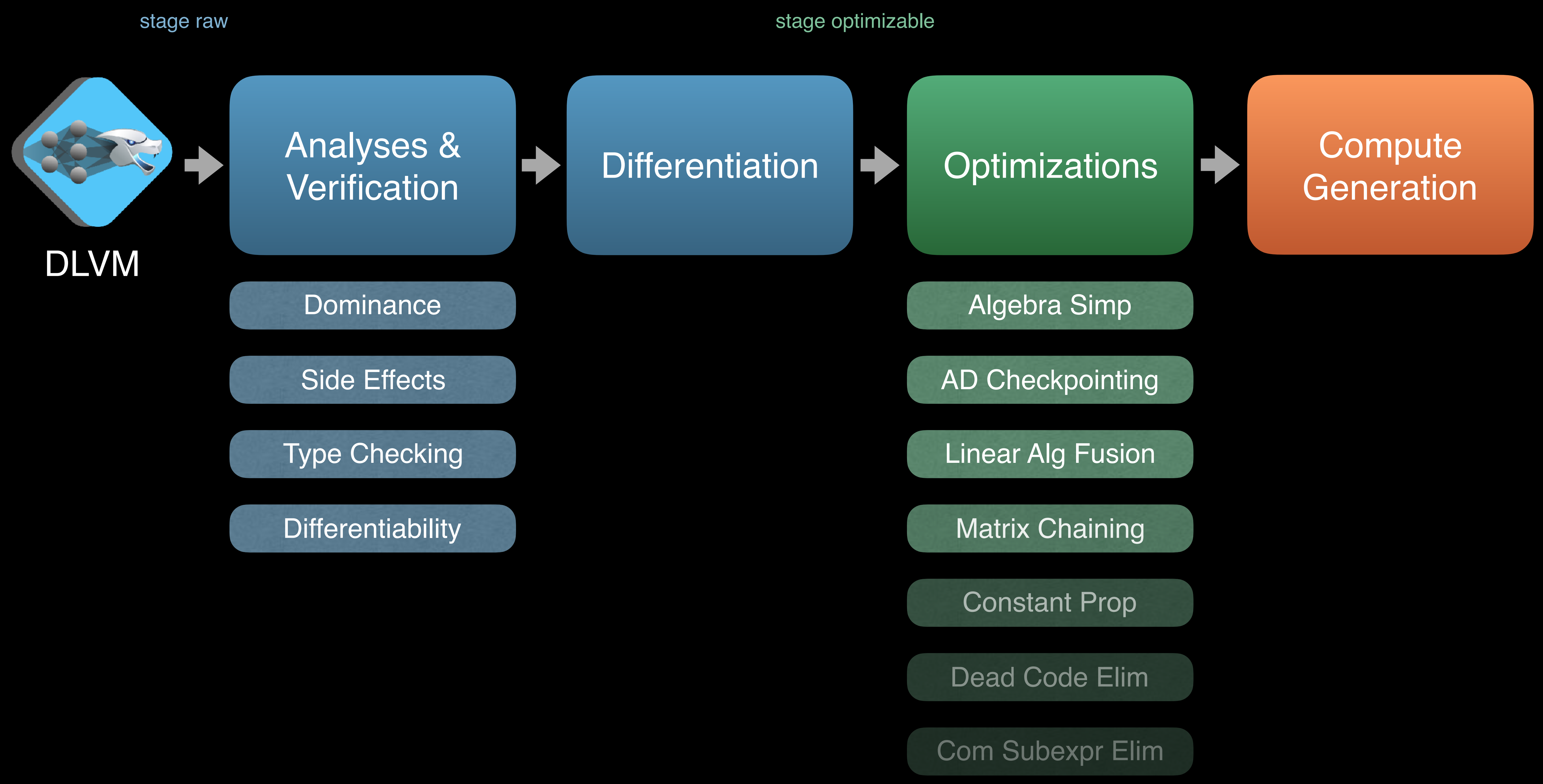
Compilation Phases



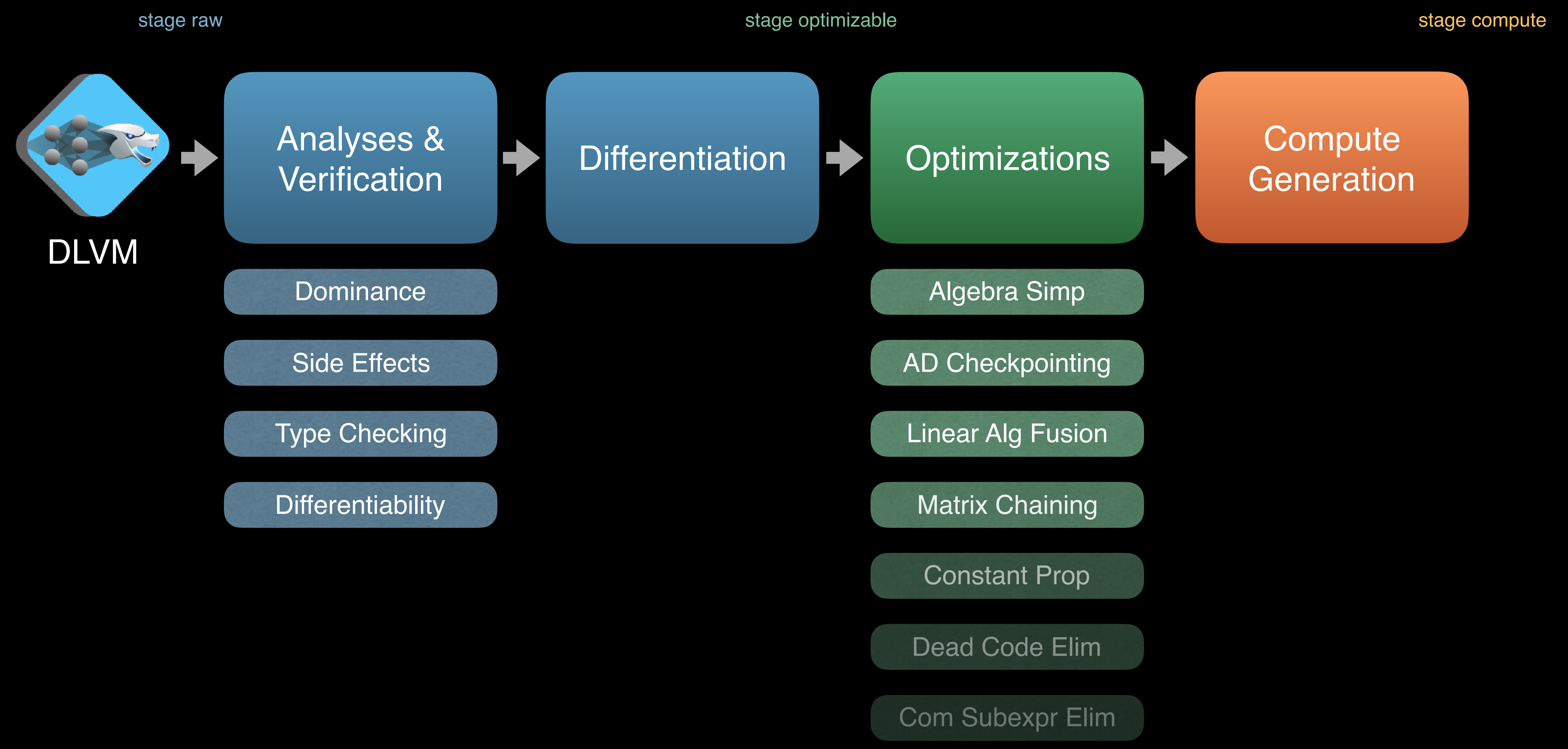
Compilation Phases



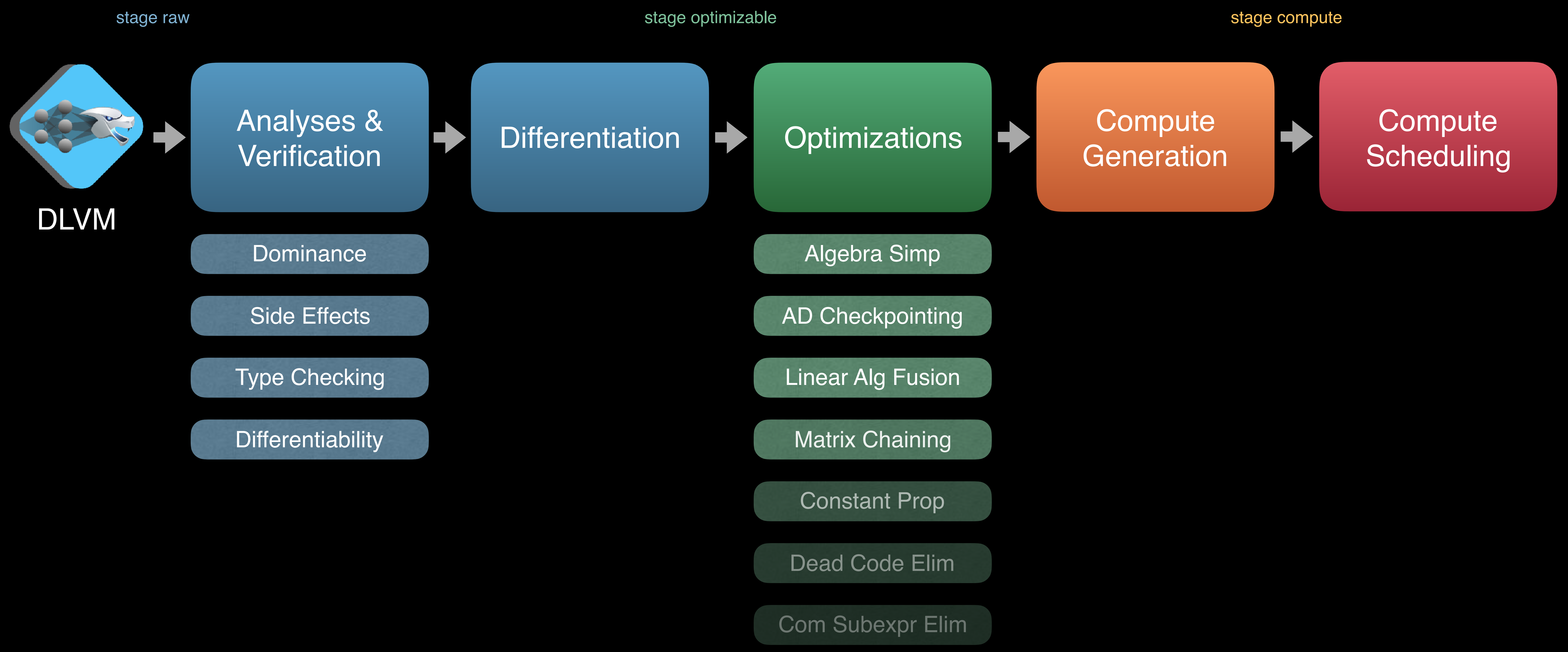
Compilation Phases



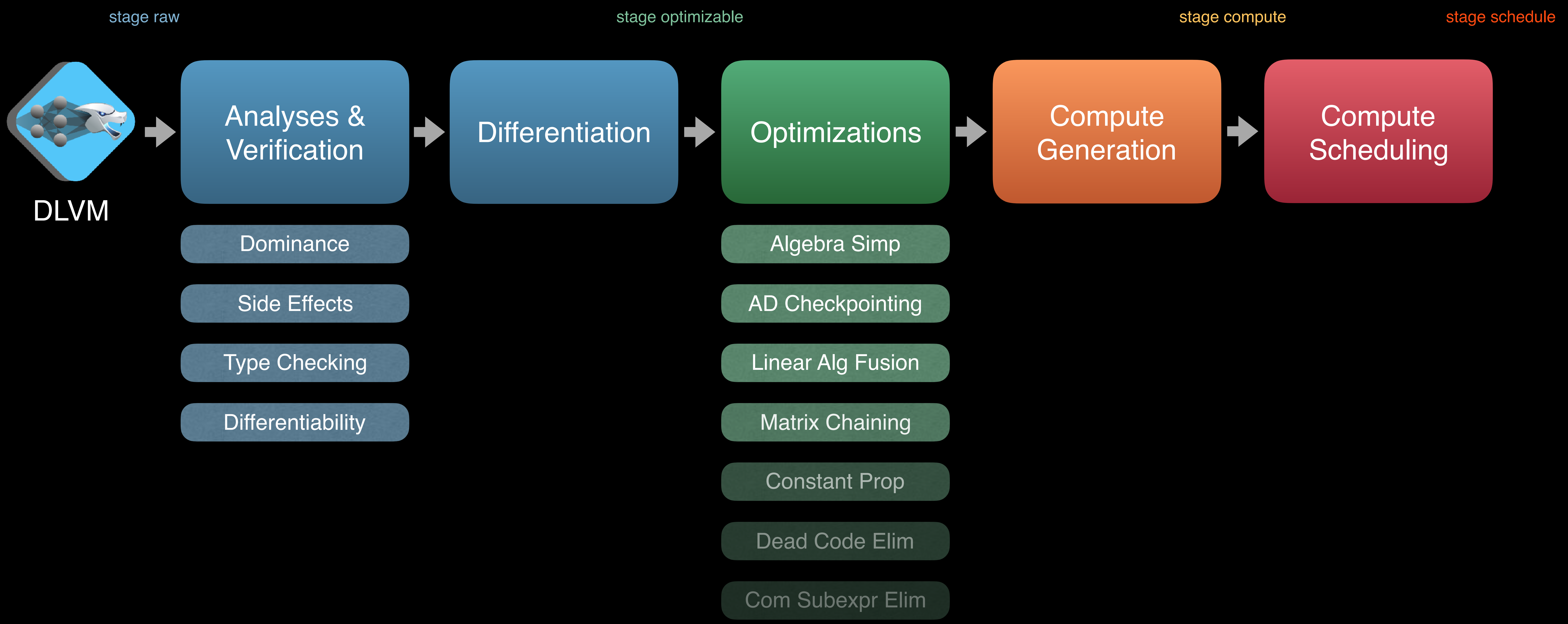
Compilation Phases



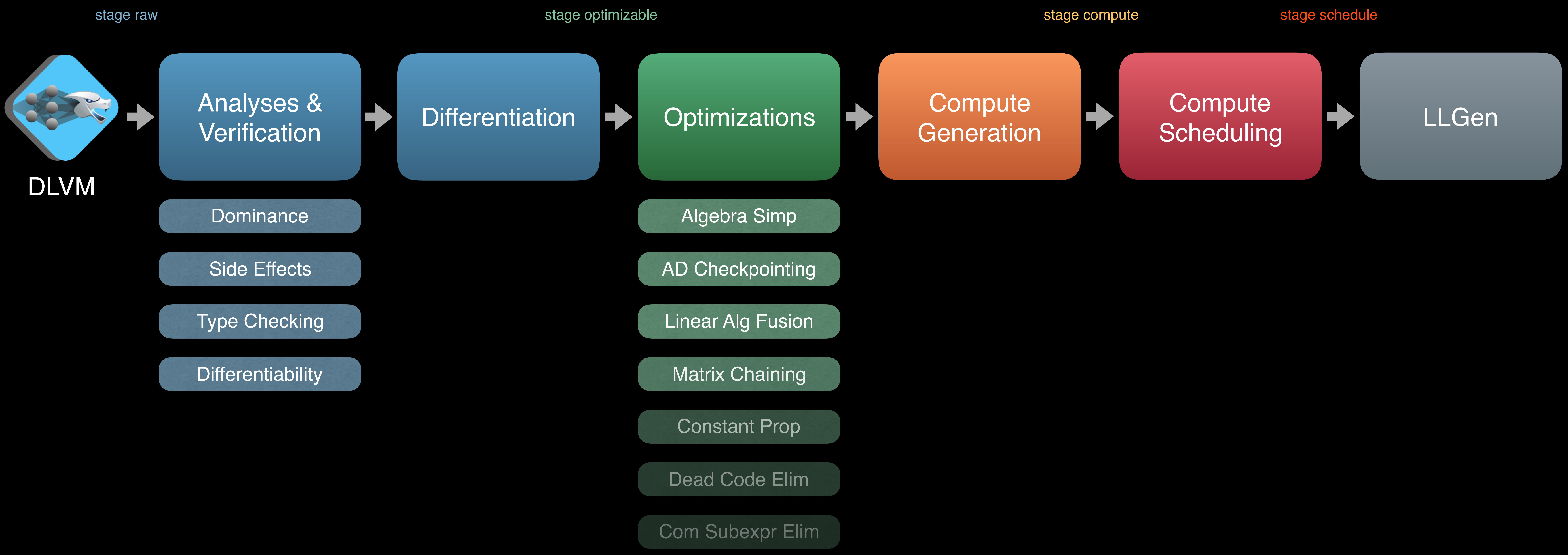
Compilation Phases



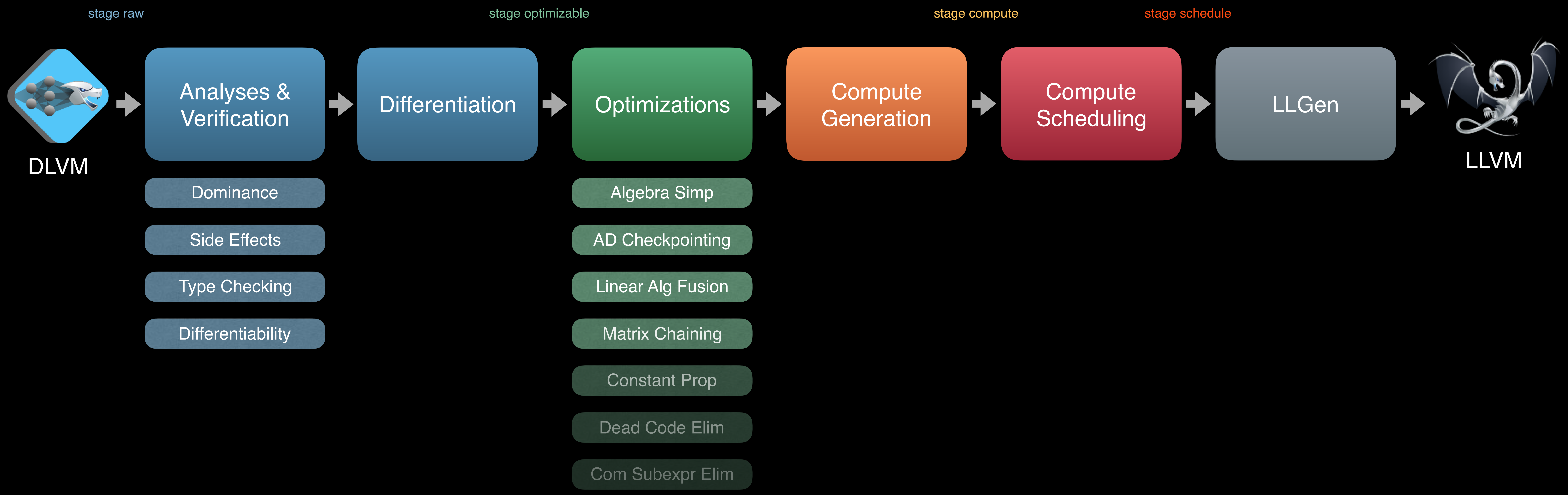
Compilation Phases



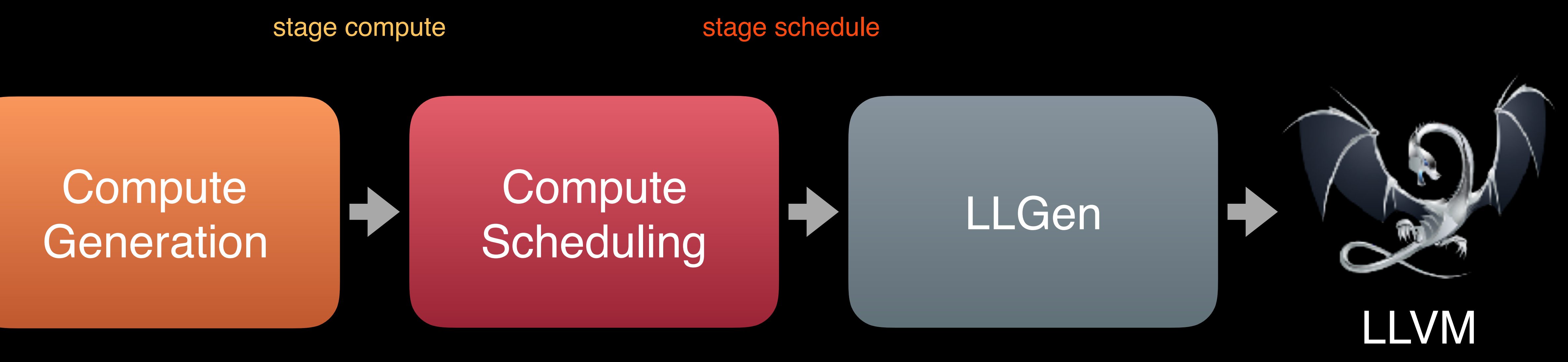
Compilation Phases



Compilation Phases



Compilation Phases



Compilation Phases

Using DLVM

NNKit: Embedded DSL in Swift

NNKit

NNKit

- Staged embedded DSL

NNKit

- Staged embedded DSL
- Type-safe

NNKit

- Staged embedded DSL
- Type-safe
- Generating DLVM IR on the fly

NNKit

NNKit

- Statically ranked tensors
  - `T, Tensor1D<T>, Tensor2D<T>, Tensor3D<T>, Tensor4D<T>`

NNKit

- Statically ranked tensors
  - `T, Tensor1D<T>, Tensor2D<T>, Tensor3D<T>, Tensor4D<T>`
- Type wrapper – `Rep<Wrapped>`
  - `Rep<Float>, Rep<Tensor1D<Float>>, Rep<Tensor2D<T>>`

NNKit

- Statically ranked tensors
  - T, Tensor1D<T>, Tensor2D<T>, Tensor3D<T>, Tensor4D<T>
- Type wrapper – Rep<Wrapped>
  - Rep<Float>, Rep<Tensor1D<Float>>, Rep<Tensor2D<T>>
- Operator overloading
  - func + <T: Numeric>(_: Rep<T>, _: Rep<T>) -> Rep<T>
  - func dot <T: Numeric>(_: Rep<Tensor2D<T>>, _: Rep<Tensor2D<T>>)
    -> Rep<Tensor2D<T>>

NNKit

NNKit

- Lambda abstraction
  - `func lambda<T, U>(_ f: (Rep<T>) -> Rep<U>) -> Rep<(T) -> U>`

NNKit

- Lambda abstraction
  - `func lambda<T, U>(_ f: (Rep<T>) -> Rep<U>) -> Rep<(T) -> U>`
- Function application
  - `subscript<T, U>(arg: Rep<T>) -> Rep<U> where Wrapped == (T) -> U`
  - `subscript<T, U>(arg: T) -> U where Wrapped == (T) -> U`
    `// JIT compilation here`

NNKit

```
typealias Float2D = Tensor2D<Float>

struct Parameters {
    var w: Float2D
    var b: Float2D
}

let inference: Rep<(Float2D, Float2D, Float2D) -> Float2D> =
    lambda { x, w, b in
        dot(x, w) + b
    }

let foo: Parameters = …
inference[foo.w, foo.b]
```

Define and execute inference function

NNKit

NNKit

- Seamless JIT compilation via DLVM

NNKit

- Seamless JIT compilation via DLVM
- Support dynamic tensor dimensionality by JIT

NNKit

- Seamless JIT compilation via DLVM
- Support dynamic tensor dimensionality by JIT
- Easy to pre-compile models to binary, when shapes are known

Recap

Recap

- Deep learning system is a compiler & language problem

Recap

- Deep learning system is a compiler & language problem
  - Frontend: application IDE/DSL, layer DSL, math DSL

Recap

- Deep learning system is a compiler & language problem
  - Frontend: application IDE/DSL, layer DSL, math DSL
  - Backend: compiler infrastructure

Recap

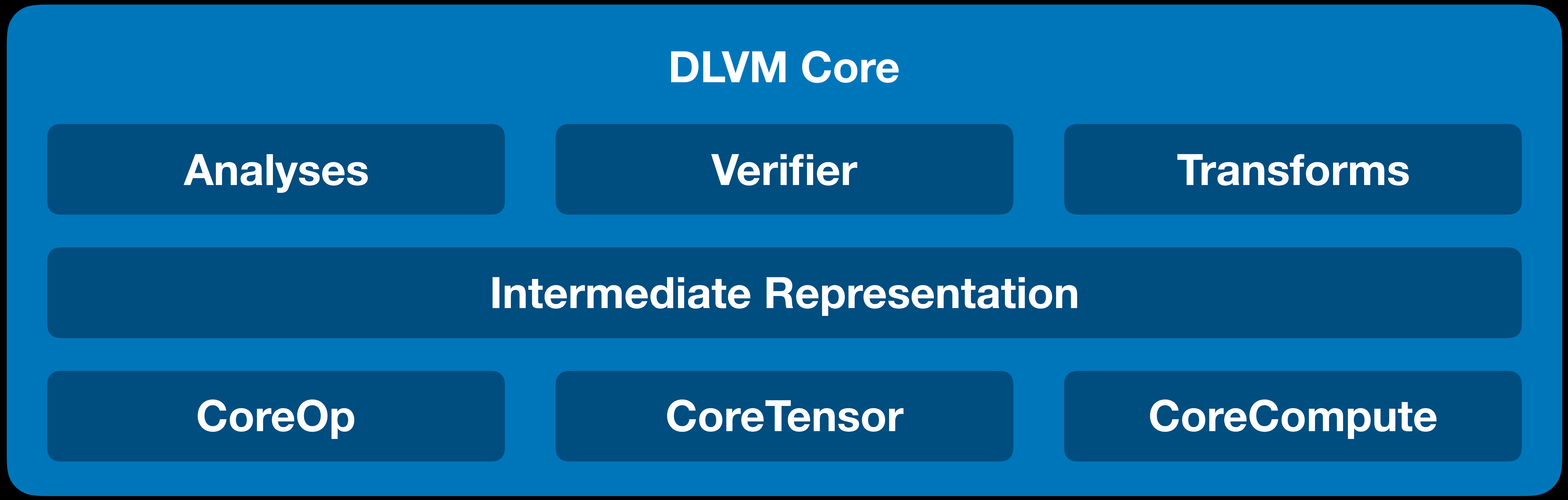
- Deep learning system is a compiler & language problem
  - Frontend: application IDE/DSL, layer DSL, math DSL
  - Backend: compiler infrastructure
- DLVM - performance & expressiveness of DSLs

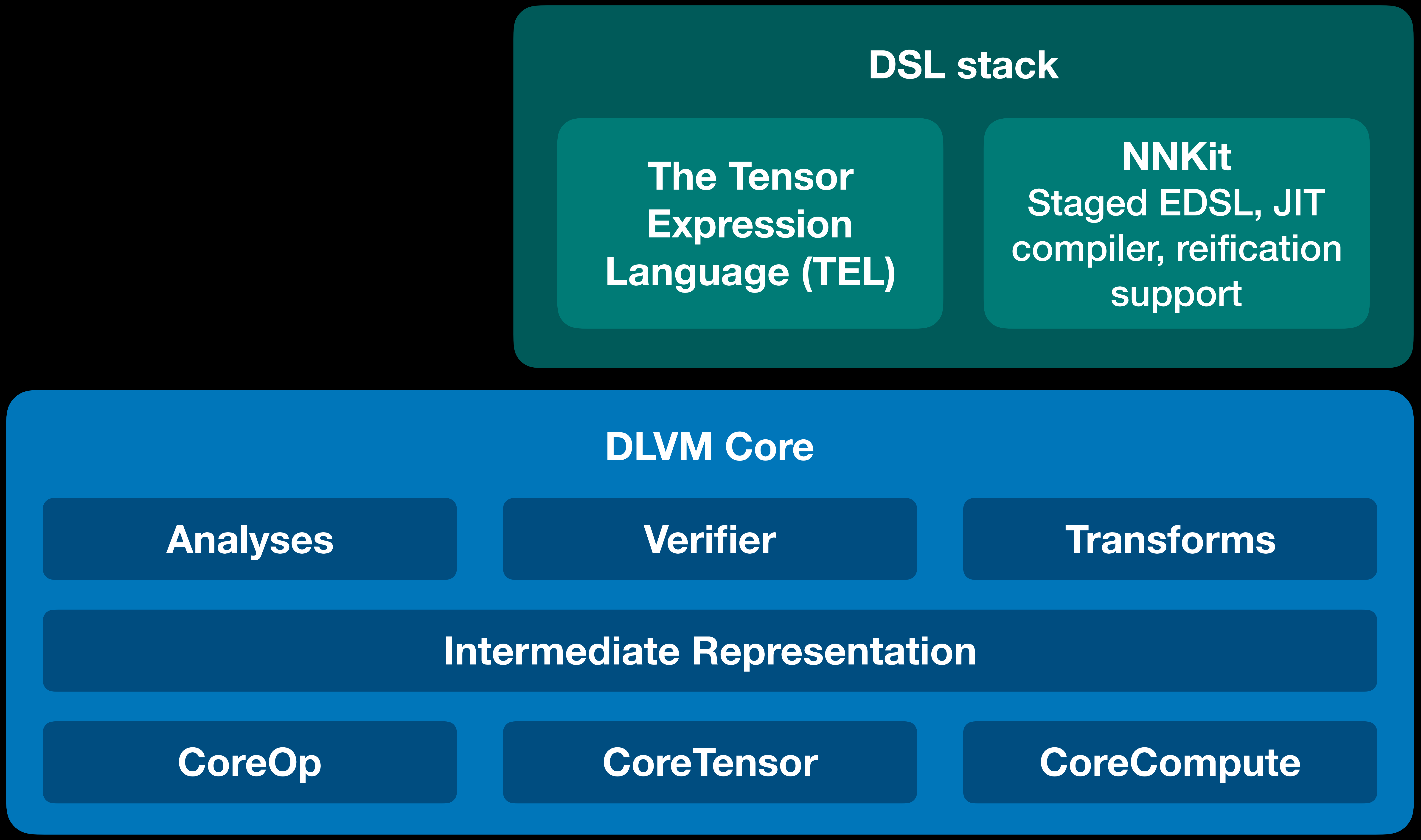
Recap

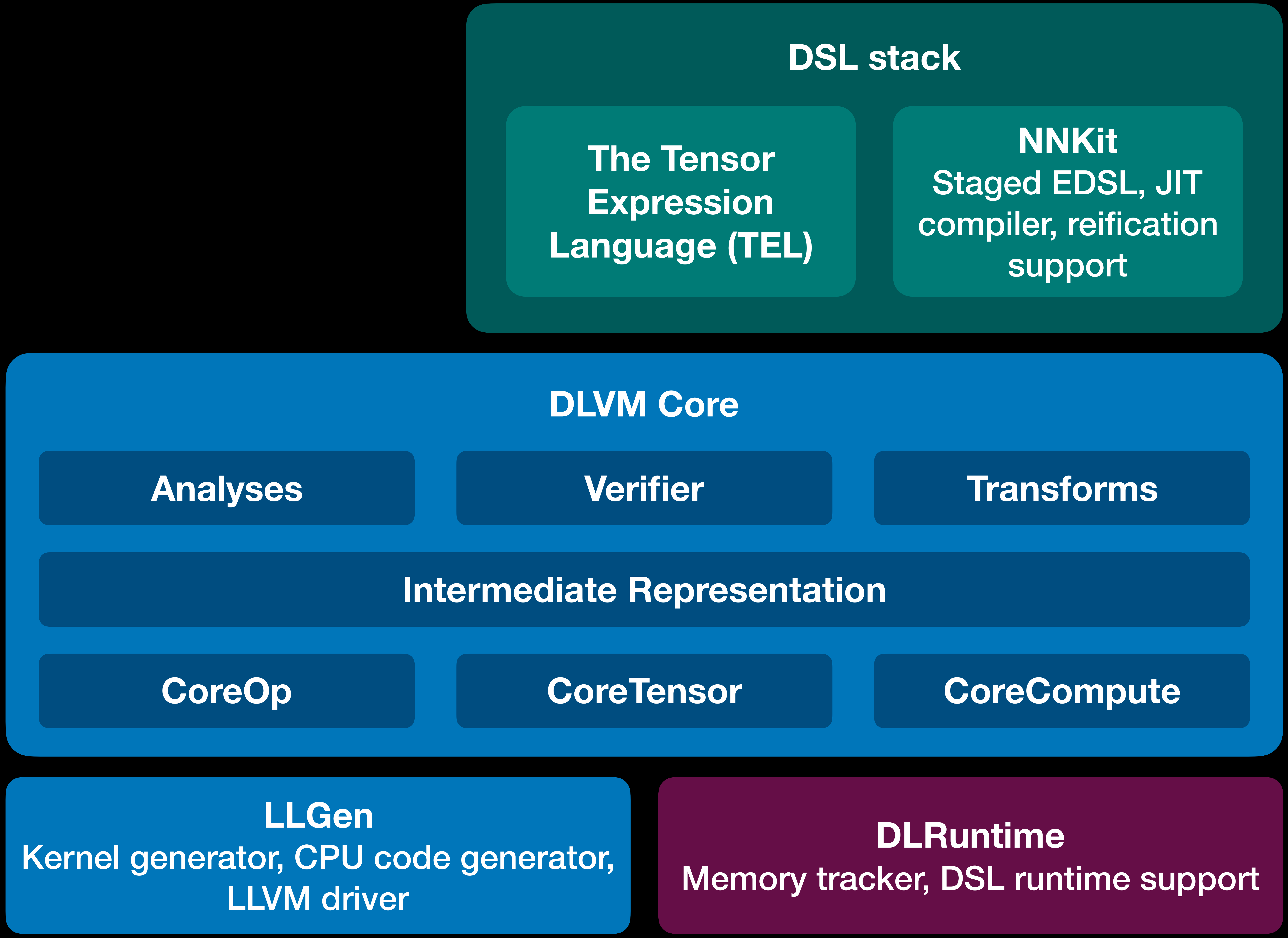
- Deep learning system is a compiler & language problem
  - Frontend: application IDE/DSL, layer DSL, math DSL
  - Backend: compiler infrastructure
- DLVM - performance & expressiveness of DSLs
- DSLs - safety, reliability & developer experience

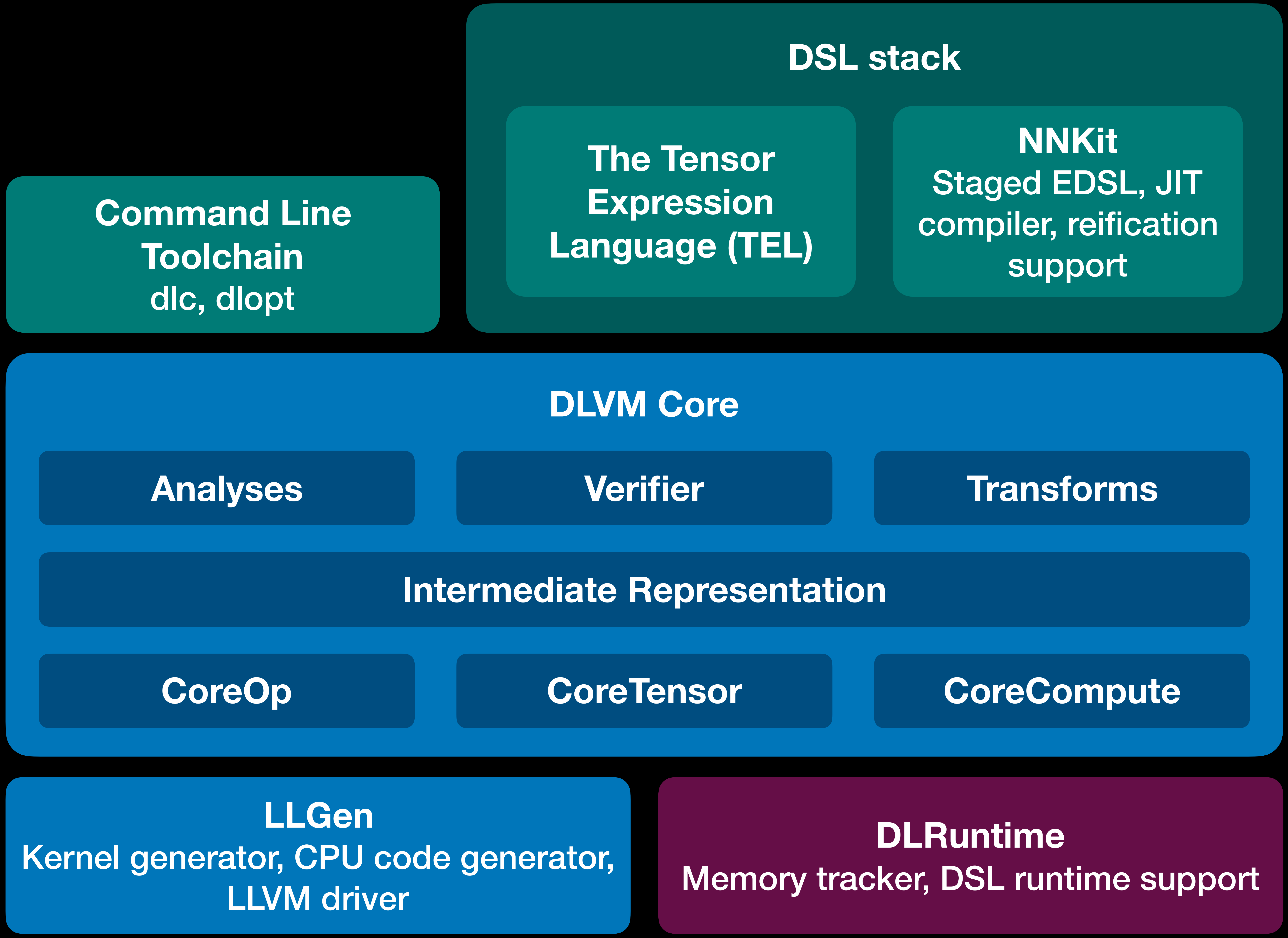
Recap

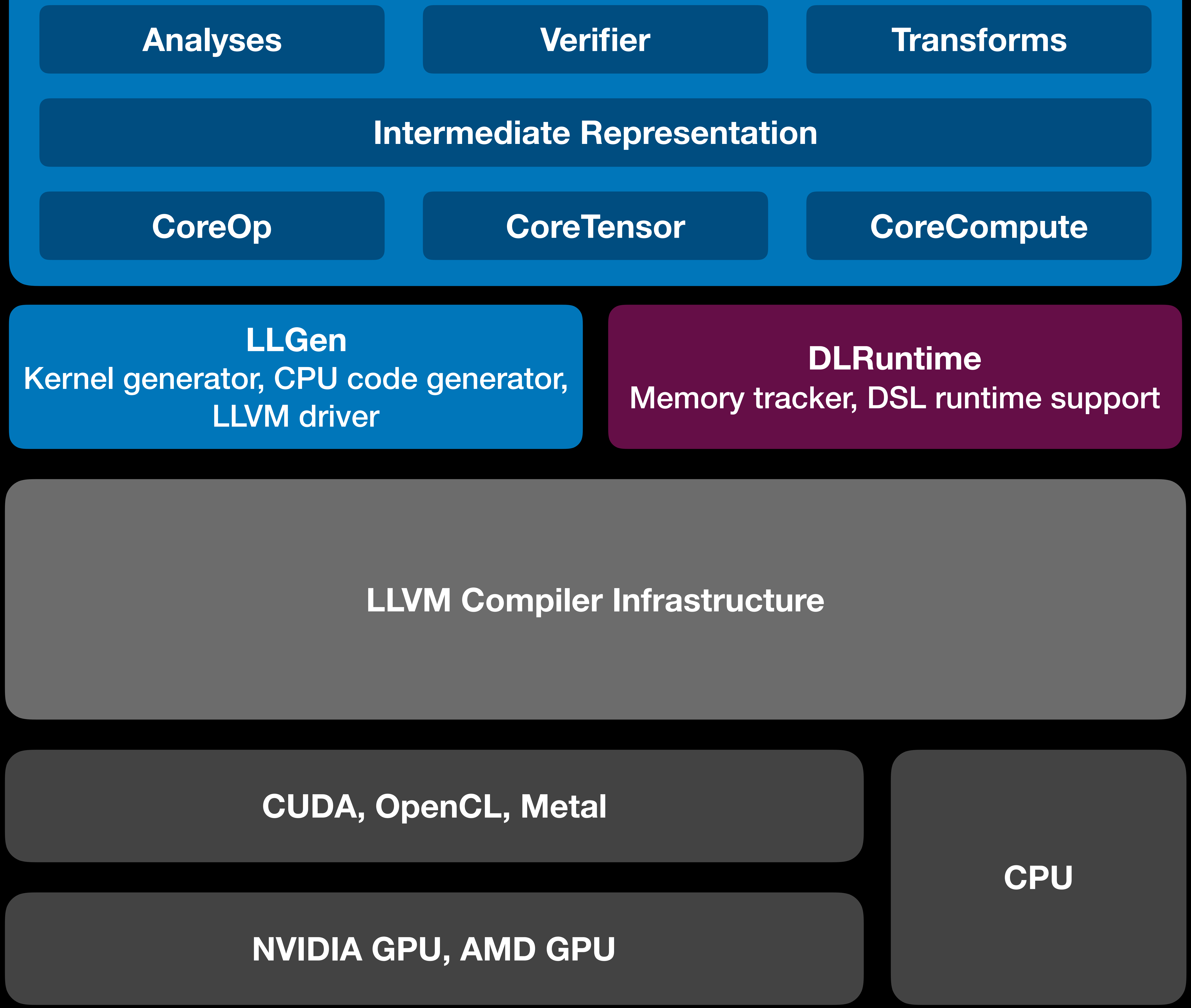
- Deep learning system is a compiler & language problem
  - Frontend: application IDE/DSL, layer DSL, math DSL
  - Backend: compiler infrastructure
- DLVM - performance & expressiveness of DSLs
- DSLs - safety, reliability & developer experience
- Bring software engineering principles into ML systems

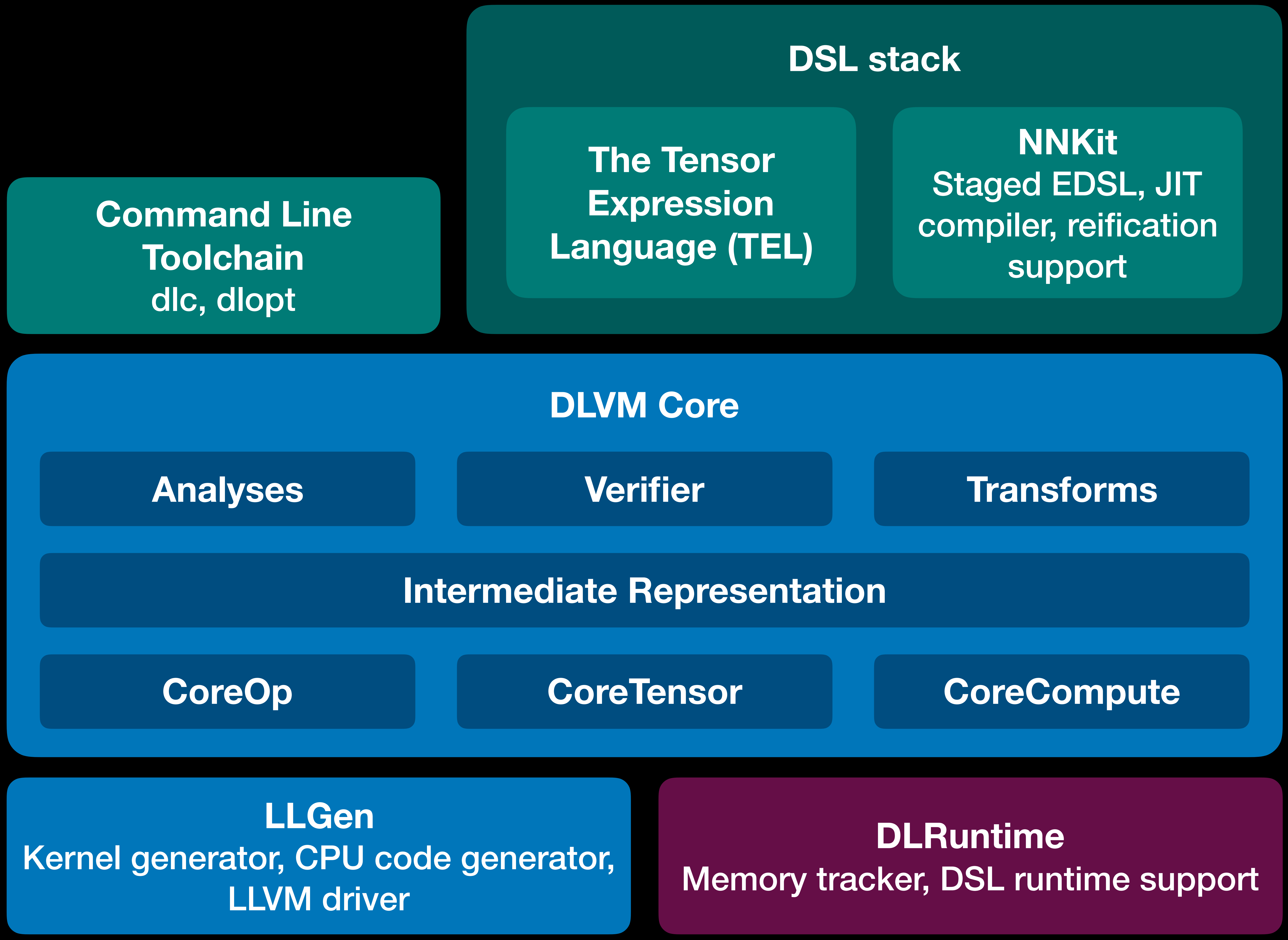

DLVM Core
Analyses
Verifier
Transforms
Intermediate Representation
CoreOp
CoreTensor
CoreCompute


DSL stack
The Tensor Expression Language (TEL)
NNKit
Staged EDSL, JIT compiler, reification support
DLVM Core
Analyses
Verifier
Transforms
Intermediate Representation
CoreOp
CoreTensor
CoreCompute

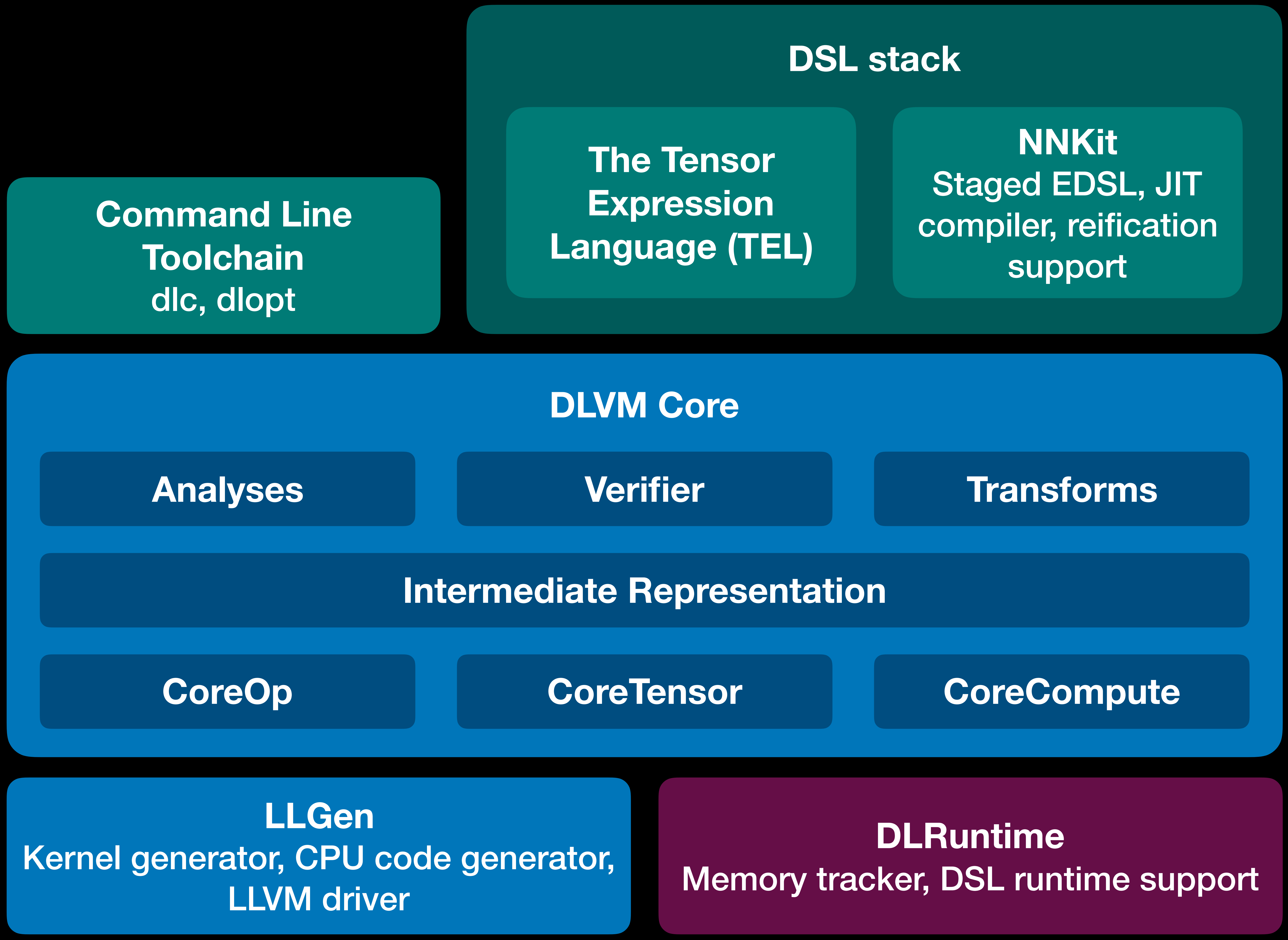

DSL stack
The Tensor Expression Language (TEL)
NNKit
Staged EDSL, JIT compiler, reification support
DLVM Core
Analyses
Verifier
Transforms
Intermediate Representation
CoreOp
CoreTensor
CoreCompute
LLGen
Kernel generator, CPU code generator, LLVM driver
DLRuntime
Memory tracker, DSL runtime support


DSL stack
The Tensor Expression Language (TEL)
NNKit
Staged EDSL, JIT compiler, reification support
Command Line Toolchain
dlc, dlopt
DLVM Core
Analyses
Verifier
Transforms
Intermediate Representation
CoreOp
CoreTensor
CoreCompute
LLGen
Kernel generator, CPU code generator, LLVM driver
DLRuntime
Memory tracker, DSL runtime support


Analyses
Verifier
Transforms
Intermediate Representation
CoreOp
CoreTensor
CoreCompute
LLGen
Kernel generator, CPU code generator, LLVM driver
DLRuntime
Memory tracker, DSL runtime support
LLVM Compiler Infrastructure
CUDA, OpenCL, Metal
NVIDIA GPU, AMD GPU
CPU



DSL stack
The Tensor Expression Language (TEL)
NNKit
Staged EDSL, JIT compiler, reification support
Command Line Toolchain
dlc, dlopt
DLVM Core
Analyses
Verifier
Transforms
Intermediate Representation
CoreOp
CoreTensor
CoreCompute
LLGen
Kernel generator, CPU code generator, LLVM driver
DLRuntime
Memory tracker, DSL runtime support



Written entirely in Swift

dlvm.org

DLVM