https://hao-ai-lab.github.io/dsc204a-f25/

DSC 204A: Scalable Data Systems
Fall 2025

Staff

Instructor: Hao Zhang

TAs: Mingjia Huo, Yuxuan Zhang

@haozhangml @haoailab

haozhang@ucsd.edu

Logistics

- All 4 Guest Lectures complete
  - Eugune Wu: classic DB + HCI researcher
  - Shreya Shankar: Modern DB + HCI researcher

  - Andrey Cheng:
    - DB researcher, but now working on LLM for optimizing DB

  - Junchen Jiang: networking, but want to propose a new type of DB

- **Fall 2025 Student Evaluations of Teaching were sent**
  - **Again: if 80% of you finish the evaluation, all will get 2 bonus points.**

Logistics

We might need 1 – 2 extra lectures (beyond scheduled) to compensate holiday interruptions

- Will decide depending on our progress
- If happen – will on Zoom

Exam:

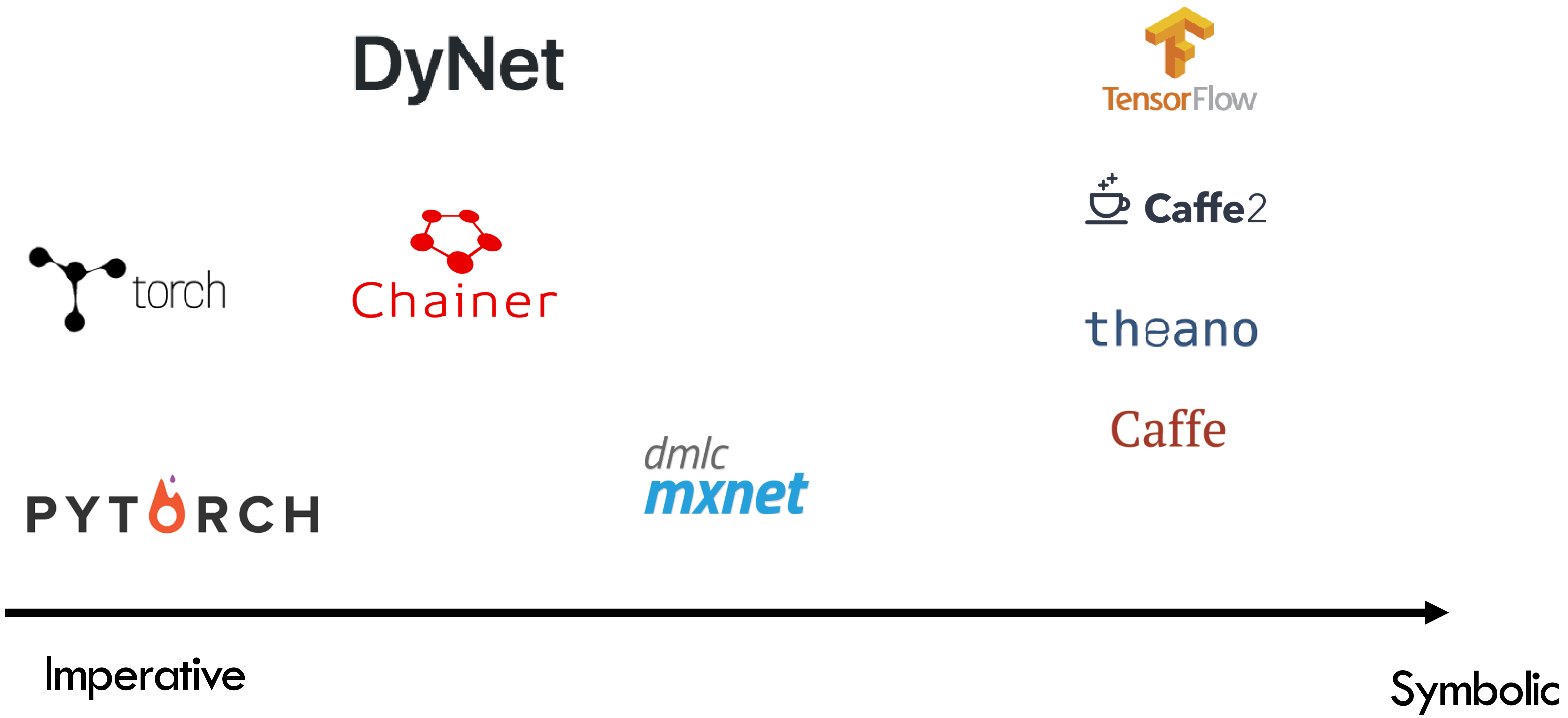
- all MCQ, TA will hold a recitation before exam
- Date and time: Dec 10, 11:30AM to 2:30PM
- Location: WLH2111

High-level Picture

Data	Model	Compute
✅ $\{x_i\}^n{}_{i=1}$	✅ Math primitives (mostly matmul)	❓Make them run on (clusters of) different kinds of hardware

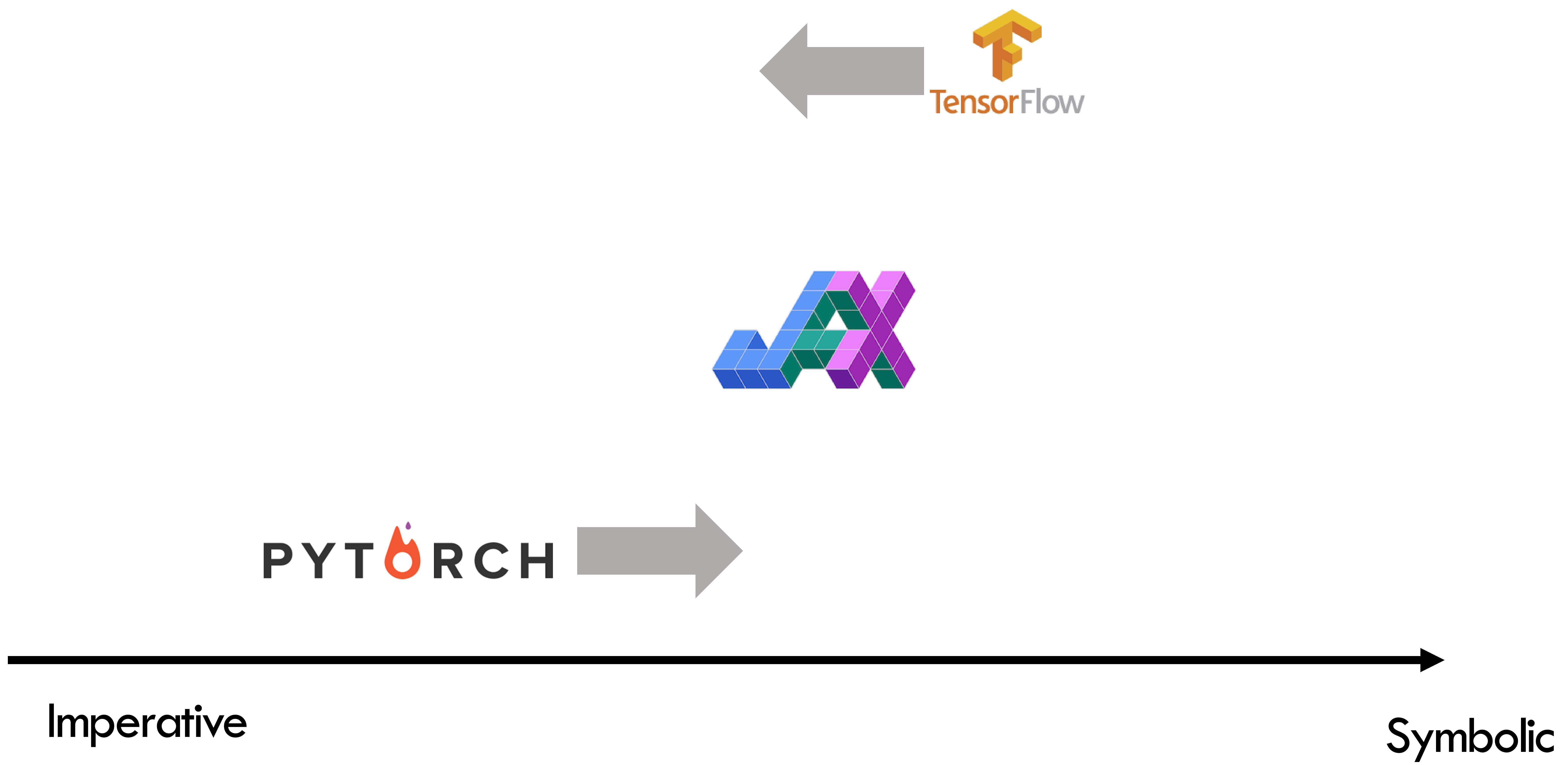
❓A repr that expresses the computation using primitives

Symbolic vs. Imperative (2016)



Symbolic vs. Imperative (2024)

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?

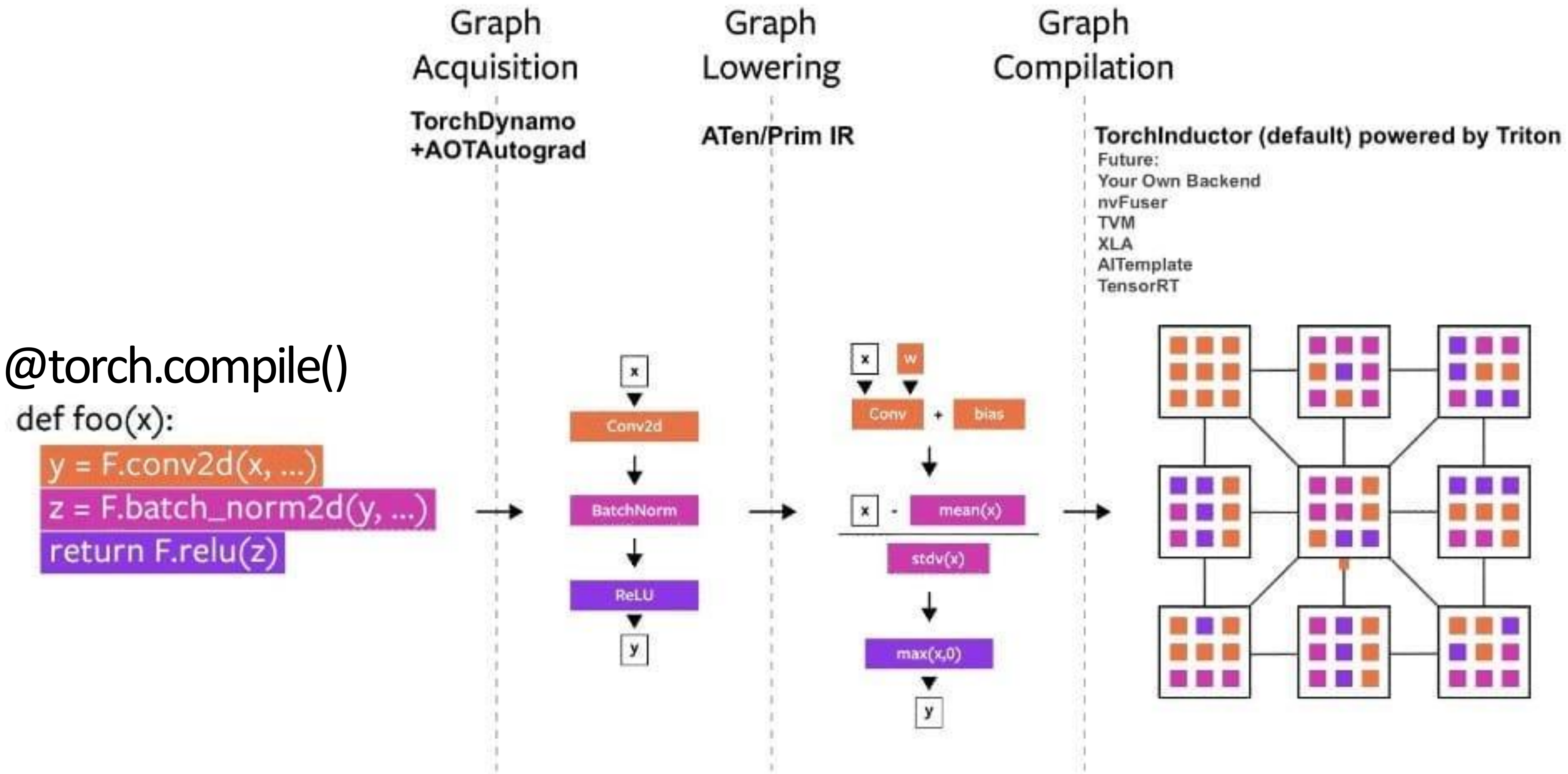
```
x = torch.Tensor([3])
y = torch.Tensor([2])
z = x - y
loss = square(z)
loss.backward()
print(x.grad)
```

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



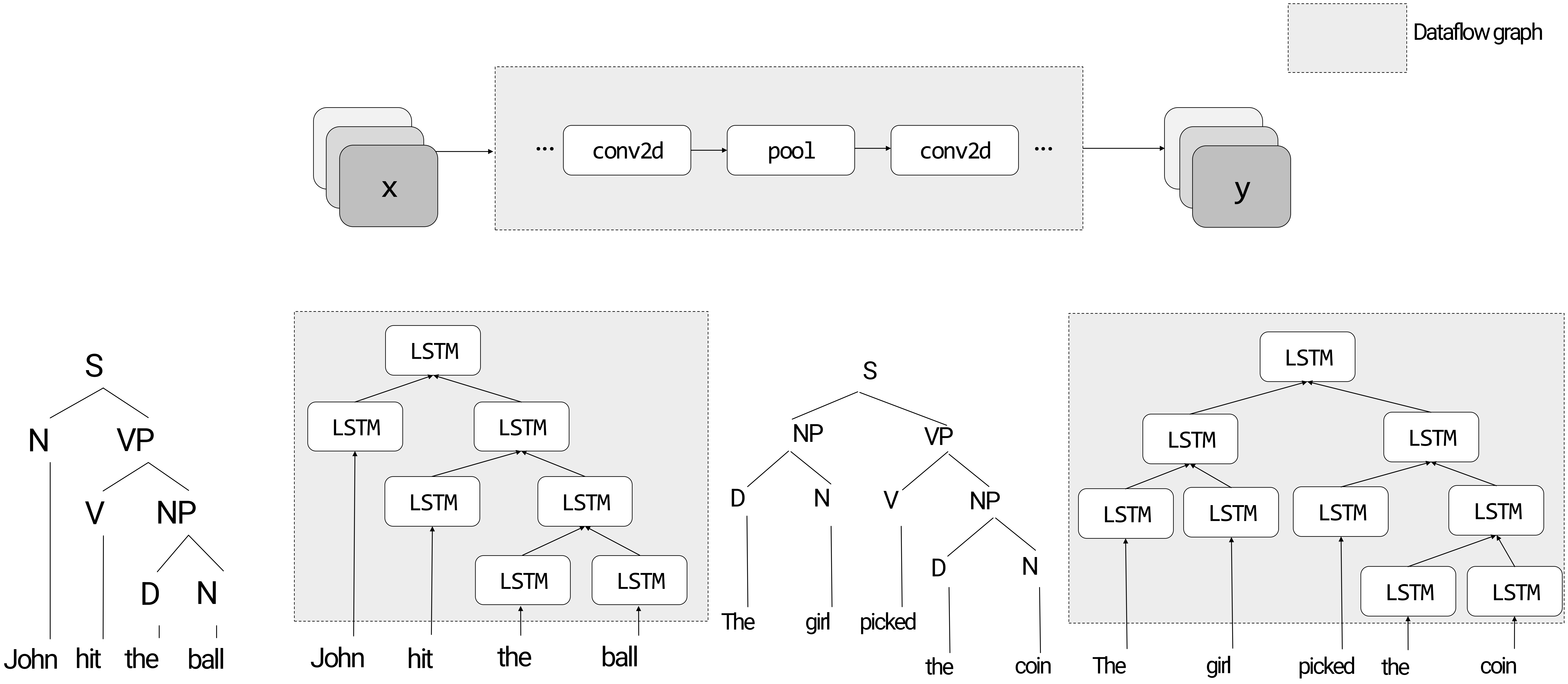
What is the problem of JIT?

Requirements for static graphs

Q: What is the problem of JIT?

A: Requirements for static graphs

Static Models vs. Dynamic Models



High-level Picture

Data

Model

Compute

✅ $\{x_i\}^n{}_{i=1}$

✅ Math primitives (mostly matmul)

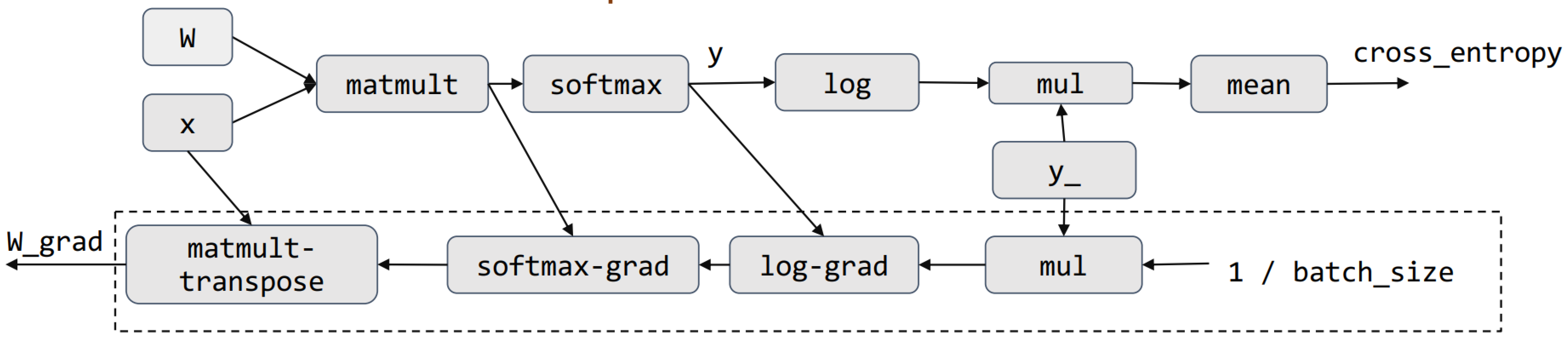
❓Make them run on (clusters of) different kinds of hardware

❓A repr that expresses the computation using primitives

What happens behind the Scene (Cond.)

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

Automatic Differentiation, more details in follow up lectures



Expand it a Bit

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

Recap: how to take derivative?

Given $f(\theta)$, what is $\frac{\partial f}{\partial \theta}$?

Instead, Symbolic Differentiation

Write down the formula, derive the gradient following PD rules

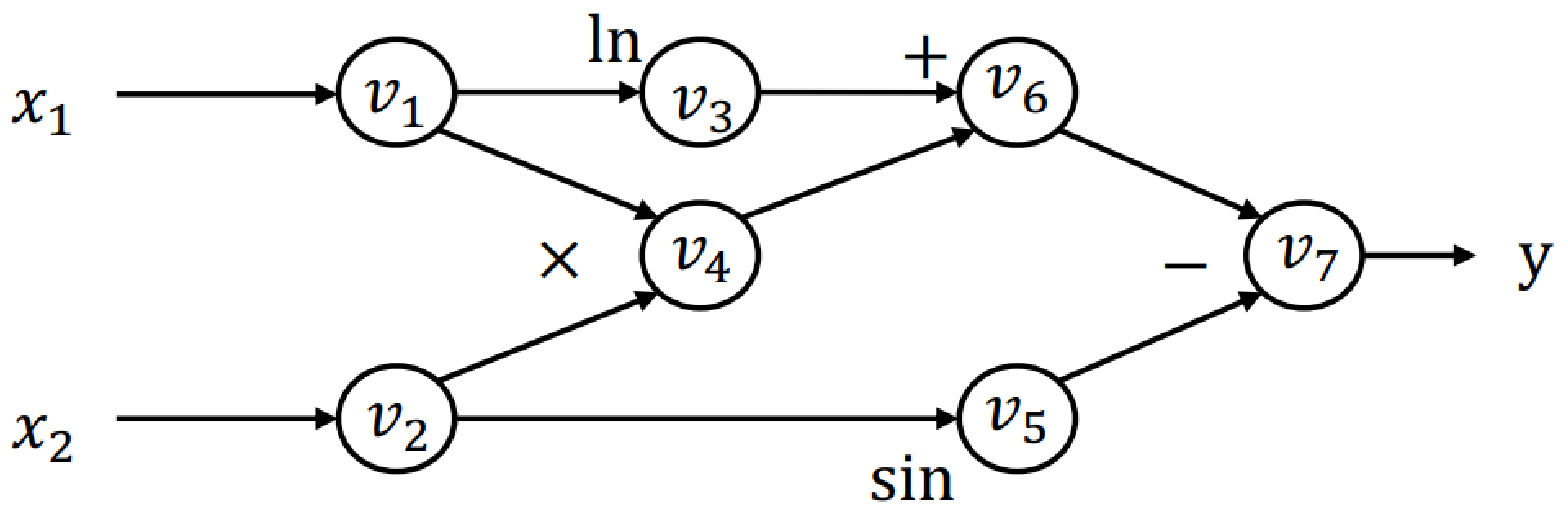
$$\frac{\partial(f(\theta) + g(\theta))}{\partial\theta} = \frac{\partial f(\theta)}{\partial\theta} + \frac{\partial g(\theta)}{\partial\theta}$$

$$\frac{\partial(f(\theta)g(\theta))}{\partial\theta} = g(\theta)\frac{\partial f(\theta)}{\partial\theta} + f(\theta)\frac{\partial g(\theta)}{\partial\theta}$$

$$\frac{\partial(f(g(\theta))}{\partial\theta} = \frac{\partial f(g(\theta))}{\partial g(\theta)}\frac{\partial g(\theta)}{\partial\theta}$$

Map autodiff rules to computational graph

$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin x_2$



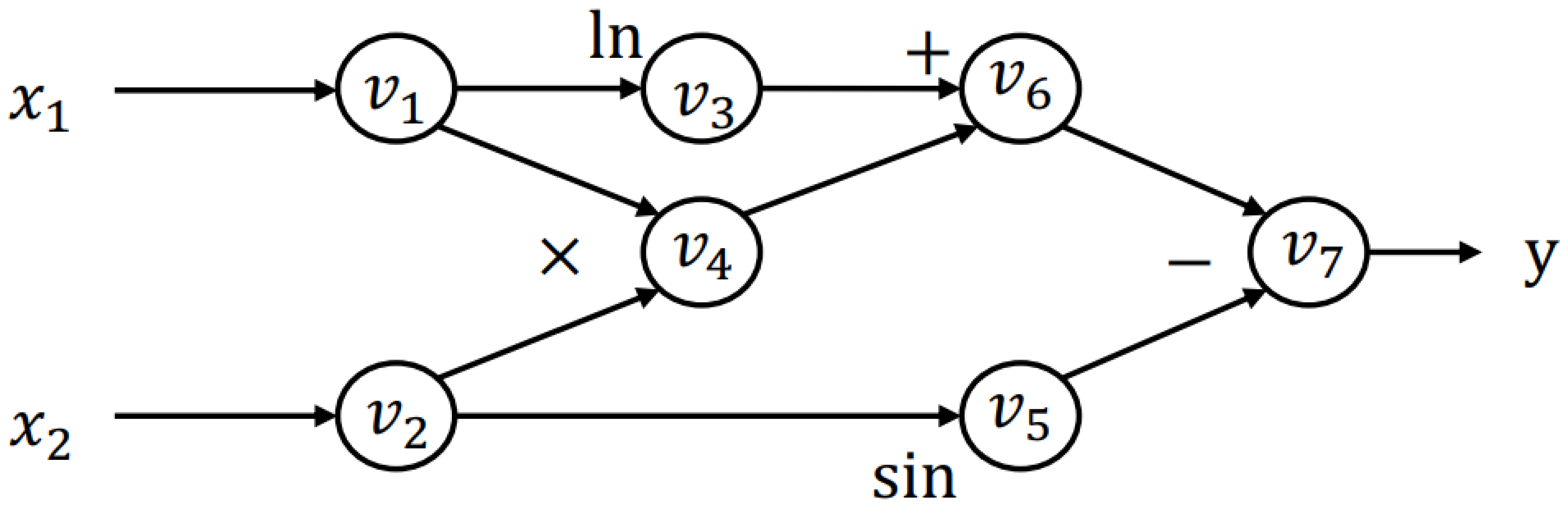
Forward evaluation trace

$v_1 = x_1 = 2$
$v_2 = x_2 = 5$
$v_3 = \ln v_1 = \ln 2 = 0.693$
$v_4 = v_1 \times v_2 = 10$
$v_5 = \sin v_2 = \sin 5 = -0.959$
$v_6 = v_3 + v_4 = 10.693$
$v_7 = v_6 - v_5 = 10.693 + 0.959 = 11.652$
$y = v_7 = 11.652$

- Q: Calculate the value of $\frac{\partial y}{\partial x_1}$
  - A: use PD and chain rules

Reverse Mode AD

$$y = f(x_1, x_2) = \ln(x_1) + x_1 x_2 - \sin x_2$$



Forward evaluation trace

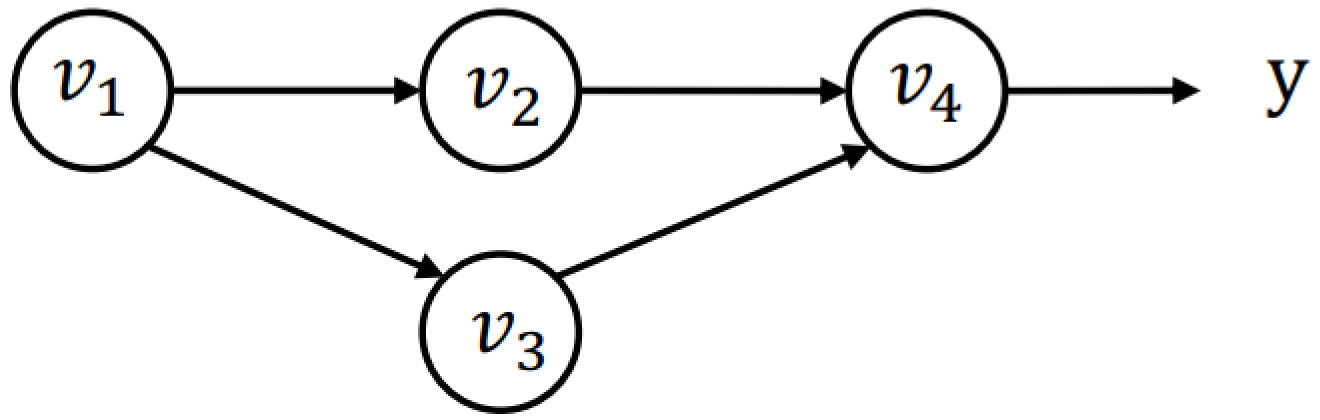
$v_1 = x_1 = 2$
$v_2 = x_2 = 5$
$v_3 = \ln v_1 = \ln 2 = 0.693$
$v_4 = v_1 \times v_2 = 10$
$v_5 = \sin v_2 = \sin 5 = -0.959$
$v_6 = v_3 + v_4 = 10.693$
$v_7 = v_6 - v_5 = 10.693 + 0.959 = 11.652$
$y = v_7 = 11.652$

- Define adjoint $\overline{v_i} = \frac{\partial y}{\partial v_i}$
- We then compute each $\bar{v}_i$ in the reverse topological order of the graph

$$\overline{v_7} = \frac{\partial y}{\partial v_7} = 1$$
$$\overline{v_6} = \overline{v_7}\frac{\partial v_7}{\partial v_6} = \overline{v_7} \times 1 = 1$$
$$\overline{v_5} = \overline{v_7}\frac{\partial v_7}{\partial v_5} = \overline{v_7} \times (-1) = -1$$
$$\overline{v_4} = \overline{v_6}\frac{\partial v_6}{\partial v_4} = \overline{v_6} \times 1 = 1$$
$$\overline{v_3} = \overline{v_6}\frac{\partial v_6}{\partial v_3} = \overline{v_6} \times 1 = 1$$
$$\overline{v_2} = \overline{v_5}\frac{\partial v_5}{\partial v_2} + \overline{v_4}\frac{\partial v_4}{\partial v_2} = \overline{v_5} \times \cos v_2 + \overline{v_4} \times v_1 = -0.284 + 2 = 1.716$$
$$\overline{v_1} = \overline{v_4}\frac{\partial v_4}{\partial v_1} + \overline{v_3}\frac{\partial v_3}{\partial v_1} = \overline{v_4} \times v_2 + \overline{v_3}\frac{1}{v_1} = 5 + \frac{1}{2} = 5.5$$

- Finally: $\frac{\partial y}{\partial x_1} = \bar{v}_1 = 5.5$

Case Study



How to derive the gradient of v_1

$$\overline{v_1} = \frac{\partial y}{\partial v_1} = \frac{\partial f(v_2,v_3)}{\partial v_2}\frac{\partial v_2}{\partial v_1} + \frac{\partial f(v_2,v_3)}{\partial v_3}\frac{\partial v_3}{\partial v_1} = \overline{v_2}\frac{\partial v_2}{\partial v_1} + \overline{v_3}\frac{\partial v_3}{\partial v_1}$$

For a v_i used by multiple consumers:

$$\overline{v_i} = \sum_{j \in next(i)} \overline{v_{i \to j}}$$, where $$\overline{v_{i \to j}} = \overline{v_j}\frac{\partial v_j}{\partial v_i}$$

Summary: Backward Mode Autodiff

- Start from the output nodes
- Derive gradient all the way back to the input node

Back to Our Question

A repr that expresses the computation using primitives

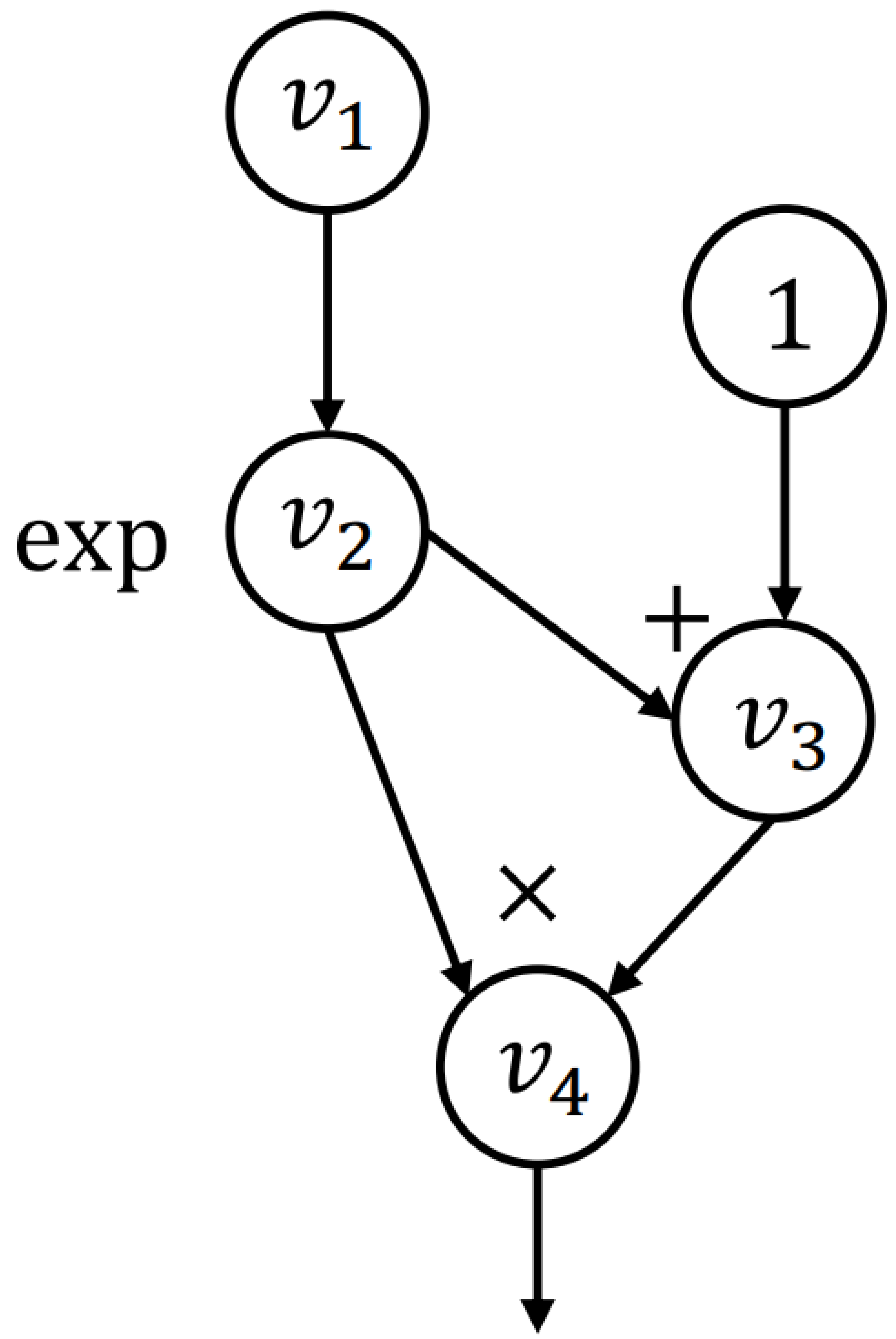
✅ A repr that expresses the forward computation using primitives

❓ A repr that expresses the backward computation using primitives

Back to our question: Construct the Backward Graph

- How can we construct a computational graph that calculates the adjoint value?

```
def gradient(out):
  node_to_grad = {out:  [1]}
  for i in reverse_topo_order(out):
    v̄_i = Σ_j v̄_{i→j} = sum(node_to_grad[i])
    for k ∈ inputs(i):
        compute v̄_{k→i} = v̄_i ∂v_i/∂v_k
        append v̄_{k→i} to node_to_grad[k]
    return adjoint of input v̄_input
```



f: $(\exp(v_1) + 1)\exp(v_1)$

How to implement reverse Autodiff (aka. BP)

def gradient(out):
  node_to_grad = {out: [1]}
  for i in reverse_topo_order(out):
    $\overline{v_i} = \sum_j \overline{v_{i \to j}}$ = sum(node_to_grad[i])
    for $k \in inputs(i)$:
      compute $\overline{v_{k \to i}} = \overline{v_i} \frac{\partial v_i}{\partial v_k}$
      append $\overline{v_{k \to i}}$ to node_to_grad[k]
    return adjoint of input $\overline{v_{input}}$

Record all partial adjoints of a node

Sum up all partial adjoints to get the gradient

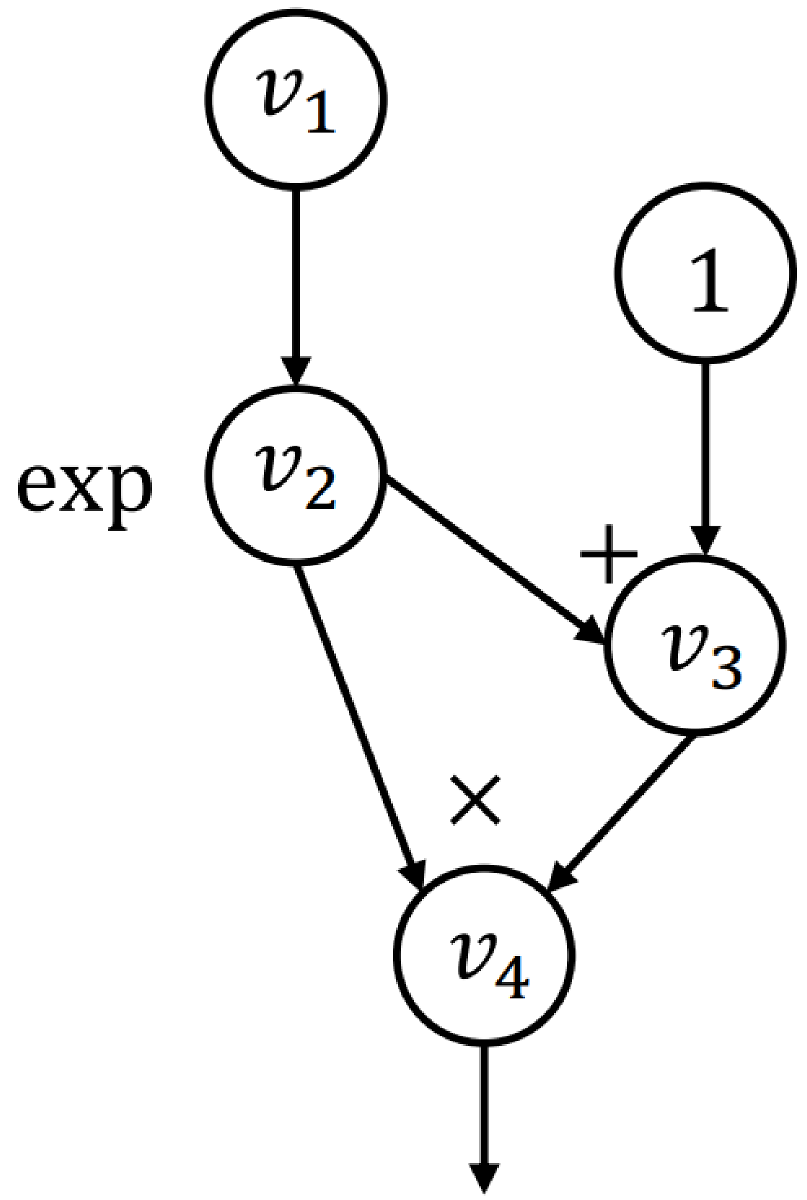
Compute and propagates partial adjoints to its inputs.

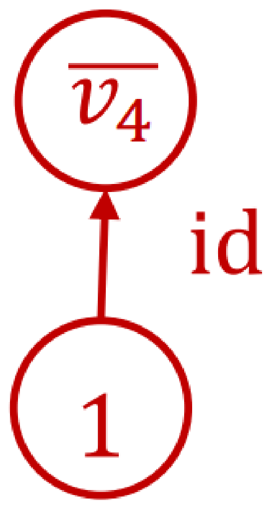
Start from v_4

i = 4: $v_4 = sum([1]) = 1$

def gradient(out):
  node_to_grad = {out: [1]}
  for i in reverse_topo_order(out):
    $\overline{v_i} = \sum_j \overline{v_{i\to j}}$ = sum(node_to_grad[i])
    for $k \in inputs(i)$:
      compute $\overline{v_{k\to i}} = \overline{v_i} \frac{\partial v_i}{\partial v_k}$
      append $\overline{v_{k\to i}}$ to node_to_grad[k]
    return adjoint of input $\overline{v_{input}}$

$i = 4$
node_to_grad: {
  4: [$\overline{v_4}$]
}





v_4: Inspect (v_2, v_4) and (v_3, v_4)

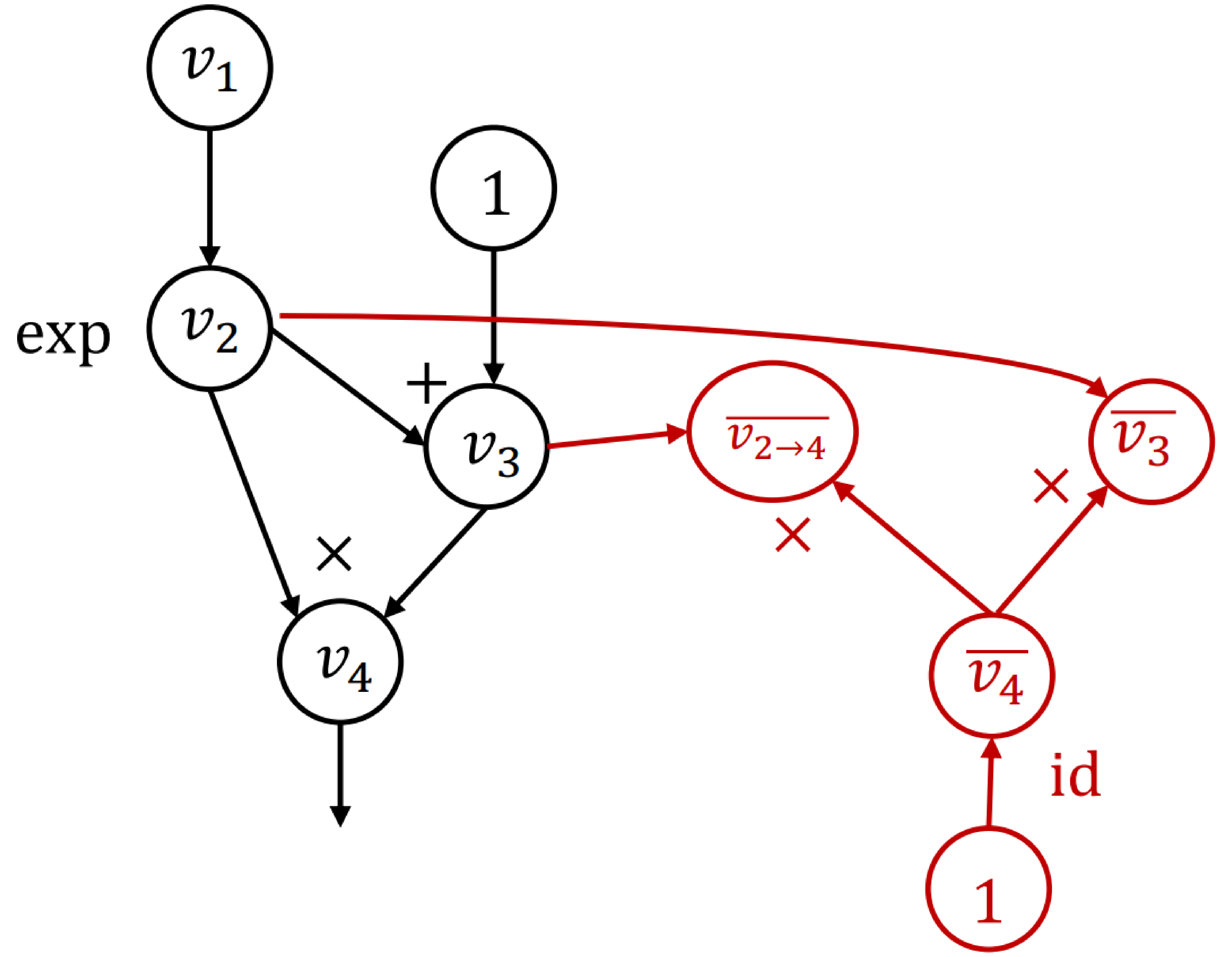
i=4: $\overline{v_4} = sum([1]) = 1$

k=2: $\overline{v_{2\to4}} = \overline{v_4}\frac{\partial v_4}{\partial v_2} = \overline{v_4}v_3$

k=3: $\overline{v_{3\to4}} = \overline{v_4}\frac{\partial v_4}{\partial v_3} = \overline{v_4}v_2, \overline{v_{3\to4}} = \overline{v_3}$

def gradient(out):
  node_to_grad = {out: [1]}
  for i in reverse_topo_order(out):
    $\overline{v_i} = \sum_j \overline{v_{i\to j}}$ = sum(node_to_grad[i])
    for $k \in inputs(i)$:
      compute $\overline{v_{k\to i}} = \overline{v_i}\frac{\partial v_i}{\partial v_k}$
      append $\overline{v_{k\to i}}$ to node_to_grad[k]
  return adjoint of input $\overline{v_{input}}$

$i = 4$
node_to_grad: {
  2: [$\overline{v_{2\to4}}$]
  3: [$\overline{v_3}$]
  4: [$\overline{v_4}$]
}



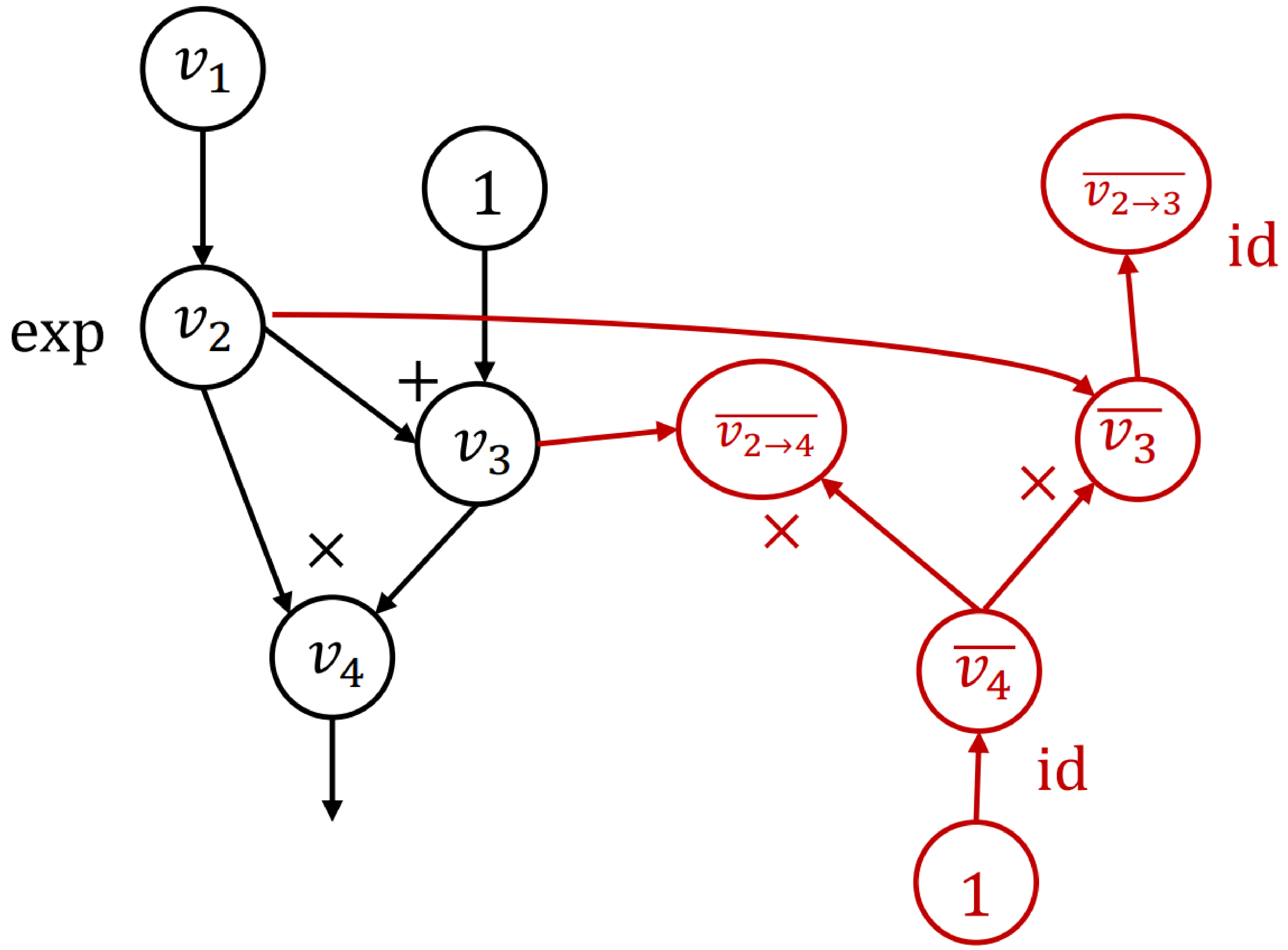
Inspect v_3

i=3: $\overline{v_3}$ done!

k=2: $\overline{v_{2\to3}} = \overline{v_3}\frac{\partial v_3}{\partial v_2} = \overline{v_3}$

```
def gradient(out):
  node_to_grad = {out:  [1]}
  for i in reverse_topo_order(out):
    v̄_i = Σ_j v̄_{i→j} = sum(node_to_grad[i])
    for k ∈ inputs(i):
      compute v̄_{k→i} = v̄_i ∂v_i/∂v_k
      append v̄_{k→i} to node_to_grad[k]
  return adjoint of input v̄_input
```

$i = 3$
node_to_grad: {
  2: $[\overline{v_{2\to4}}, \overline{v_{2\to3}}]$
  3: $[\overline{v_3}]$
  4: $[\overline{v_4}]$
}



$i{=}2\text{:}\ \overline{v_2} = \overline{v_{2\to3}} + \overline{v_{2\to4}}$

Inspect v_2

```
def gradient(out):
  node_to_grad = {out:  [1]}
  for i in reverse_topo_order(out):
➡   v̄_i = Σ_j v̄_{i→j} = sum(node_to_grad[i])
    for k ∈ inputs(i):
        compute v̄_{k→i} = v̄_i ∂v_i/∂v_k
        append v̄_{k→i} to node_to_grad[k]
  return adjoint of input v̄_input
```

$i = 2$

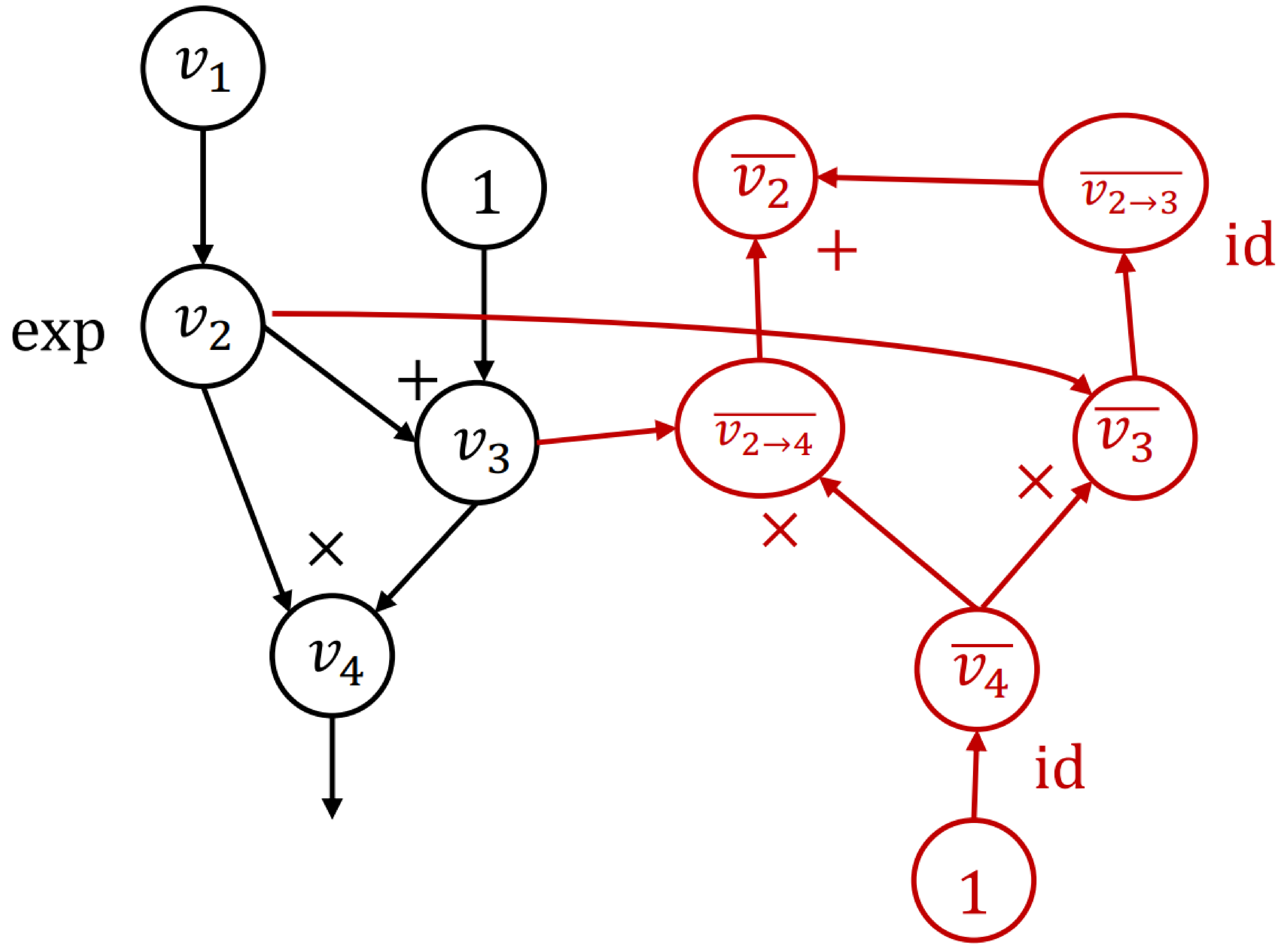
node_to_grad: {

2: $[\overline{v_{2\to4}}, \overline{v_{2\to3}}]$

3: $[\overline{v_3}]$

4: $[\overline{v_4}]$

}



Inspect (v_1, v_2)

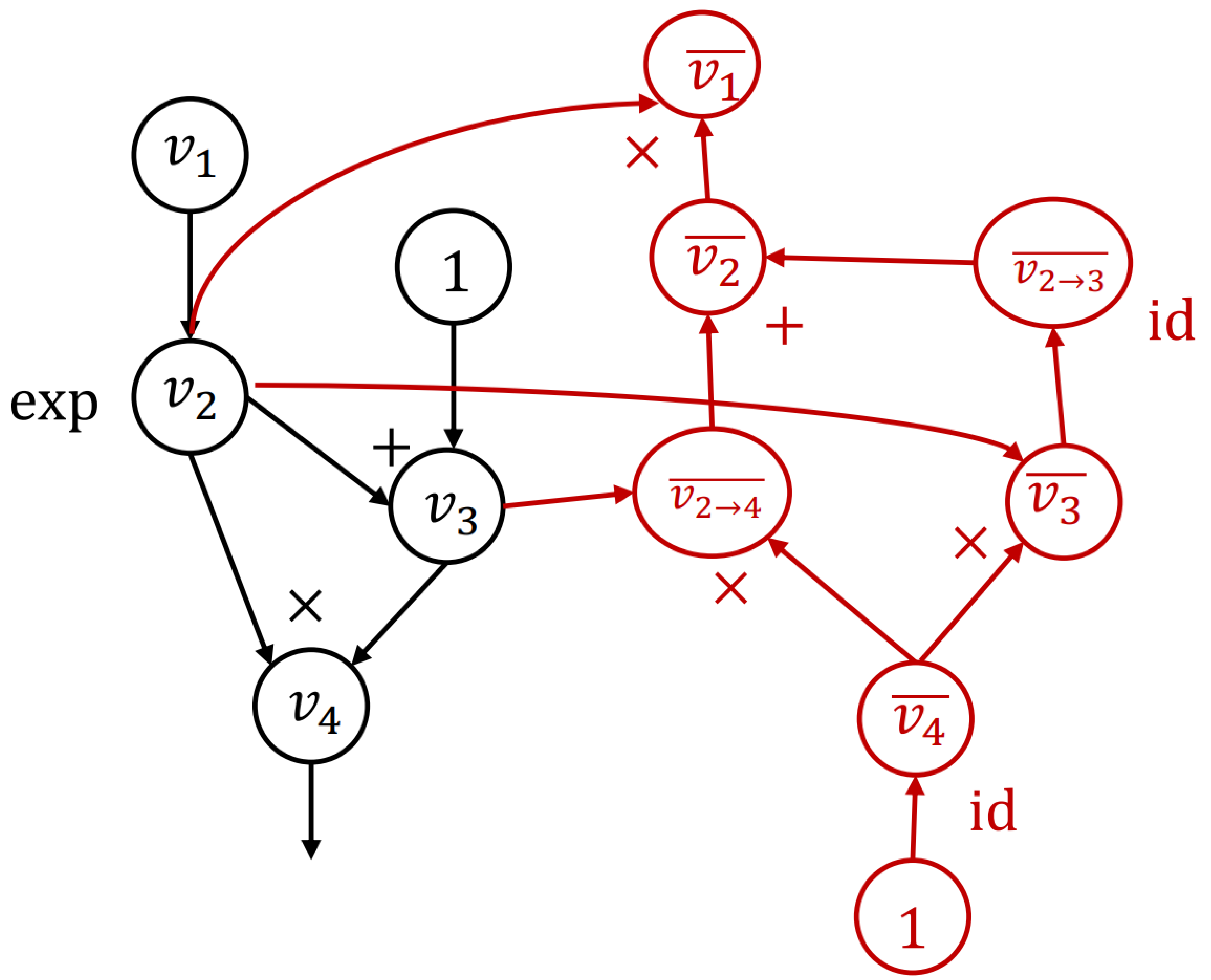
i=2: $\overline{v_2} = \overline{v_{2\to3}} + \overline{v_{2\to4}}$

k=1: $\overline{v_{1\to2}} = \overline{v_2}\frac{\partial v_2}{\partial v_1} = \overline{v_2}\exp(v_1)$,

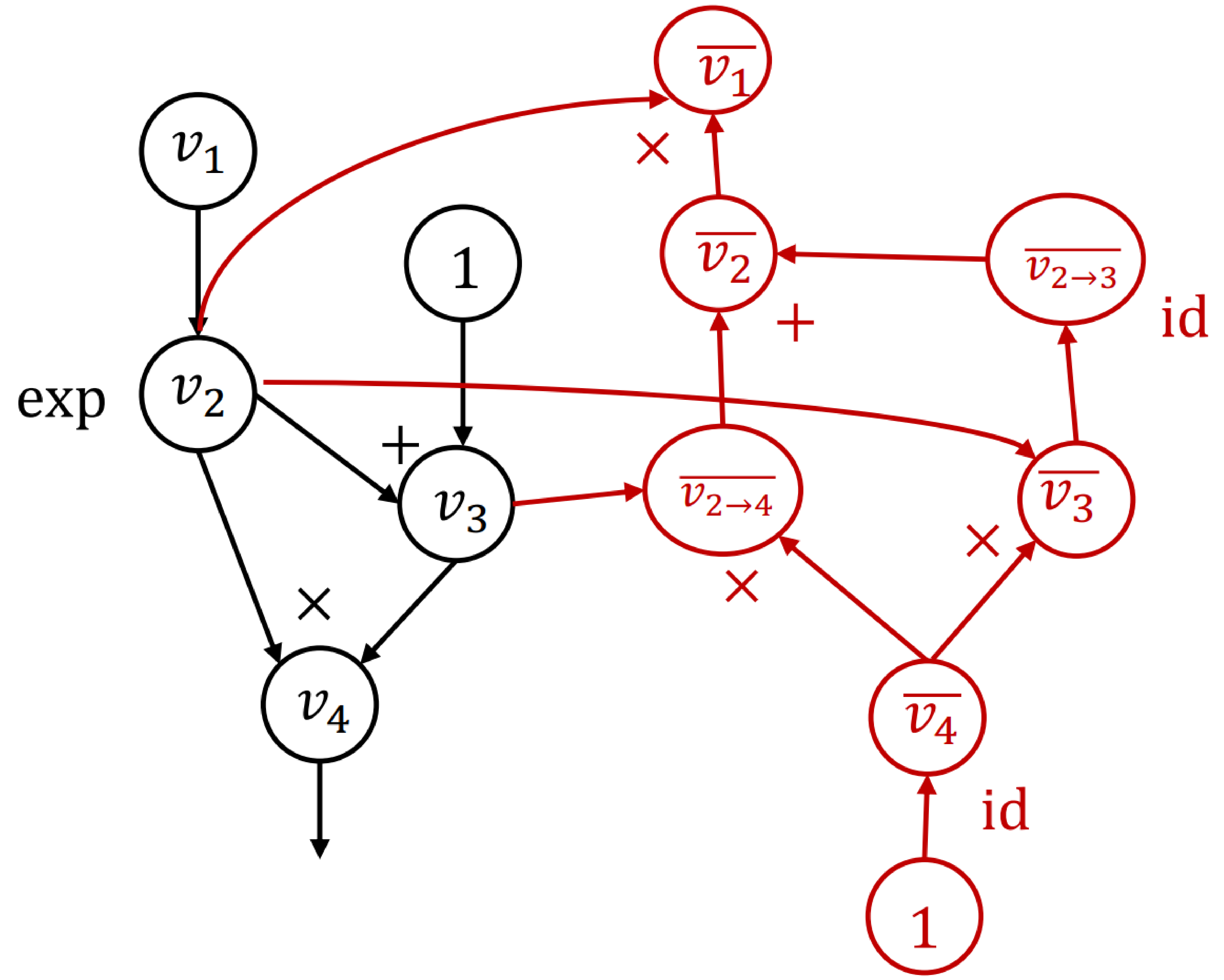
$\overline{v_1} = \overline{v_{1\to2}}$

def gradient(out):
  node_to_grad = {out: [1]}
  for i in reverse_topo_order(out):
    $\overline{v_i} = \sum_j \overline{v_{i\to j}}$ = sum(node_to_grad[i])
    for $k \in inputs(i)$:
      compute $\overline{v_{k\to i}} = \overline{v_i}\frac{\partial v_i}{\partial v_k}$
      append $\overline{v_{k\to i}}$ to node_to_grad[k]
    return adjoint of input $\overline{v_{input}}$

$i = 2$
node_to_grad: {
  1: [$\overline{v_1}$]
  2: [$\overline{v_{2\to4}}, \overline{v_{2\to3}}$]
  3: [$\overline{v_3}$]
  4: [$\overline{v_4}$]
}

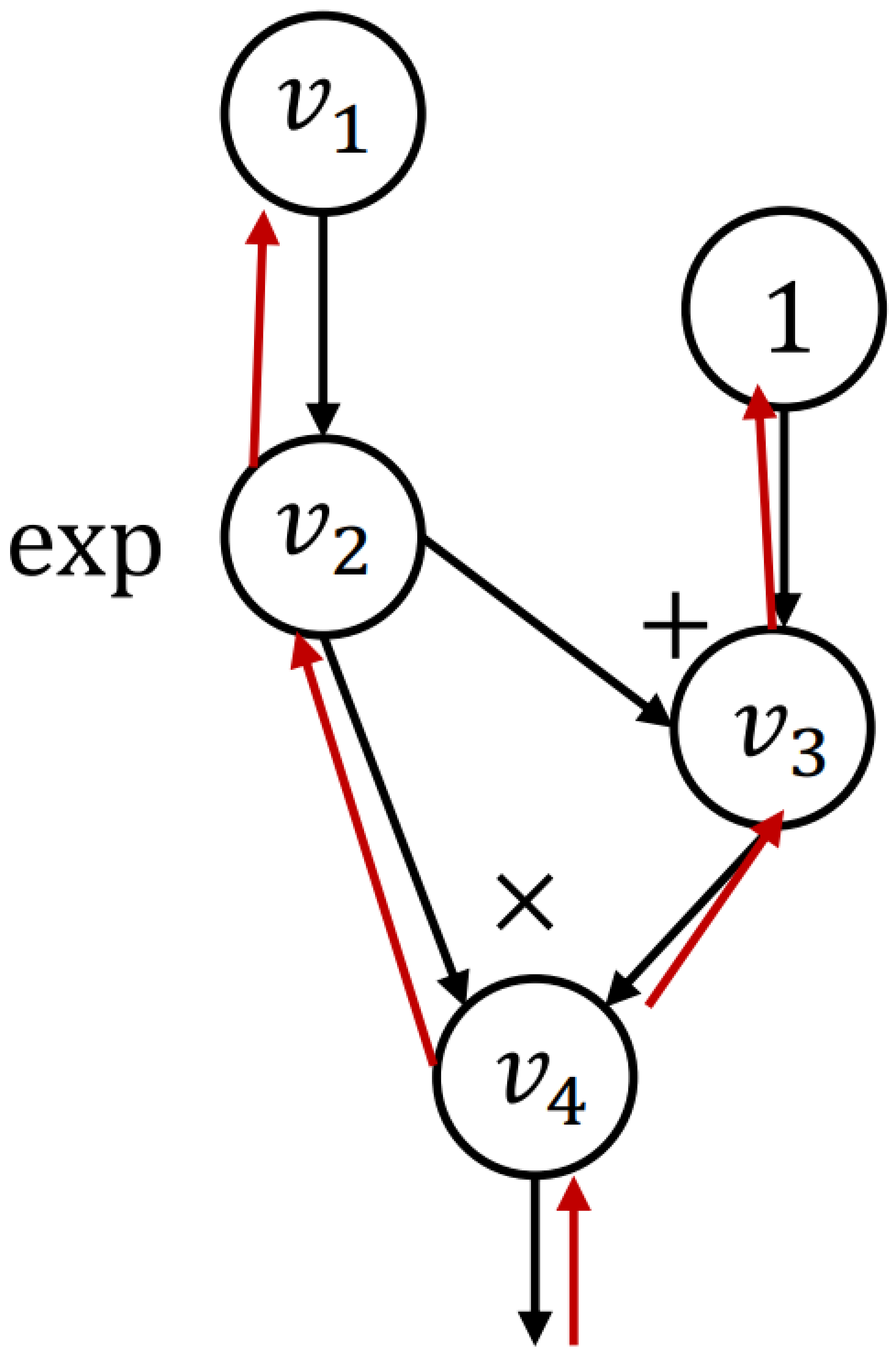


Summary: Backward AD

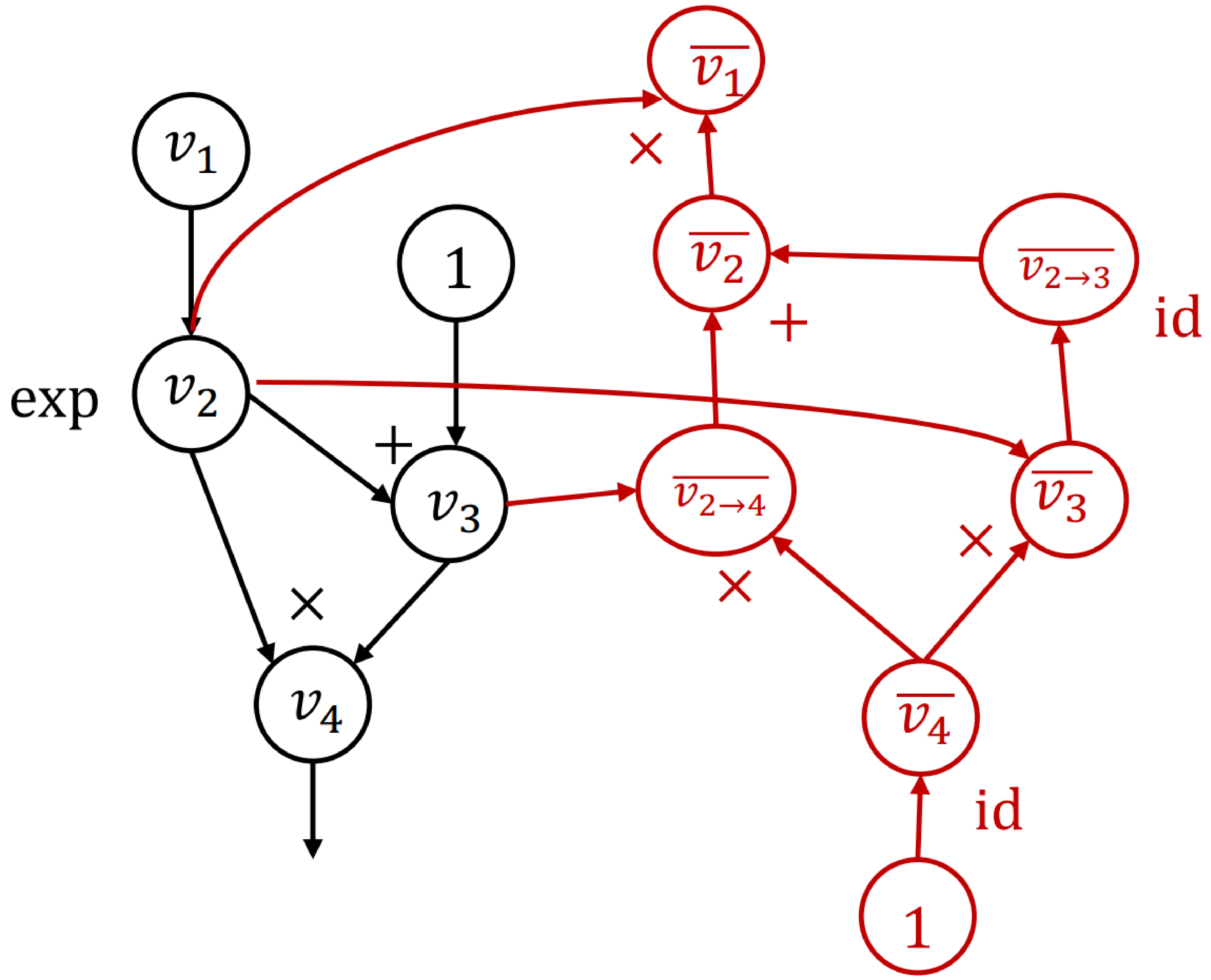


- Construct backward graph in a symbolic way (instead of concrete values)
- This graph can be reused by different input values

Backpropagation vs. Reverse-mode AD



vs.

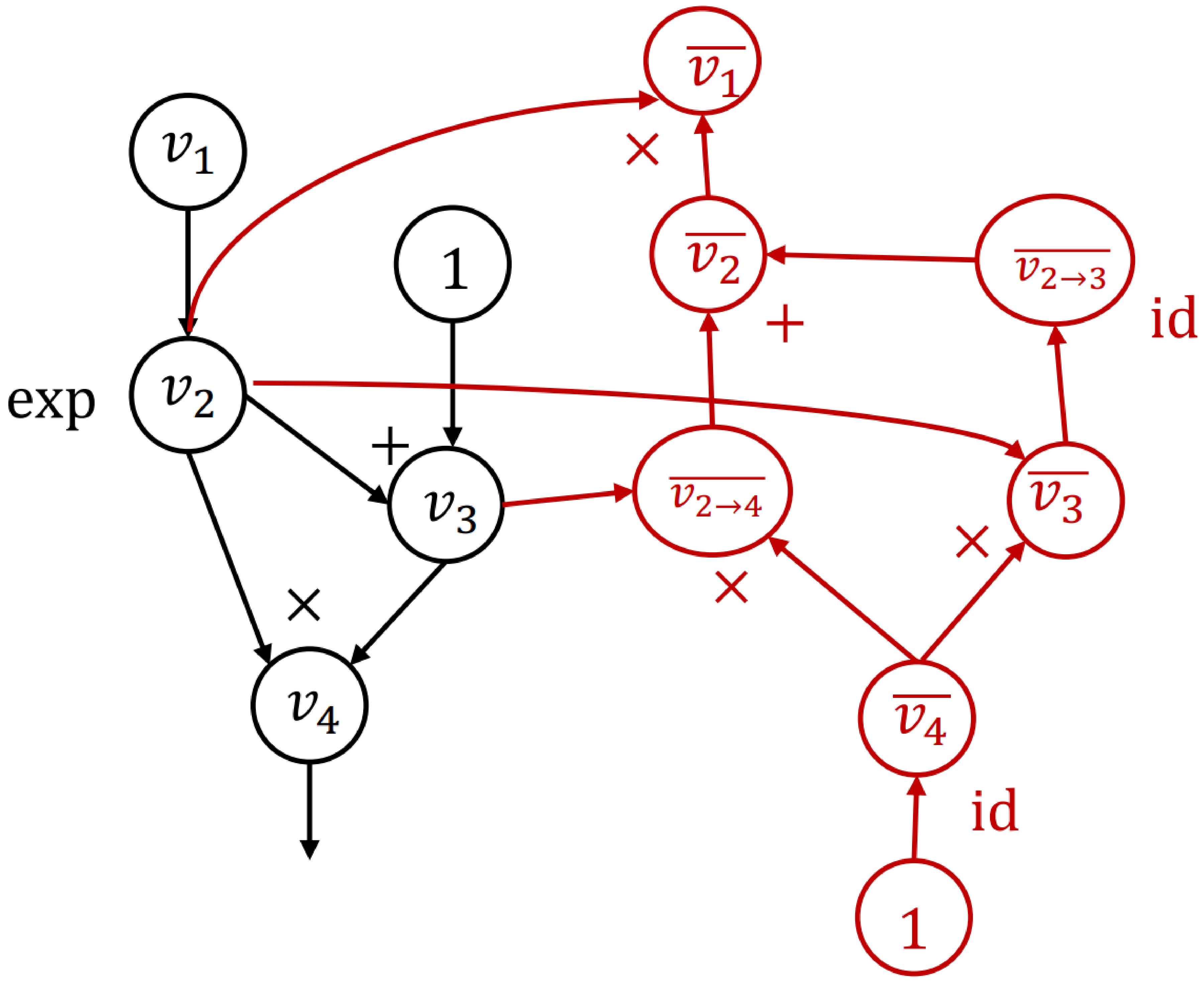


- Run backward through the forward graph
- Caffe/cuda-convnet

- Construct backward graph
- Used by TensorFlow, PyTorch

Incomplete yet?

- What is the missing from the following graph for ML training?



Recall Our Master Equation

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

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

$$f(\theta, \nabla_L) = \theta - \nabla_L$$

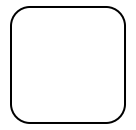
Forward	Backward	Weight update
$L(\cdot)$	$\nabla_L(\cdot)$	$f(\cdot)$

Put in Practice

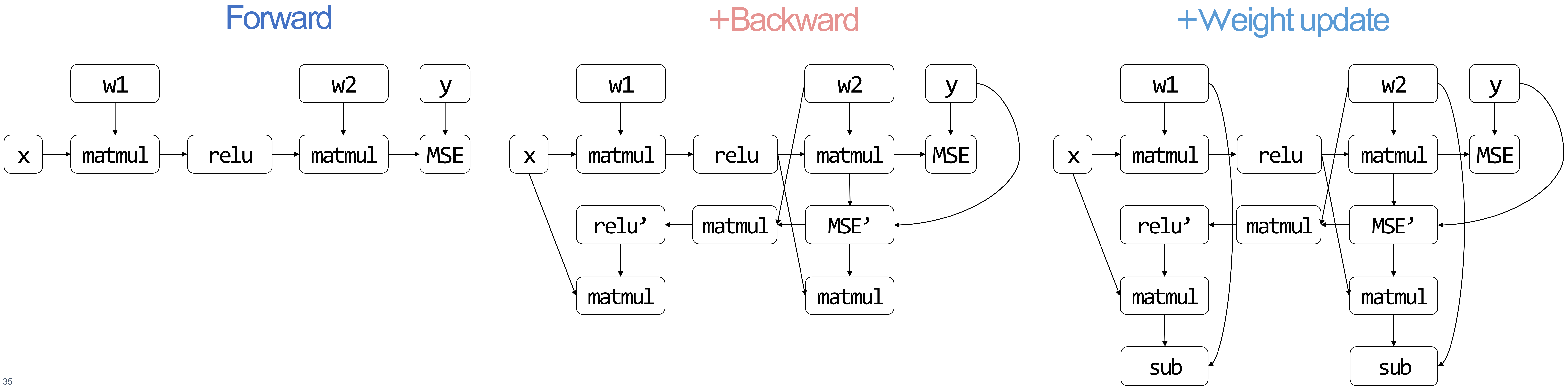
$$\theta^{(t+1)} = f\left(\theta^{(t)},\ \nabla_L\left(\theta^{(t)},\ D^{(t)}\right)\right)$$

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

$$f(\theta, \nabla_L) = \theta - \nabla_L$$


Operator / its output tensor

Data flowing direction



Homework: How to derive gradients for

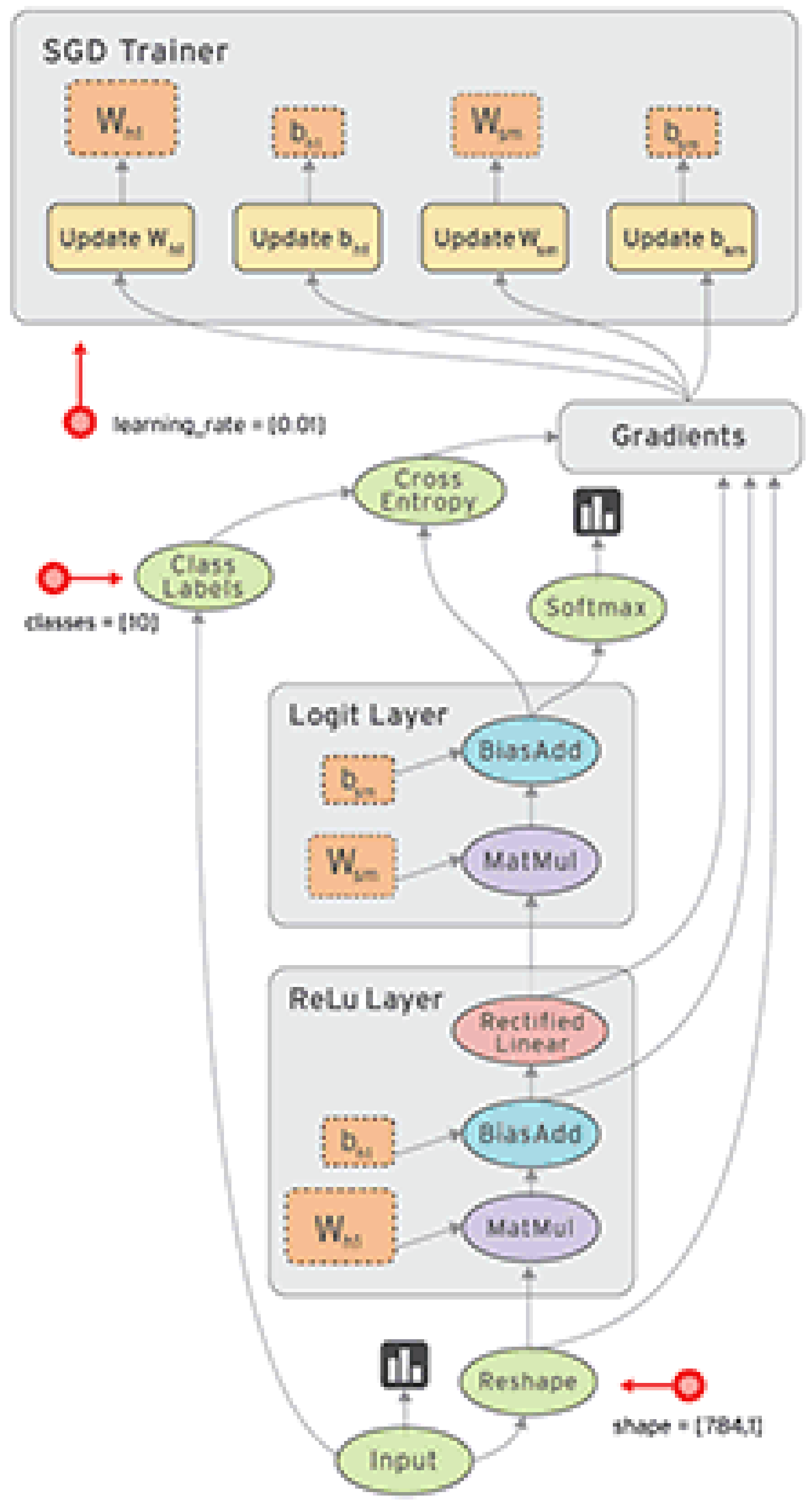
- Softmax cross entropy:

$$L = -\sum t_i \log(y_i)\,, y_i = softmax(\boldsymbol{x})_i = \frac{e^{x_i}}{\sum e^{x_d}}$$

Today

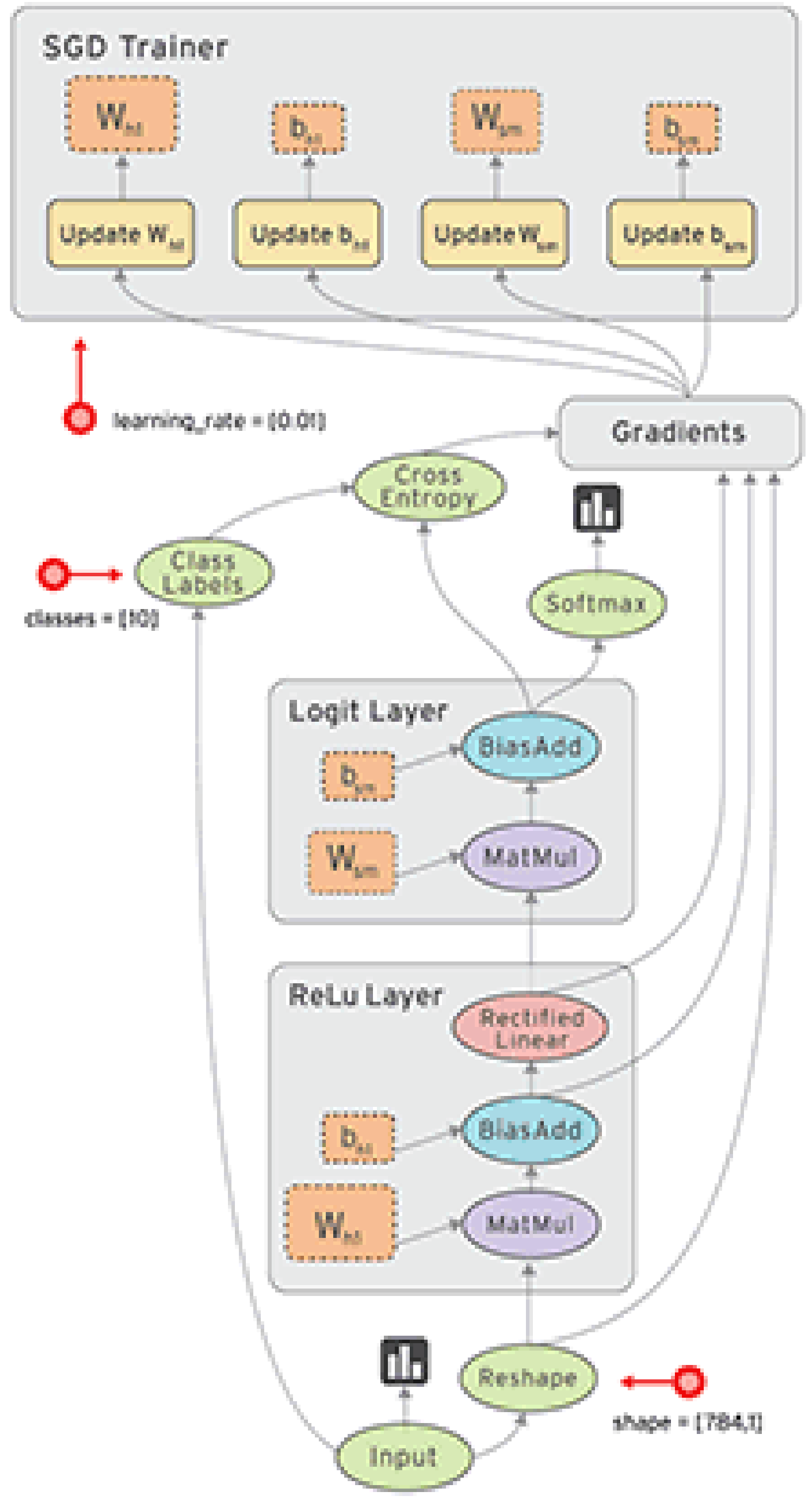
- Autodiff
- **Architecture Overview**

MLSys' Grand problem

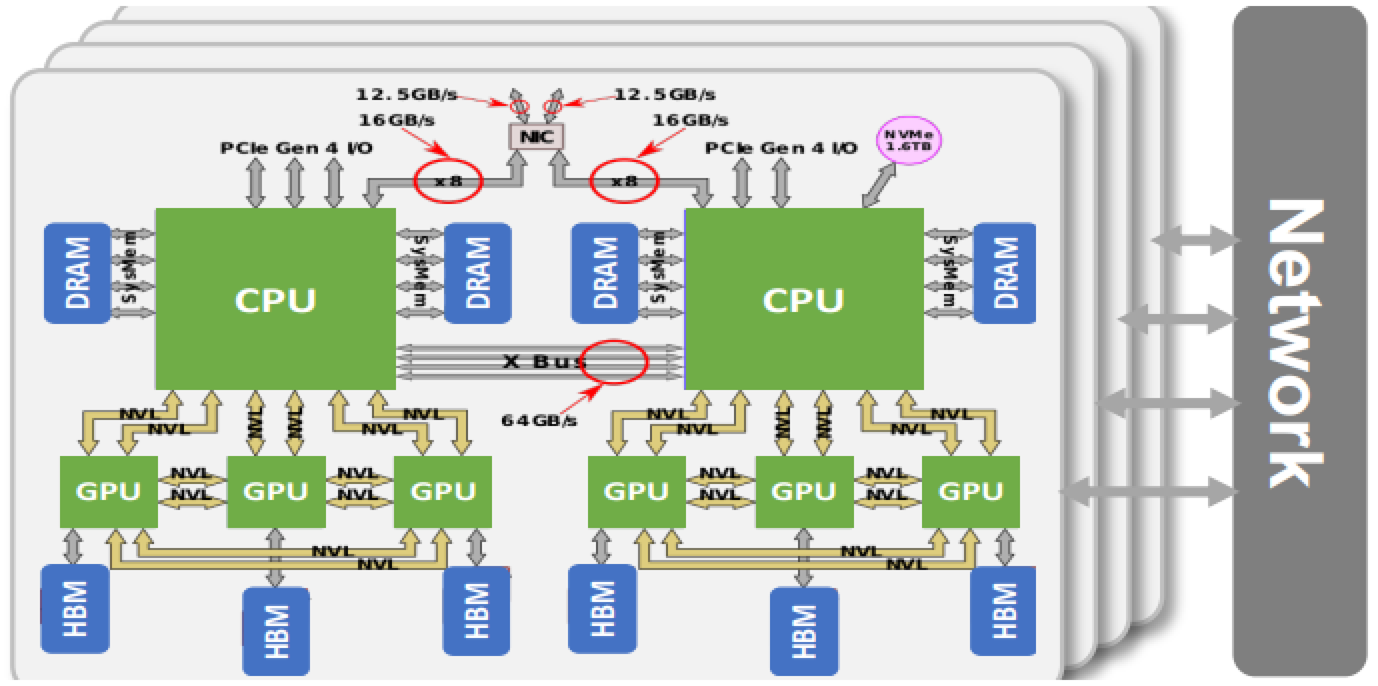


- Our system goals:
  - Fast
  - Scale
  - Memory-efficient
  - Run on diverse hardware
  - Energy-efficient
  - Easy to program/debug/deploy

ML System Overview



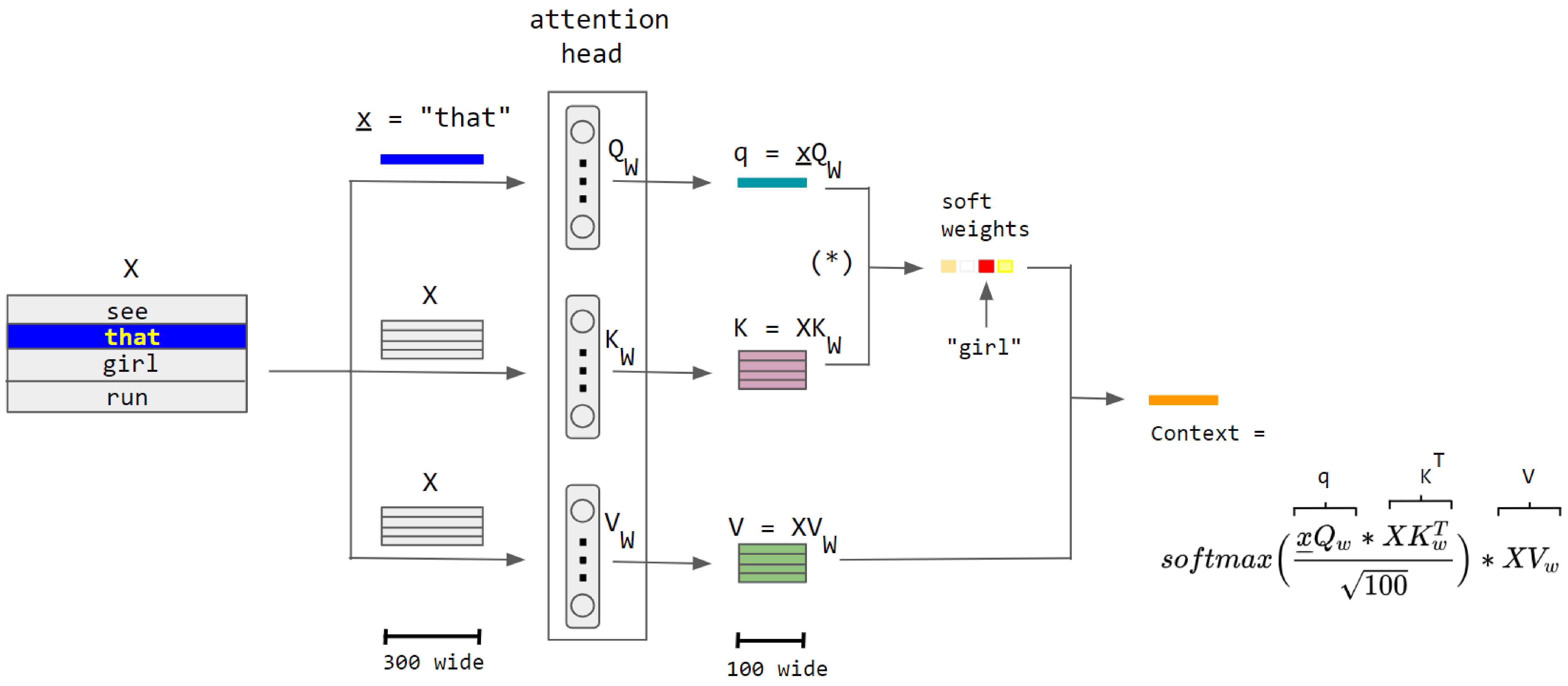
- Dataflow Graph
- Autodiff
- Graph Optimization
- Parallelization
- Runtime: schedule / memory
- Operator optimization/compilation



Graph Optimization

- Goal:
  - Rewrite the original Graph G to G'
  - G' runs faster than G

Motivating Example: Attention




```
# Original
Q = matmul(W_q, h)
K = matmul(W_k, h)
V = matmul(W_v, h)

# Merged QKV
QKV = matmul(concat(W_q, W_k, W_v), h)
```

- Why merged QKV is faster?

Arithmetic Intensity

$$AI = \#ops / \#bytes$$

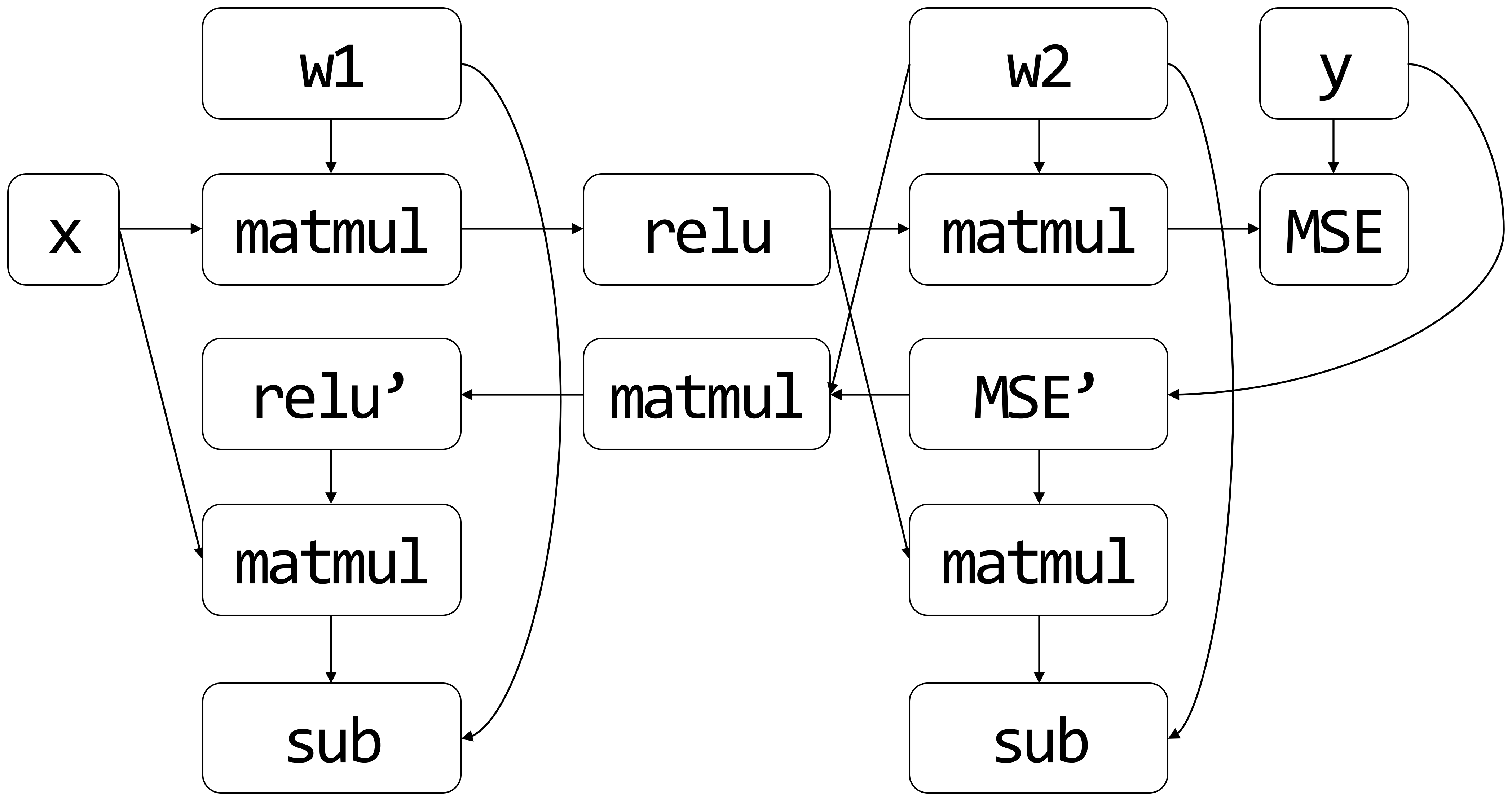
How to perform graph optimization?

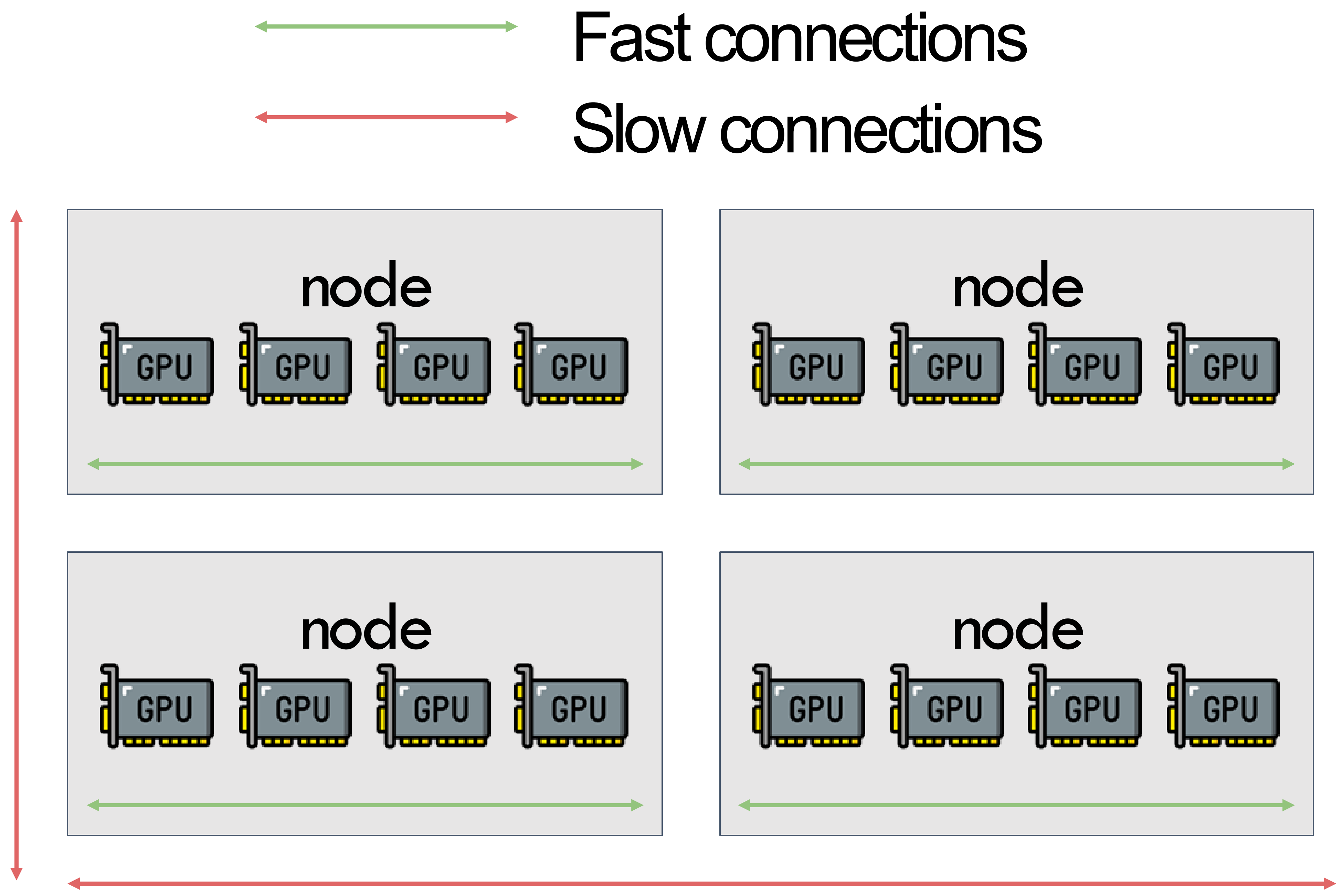
- Writing rules / template
- Auto discovery

Parallelization

- Goal: parallelize the graph compute over multiple devices

How to partition the computational graph on the device cluster?





Parallelization Problems

- How to partition
- How to communicate
- How to schedule
- Consistency
- How to auto-parallelize?

Runtime and Scheduling

- Goal: schedule the compute/communication/memory in a way that
  - As fast as possible
  - Overlap communication with compute
  - Subject to memory constraints

Operator Implementation

- Goal: get the fastest possible implementation of
  - Matmul
  - Conv2d?
- For different hardware: V100, A100, H100, phone, TPU
- For different precision: fp32, fp16, fp8, fp4
- For different shape: conv2d_3x3, conv2d_5x5, matmul2D, 3D, attention

High-level Picture

Data	Model	Compute
✅ $\{x_i\}^n{}_{i=1}$	✅ Math primitives (mostly matmul)	❓Make them run on (clusters of) different kinds of hardware
	✅A repr that expresses the computation using primitives	