Automatic Differentiation (under the Hood)

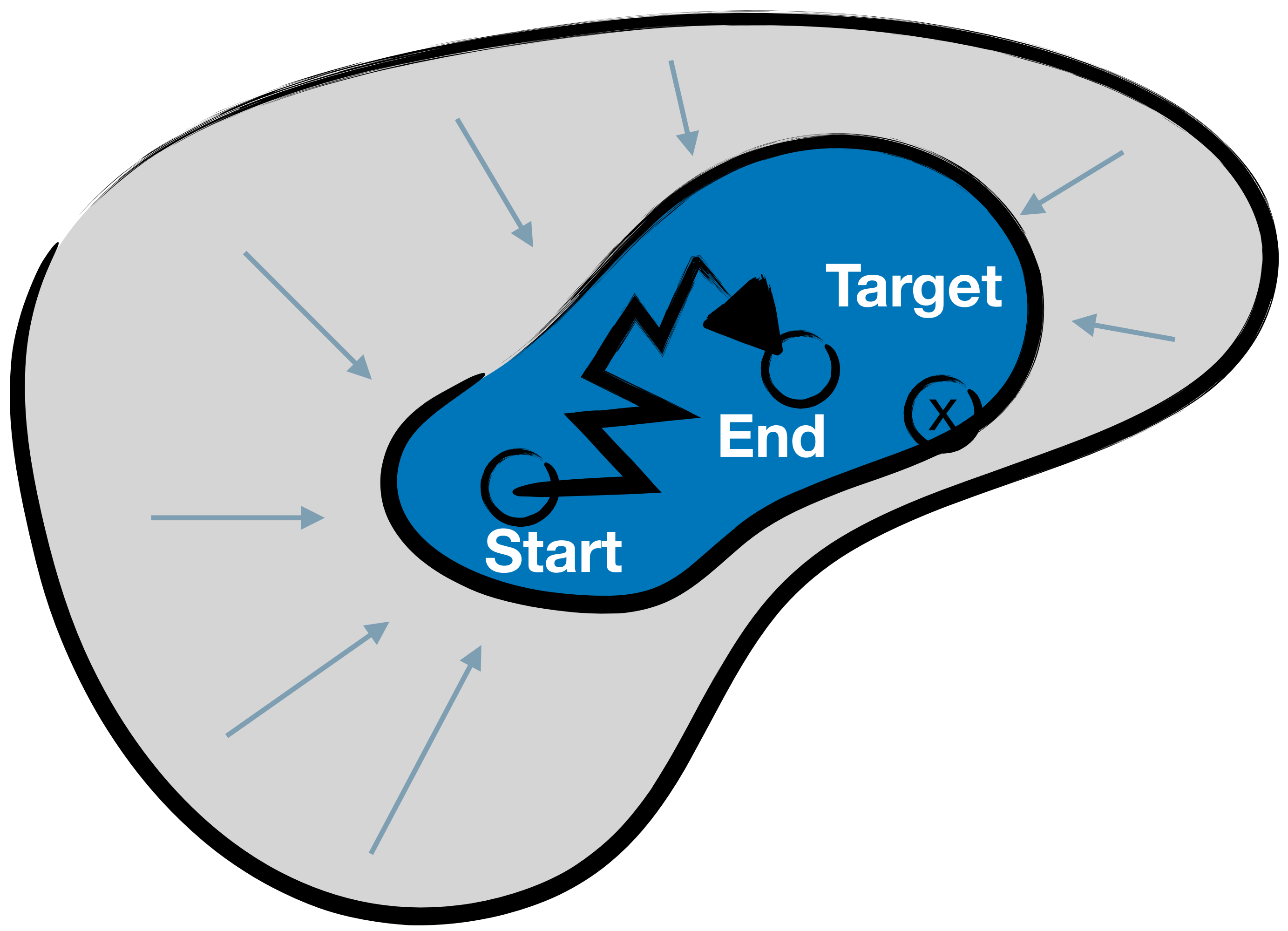
HighRR Lecture Week

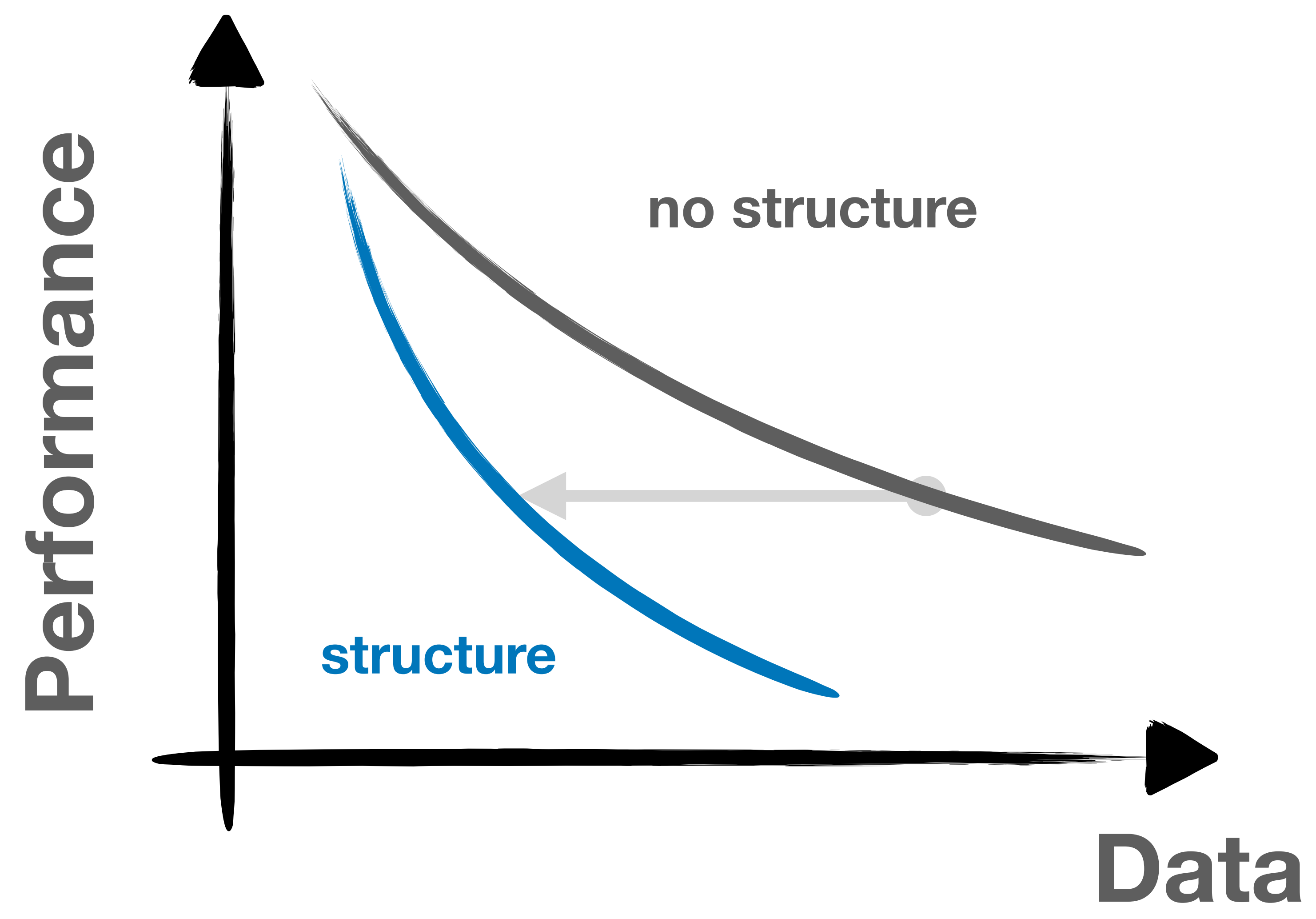

CNN
GNN
TRANSFORMER
DEEP SETS
RNN

The point of Architecture

Why introduce architecture despite universal function approximatation?

- **Data and Training Efficiency**
- **Physics Inductive Bias**

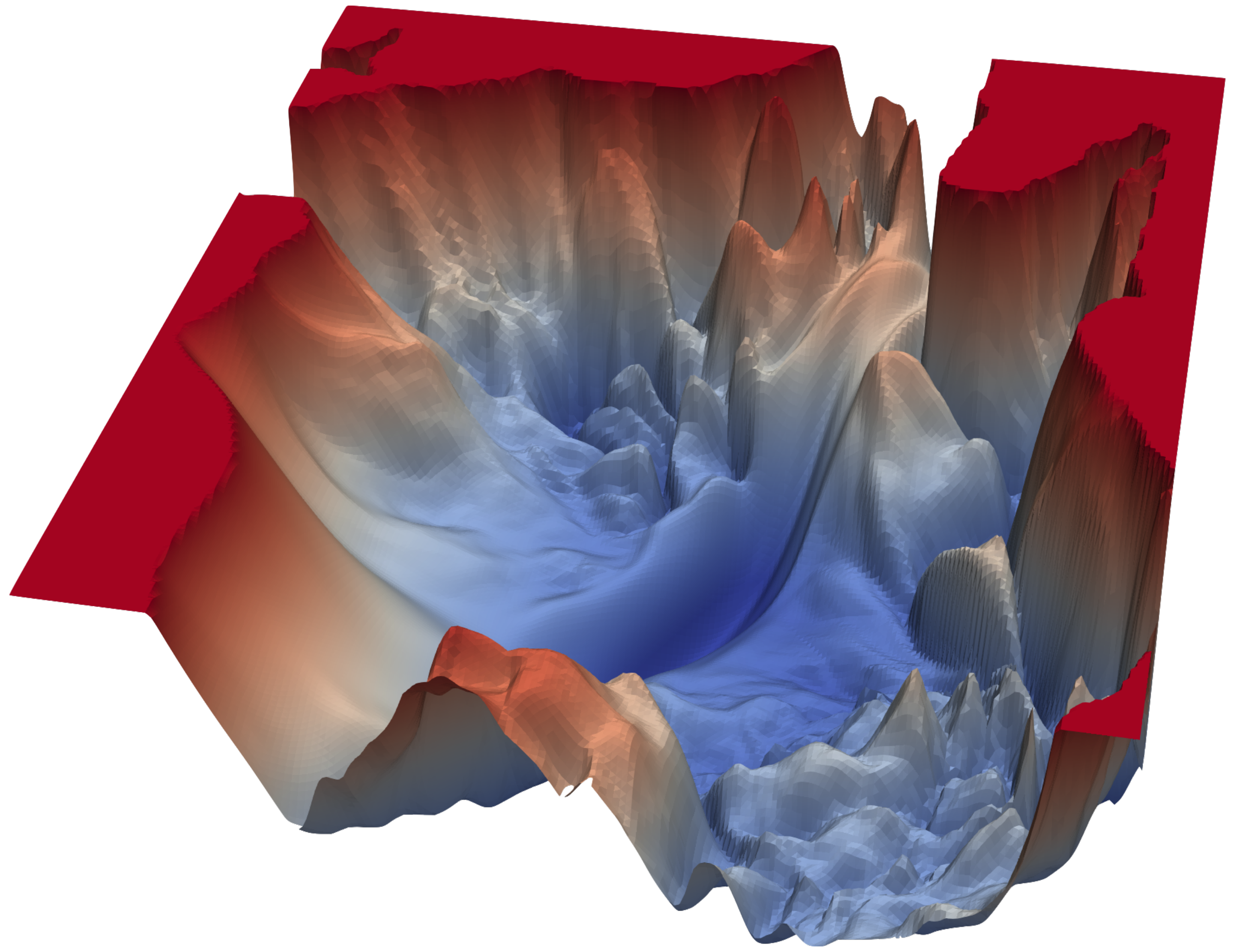


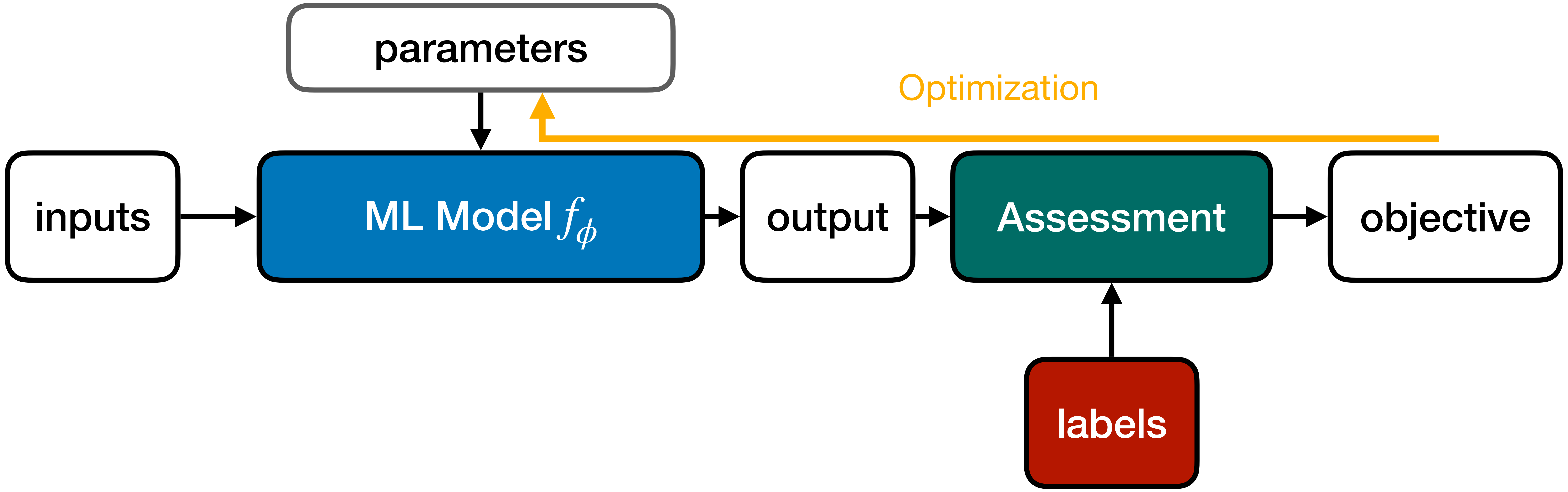


The point of Architecture

So we want structure, but we also want learning…

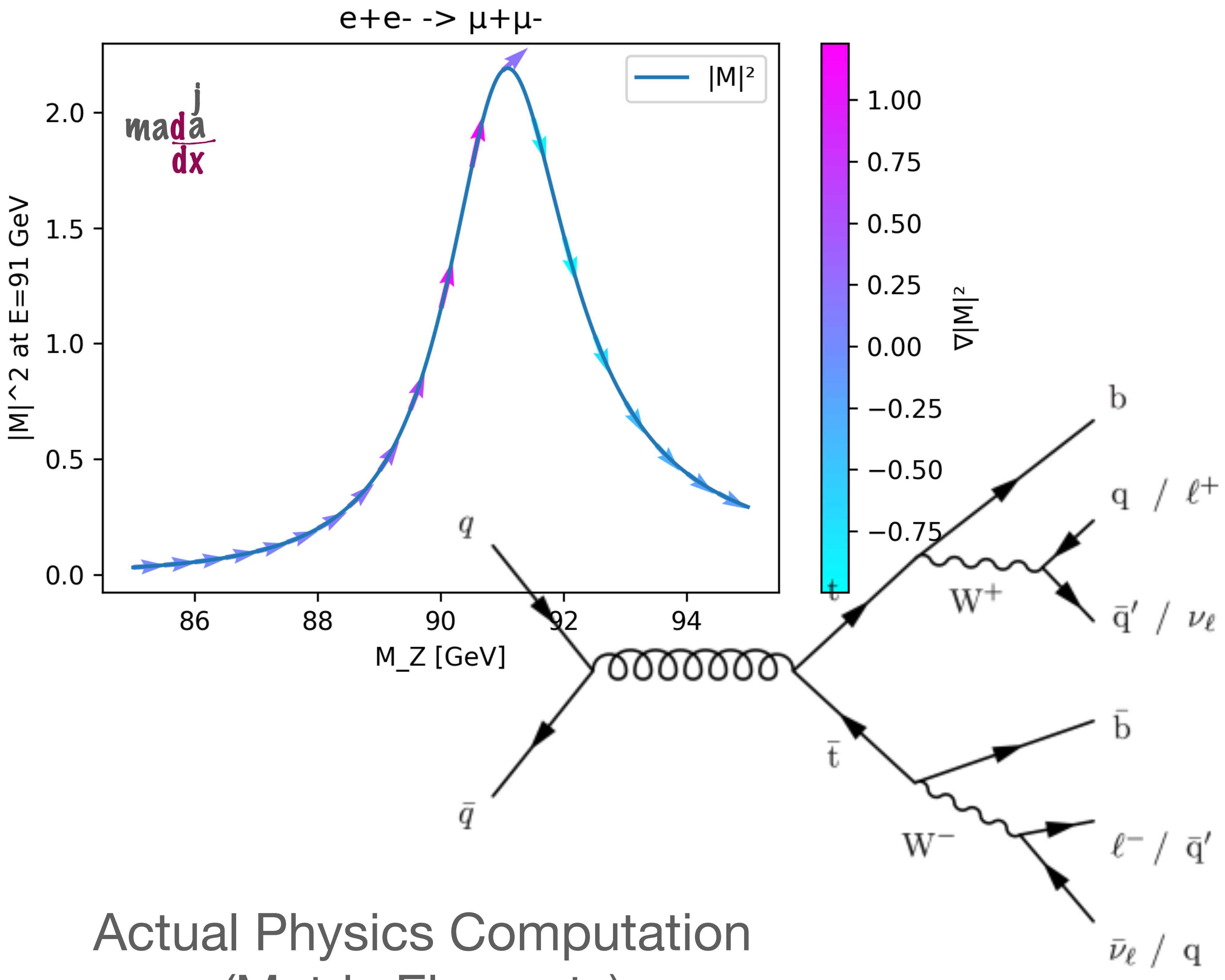
… and that means **gradients** → **differentiable structures**





The point of Architecture

Adding physics information may be much more than just adding symmetries



Actual Physics Computation
(Matrix Elements)

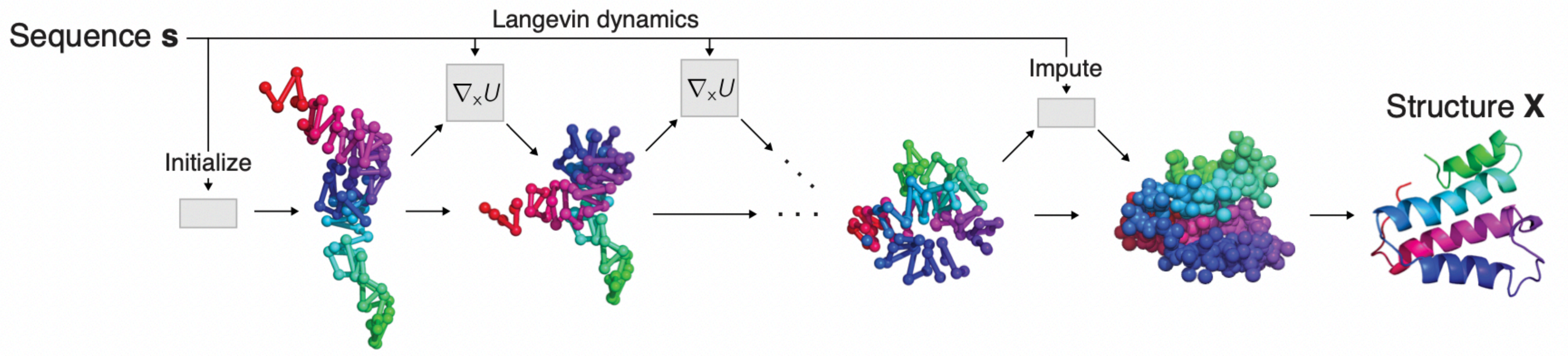


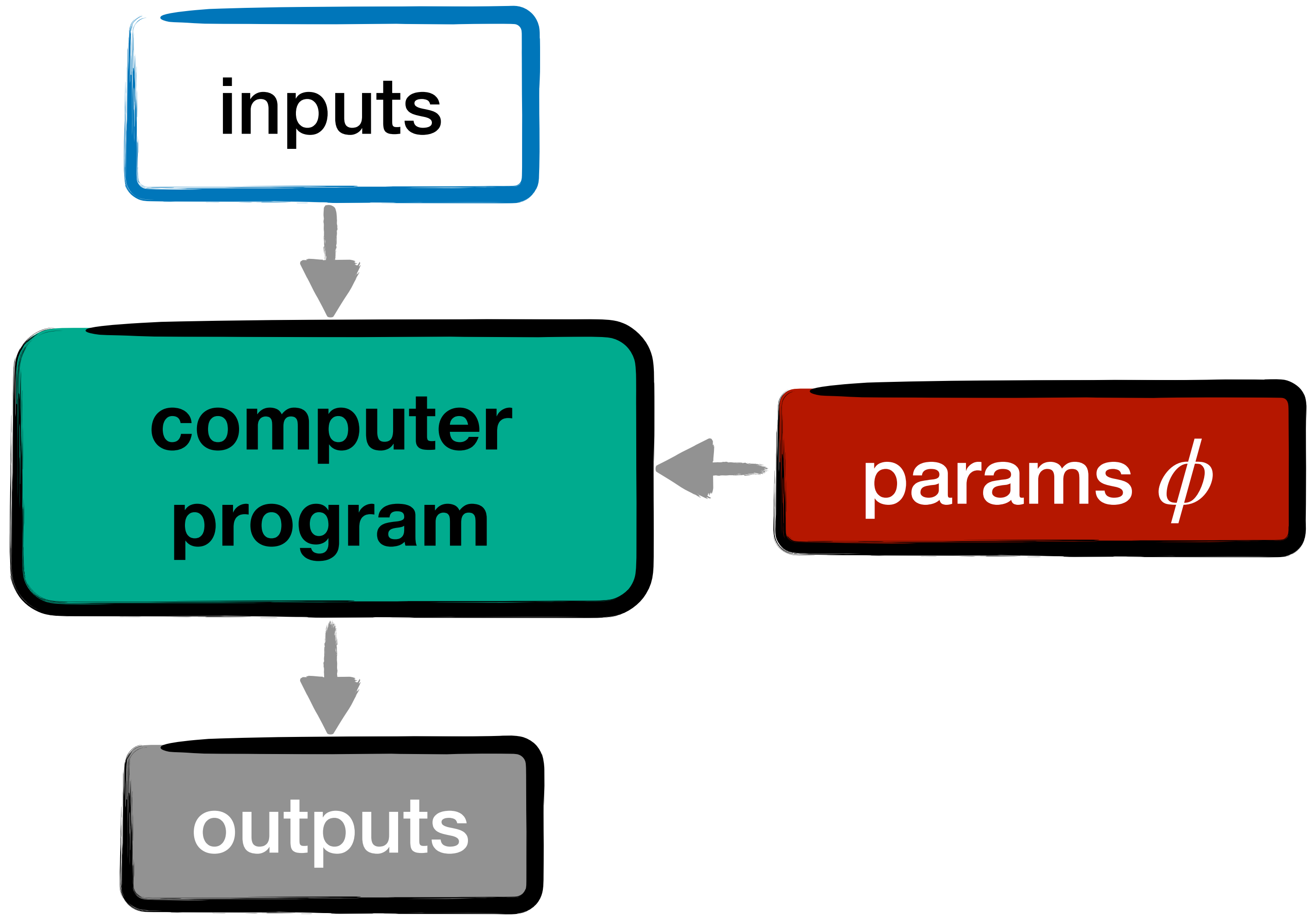
Figure 1: **An unrolled simulator as a model for protein structure.** NEMO combines a neural energy function for coarse protein structure, a stochastic simulator based on Langevin dynamics with learned (amortized) initialization, and an atomic imputation network to build atomic coordinate output from sequence information. It is trained end-to-end by backpropagating through the *unrolled* folding simulation.

Dynamical Layers
(ODE)

Gradients

So what we really want is gradients of arbitrary programs

∇_{ϕ}



?

A big underlying reason for the success of Deep Learning

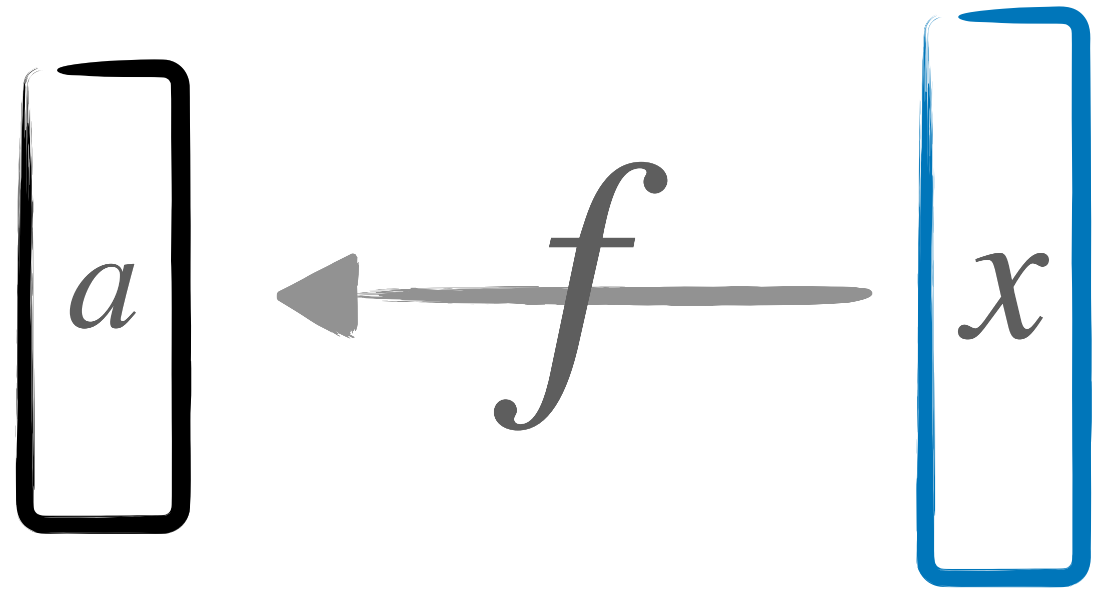
Differentiable Programming

Start Simple

Consider a simple function

$$\mathbb{R}^{n_1} \xrightarrow{f} \mathbb{R}^{n_2}$$

```python
def program(x):
    a = f(x)
    return a
```



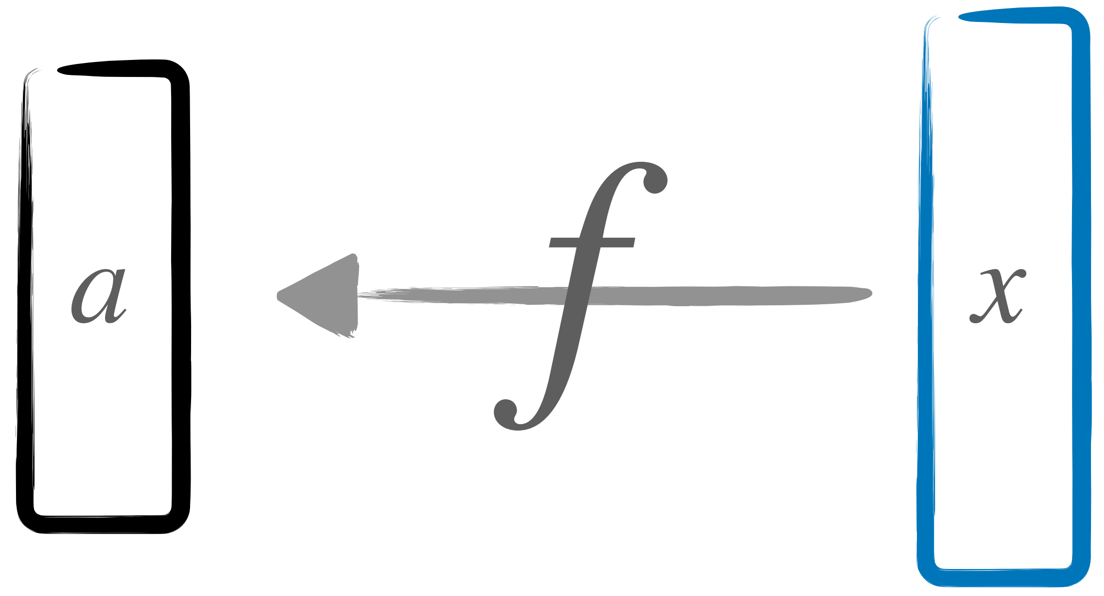
What can we say about the differentials?

Start Simple

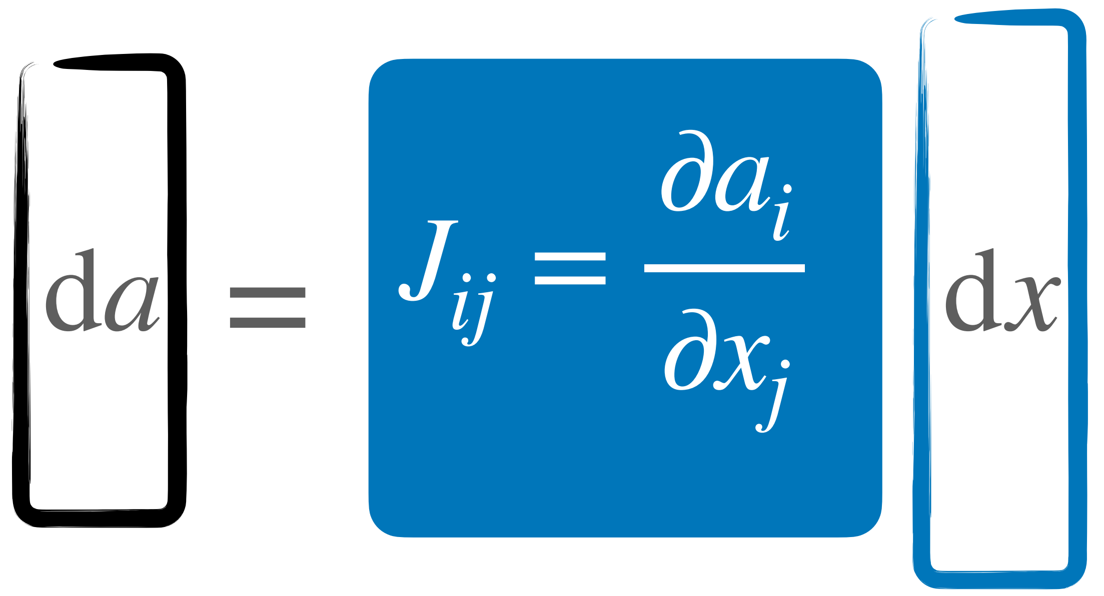
Consider a simple function

$$\mathbb{R}^{n_1} \xrightarrow{f} \mathbb{R}^{n_2}$$

```python
def program(x):
    a = f(x)
    return a
```



Differentials (small changes) transform linearly between input and output



- **scale factor is the Jacobian Matrix**
- **transform is just a matrix multiplication**

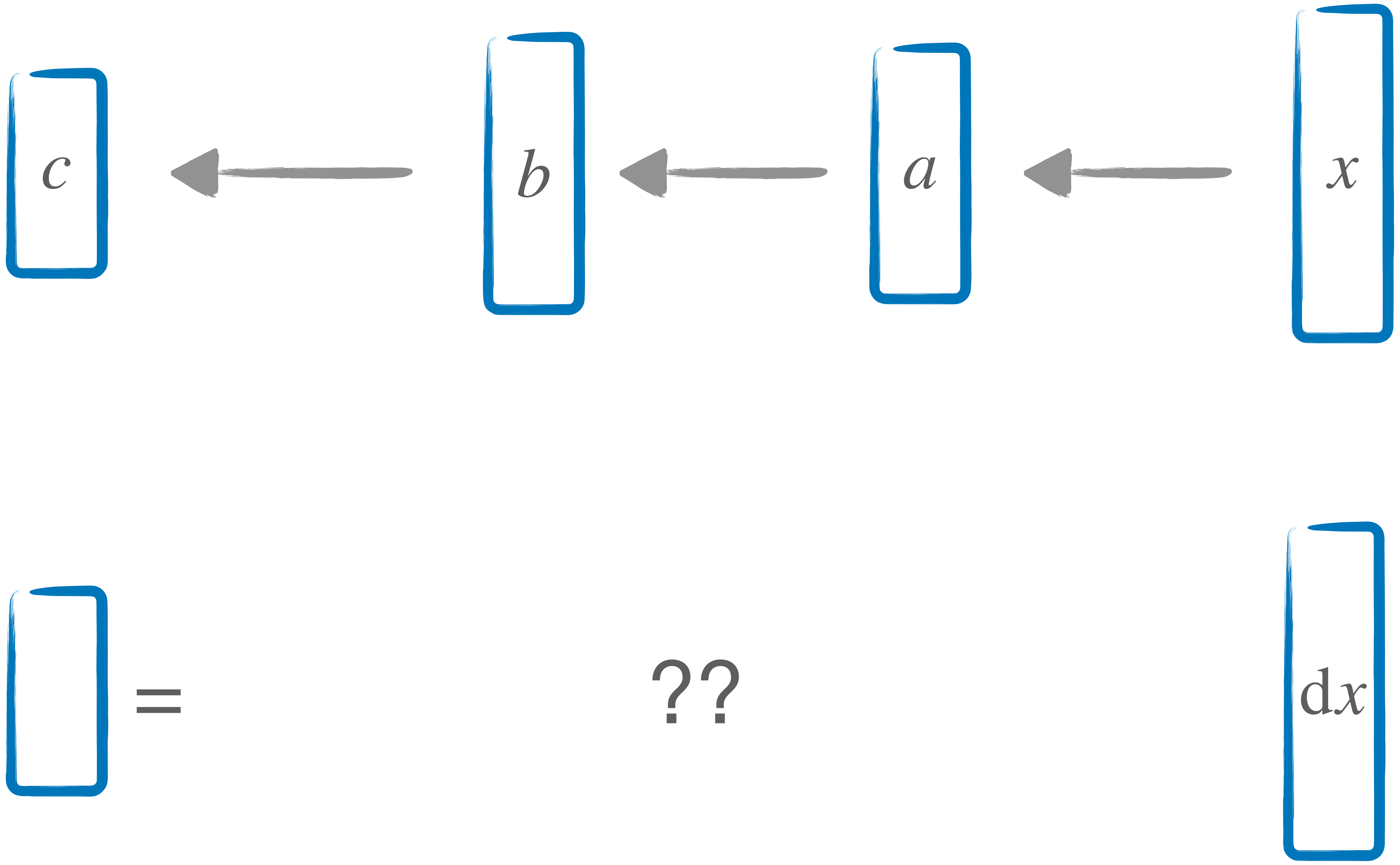
$$\mathrm{d}a_i = J_{ij}\mathrm{d}x_j$$

Start Simple

How about composition?

$$\mathbb{R}^{n_1} \xrightarrow{f} \mathbb{R}^{n_2} \xrightarrow{g} \mathbb{R}^{n_3} \xrightarrow{h} \mathbb{R}^{n_4}$$

```python
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```

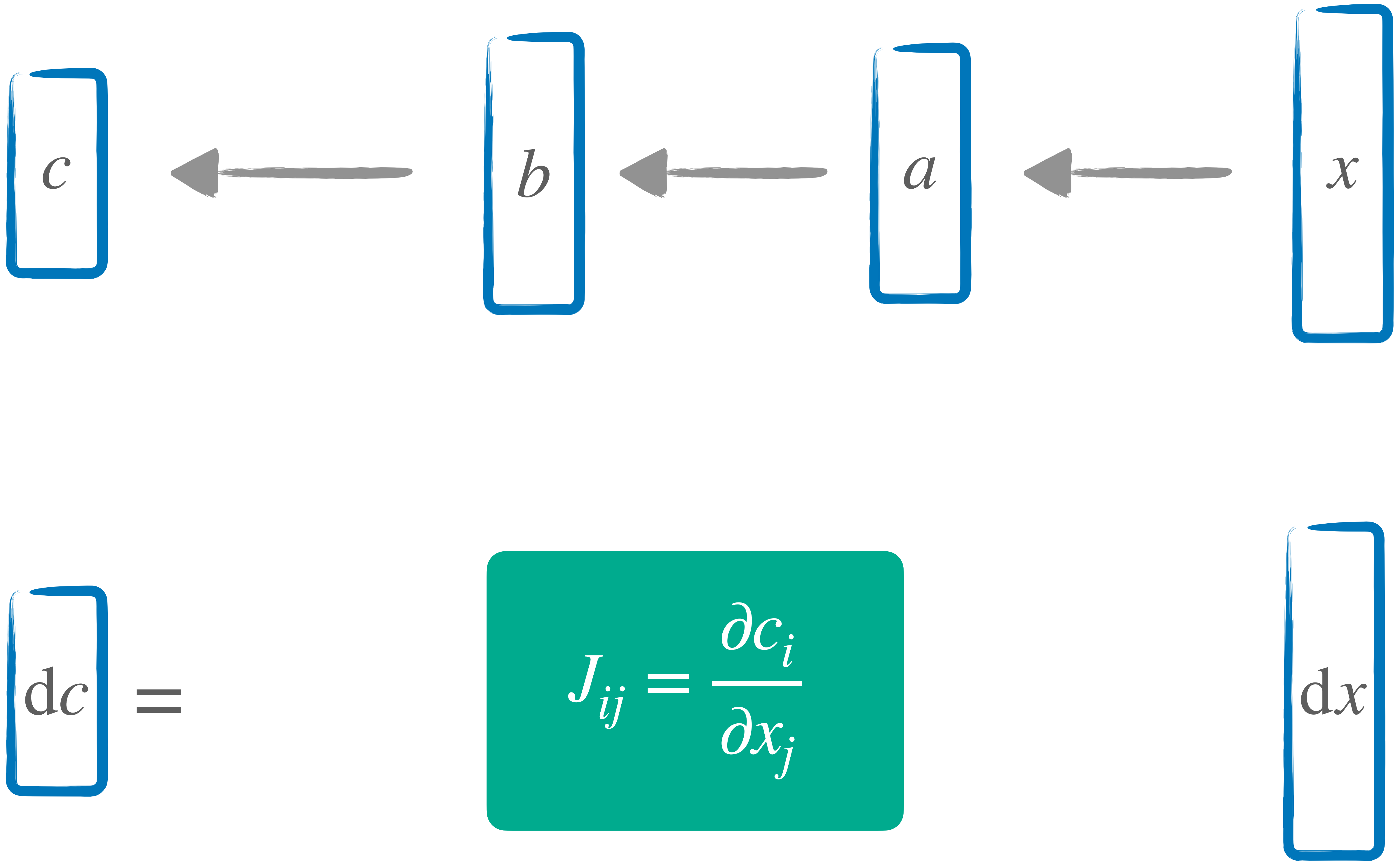


Start Simple

How about composition?

$$\mathbb{R}^{n_1} \xrightarrow{f} \mathbb{R}^{n_2} \xrightarrow{g} \mathbb{R}^{n_3} \xrightarrow{h} \mathbb{R}^{n_4}$$

```python
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```



Start Simple

How about composition?

$$\mathbb{R}^{n_1} \xrightarrow{f} \mathbb{R}^{n_2} \xrightarrow{g} \mathbb{R}^{n_3} \xrightarrow{h} \mathbb{R}^{n_4}$$

```python
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```

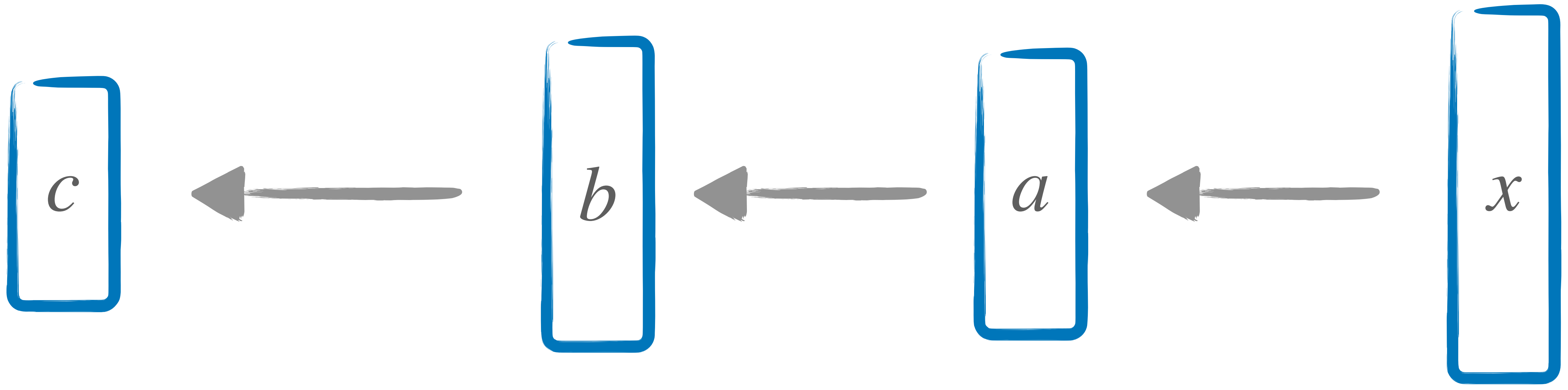
$$c \xleftarrow{h} b \xleftarrow{g} a \xleftarrow{f} x$$

$$\mathrm{d}c = \boxed{J^h_{ik} = \frac{\partial c_i}{\partial b_k}}\ \mathrm{d}b\ \boxed{J^g_{km} = \frac{\partial b_k}{\partial a_m}}\ \mathrm{d}a\ \boxed{J^f_{mj} = \frac{\partial a_m}{\partial x_j}}\ \mathrm{d}x$$

Just the Chain Rule

How about composition?

$$\mathbb{R}^{n_1} \xrightarrow{f} \mathbb{R}^{n_2} \xrightarrow{g} \mathbb{R}^{n_3} \xrightarrow{h} \mathbb{R}^{n_4}$$




```
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```

$$J^{x\to y}_{ij} = \frac{\partial c_i}{\partial x_j} \quad = \quad J^h_{ik} = \frac{\partial c_i}{\partial b_k} \quad J^g_{km} = \frac{\partial b_k}{\partial a_m} \quad J^f_{mj} = \frac{\partial a_m}{\partial x_j}$$

$$\mathrm{d}c = J^{x\to y}\mathrm{d}x = J^h J^g J^f \mathrm{d}x$$

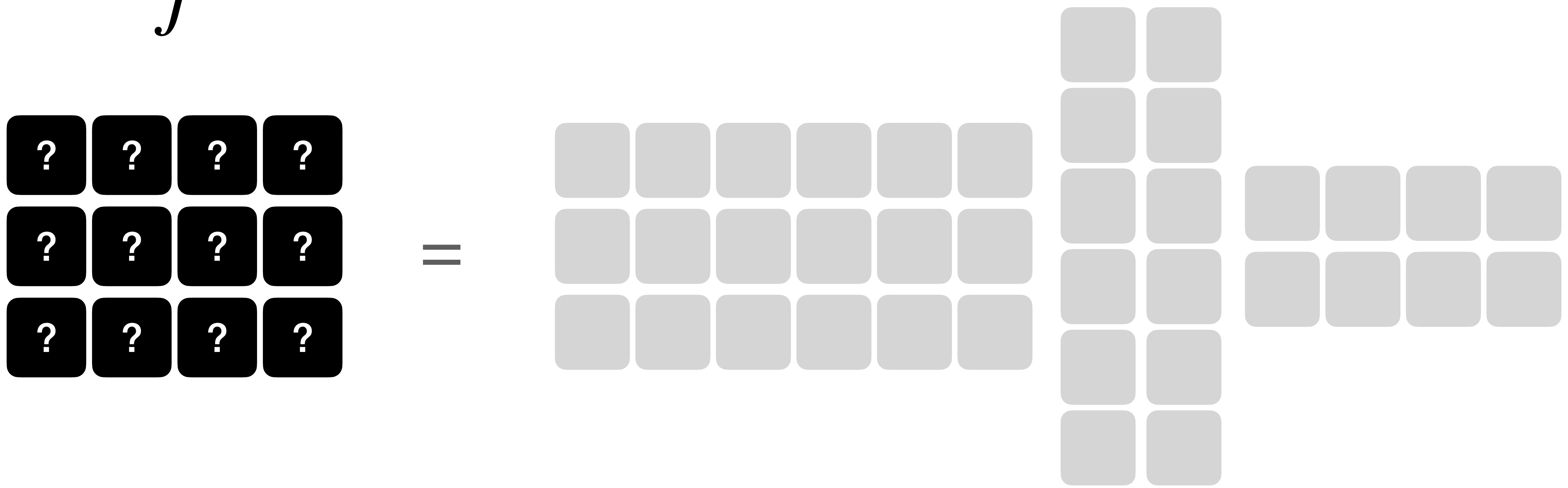
This is just the chain rule $\partial_x\, f(y = g(x)) = \partial_y f \cdot \partial_x g$

So, it’s just about Matrix Multiplication

To know the **gradients (i.e. Jacobians)** of a composed program, we need a good way to characterize **products of Jacobians** matrices

```python
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```

$$\frac{\partial c_i}{\partial x_j} = J^{x \to y} = J^h J^g J^f$$



It’s all about Matrices

Naive Multiplication

Explicitly constructing matrices and naively multiplying doesn't scale, details on how we compute matter

2 x 50 | 50 x 100 | 100 x 200

$$(AB)C \qquad\qquad A(BC)$$

100k FLOPS | 2M FLOPS

Best option: Matrix Multiply without multiplying Matrices

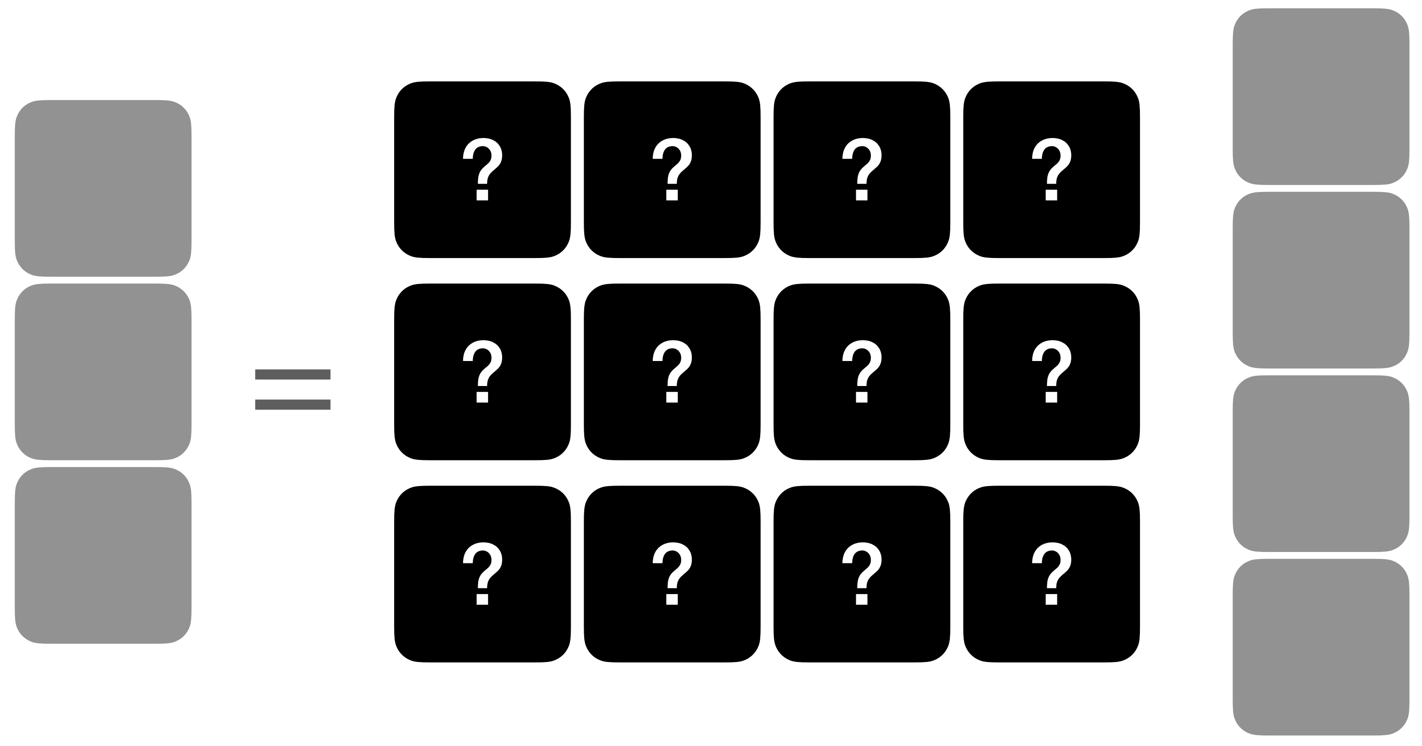
So let’s talk about Matrices

Inspecting a Matrix

Matrices encode transformations of vectors

Conversely: we can recover all information about a matrix by observing how vectors get transformed

- **i.e. through Matrix-Vector Products.**

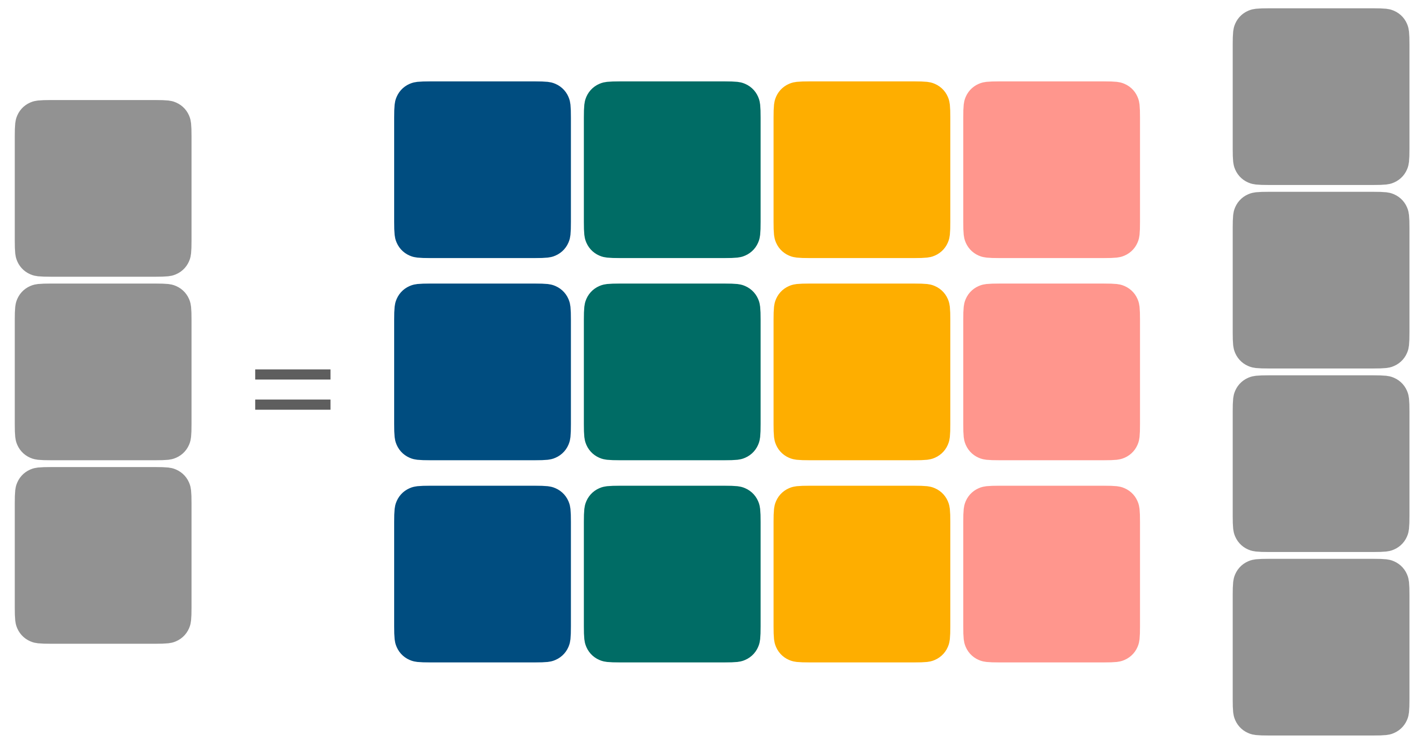


Inspecting a Matrix

Matrices encode transformations of vectors

Conversely: we can recover all information about a matrix by observing how vectors get transformed

- **i.e. through Matrix-Vector Products.**

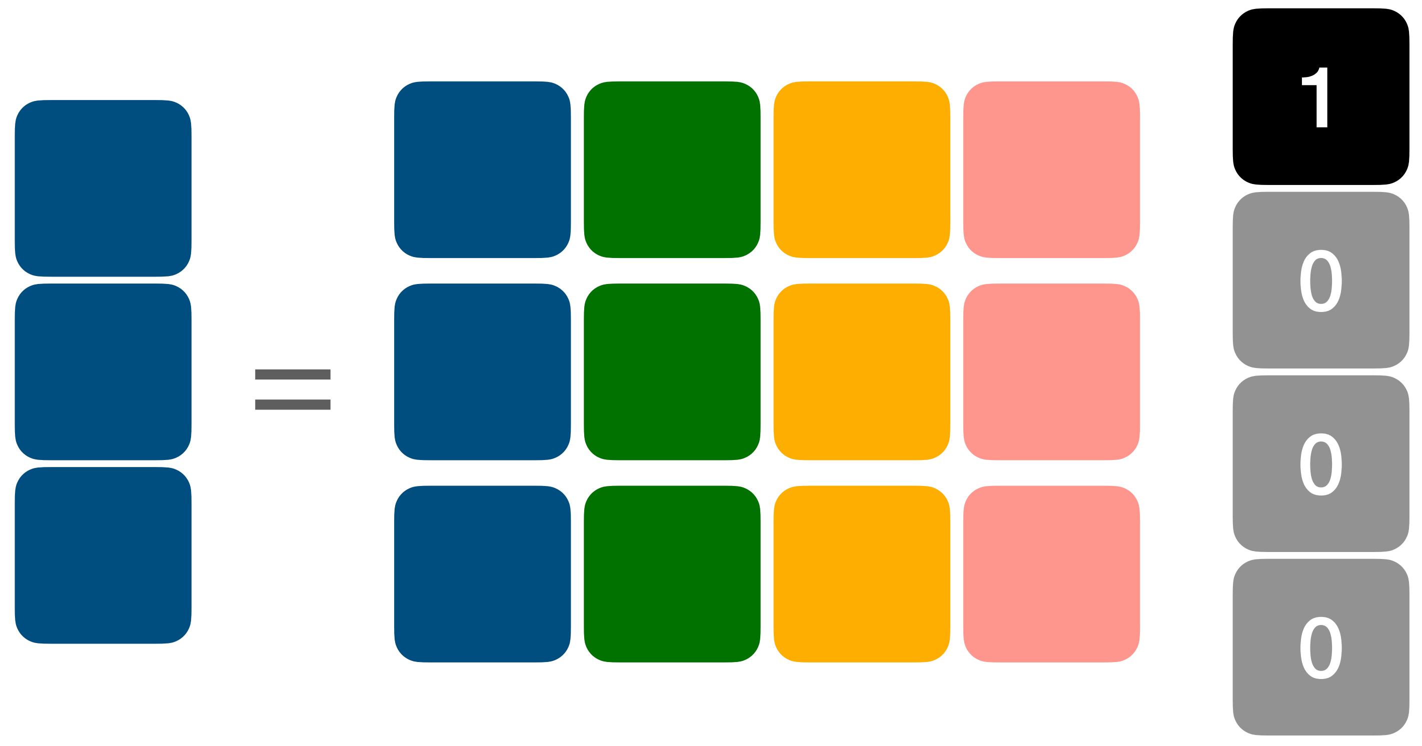


Inspecting a Matrix

Matrices encode transformations of vectors

Conversely: we can recover all information about a matrix by observing how vectors get transformed

- **i.e. through Matrix-Vector Products.**

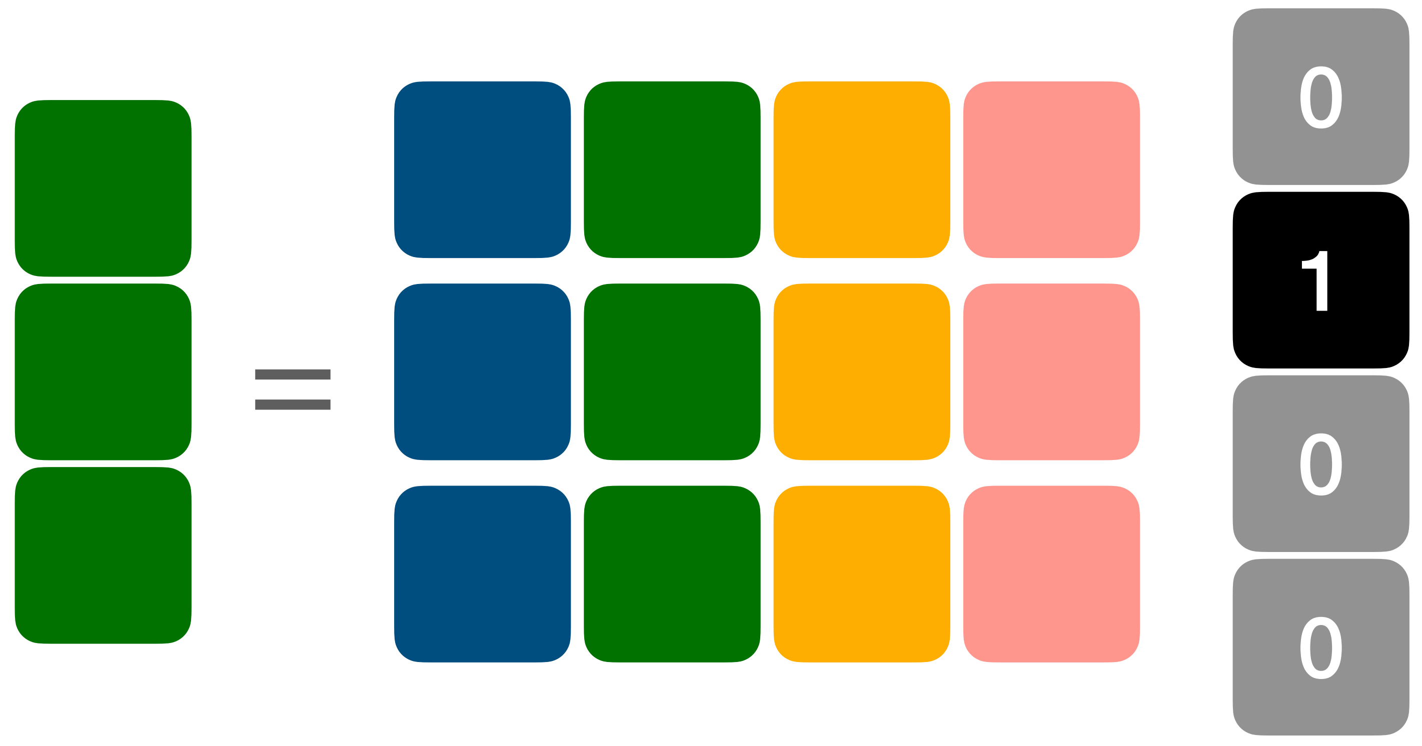


Inspecting a Matrix

Matrices encode transformations of vectors

Conversely: we can recover all information about a matrix by observing how vectors get transformed

- **i.e. through Matrix-Vector Products.**

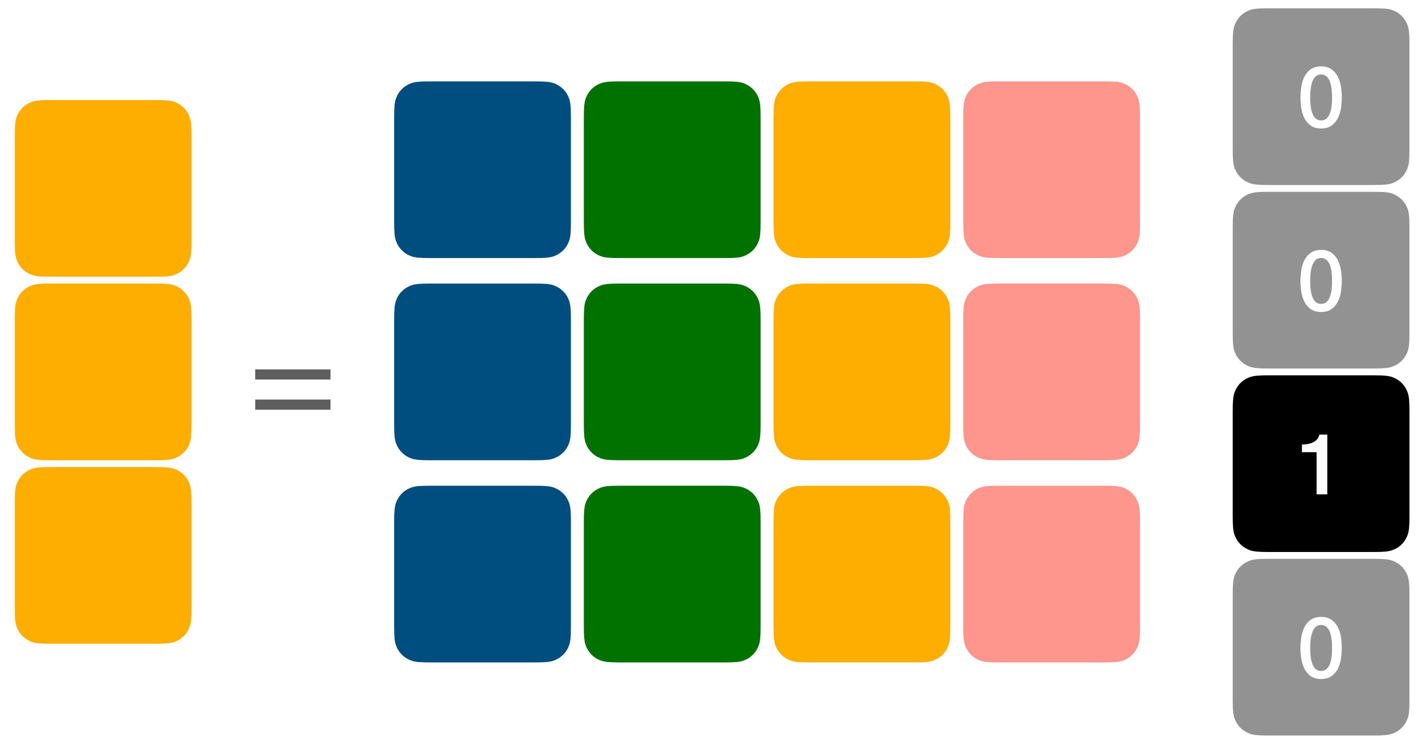


Inspecting a Matrix

Matrices encode transformations of vectors

Conversely: we can recover all information about a matrix by observing how vectors get transformed

- **i.e. through Matrix-Vector Products.**

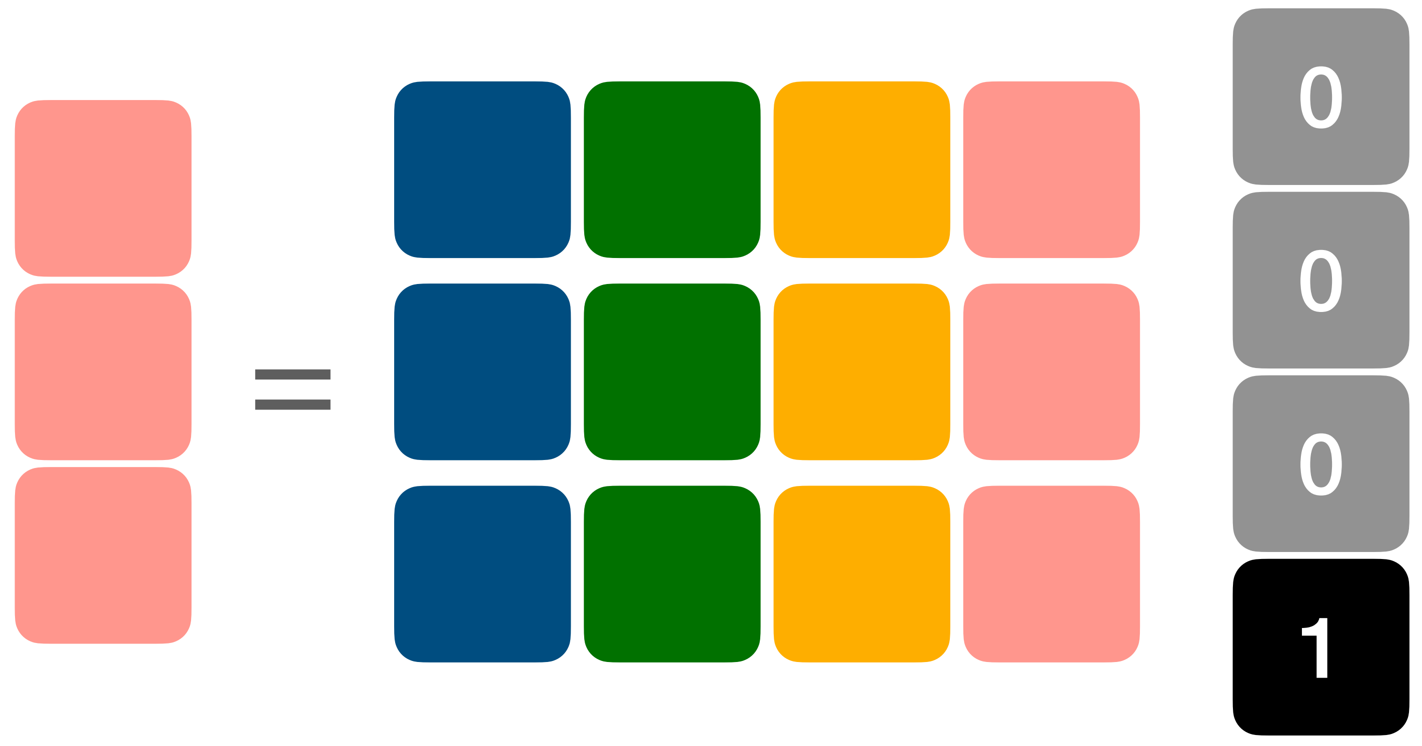


Inspecting a Matrix

Matrices encode transformations of vectors

Conversely: we can recover all information about a matrix by observing how vectors get transformed

- **i.e. through Matrix-Vector Products.**



A Matrix as a Program

Gives us an efficient way to "store"/express matrices:

matrix ↔ a program that computes transforms vectors

$$\begin{bmatrix} 2 & 3 & 0 \\ 0 & 5 & 4 \end{bmatrix}$$

```python
def mvp(inp):
    x,y,z = inp
    return np.array([
        2*x + 3*y,
        5*y + 4*z
    ])

mvp([1,2,3])
```

```
array([ 8, 22])
```

```python
def explicit(inp):
    matrix = np.array([
        [2,3,0],
        [0,5,4]
    ])
    return matrix @ inp

explicit([1,2,3])
```

```
array([ 8, 22])
```

A Matrix as a Program

Recover full Matrix through three MVPs

$$\begin{bmatrix} 2 & 3 & 0 \\ 0 & 5 & 4 \end{bmatrix}$$

```python
def explicit(inp):
    matrix = np.array([
        [2,3,0],
        [0,5,4]
    ])
    return matrix @ inp
```

```python
explicit([0,0,1])
```

```
array([0, 4])
```

```python
explicit([0,1,0])
```

```
array([3, 5])
```

```python
explicit([1,0,0])
```

```
array([2, 0])
```

```python
def mvp(inp):
    x,y,z = inp
    return np.array([
        2*x + 3*y,
        5*y + 4*z
    ])
```

```python
mvp([0,0,1])
```

```
array([0, 4])
```

```python
mvp([0,1,0])
```

```
array([3, 5])
```

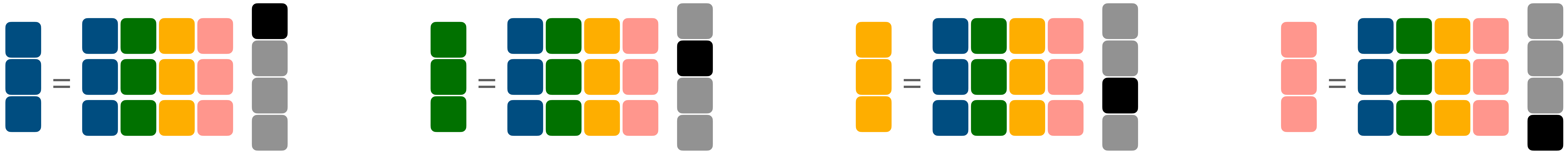
```python
mvp([1,0,0])
```

```
array([2, 0])
```

Inspecting a Matrix

Ability to compute Matrix-Vector Products (MVP) is sufficient to extract any information we want from a matrix. Note:

- do not need the explicit matrix, just ability to compute MVPs
- to get full matrix we need do N_{column} MVPs computations



Or we can go the other way around…

Matrices encode transformations of vectors

Conversely: we can recover all information about a matrix by throwing specific vectors at it, <u>from the left.</u>

- **i.e. through Vector-Matrix Products.**

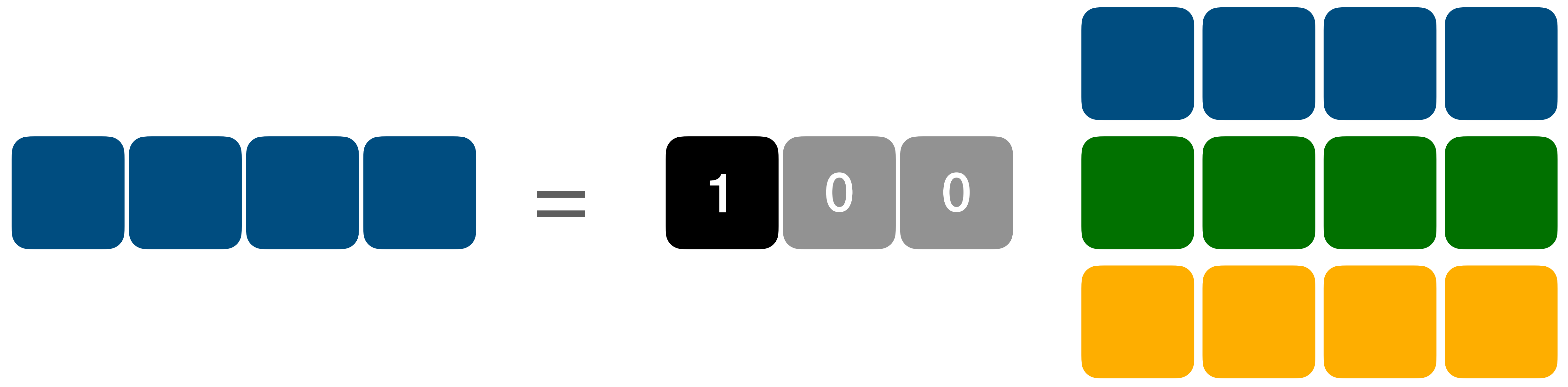
$$\begin{bmatrix} \square & \square & \square & \square \end{bmatrix} = \begin{bmatrix} \square & \square & \square \end{bmatrix} \begin{bmatrix} ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{bmatrix}$$

Or we can go the other way around…

Matrices encode transformations of vectors

Conversely: we can recover all information about a matrix by throwing specific vectors at it, <u>from the left.</u>

- **i.e. through Vector-Matrix Products.**

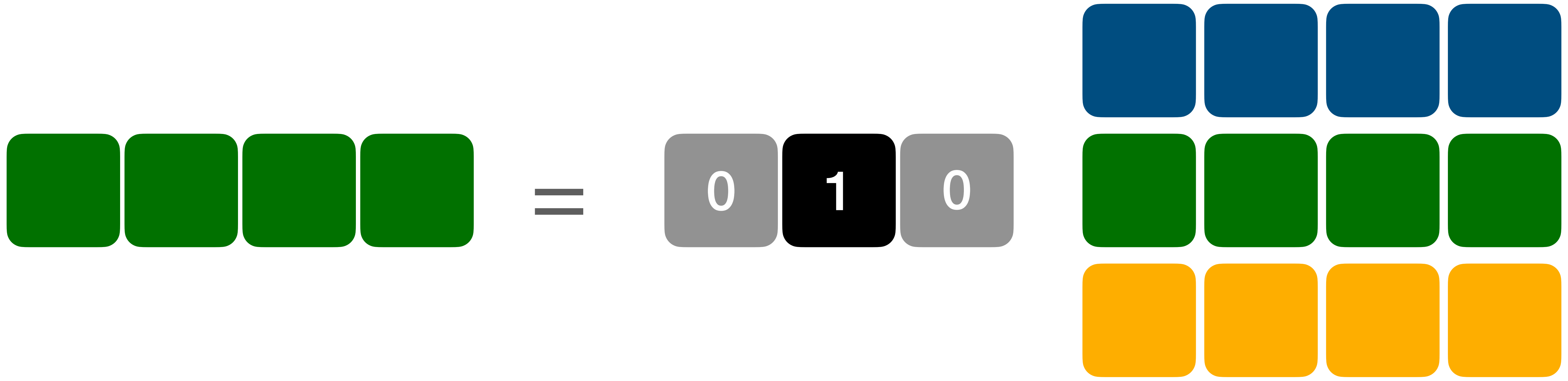


Or we can go the other way around…

Matrices encode transformations of vectors

Conversely: we can recover all information about a matrix by throwing specific vectors at it, from the left.

- **i.e. through Vector-Matrix Products.**

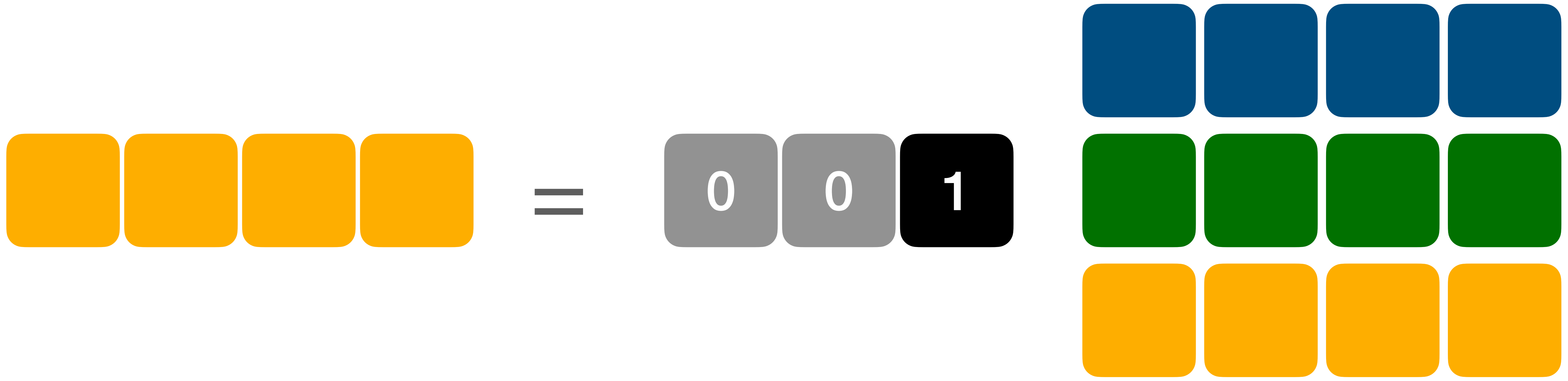


Or we can go the other way around…

Matrices encode transformations of vectors

Conversely: we can recover all information about a matrix by throwing specific vectors at it, <u>from the left.</u>

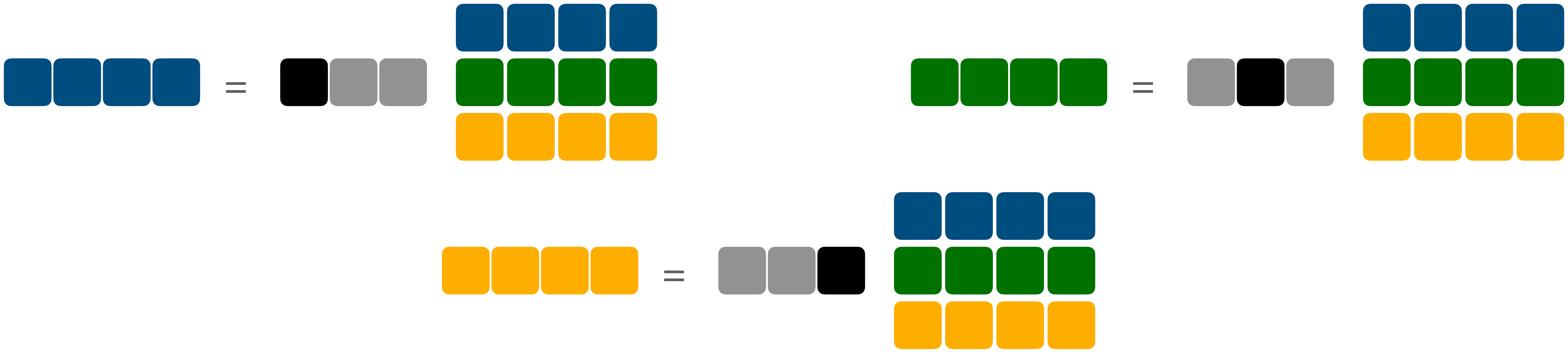
- **i.e. through Vector-Matrix Products.**



Or we can go the other way around…

Ability to compute Vector Products (MVP) is sufficient to extract any information we want from a matrix. Note:

- do not need the explicit matrix, just ability to compute VMPs
- to get full matrix we need to compute N_{row} VMPs computations



Again as Programs

Again, having a program that computes vector-matrix products (VMPs) is equivalent to having the full matrix.

$$\begin{bmatrix} 2 & 3 & 0 \\ 0 & 5 & 4 \end{bmatrix}$$

```python
def vmp(out):
    a,b  = out
    return np.array([
        2*a,
        3*a + 5*b,
        4*b
    ])
```

```python
vmp([2,3])
```

```
array([ 4, 21, 12])
```

```python
def explicit(out):
    matrix = np.array([
        [2,3,0],
        [0,5,4]
    ])
    return np.matmul(np.array(out).T,matrix)
```

```python
explicit([2,3])
```

```
array([ 4, 21, 12])
```

Again as Programs

Recover full Matrix through two VMPs

$$\begin{bmatrix} 2 & 3 & 0 \\ 0 & 5 & 4 \end{bmatrix}$$

```python
def explicit(out):
    matrix = np.array([
        [2,3,0],
        [0,5,4]
    ])
    return out @ matrix
```

```python
explicit([1,0])
```

```
array([2, 3, 0])
```

```python
explicit([0,1])
```

```
array([0, 5, 4])
```

```python
def vmp(out):
    a,b = out
    return np.array([
        2*a,
        3*a + 5*b,
        4*b
    ])
```

```python
explicit([1,0])
```

```
array([2, 3, 0])
```

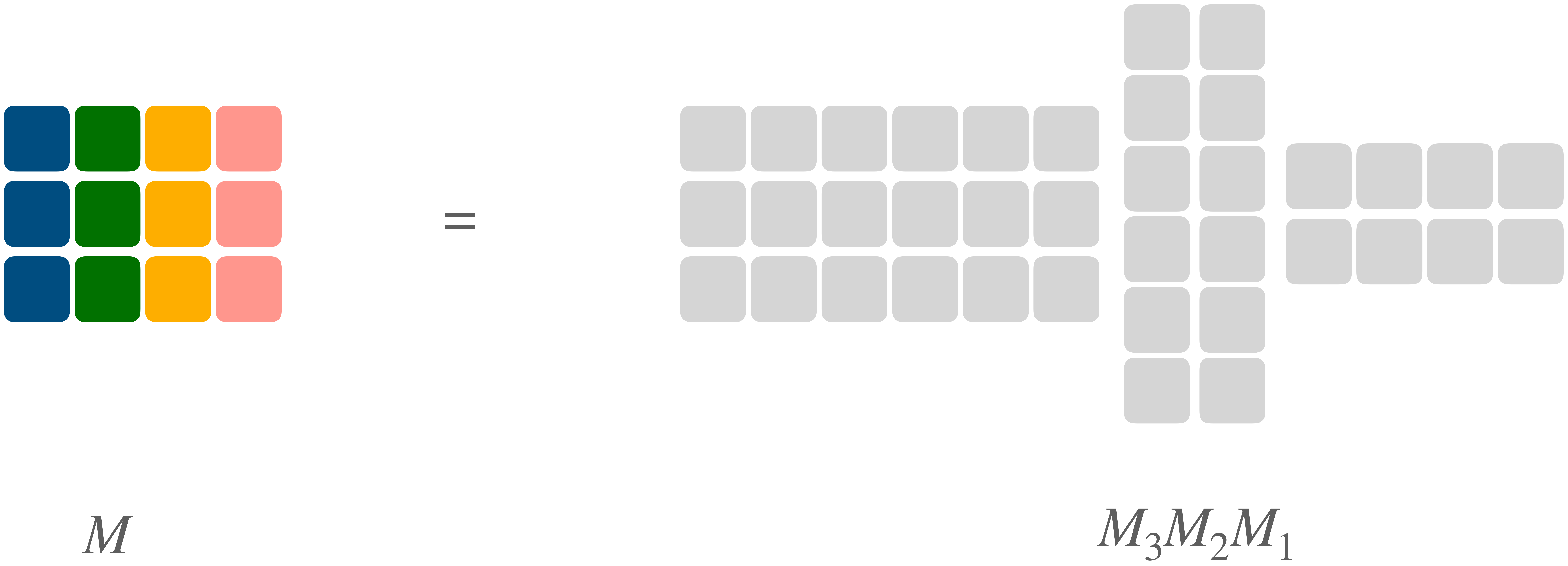
```python
explicit([0,1])
```

```
array([0, 5, 4])
```

Composition of Matrices

The MVP, VMP picture still works if we have compositions

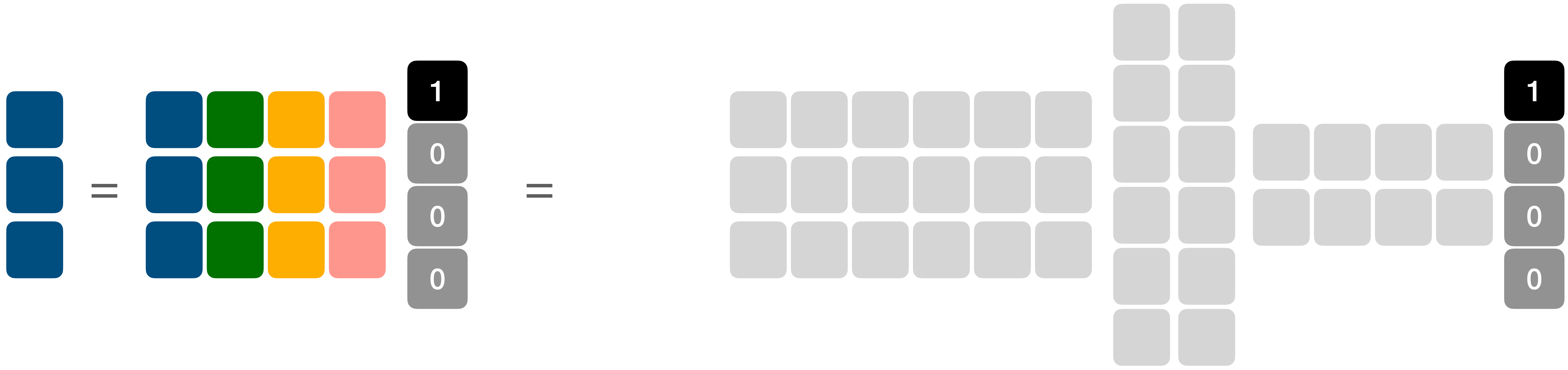
- to characterize $M = M_3 M_2 M_1$ we just need MVP/MVP with M_i



Composition of Matrices

The MVP, VMP picture still works if we have compositions

- to characterize $M = M_3 M_2 M_1$ we just need MVP/MVP with M_i



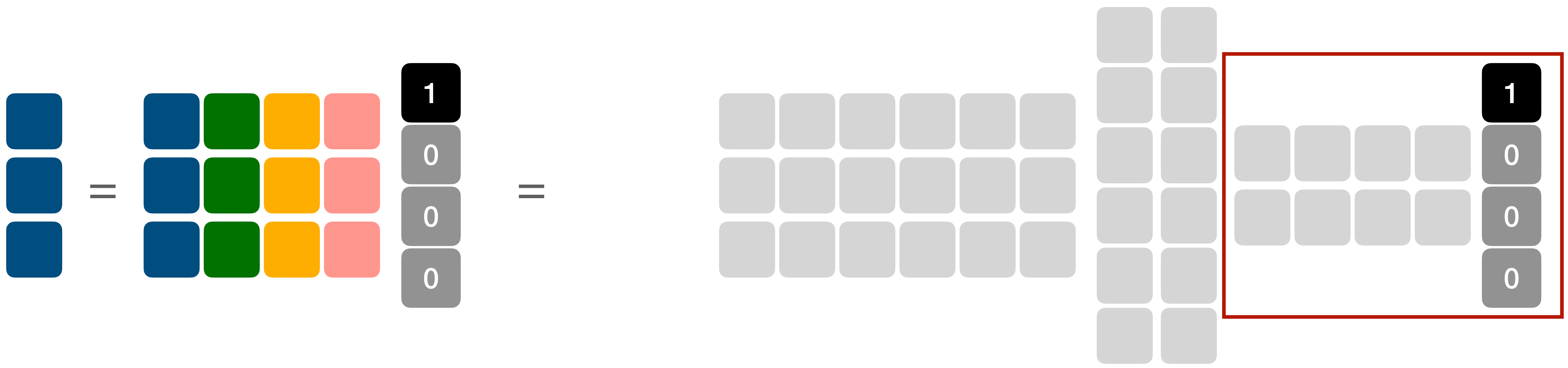
$c_i = Me_i$

$c_i = Me_i = M_3 M_2 M_1 e_i$

Composition of Matrices

The MVP, VMP picture still works if we have compositions

- to characterize $M = M_3 M_2 M_1$ we just need MVP/MVP with M_i



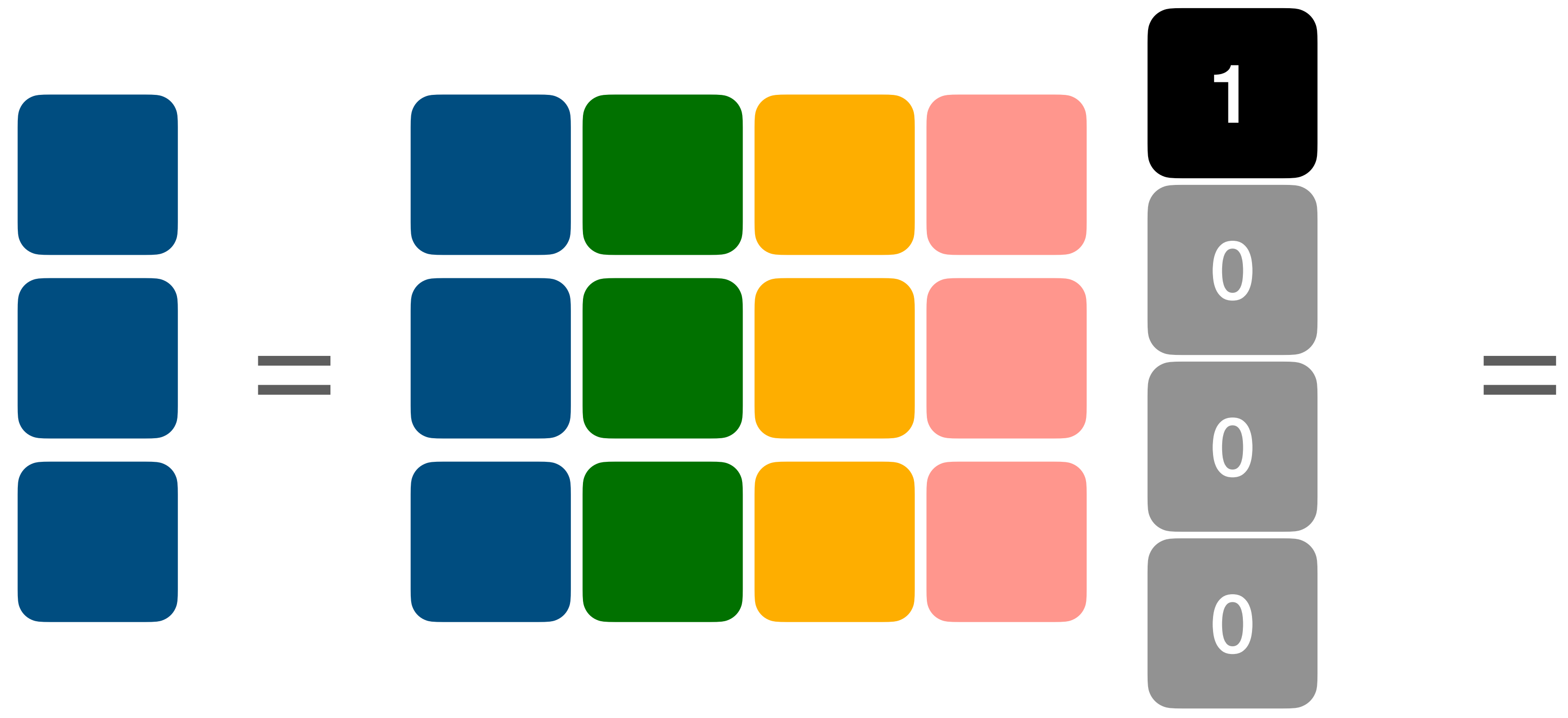
$c_i = Me_i$

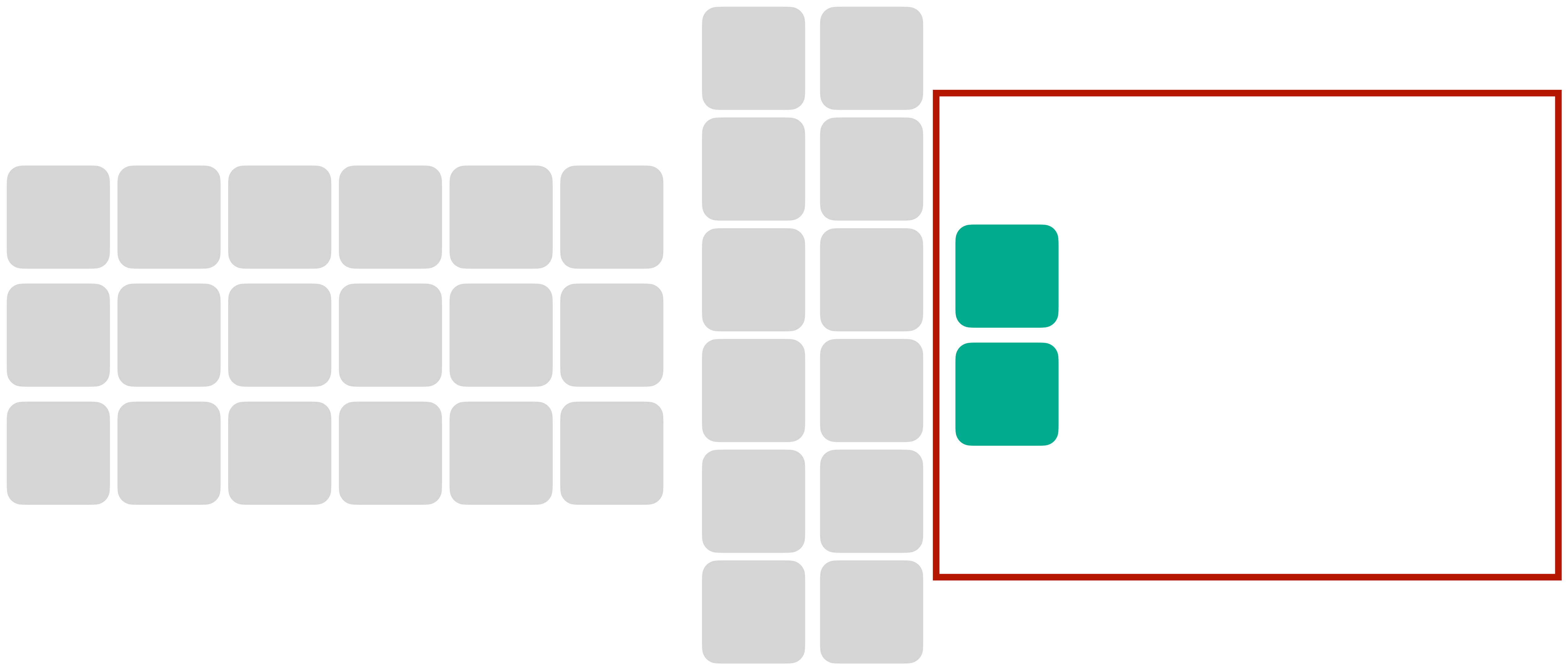
$c_i = Me_i = M_3 M_2 \boxed{M_1 e_i}$

Composition of Matrices

The MVP, VMP picture still works if we have compositions

- to characterize $M = M_3 M_2 M_1$ we just need MVP/MVP with M_i





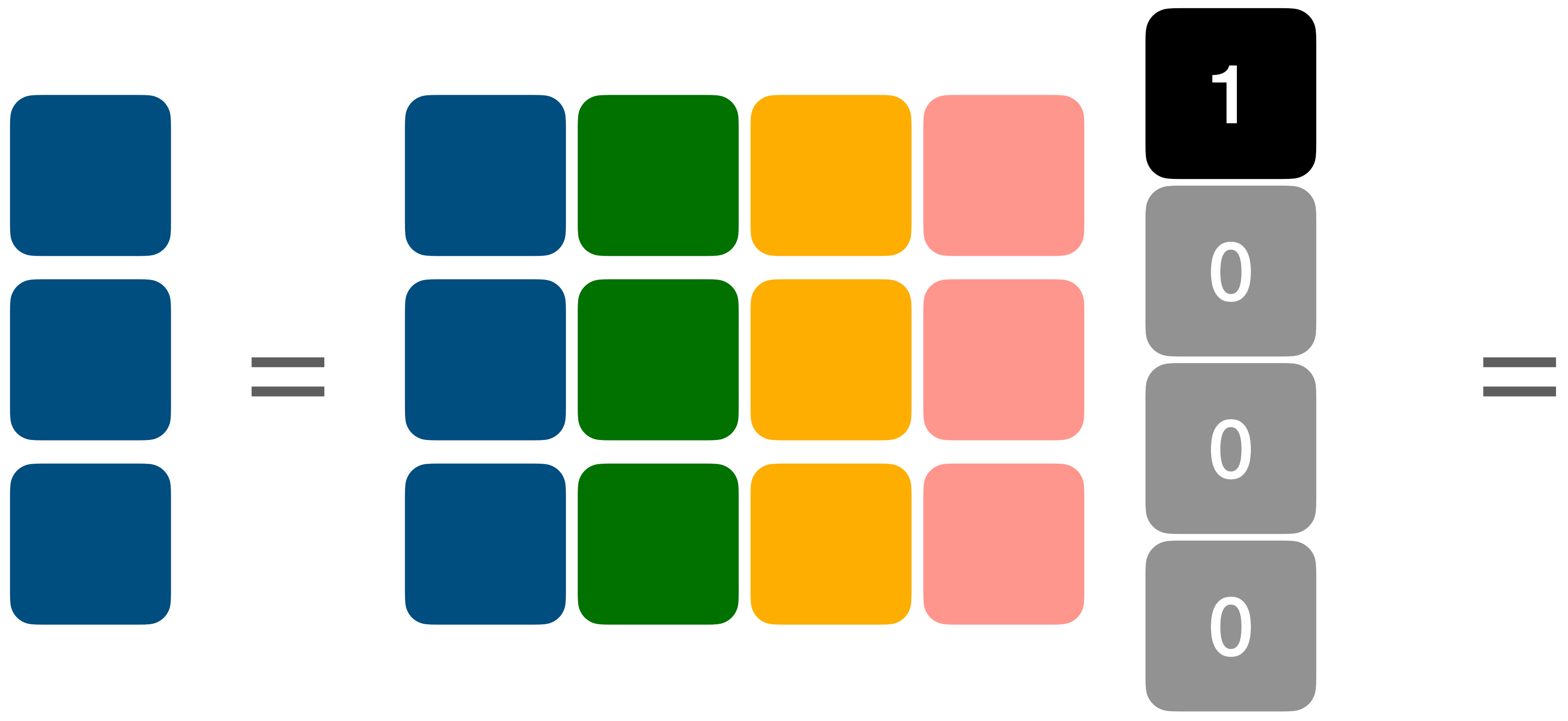
$c_i = Me_i$

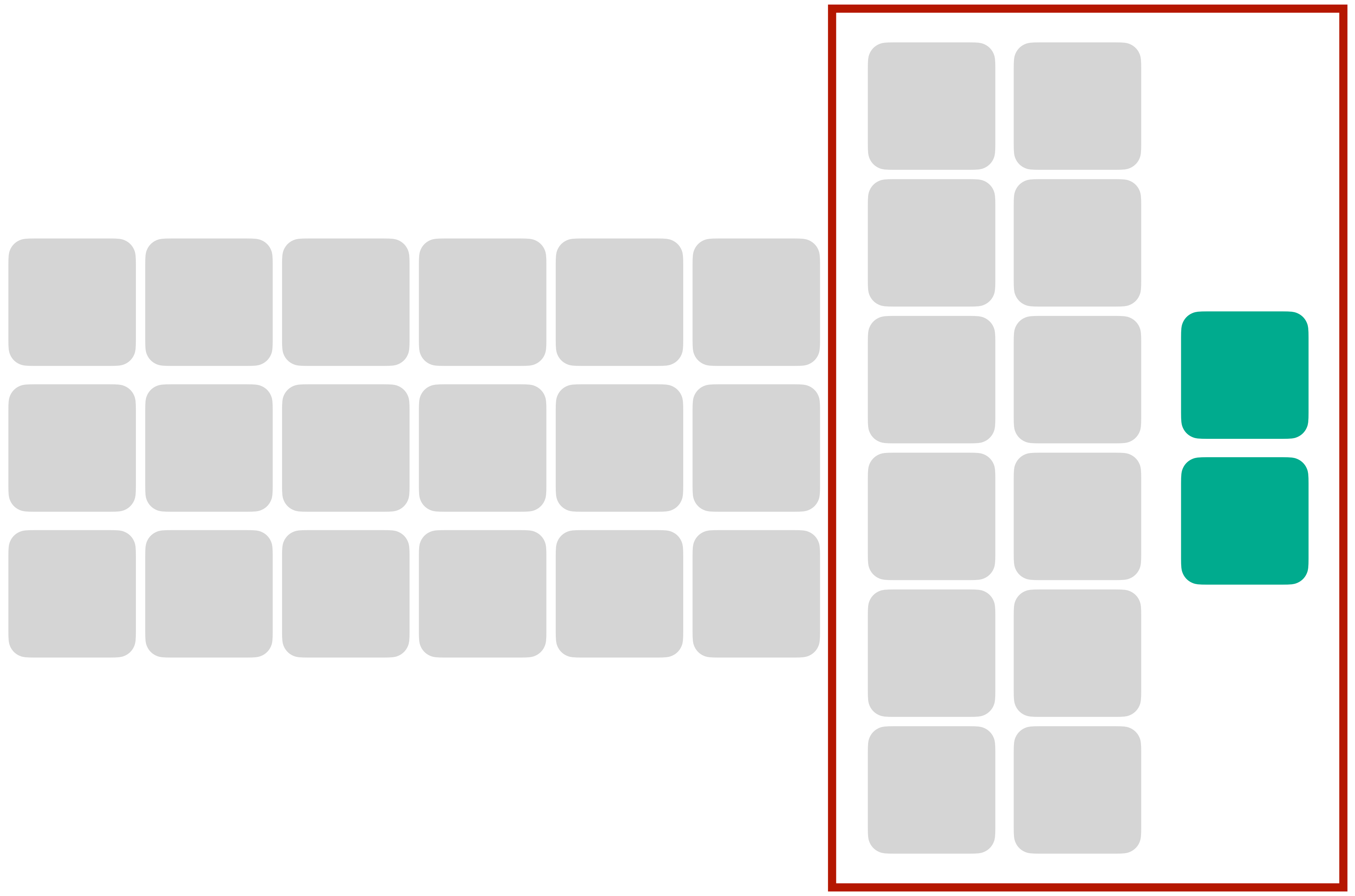
$c_i = Me_i = M_3 M_2 \boxed{v_1}$

Composition of Matrices

The MVP, VMP picture still works if we have compositions

- to characterize $M = M_3M_2M_1$ we just need MVP/MVP with M_i





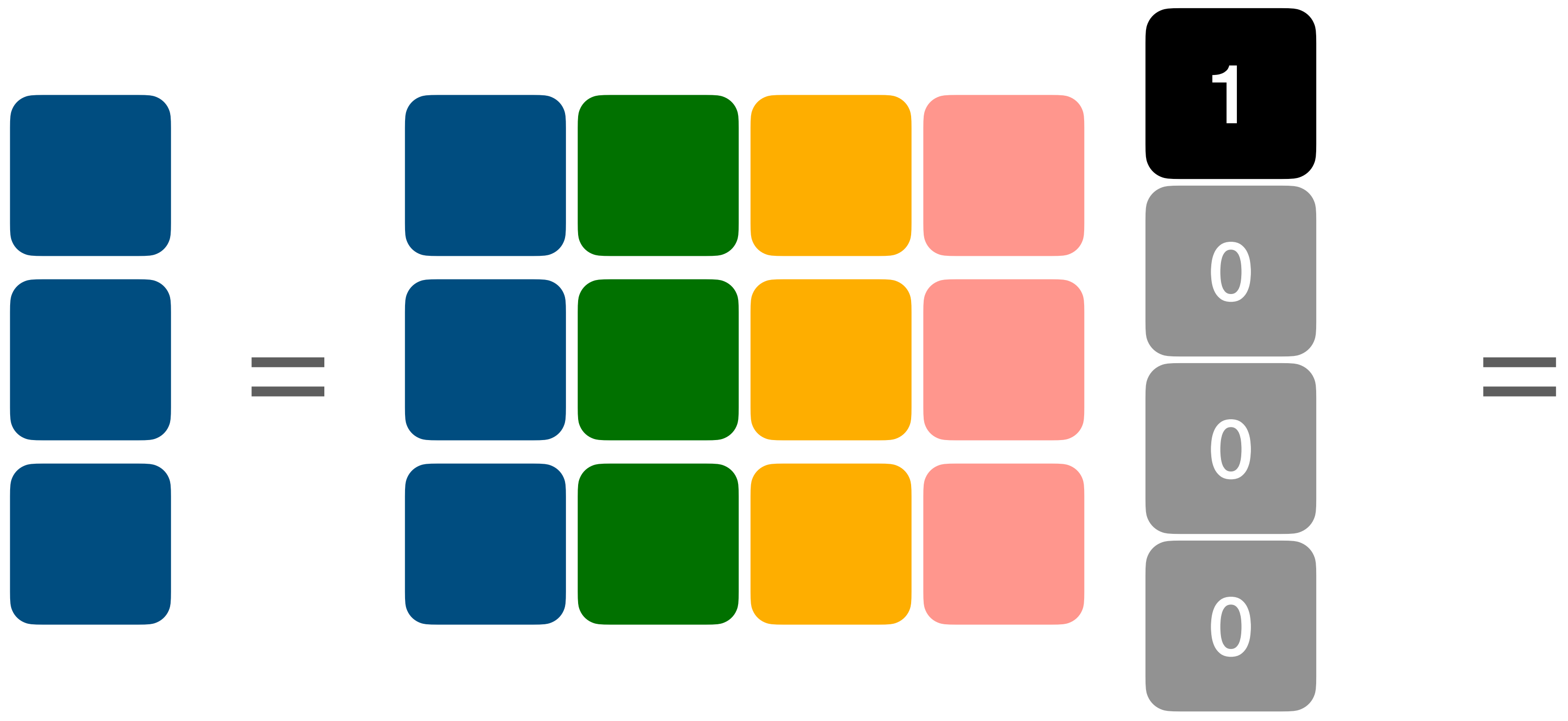
$c_i = Me_i$

$c_i = Me_i = M_3\boxed{M_2v_1}$

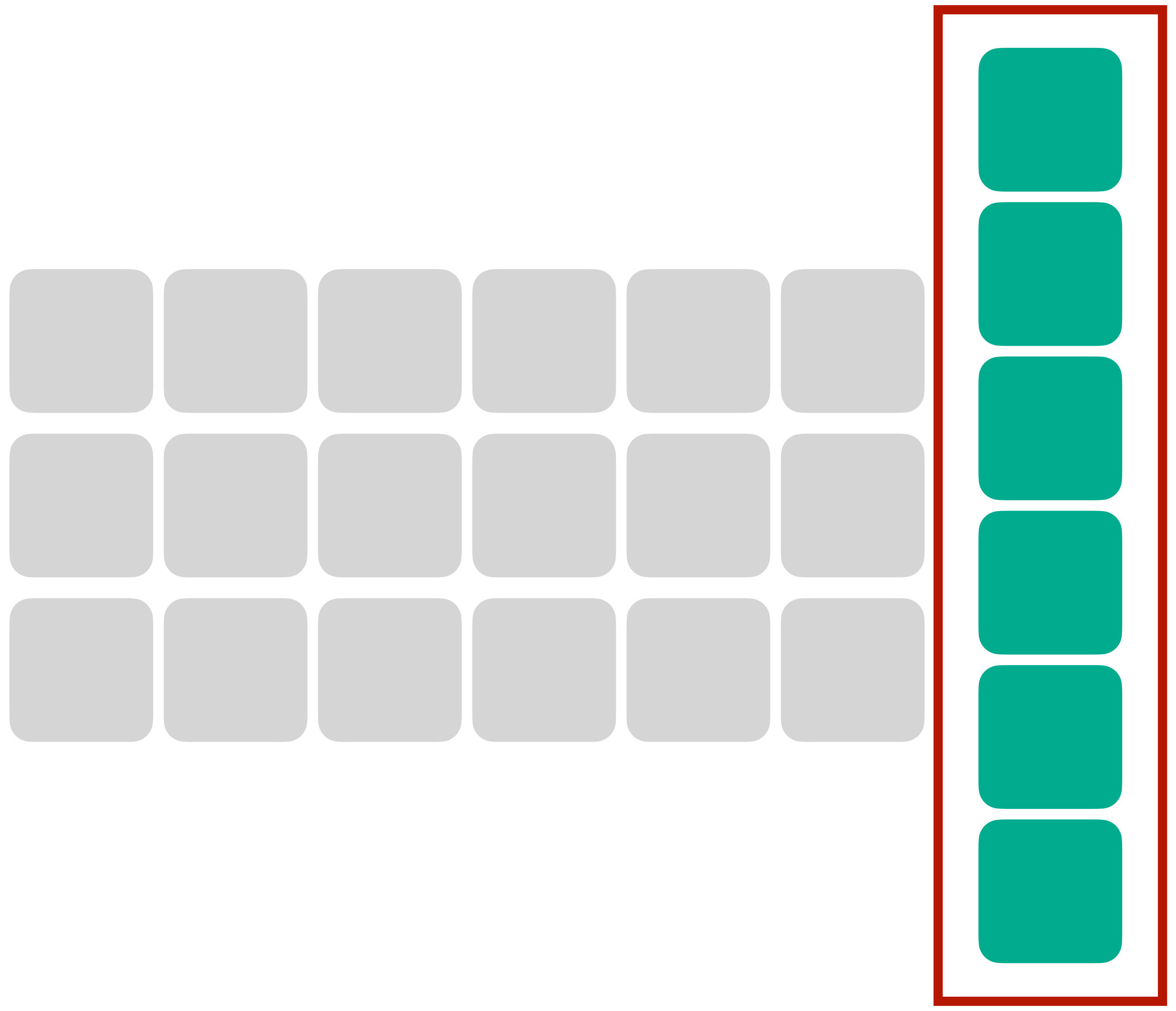
Composition of Matrices

The MVP, VMP picture still works if we have compositions

- to characterize $M = M_3 M_2 M_1$ we just need MVP/MVP with M_i



$c_i = Me_i$

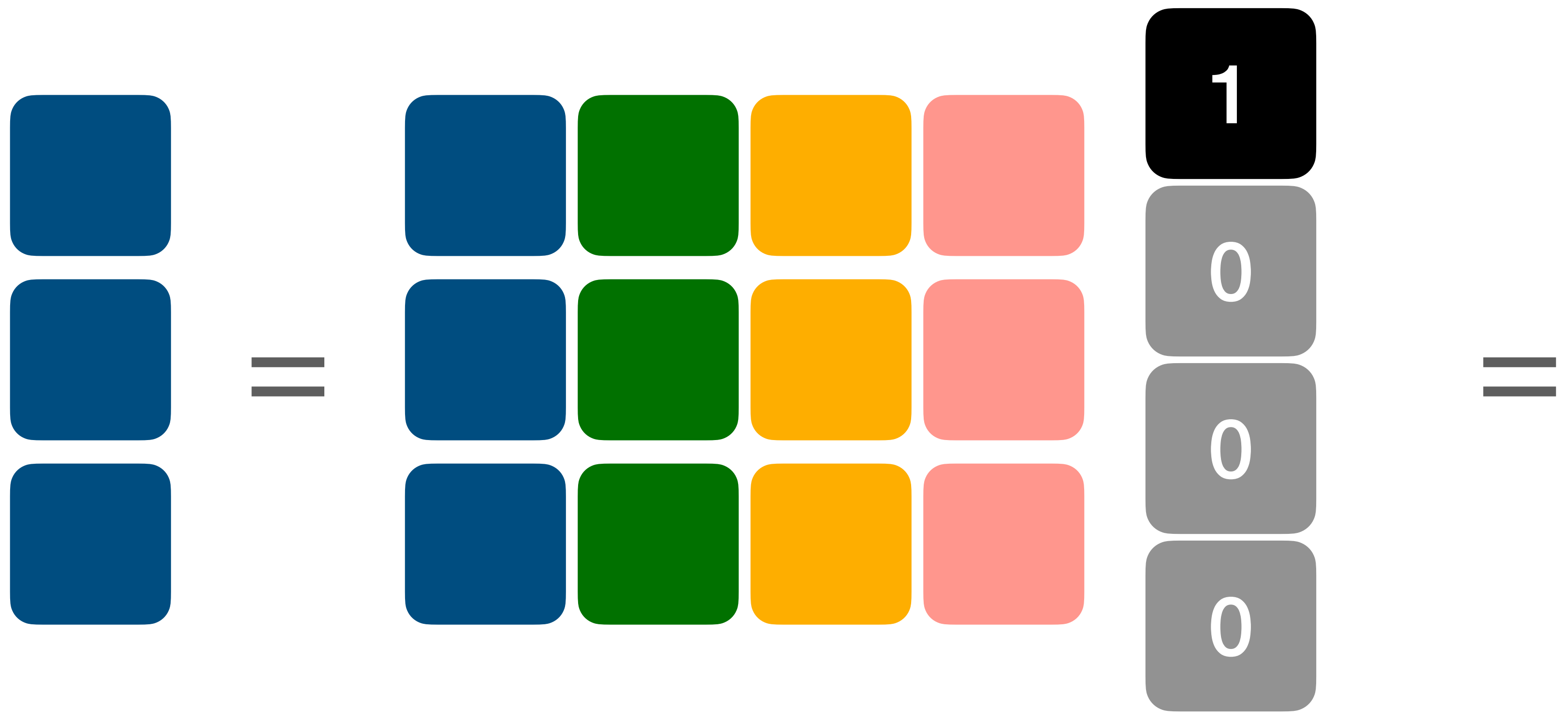


$c_i = Me_i = M_3 \boxed{v_2}$

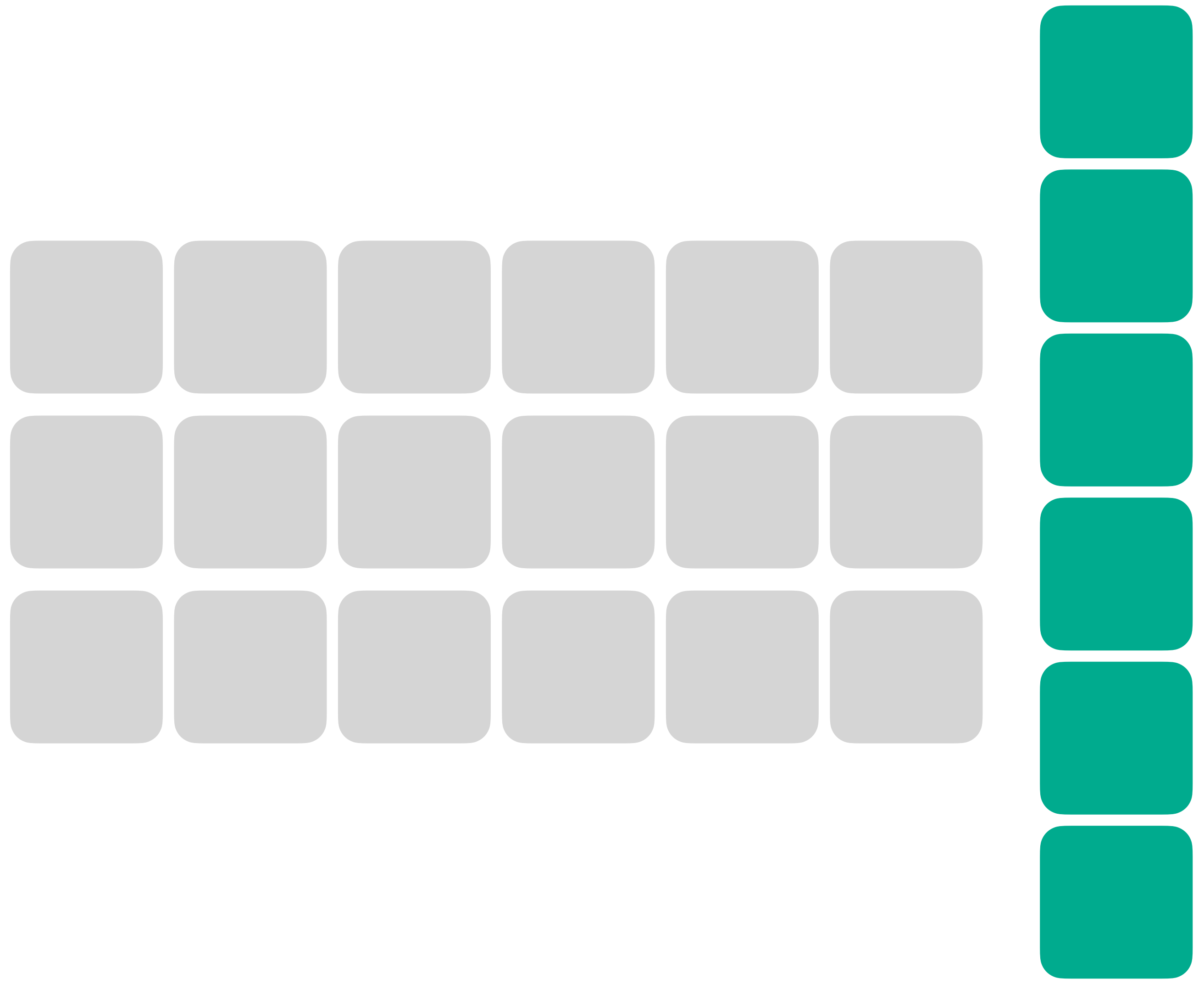
Composition of Matrices

The MVP, VMP picture still works if we have compositions

- to characterize $M = M_3 M_2 M_1$ we just need MVP/MVP with M_i



$c_i = Me_i$

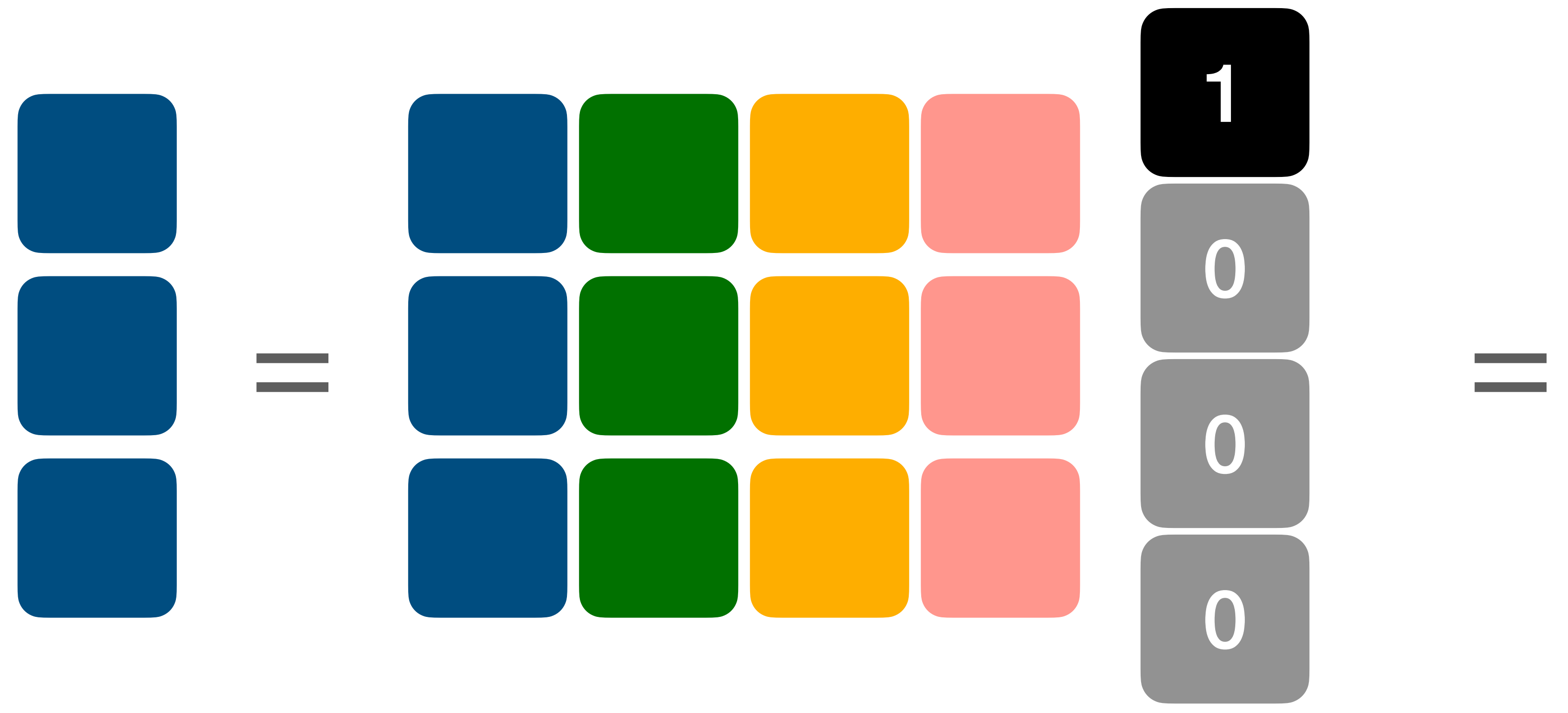


$c_i = Me_i = M_3 v_2$

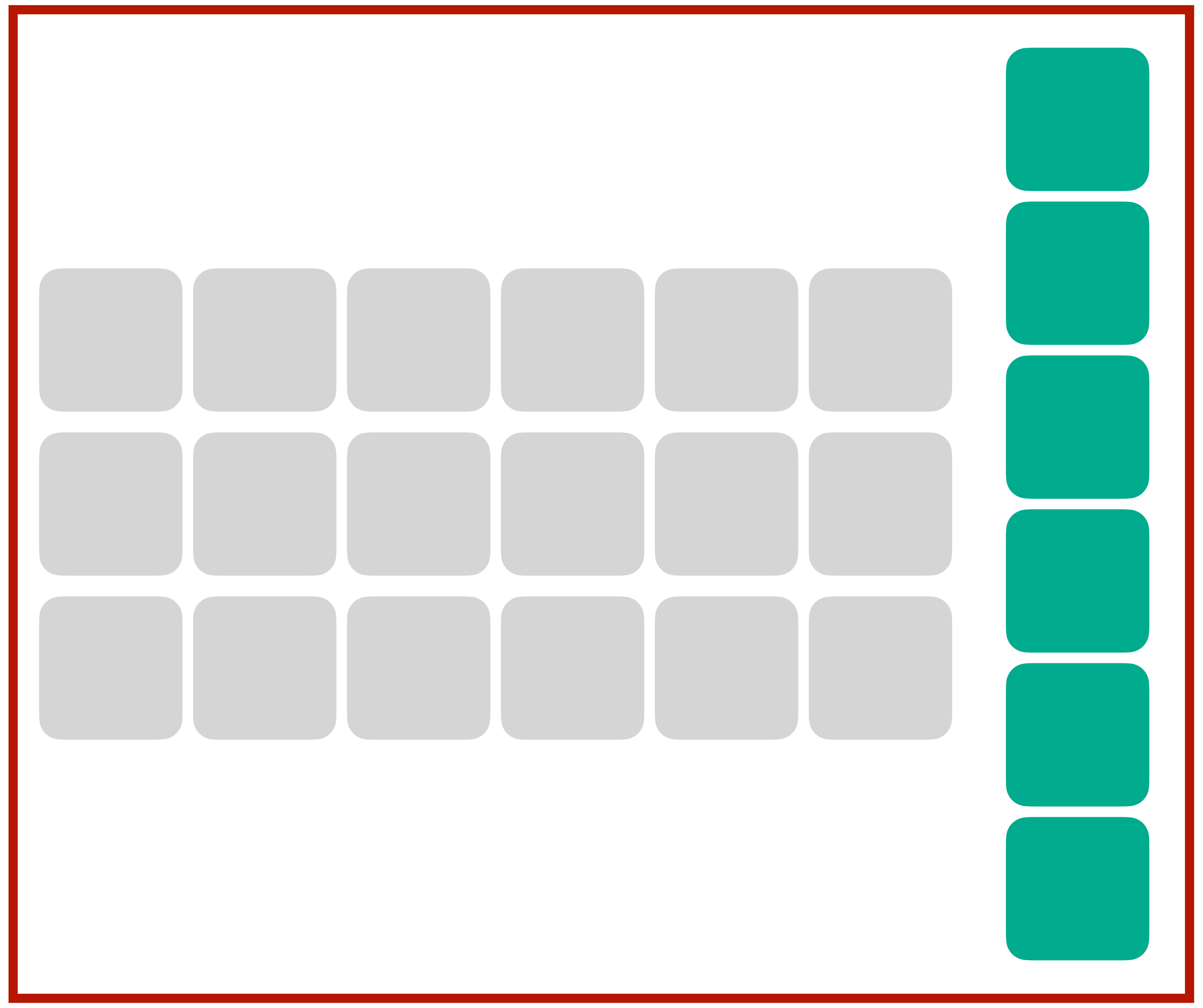
Composition of Matrices

The MVP, VMP picture still works if we have compositions

- to characterize $M = M_3M_2M_1$ we just need MVP/MVP with M_i



$c_i = Me_i$

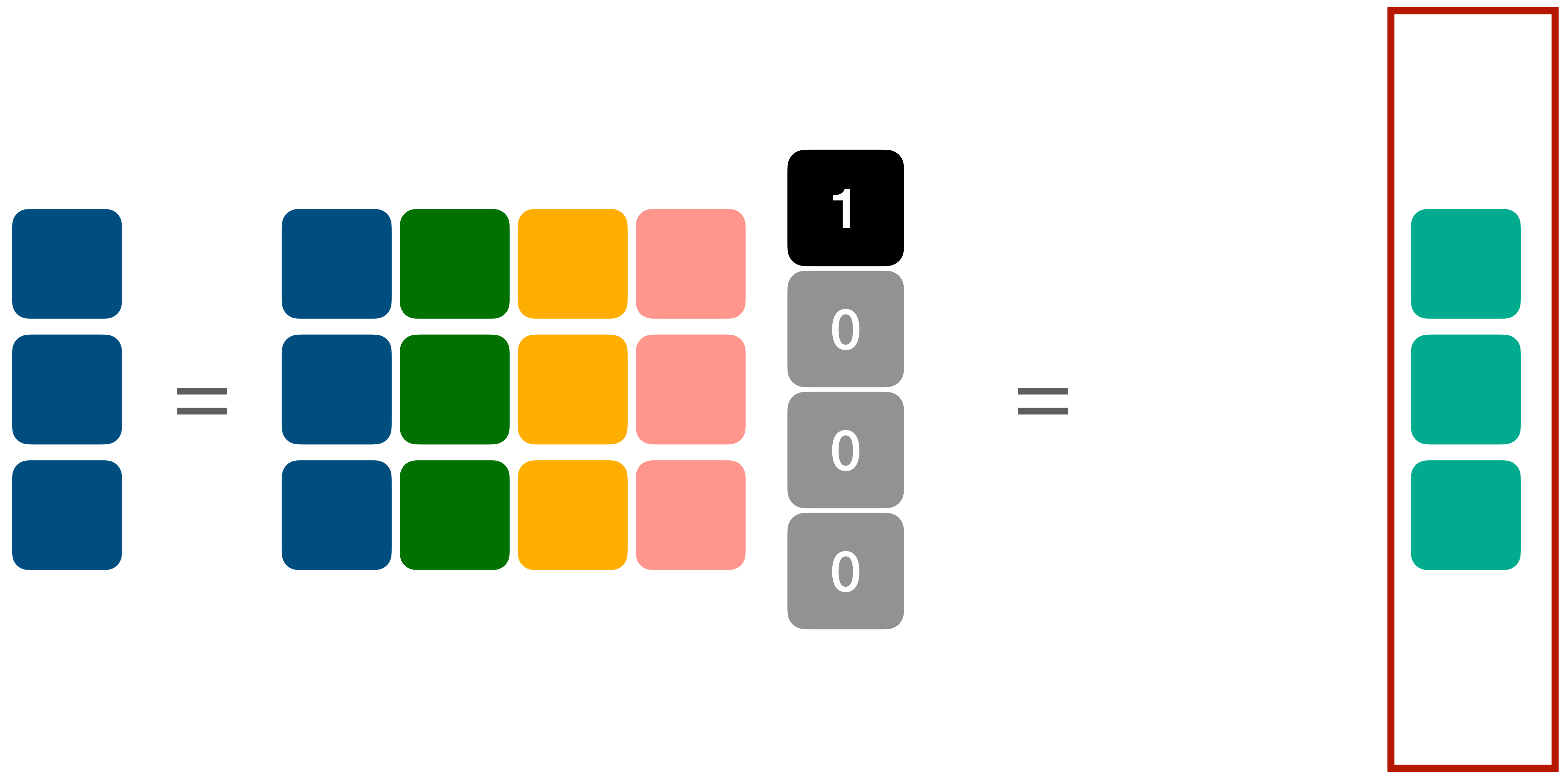


$c_i = Me_i = \boxed{M_3v_2}$

Composition of Matrices

The MVP, VMP picture still works if we have compositions

- to characterize $M = M_3 M_2 M_1$ we just need MVP/MVP with M_i



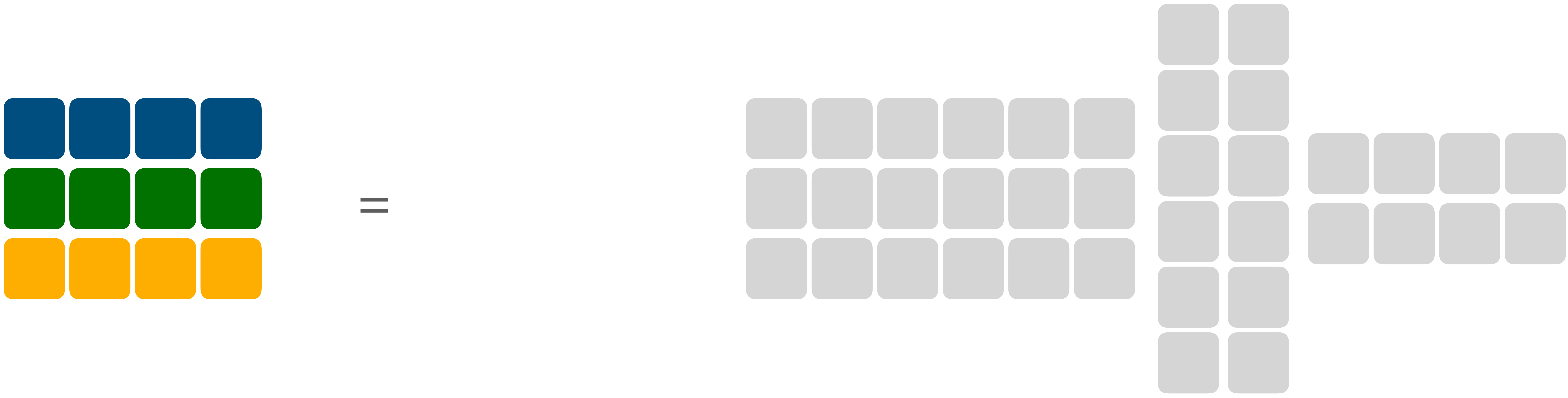
$c_i = Me_i$

$c_i = Me_i = \boxed{v_3}$

Composition of Matrices

The MVP, VMP picture still works if we have compositions

- to characterize $M = M_3 M_2 M_1$ we just need MVP/MVP with M_i



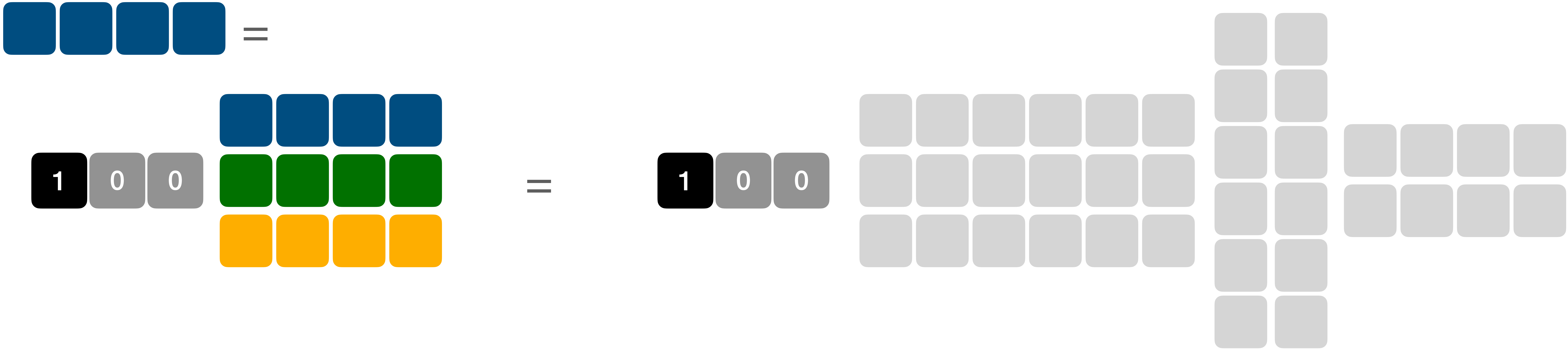
M

$M = M_1 M_2 M_3$

Composition of Matrices

The MVP, VMP picture still works if we have compositions

- to characterize $M = M_3 M_2 M_1$ we just need MVP/MVP with M_i



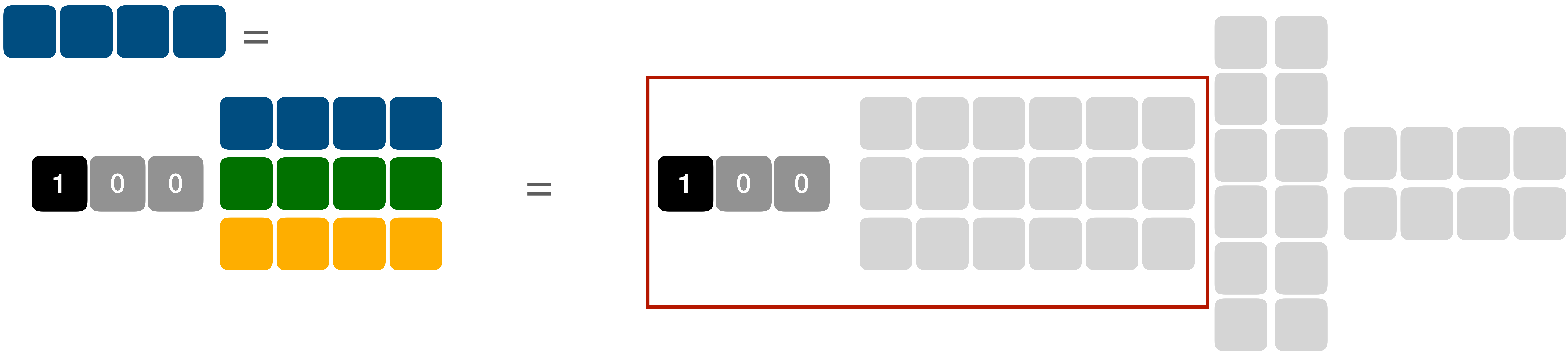
$r_i = e_i^T M$

$r_i = e_i^T M = e_i^T M_3 M_2 M_1$

Composition of Matrices

The MVP, VMP picture still works if we have compositions

- to characterize $M = M_3 M_2 M_1$ we just need MVP/MVP with M_i



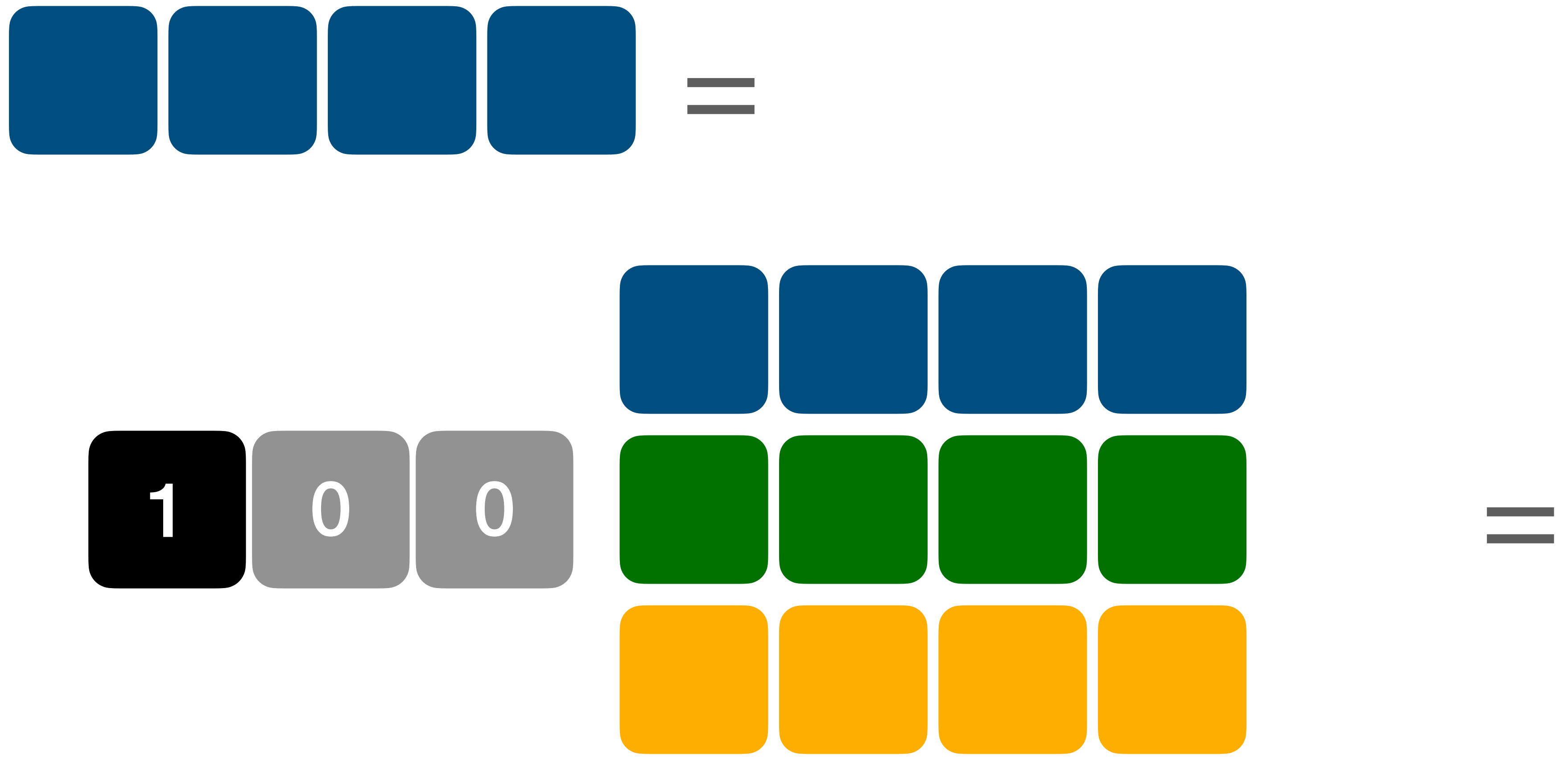
$$r_i = e_i^T M$$

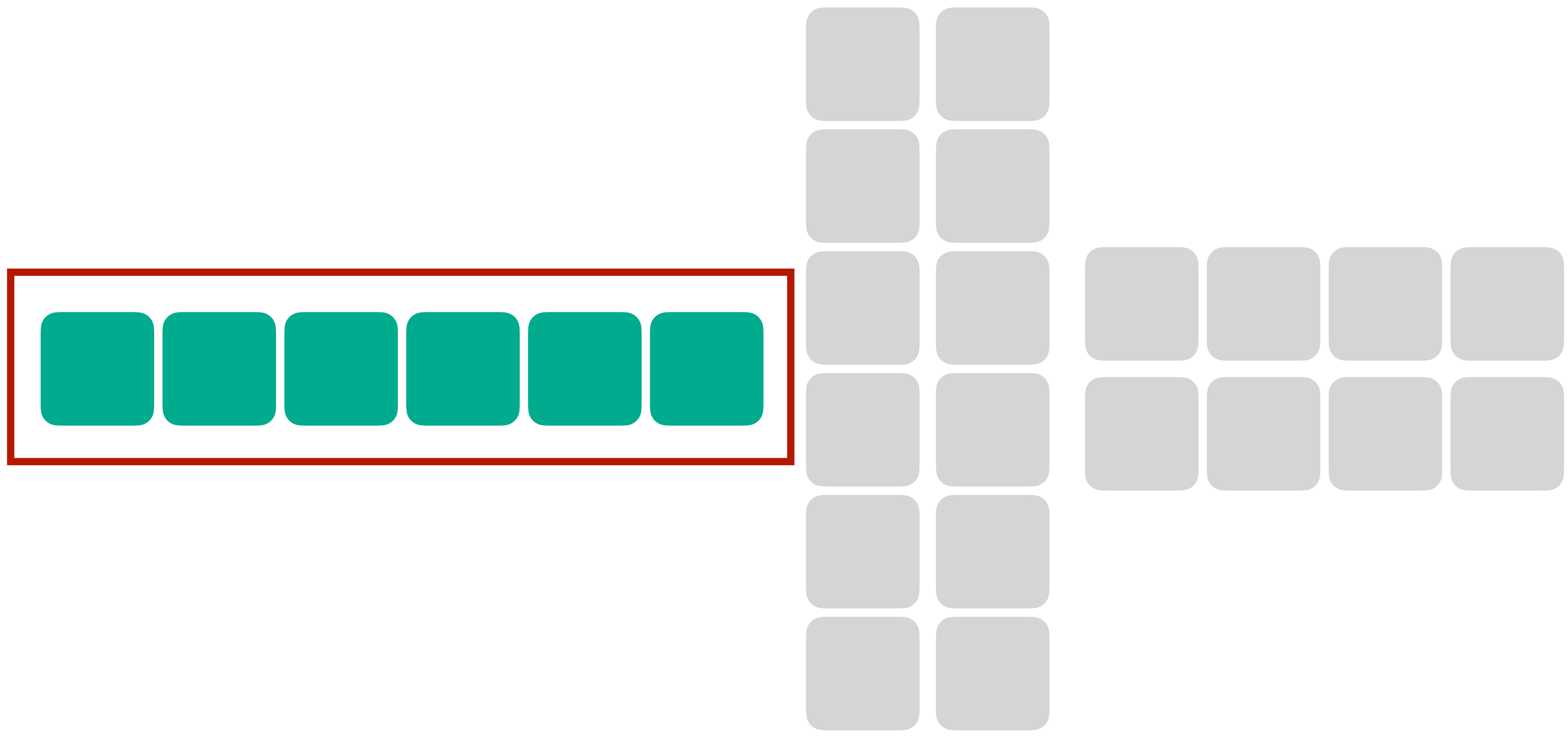
$$r_i = e_i^T M = \boxed{e_i^T M_3} M_2 M_1$$

Composition of Matrices

The MVP, VMP picture still works if we have compositions

- to characterize $M = M_3 M_2 M_1$ we just need MVP/MVP with M_i





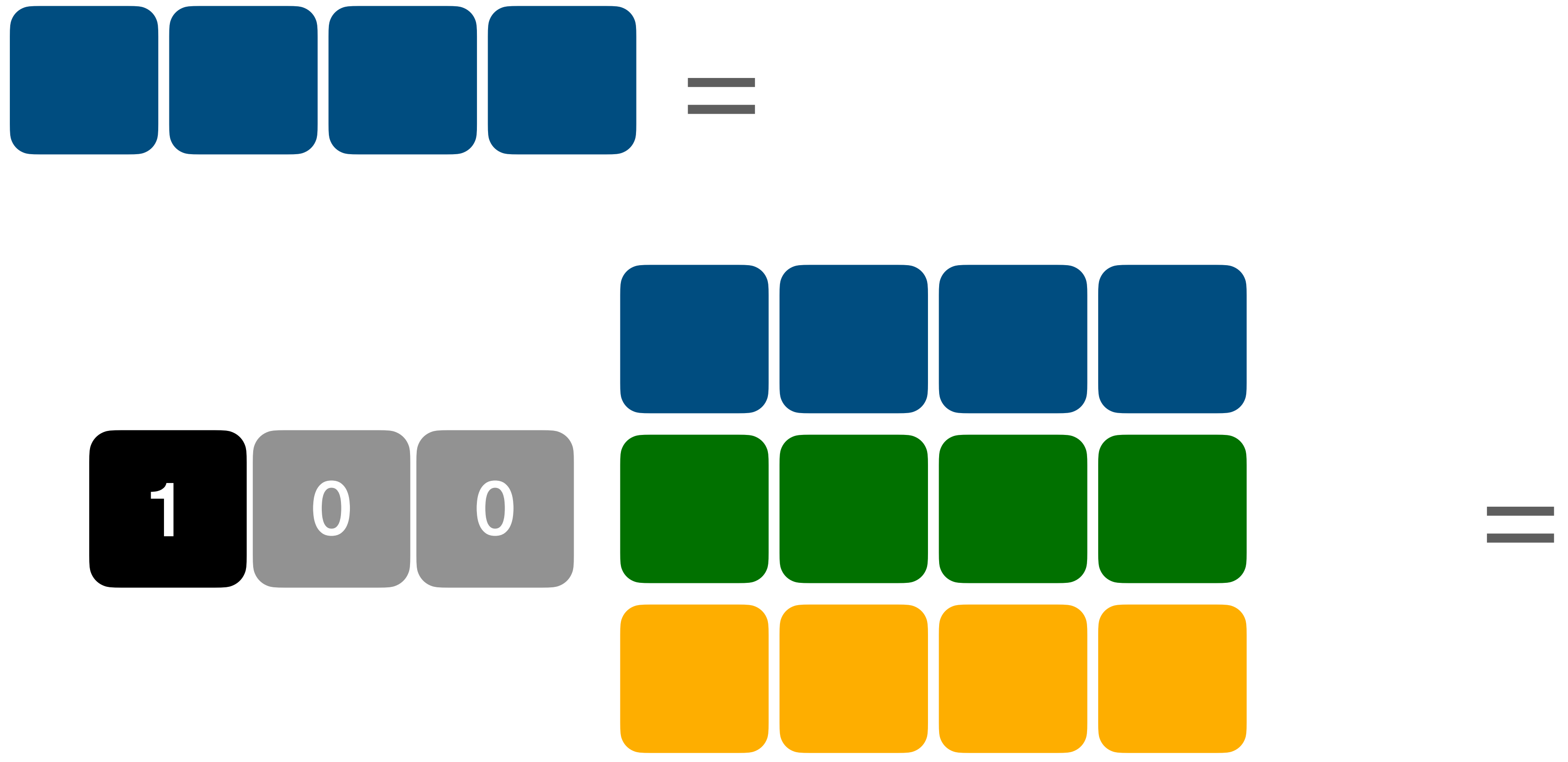
$r_i = e_i^T M$

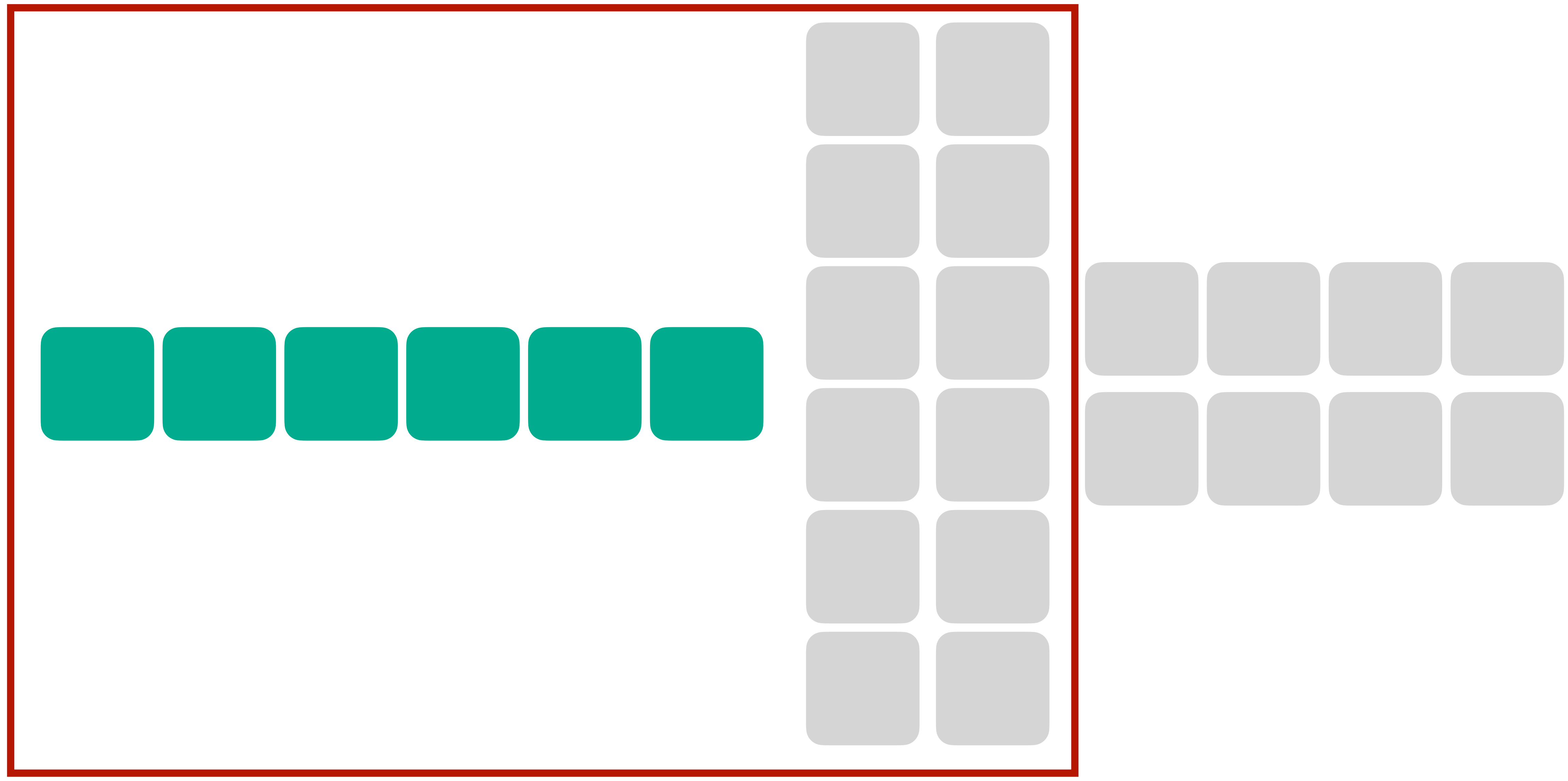
$r_i = e_i^T M = \boxed{\bar{v}_1} M_2 M_1$

Composition of Matrices

The MVP, VMP picture still works if we have compositions

- to characterize $M = M_3 M_2 M_1$ we just need MVP/MVP with M_i





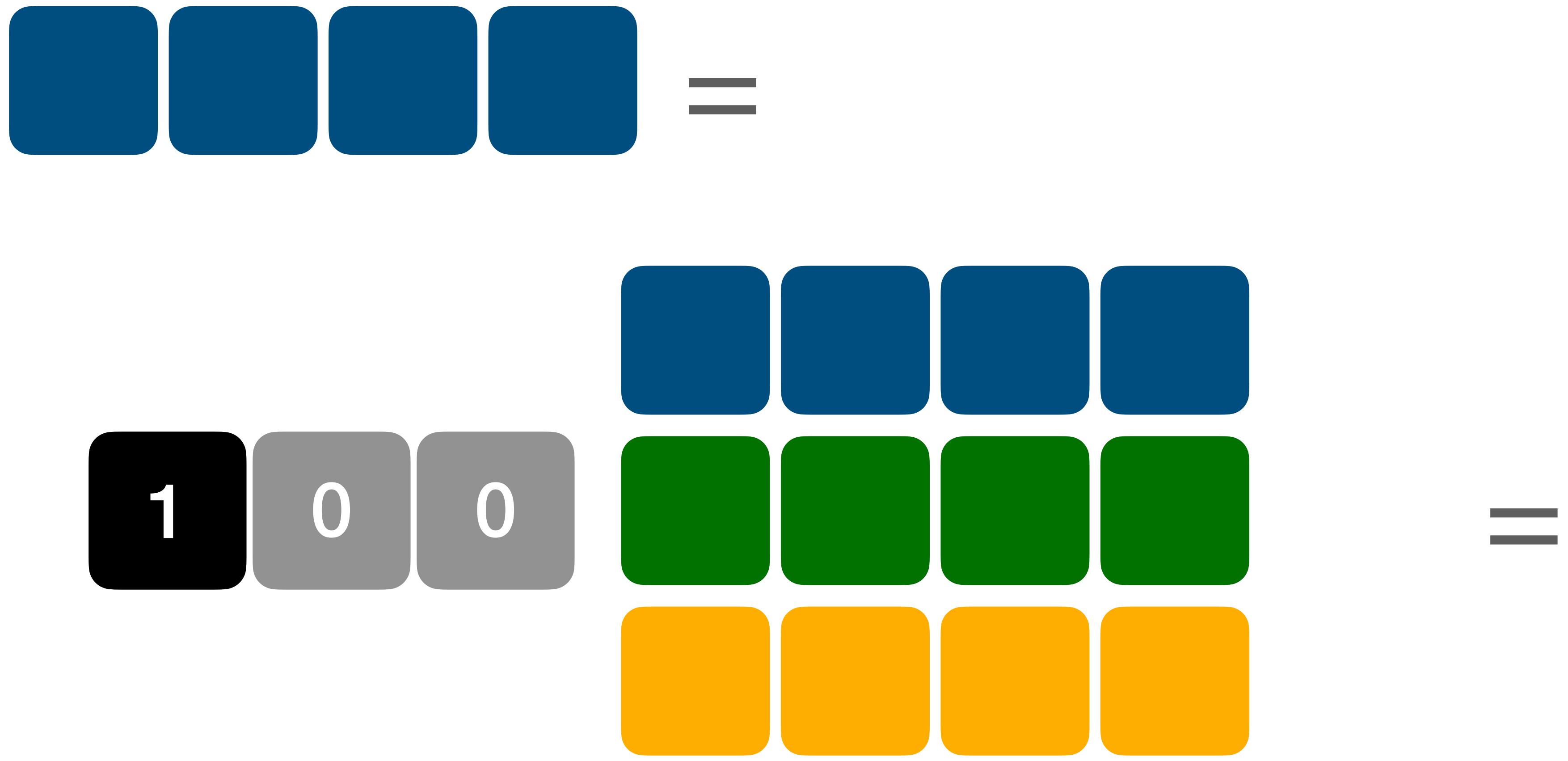
$$r_i = e_i^T M$$

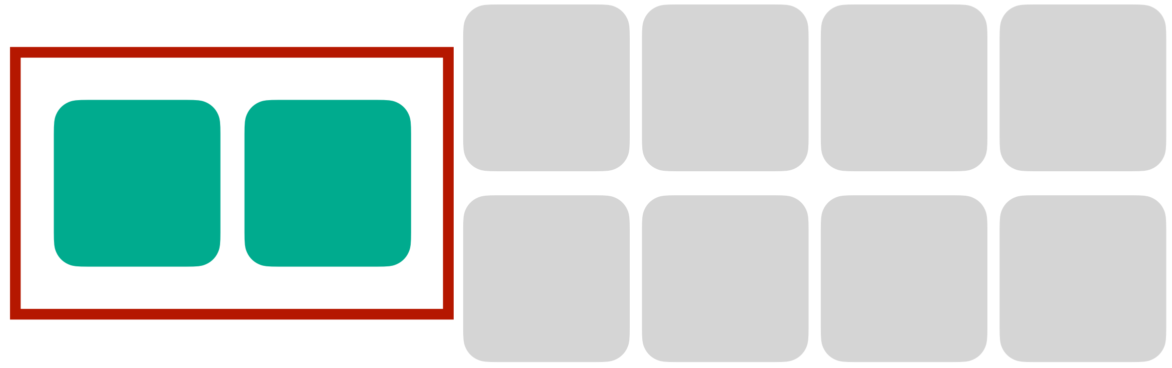
$$r_i = e_i^T M = \boxed{\bar{v}_1 M_2} M_1$$

Composition of Matrices

The MVP, VMP picture still works if we have compositions

- to characterize $M = M_3 M_2 M_1$ we just need MVP/MVP with M_i





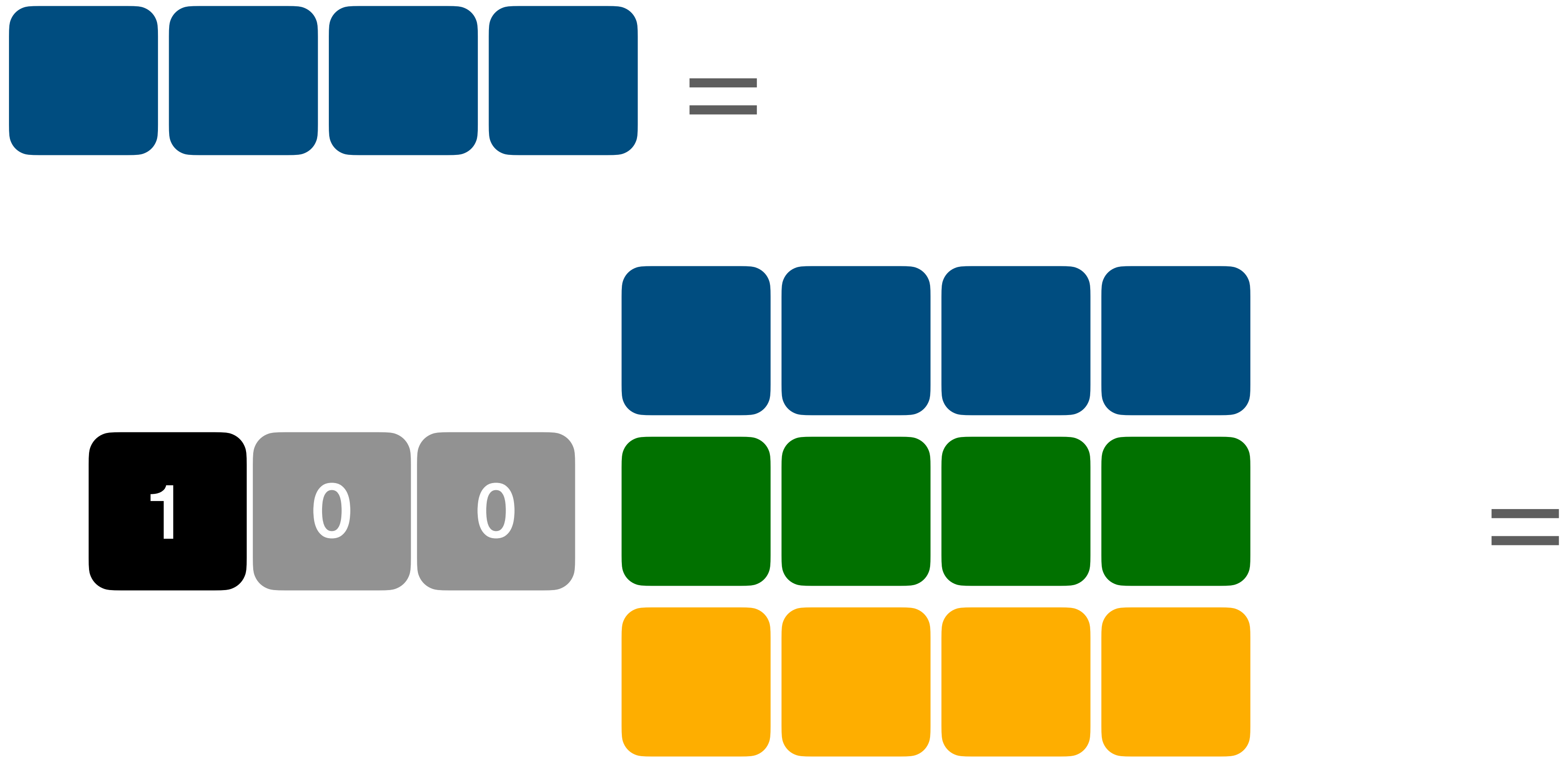
$r_i = e_i^T M$

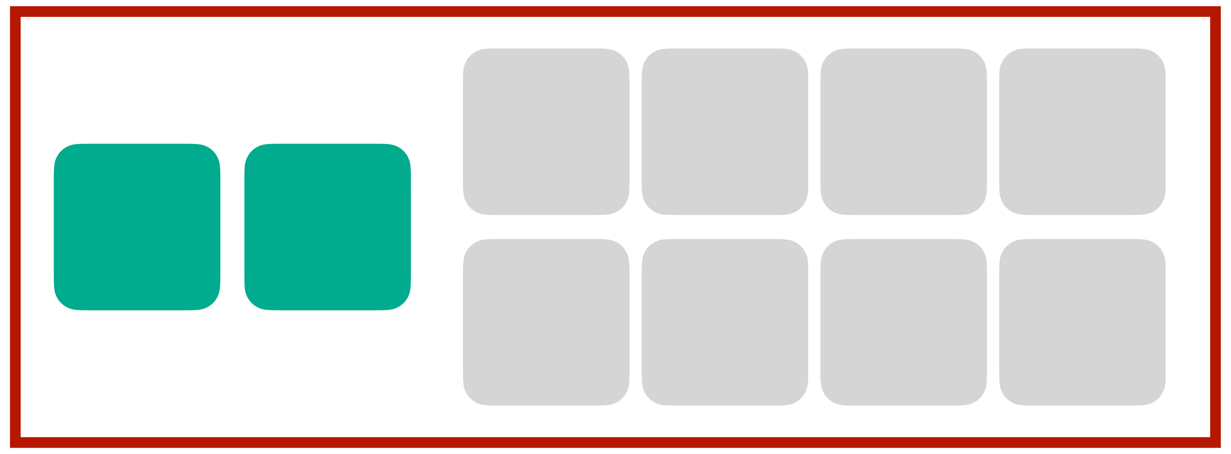
$r_i = e_i^T M = \boxed{\bar{v}_2} M_1$

Composition of Matrices

The MVP, VMP picture still works if we have compositions

- to characterize $M = M_3 M_2 M_1$ we just need MVP/MVP with M_i





$r_i = e_i^T M$

$r_i = e_i^T M = \boxed{\bar{v}_2 M_1}$

Composition of Matrices

The MVP, VMP picture still works if we have compositions

- to characterize $M = M_3 M_2 M_1$ we just need MVP/MVP with M_i

$r_i = e_i^T M$

$r_i = e_i^T M = \boxed{\bar{v}_3}$

Upshot: Forward and Backward

With MVPs/VMPs can characterize a Products of Matrices

- to get a row/column we never need explicit representations of M_i.
- programs for MVP/VMPs is all we need ("matrix-free" approach)

$$c_i = Me_i = M_3 M_2 M_1 e_i$$

$$c_i = Me_i = M_3 M_2 v_1$$

$$c_i = Me_i = M_3 v_2$$

$$c_i = Me_i = v_3$$

forward

$$r_i = e_i^T M = e_i^T M_3 M_2 M_1$$

$$r_i = e_i^T M = \bar{v}_1 M_2 M_1$$

$$r_i = e_i^T M = \bar{v}_2 M_1$$

$$r_i = e_i^T M = \bar{v}_3$$

backward (or reverse)

Back to Derivatives

A strategy for Derivatives of Programs

Our main job is to characterize the product of Jacobians:

$$\frac{\partial c_i}{\partial x_j} = J^{x \to y} = J^h J^g J^f$$

We know know that we compute it purely through programs that give us Jacobian Vector Products (JVP) or Vector-Jacobian products (VJP)

$$\mathrm{jvp}_f(x) = J_f x \qquad \mathrm{vjp}_f(x) = x^T J_f$$

A strategy for Derivatives of Programs

If we have all the $\mathrm{jvp}(x), \mathrm{vjp}(x)$ for all transforms h, g, f we can easily and mechanically create programs that compute derivatives

```
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```

A strategy for Derivatives of Programs

If we have all the $\mathrm{jvp}(x), \mathrm{vjp}(x)$ for all transforms h, g, f we can easily and mechanically create programs that compute derivatives

```python
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```

$$J^{x \to y} e_i = J^h J^g J^f e_i$$

```python
def jacobian_column(c0):
    c1 = jvp_f(c0)
    c2 = jvp_g(c1)
    c3 = jvp_h(c2)
    return c3
```

Forward-Mode Differentiation

A strategy for Derivatives of Programs

If we have all the $\mathrm{jvp}(x), \mathrm{vjp}(x)$ for all transforms h, g, f we can easily and mechanically create programs that compute derivatives

```
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```

$$e_i^T J^{x \to y} = e_i^T J^h J^g J^f$$

```
def jacobian_row(r0):
    r1 = vjp_h(r0)
    r2 = vjp_g(r1)
    r3 = vjp_f(r2)
    return r3
```

Reverse-Mode Differentiation

$$J^{x \to y} e_i = J^h J^g J^f e_i$$

```
def jacobian_column(c0):
    c1 = jvp_f(c0)
    c2 = jvp_g(c1)
    c3 = jvp_h(c2)
    return c3
```

Forward-Mode Differentiation

A strategy for Derivatives of Programs

For example, we can collect all the right programs while we are running through our main program as additional output

```
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```

Main Program

A strategy for Derivatives of Programs

For example, we can collect all the right programs while we are running through our main program as additional output

```python
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```

$\mathrm{jvp}_f(x)$

Main Program

A strategy for Derivatives of Programs

For example, we can collect all the right programs while we are running through our main program as additional output

```python
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```

$\mathrm{jvp}_f(x)$

$\mathrm{jvp}_g(x)$

Main Program

A strategy for Derivatives of Programs

For example, we can collect all the right programs while we are running through our main program as additional output

```
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```

$\mathrm{jvp}_f(x)$

$\mathrm{jvp}_g(x)$

$\mathrm{jvp}_h(x)$

Main Program

A strategy for Derivatives of Programs

For example, we can collect all the right programs while we are running through our main program as additional output

... and from those assemble the derivative program

```python
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```

$\mathrm{jvp}_f(x)$

$\mathrm{jvp}_g(x)$

$\mathrm{jvp}_h(x)$

```python
def jacobian_column(c0):
    c1 = jvp_f(c0)
    c2 = jvp_g(c1)
    c3 = jvp_h(c2)
    return c3
```

Main Program

Derivative Program

A strategy for Derivatives of Programs

Or we can go backwards as well...

```
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```

Main Program

A strategy for Derivatives of Programs

Or we can go backwards as well…

```
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```

$\mathrm{vjp}_f(x)$

Main Program

A strategy for Derivatives of Programs

Or we can go backwards as well…

```
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```

$\mathrm{vjp}_f(x)$

$\mathrm{vjp}_g(x)$

Main Program

A strategy for Derivatives of Programs

Or we can go backwards as well…

```
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```

$\mathrm{vjp}_f(x)$

$\mathrm{vjp}_g(x)$

$\mathrm{vjp}_h(x)$

Main Program

A strategy for Derivatives of Programs

Or we can go backwards as well…

```python
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```

$\mathrm{vjp}_f(x)$

$\mathrm{vjp}_g(x)$

$\mathrm{vjp}_h(x)$

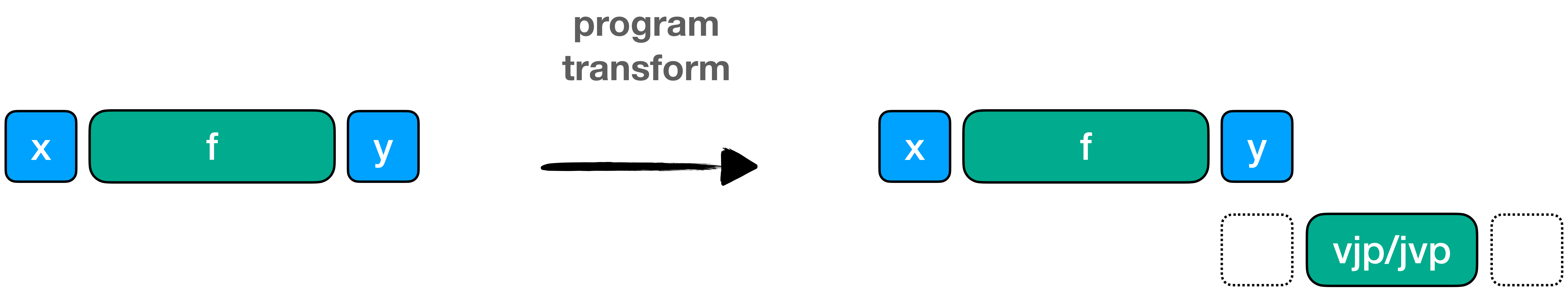
```python
def jacobian_row(r0):
    r1 = vjp_h(r0)
    r2 = vjp_g(r1)
    r3 = vjp_f(r2)
    return r3
```

Main Program

Derivative Program

A strategy for Derivatives of Programs

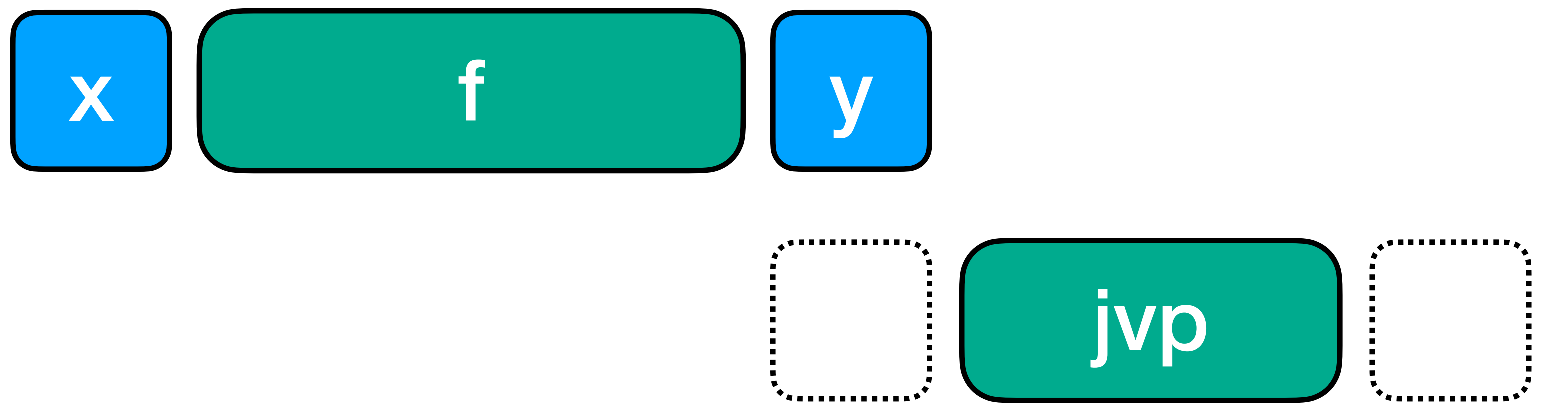
That means, if we have a system that can modify a function such that it returns not only the output value but also a vjp/jvp function



Then, we can almost blindly assemble a corresponding derivative program!

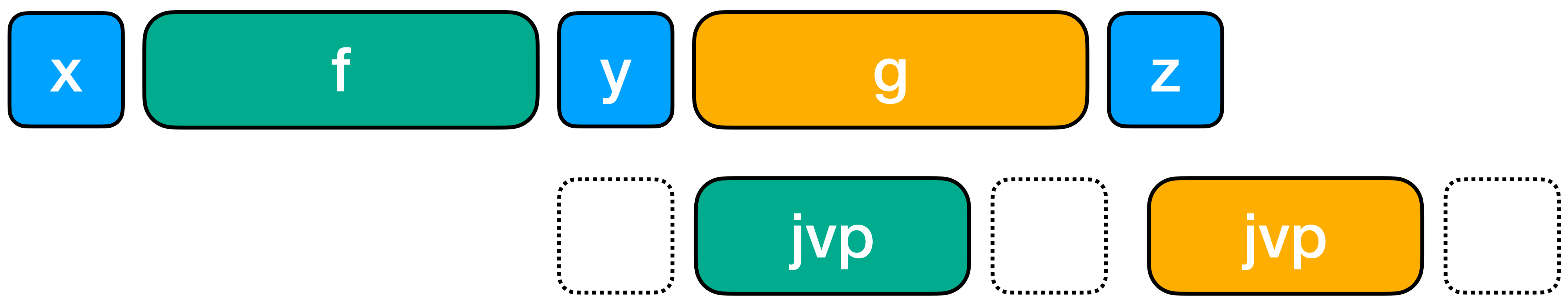
A strategy for Derivatives of Programs

That means, if we have a system that can modify a function such that it returns not only the output value but also a vjp/jvp function



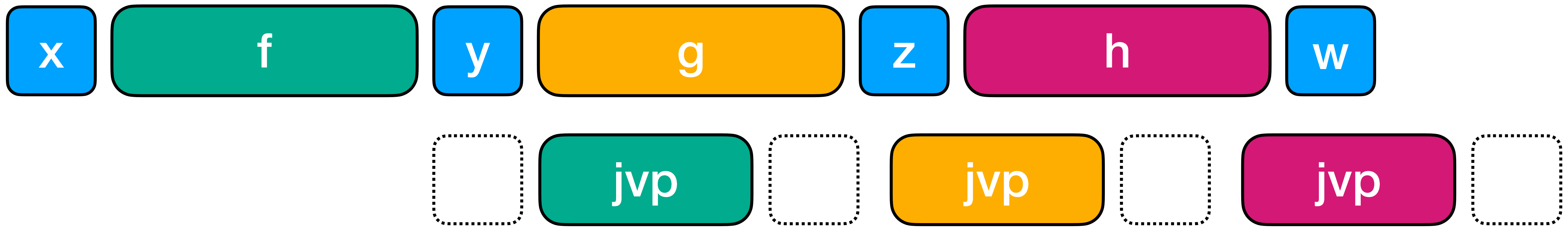
A strategy for Derivatives of Programs

That means, if we have a system that can modify a function such that it returns not only the output value but also a vjp/jvp function



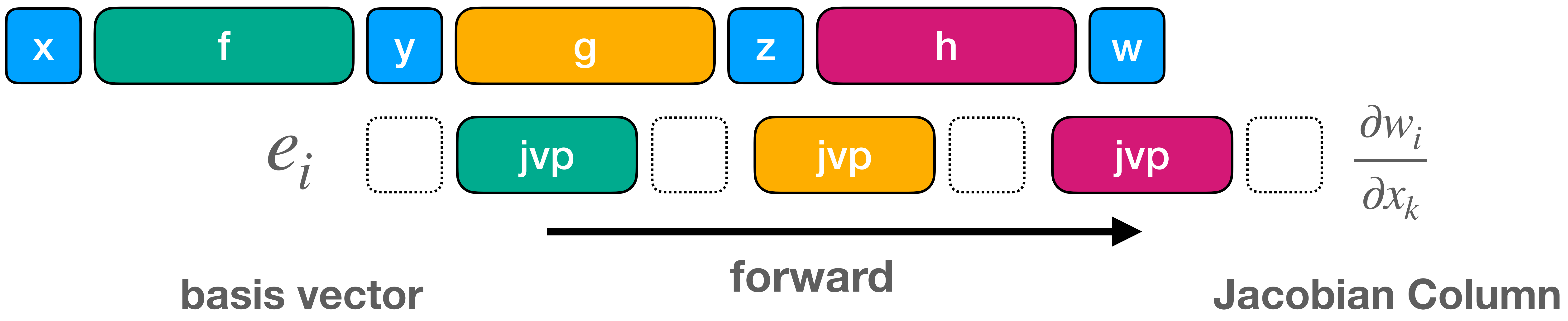
A strategy for Derivatives of Programs

That means, if we have a system that can modify a function such that it returns not only the output value but also a vjp/jvp function



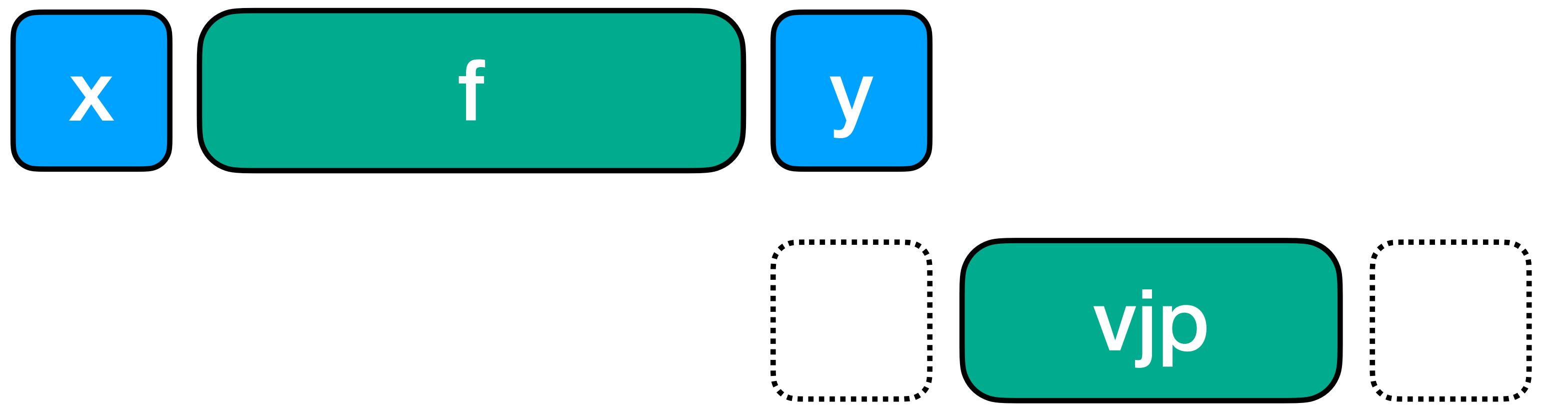
A strategy for Derivatives of Programs

That means, if we have a system that can modify a function such that it returns not only the output value but also a vjp/jvp function



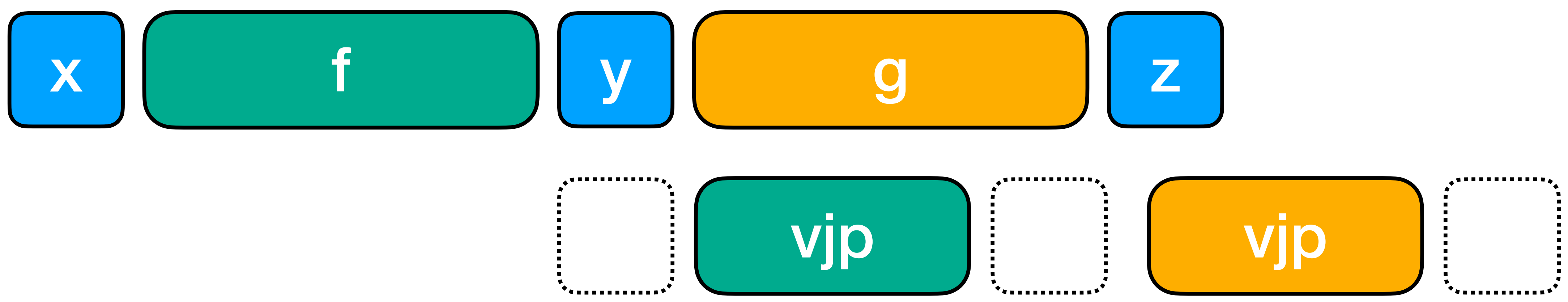
A strategy for Derivatives of Programs

That means, if we have a system that can modify a function such that it returns not only the output value but also a vjp/jvp function



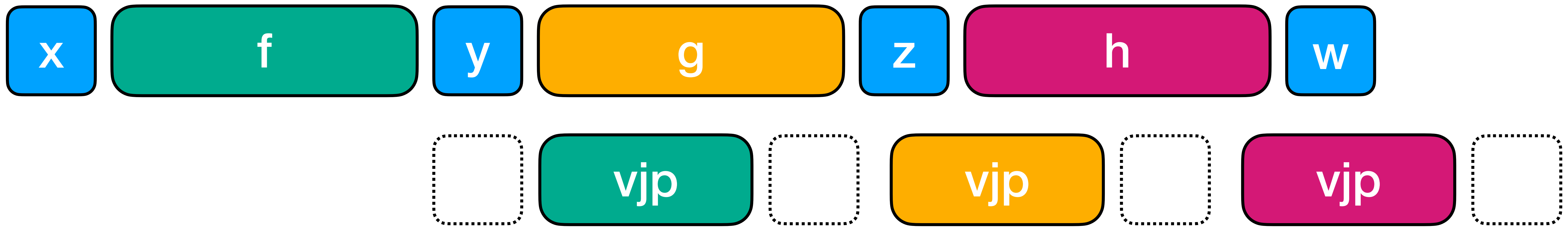
A strategy for Derivatives of Programs

That means, if we have a system that can modify a function such that it returns not only the output value but also a vjp/jvp function



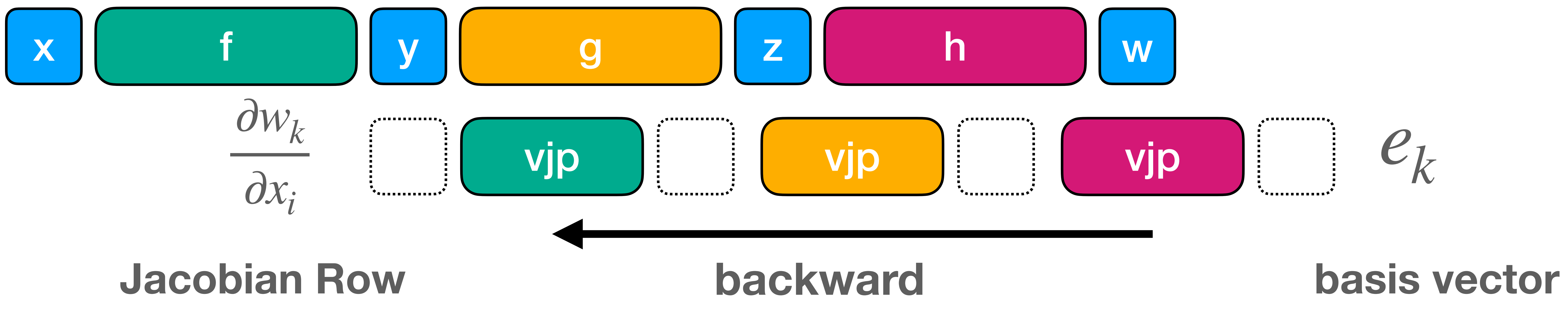
A strategy for Derivatives of Programs

That means, if we have a system that can modify a function such that it returns not only the output value but also a vjp/jvp function



A strategy for Derivatives of Programs

That means, if we have a system that can modify a function such that it returns not only the output value but also a vjp/jvp function



Forward vs Backward

We have two methods to derive gradients, which one should we use?

For ML, what's $f : \mathbb{R}^n \to \mathbb{R}^m$?

It's our empirical risk as function of parameters, i.e.

$$L(\phi) : \mathbb{R}^{\text{a lot}} \to \mathbb{R}^1$$

Forward vs Backward

Given $L(\phi) : \mathbb{R}^{\text{a lot}} \rightarrow \mathbb{R}^1$, what's the shape of the Jacobian?

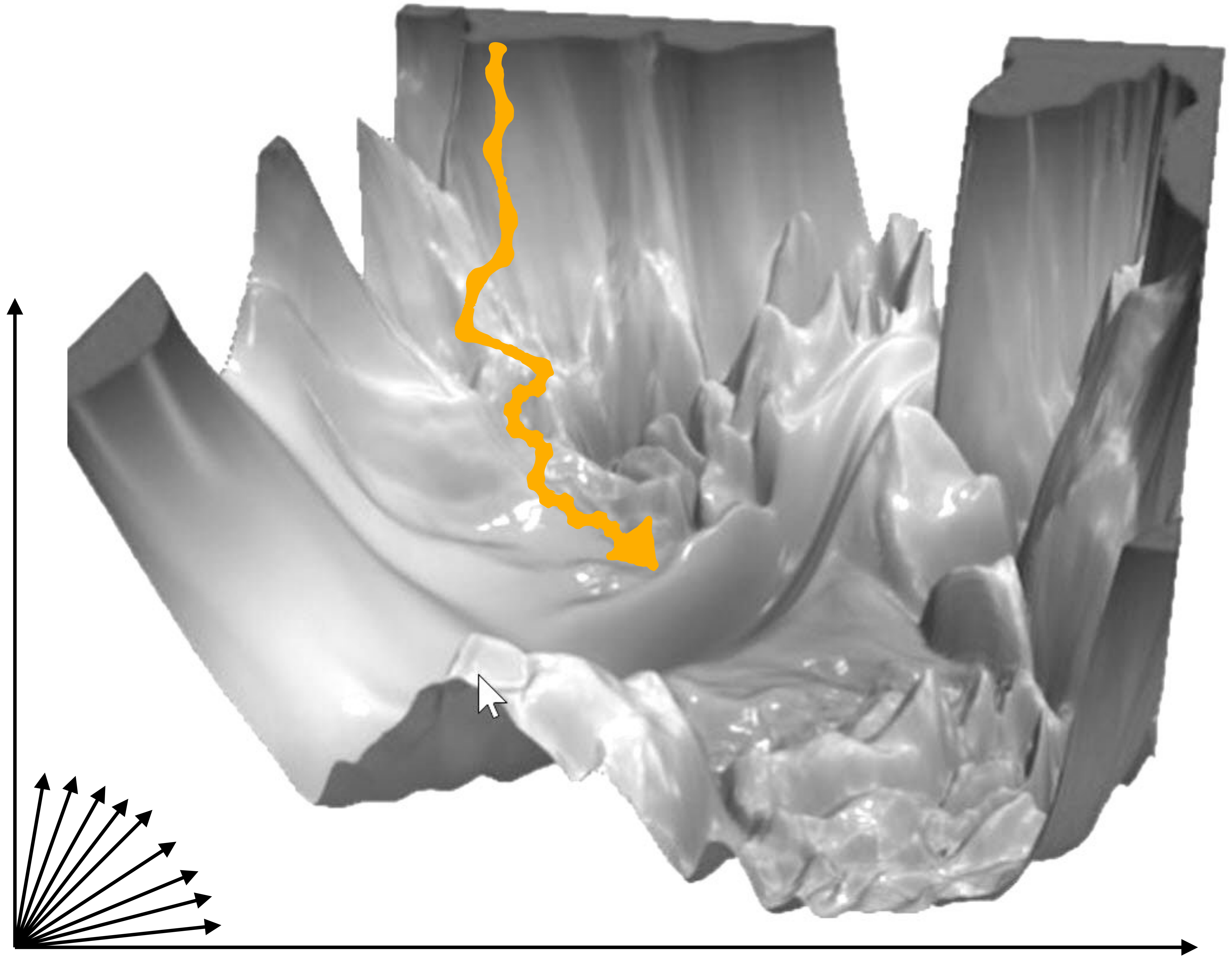
A single row

Can compute it with a single pass of reverse-mode differentiation

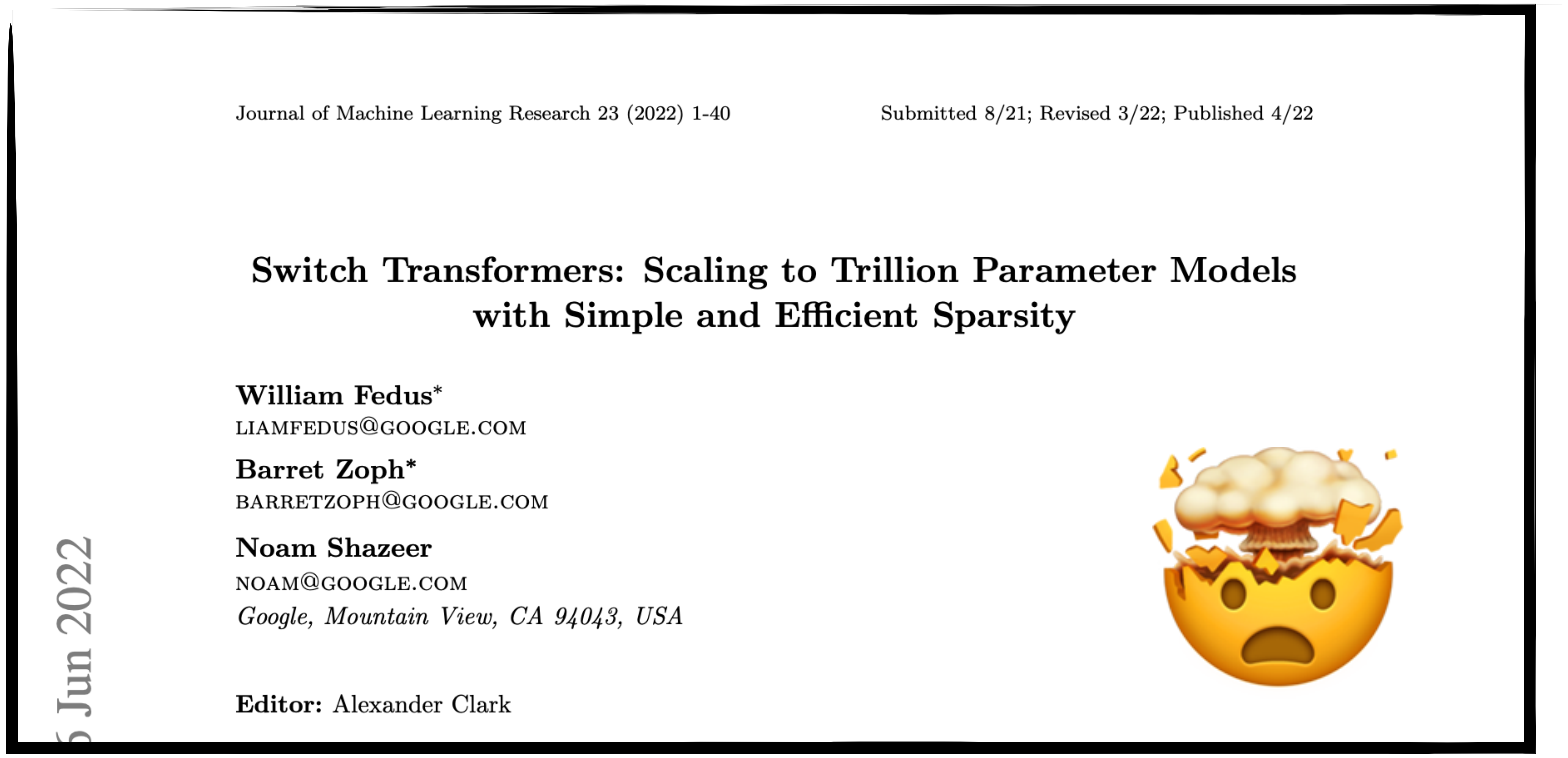
Scaling to Billions of Parameters

By using reverse-mode, we can scale to billions of dimensions

- gradient computation requires roughly same time as main program



To deal with hyper-planes in a 14-dimensional space, visualize a 3D space and say 'fourteen' to yourself very loudly. -Hinton



Journal of Machine Learning Research 23 (2022) 1-40 Submitted 8/21; Revised 3/22; Published 4/22

Switch Transformers: Scaling to Trillion Parameter Models with Simple and Efficient Sparsity

William Fedus*
LIAMFEDUS@GOOGLE.COM
Barret Zoph*
BARRETZOPH@GOOGLE.COM
Noam Shazeer
NOAM@GOOGLE.COM
Google, Mountain View, CA 94043, USA

Editor: Alexander Clark

Gives us a good sense of direction in billion-D spaces

Forward vs Backward

For Machine Learning, mostly the reverse-mode differentiation via vector-Jacobian products is relevant. Rediscovered by ML in 80s.

Backpropagation

1970

Seppo Linnainmaa

Seppo Linnainmaa

ALGORITMIN KUMULATIIVINEN PYÖRISTYSVIRHE

YKSITTÄISTEN PYÖRISTYSVIRHEIDEN TAYLOR-KEHITELMÄNÄ

Pro gradu-tutkielma • ohjaaja professori M.Tienari

1986

Published: 09 October 1986

Geoff Hinton

Learning representations by back-propagating errors

David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams

Nature **323**, 533–536 (1986) | Cite this article

95k Accesses | **13696** Citations | **255** Altmetric | Metrics

Abstract

We describe a new learning procedure, back-propagation, for networks of neurone units. The procedure repeatedly adjusts the weights of the connections in the netw as to minimize a measure of the difference between the actual output vector of the the desired output vector. As a result of the weight adjustments, internal 'hidden' u which are not part of the input or output come to represent important features of t domain, and the regularities in the task are captured by the interactions of these units. The ability to create useful new features distinguishes back-propagation from earlier, simpler methods such as the perceptron-convergence procedure[1].

Ok, but…

Beyond Sequences

You might ask: *"My programs look a bit more complicated than a sequence of function calls":* control flow, loops, ..

```python
def program(x):
    a = f(x)
    b = g(a)
    c = h(b)
    return c
```

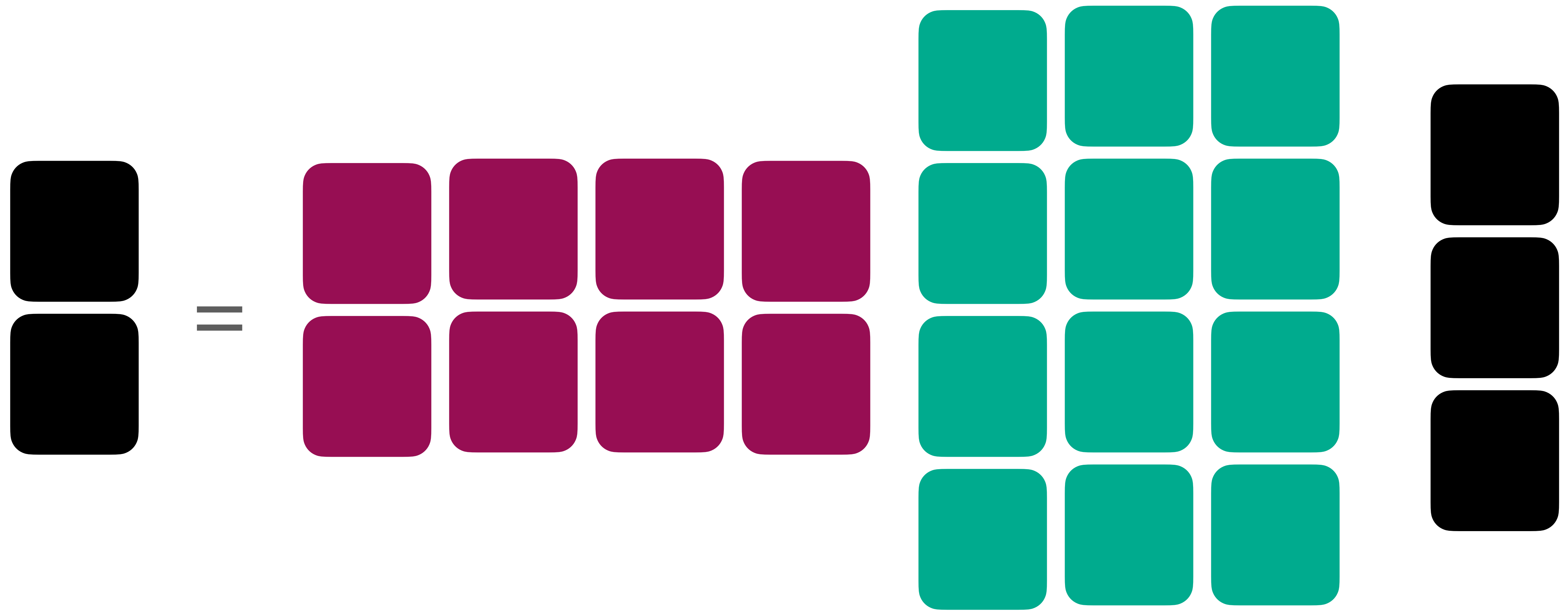
We know how to do this

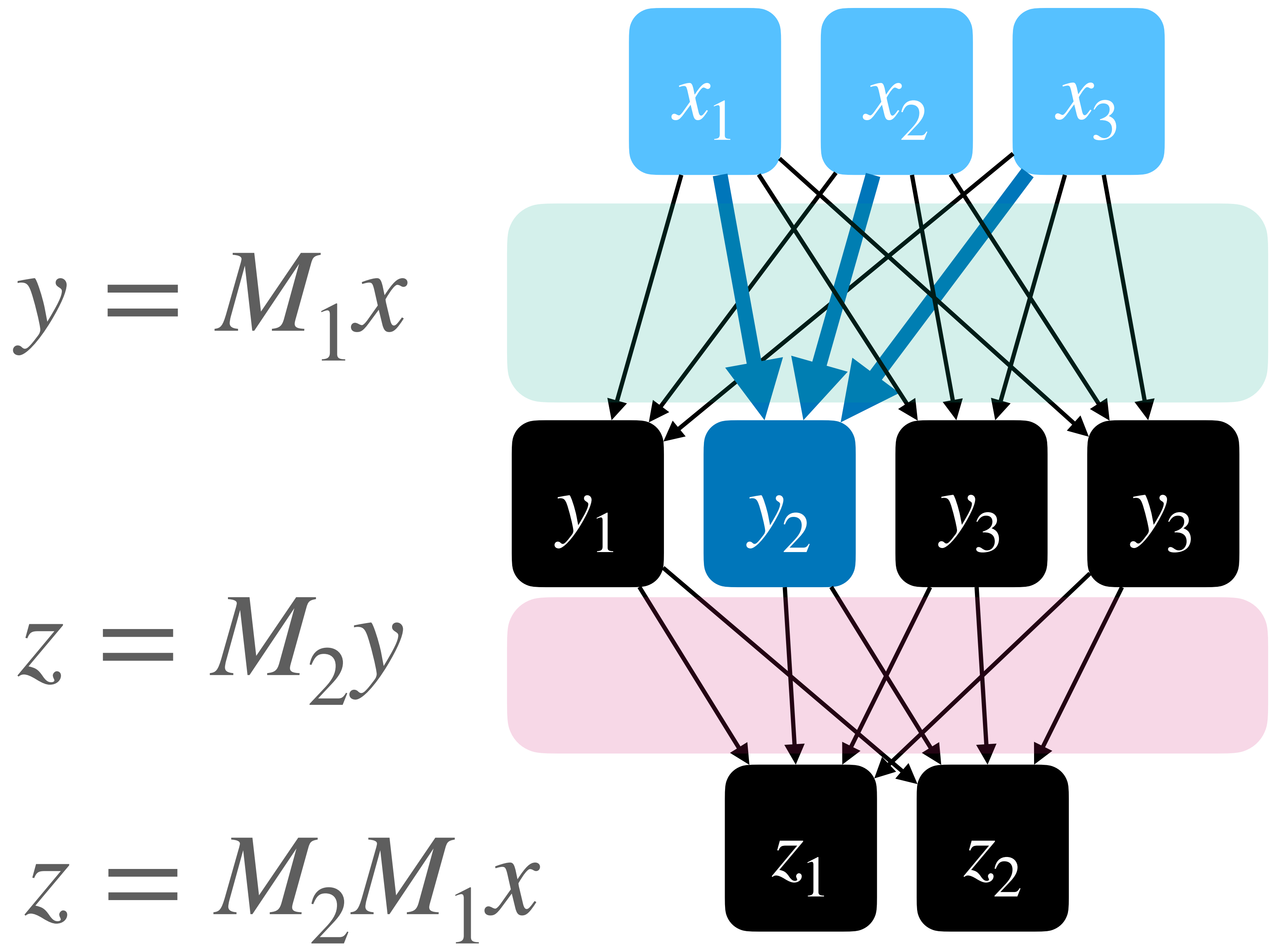
```python
def program(x):
    v=x
    for i in range(3)
        v=4*v*(1-v)
    return v
```

How do we do this?

A Graph Picture of Matrix Multiplication

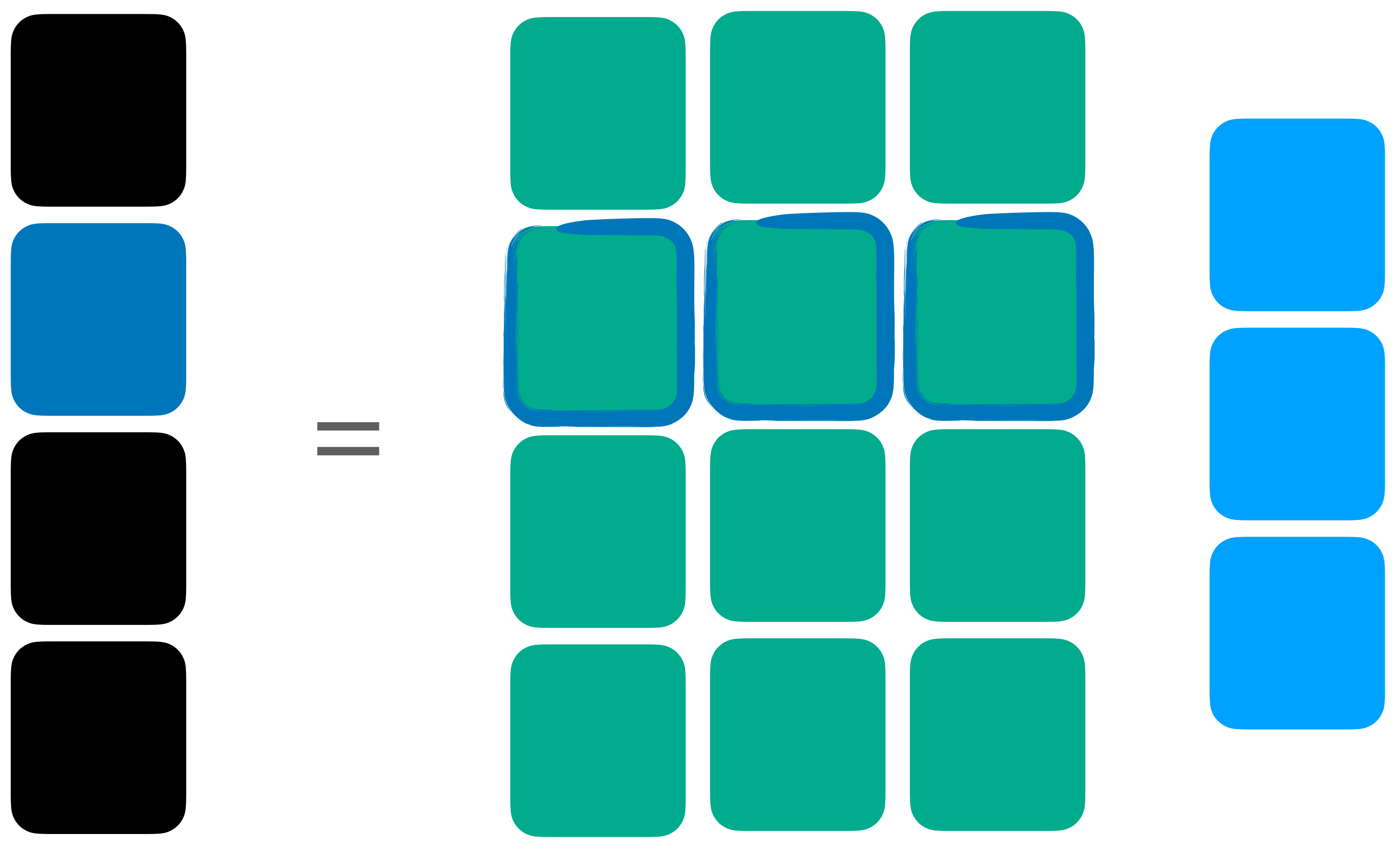
We can get a hint, by looking at Matrix Multiplication as a graph





A Graph Picture of Matrix Multiplication

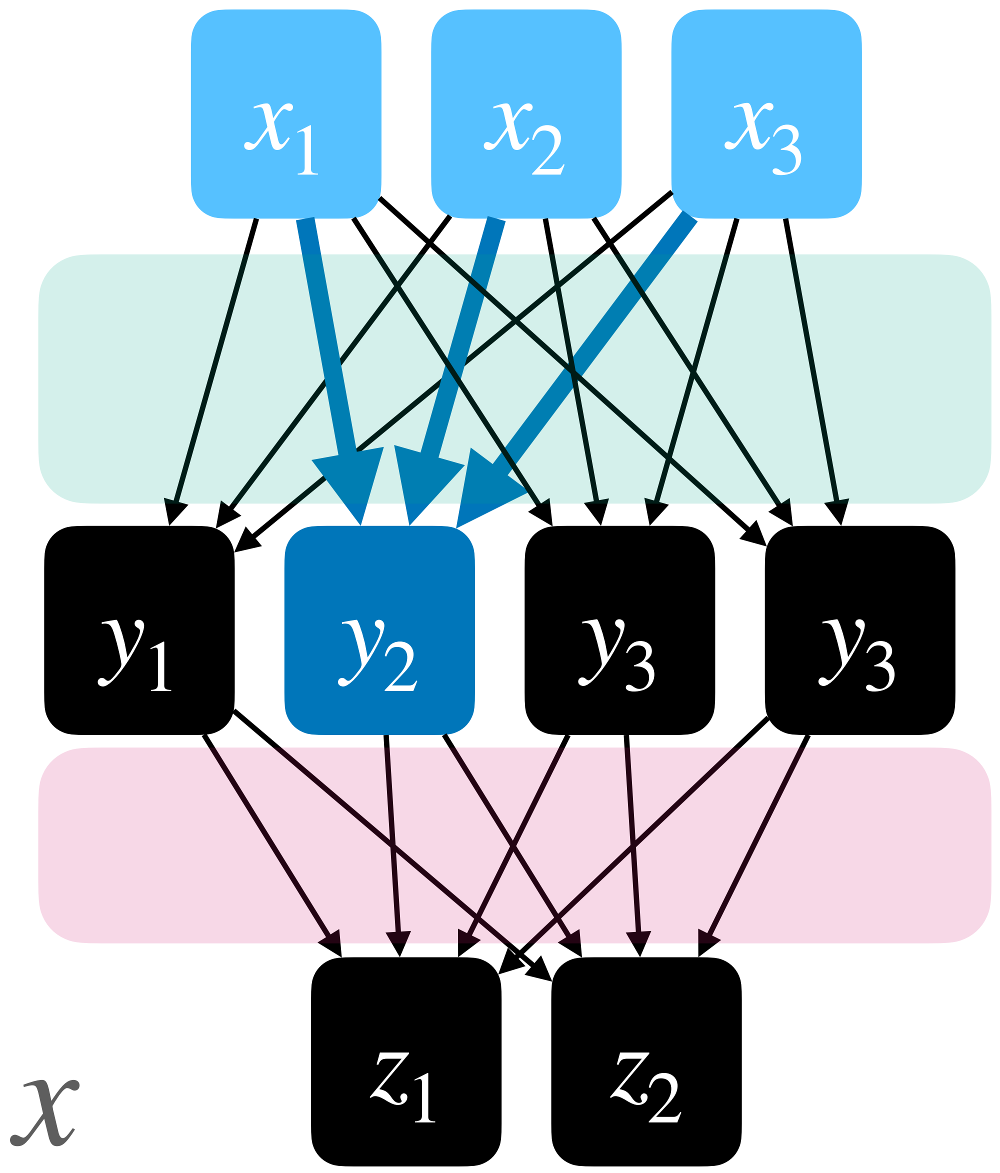
What do Matrix-Vector Products look in this picture ?



$$y = M_1 x$$

$$z = M_2 y$$

$$z = M_2 M_1 x$$

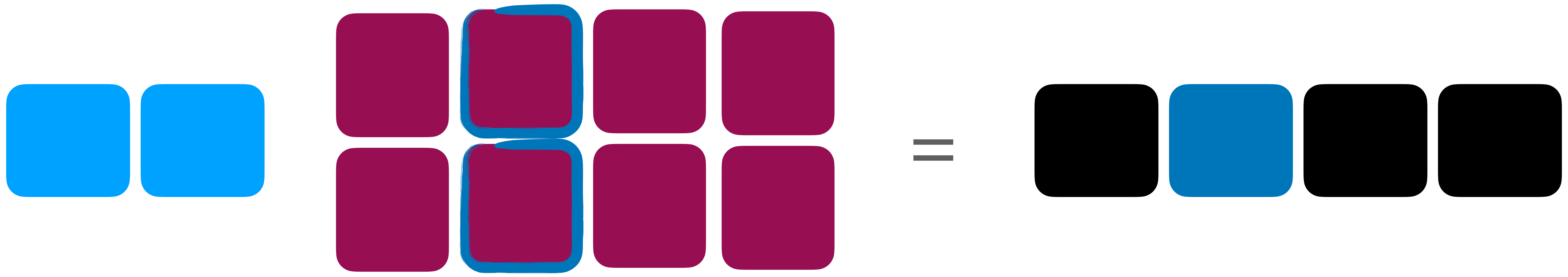


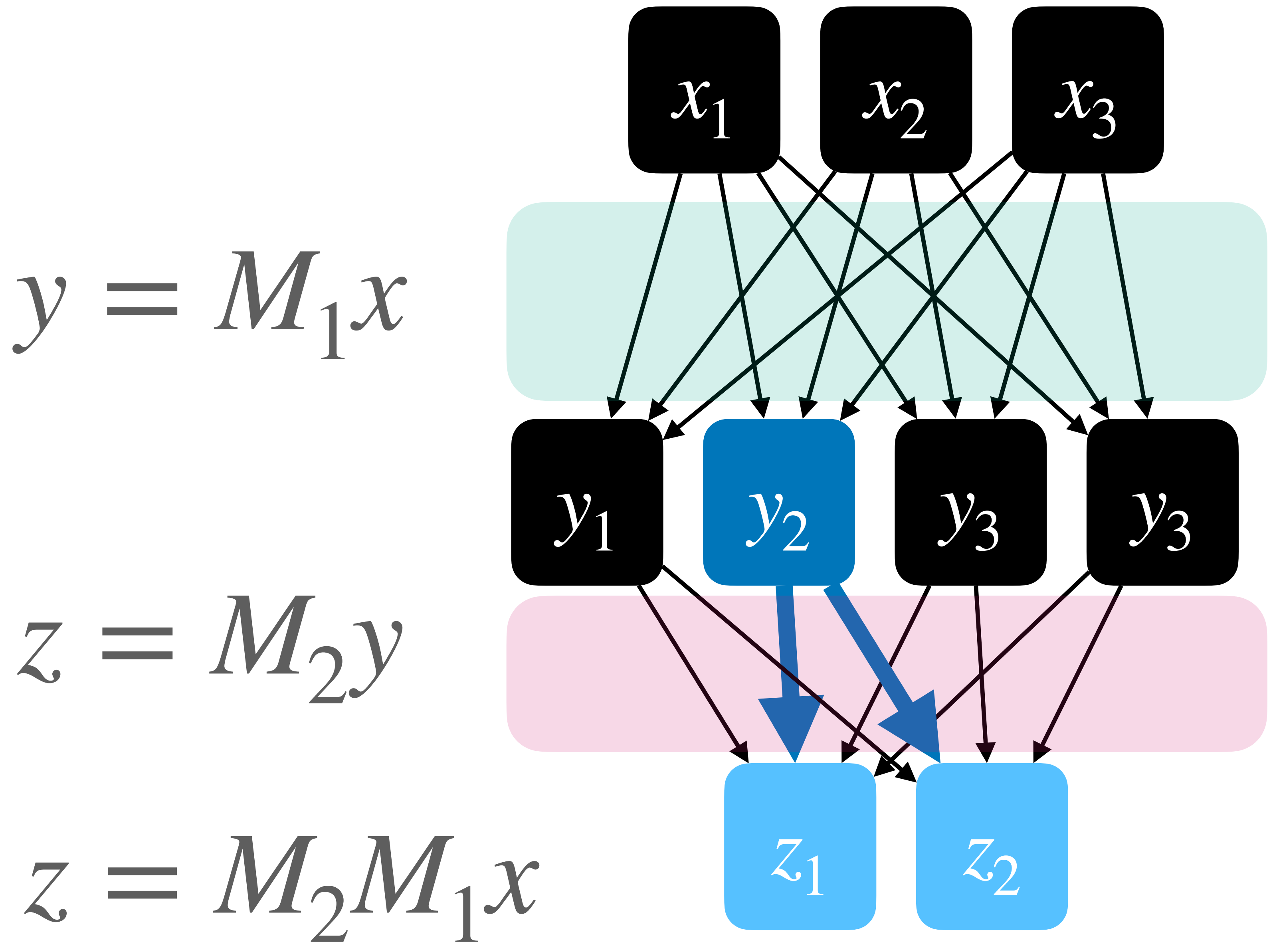
$$y_i = (Mx)_i = \sum_k M_{ik} x_k$$

$$y_i = (Mx)_i = \sum_{p \in \text{parents}(y_i)} M_{ip} x_p$$

A Graph Picture of Matrix Multiplication

What do Vector-Matrix Products look like?





$$y_i = (z^T M)_i = \sum_k z_k M_{ki}$$

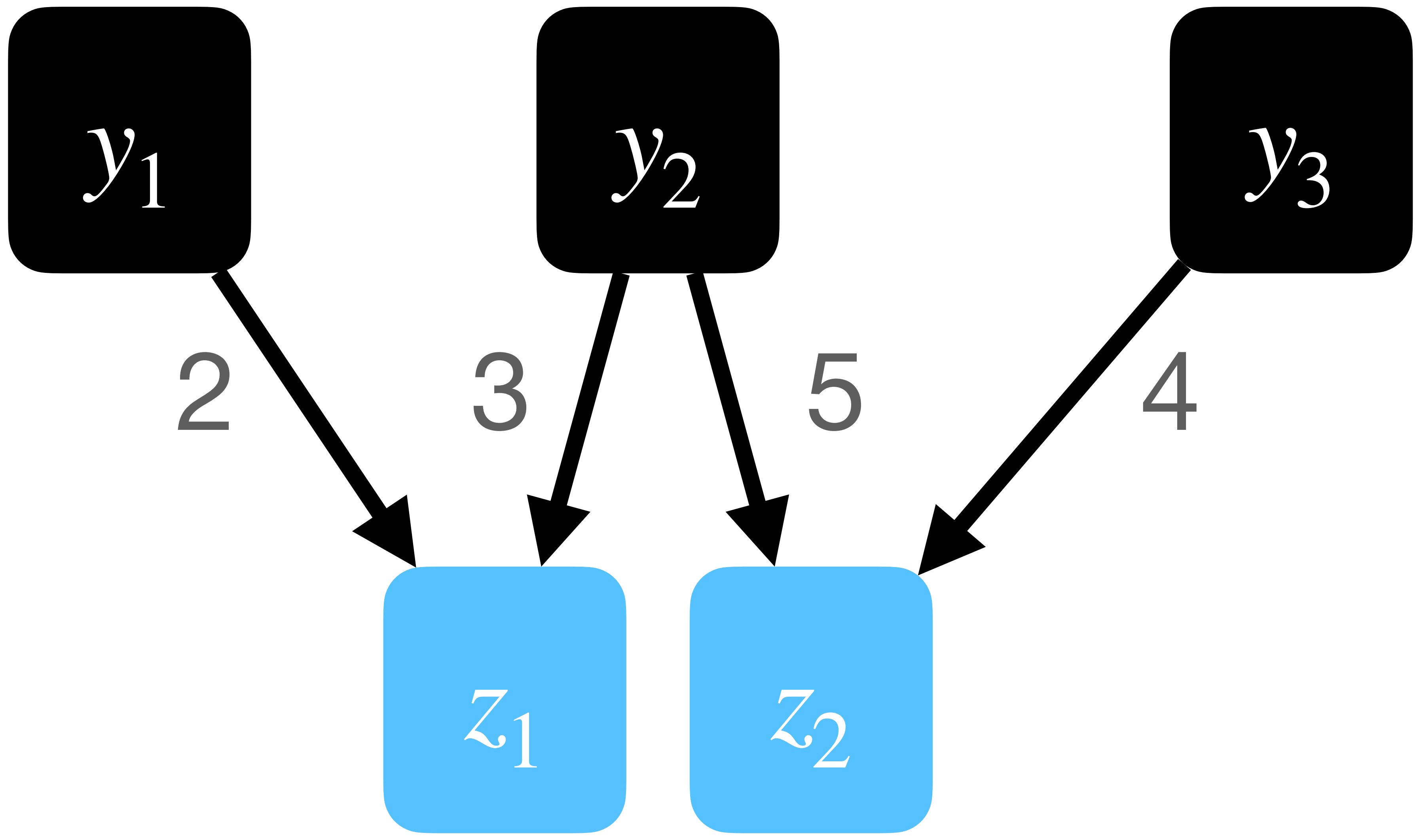
$$y_i = (z^T M)_i = \sum_{p \in \text{children}(y_i)} z_p M_{pi}$$

JVP/VJP in graph

There is a correspondence between

- Matrix Elements and Edges
- Linear Programs <> Matrices <> annotated edges

$$\begin{bmatrix} 2 & 3 & 0 \\ 0 & 5 & 4 \end{bmatrix}$$



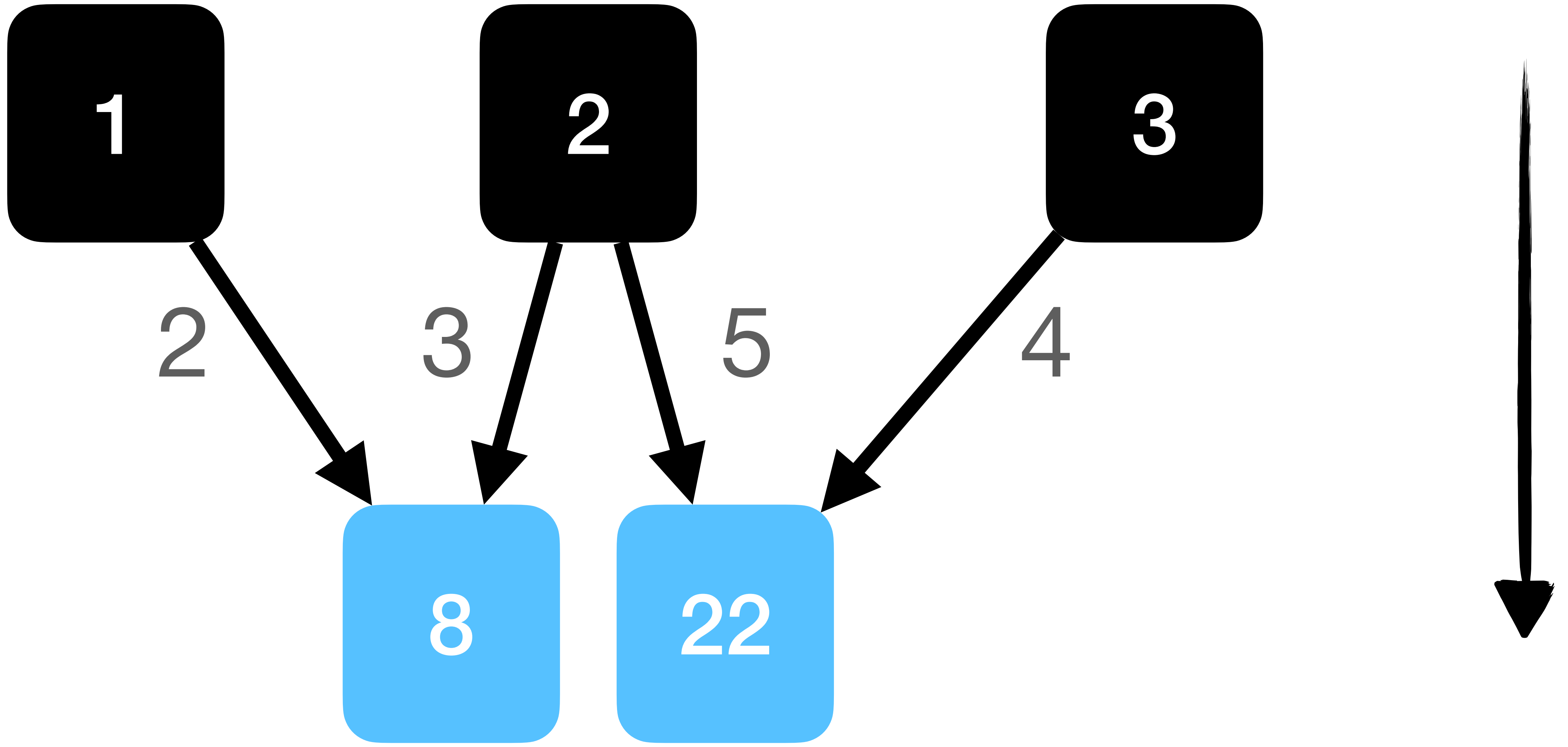
JVP/VJP in graph

```python
def mvp(inp):
    x,y,z = inp
    return np.array([
        2*x + 3*y,
        5*y + 4*z
    ])

mvp([1,2,3])
```

```
array([ 8, 22])
```

$$y_i = (Mx)_i = \sum_{p \in \text{parents}(y)} M_{ip} x_p$$

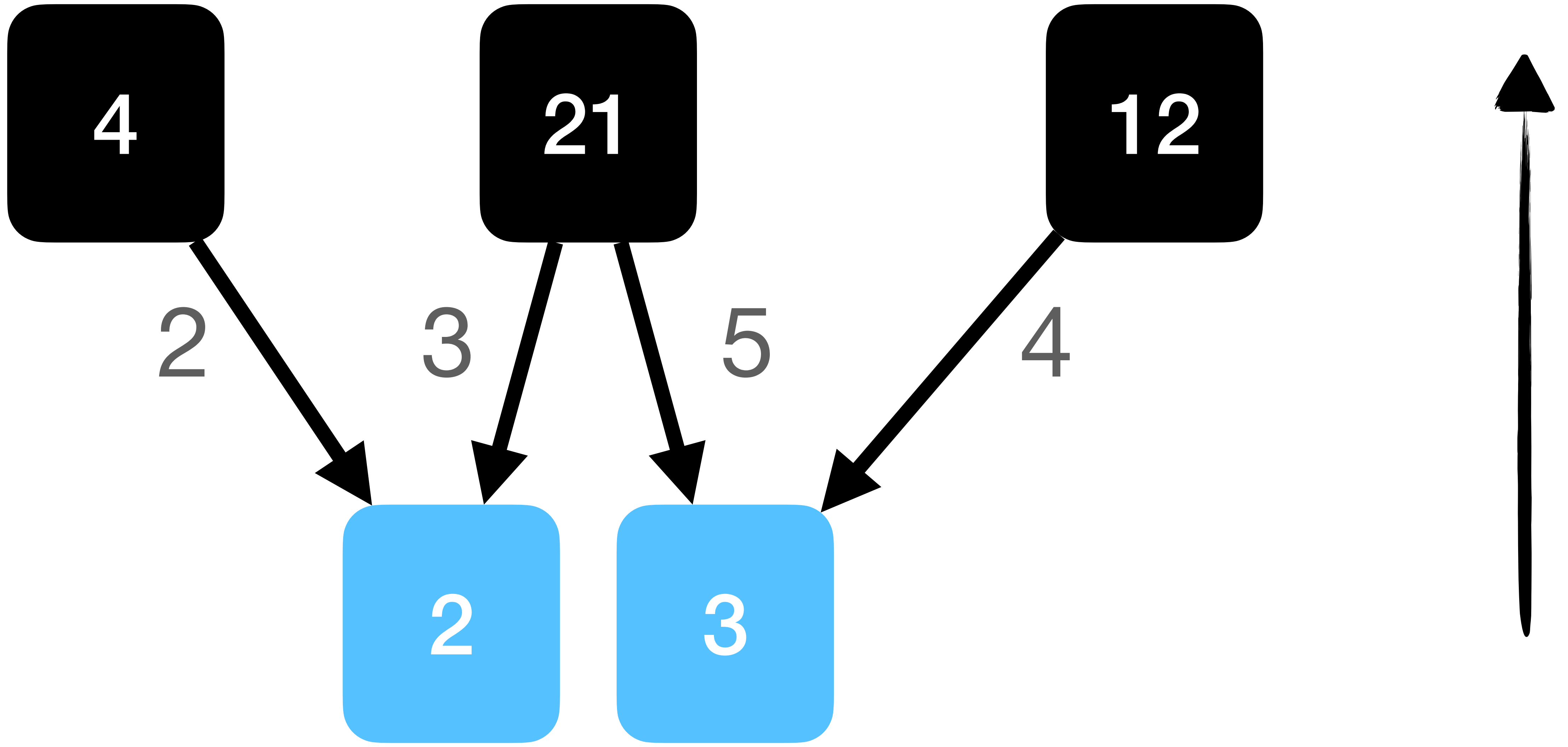


JVP/VJP in graph

```python
def vmp(out):
    a,b = out
    return np.array([
        2*a,
        3*a + 5*b,
        4*b
    ])

vmp([2,3])
```

```
array([ 4, 21, 12])
```

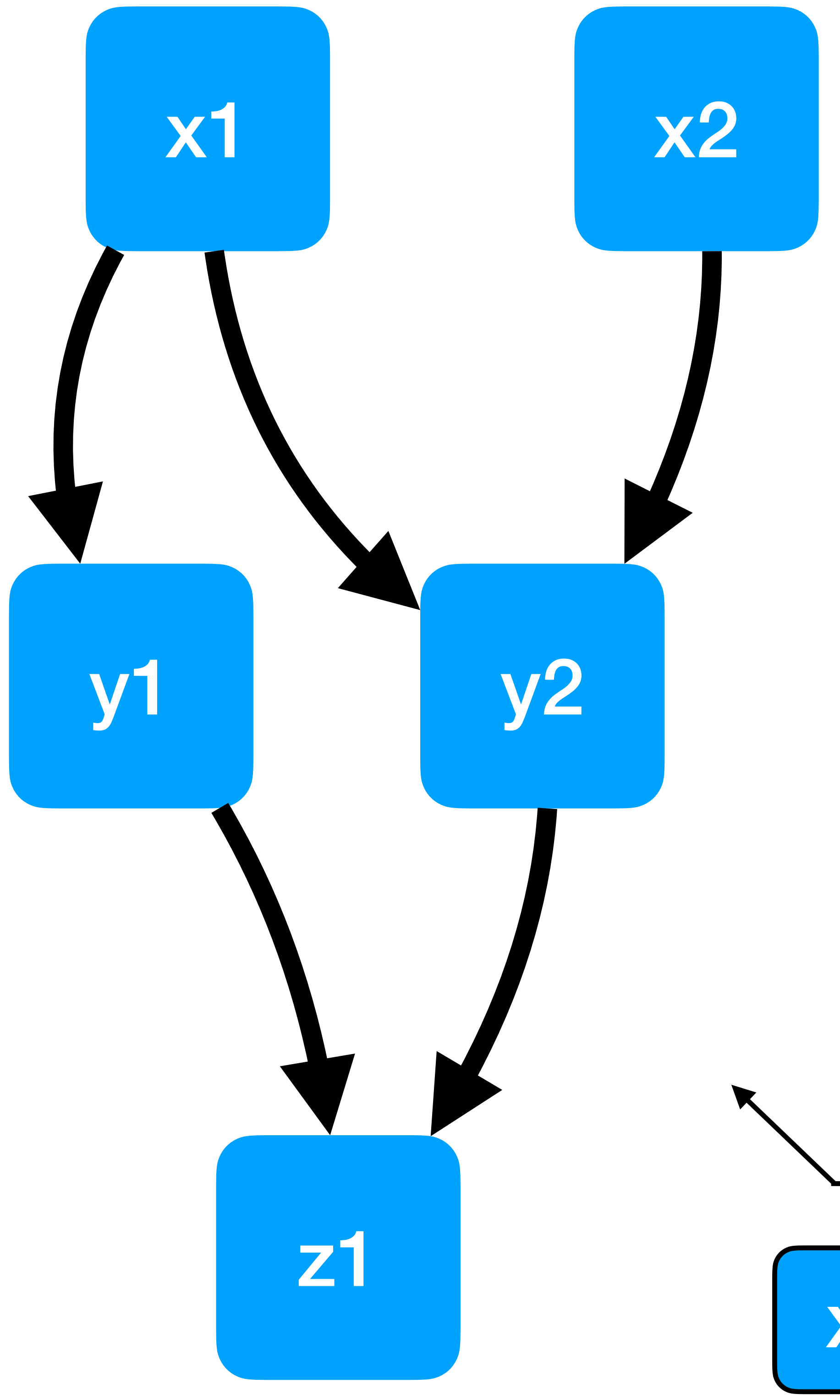


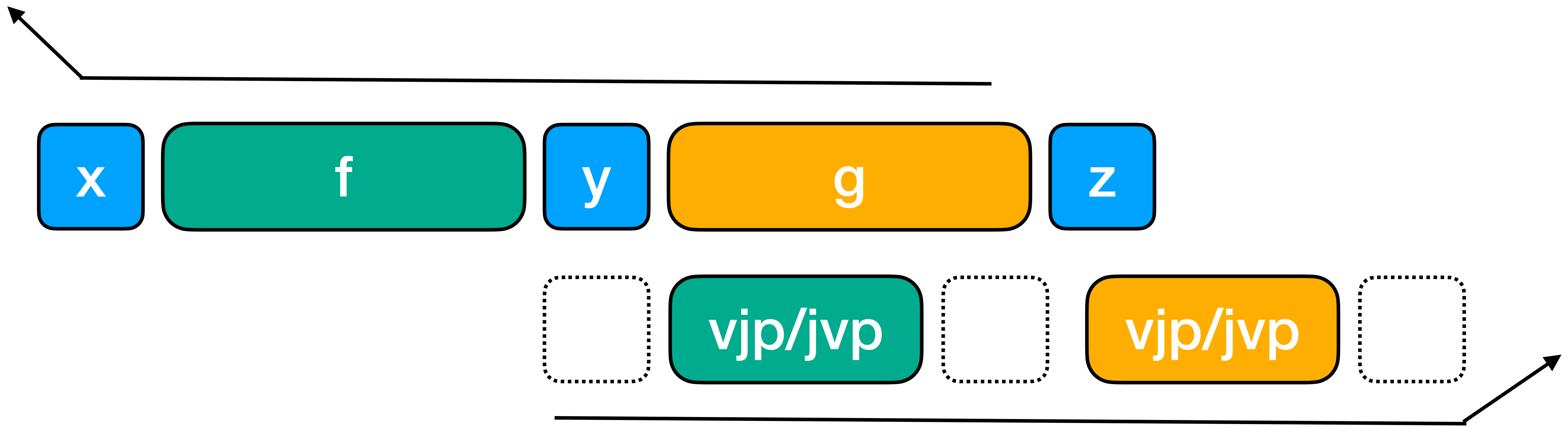
$$y_i = (z^T M)_i = \sum_{p \in \mathrm{children}(y)} z_p M_{pi}$$

JVP/VJP in graph

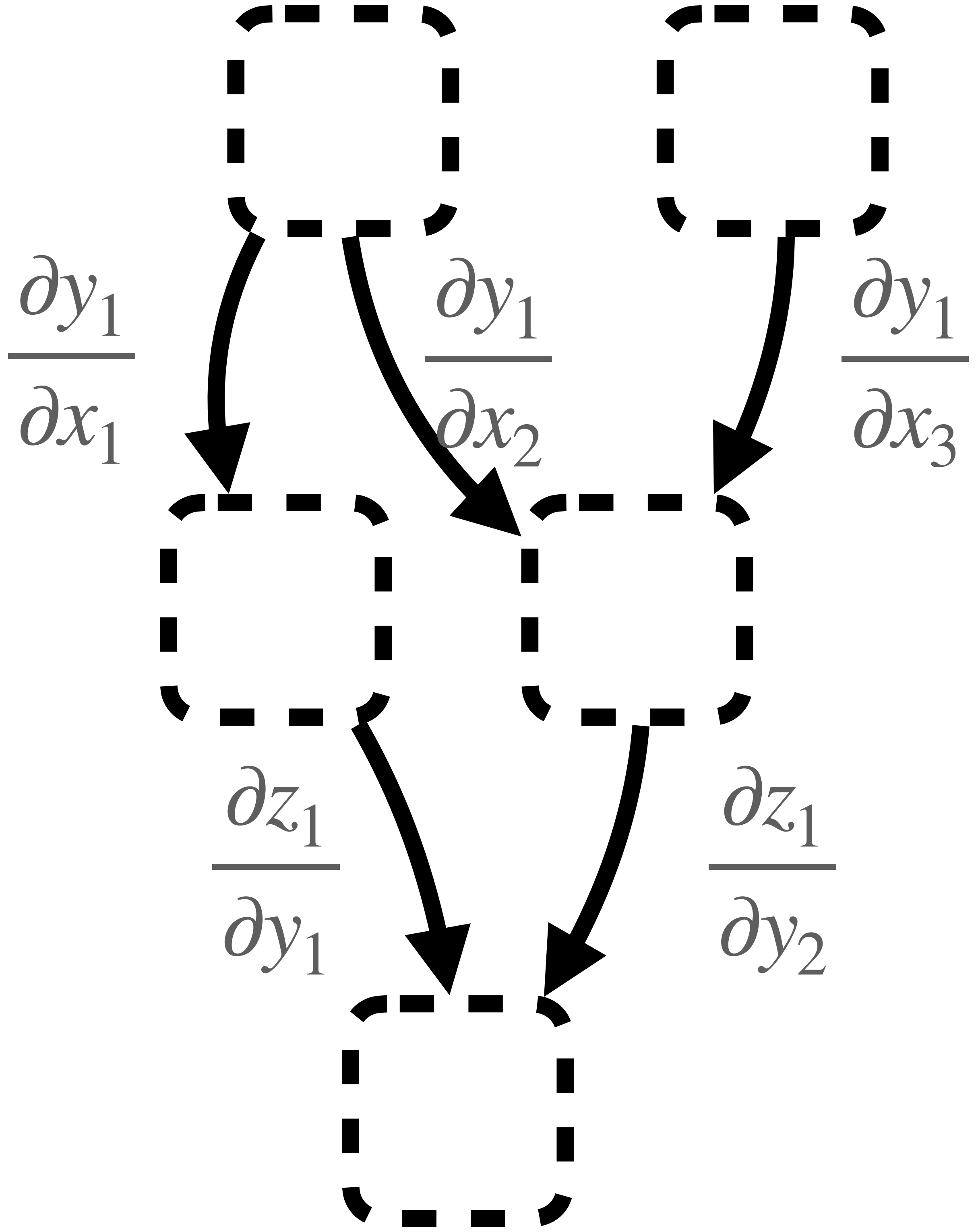
“creating a JVP/VJP program” is just annotating the graph edges with partial derivatives

main
non-linear
program



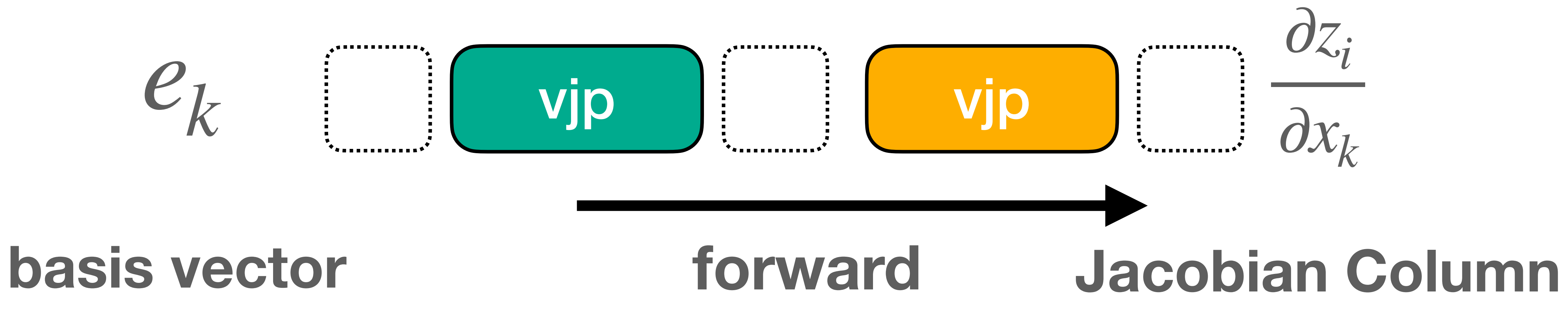


JVP/JVP
graph

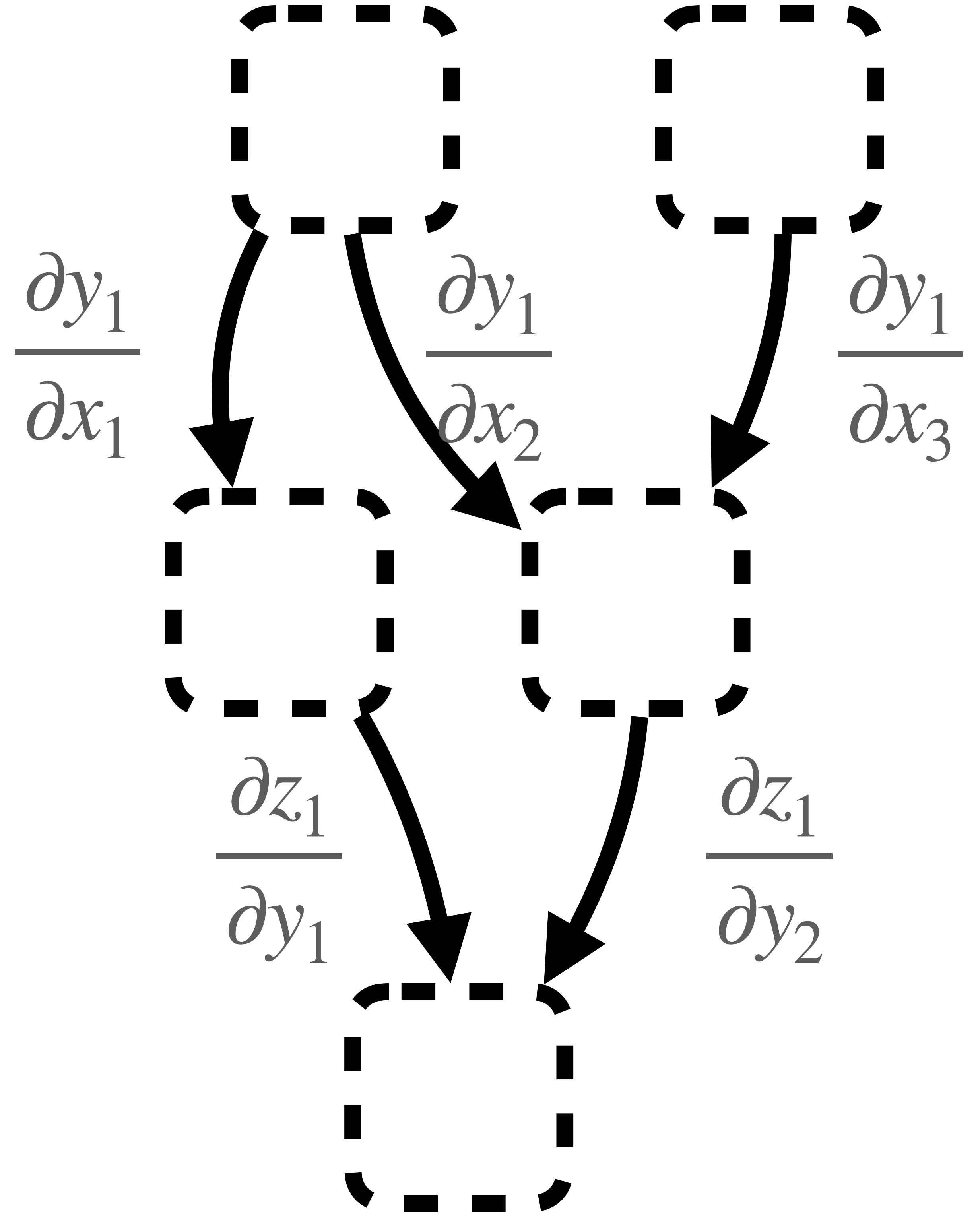


JVP/VJP in graph

Compute derivatives by propagating basis vectors through the graph



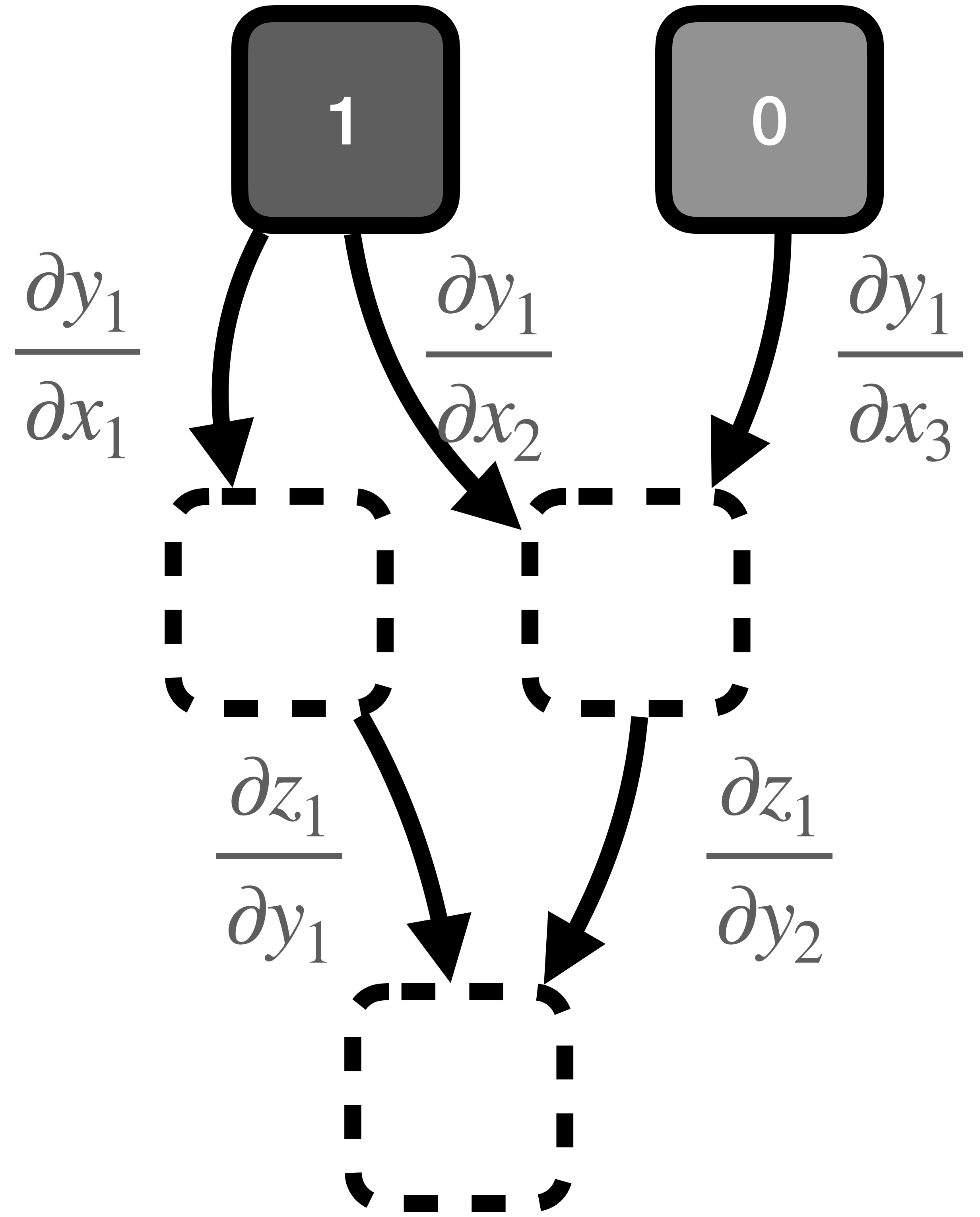
$$a_i = (Jb)_i = \sum_{p \in \text{parents}(y)} \frac{\partial u_i}{\partial v_p} b_p$$



JVP/VJP in graph

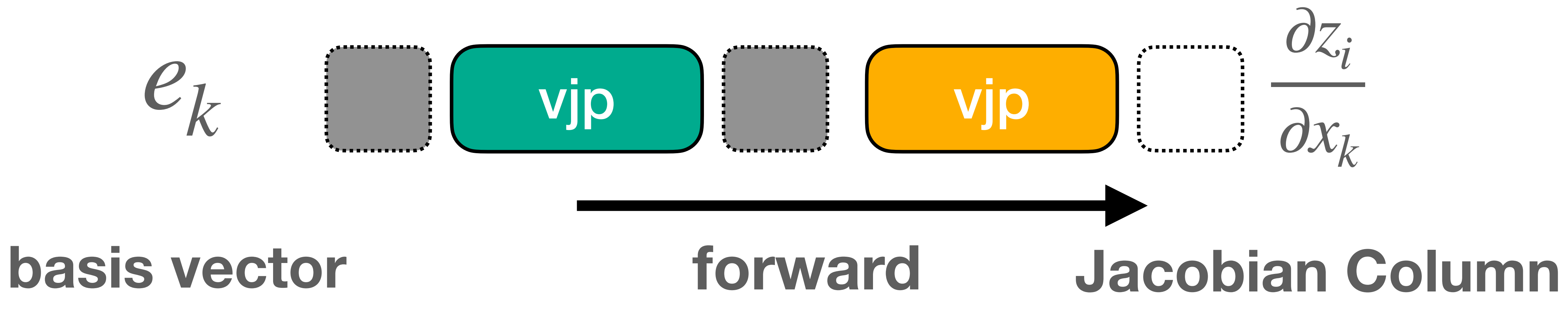
Compute derivatives by propagating basis vectors through the graph

$$a_i = (Jb)_i = \sum_{p \in \text{parents}(y)} \frac{\partial u_i}{\partial v_p} b_p$$

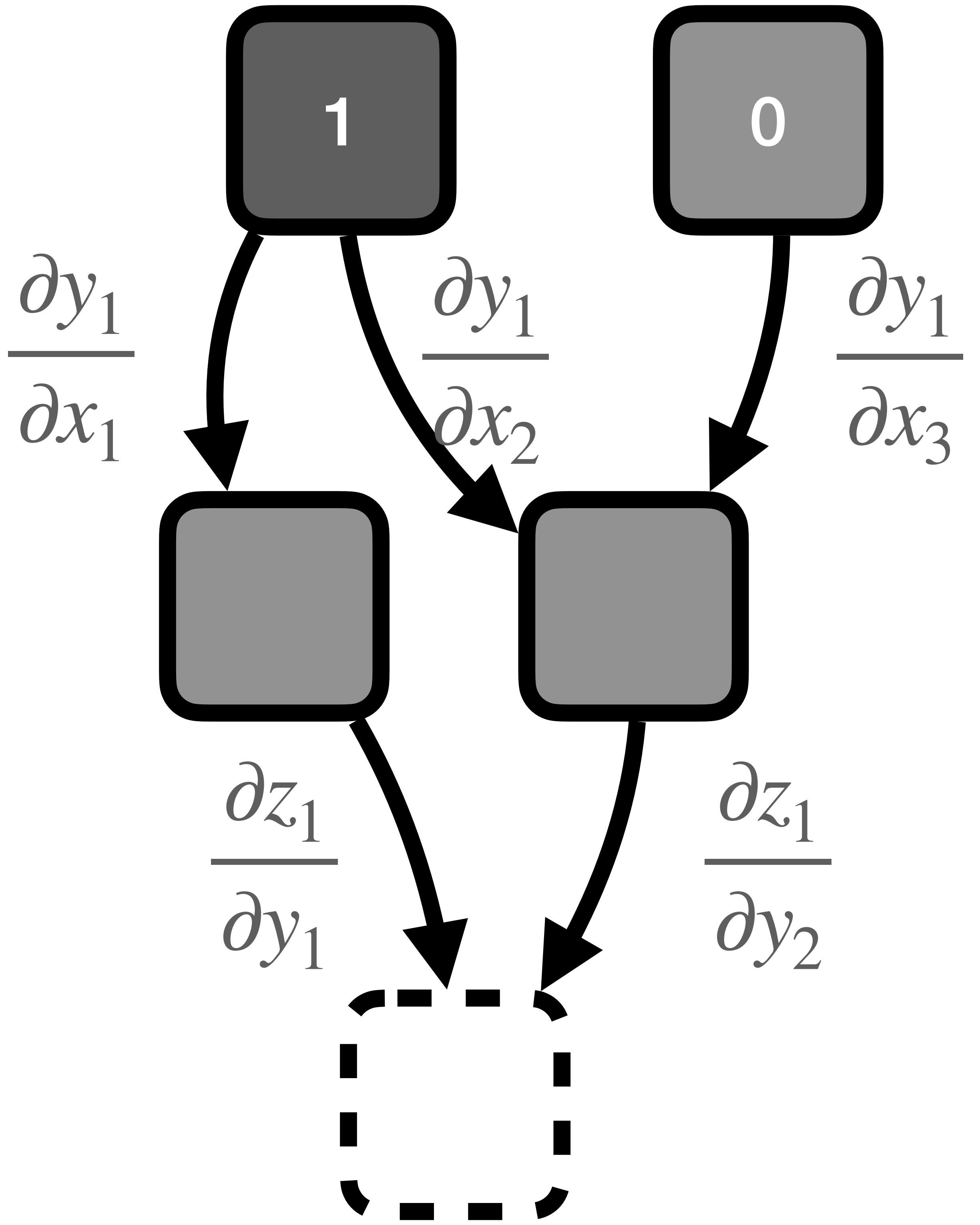


JVP/VJP in graph

Compute derivatives by propagating basis vectors through the graph

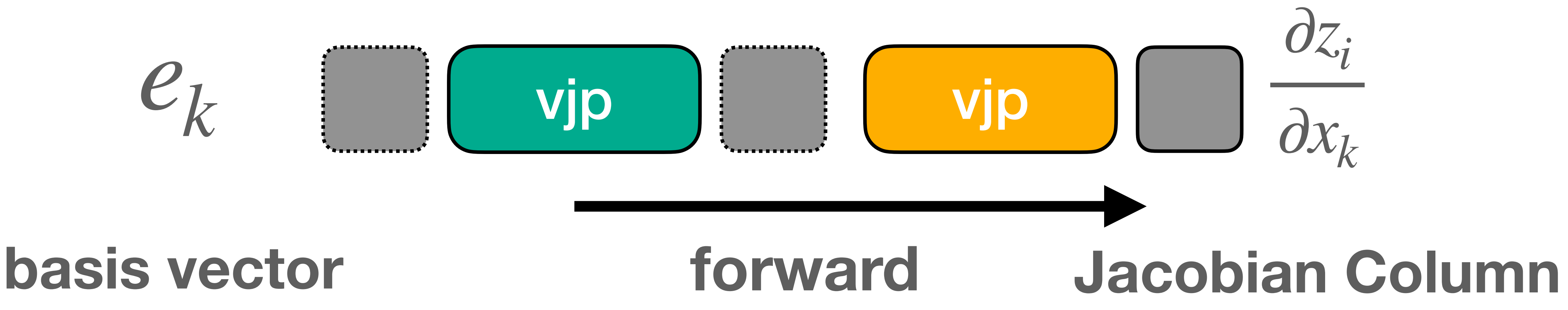


$$a_i = (Jb)_i = \sum_{p \in \text{parents}(y)} \frac{\partial u_i}{\partial v_p} b_p$$

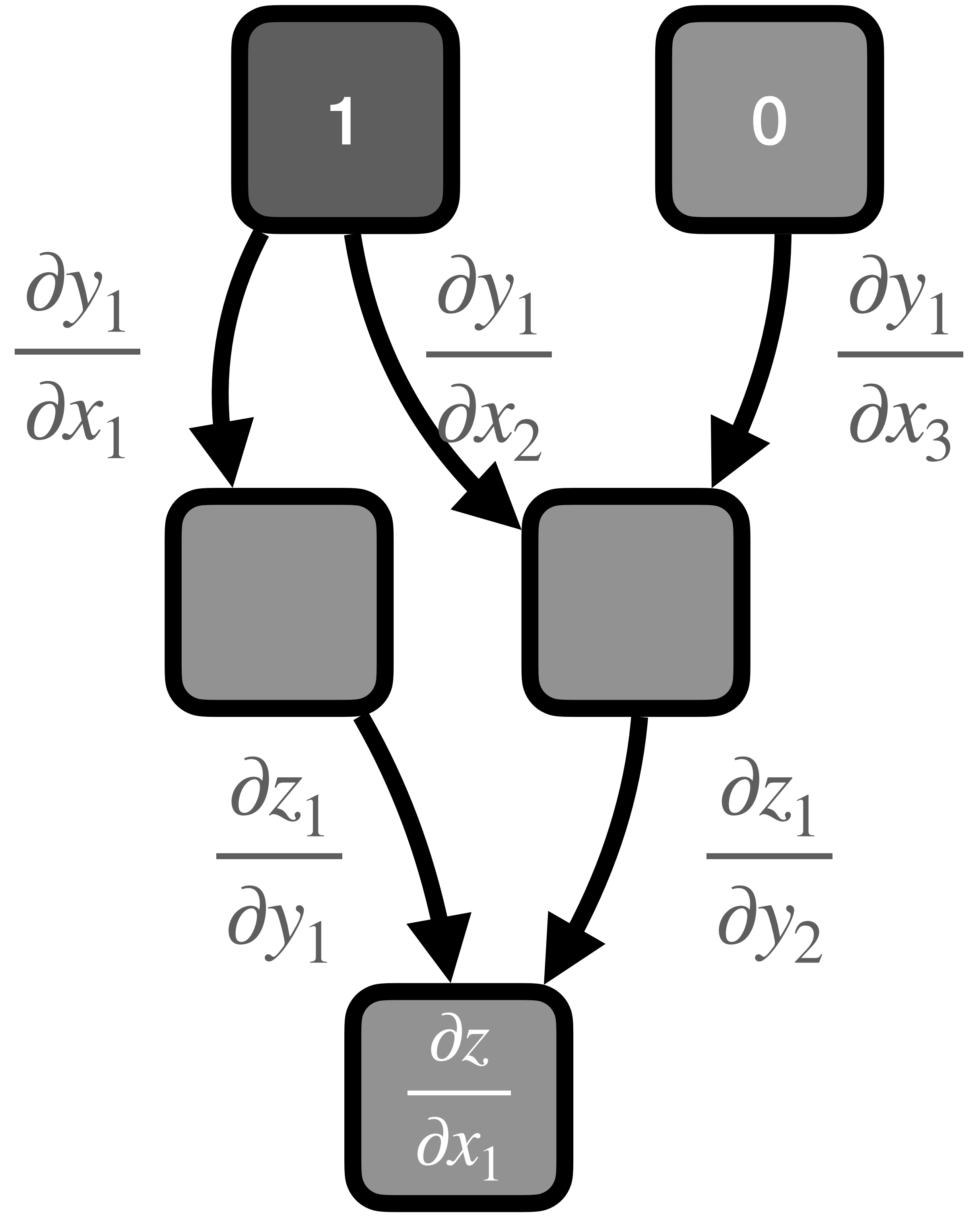


JVP/VJP in graph

Compute derivatives by propagating basis vectors through the graph

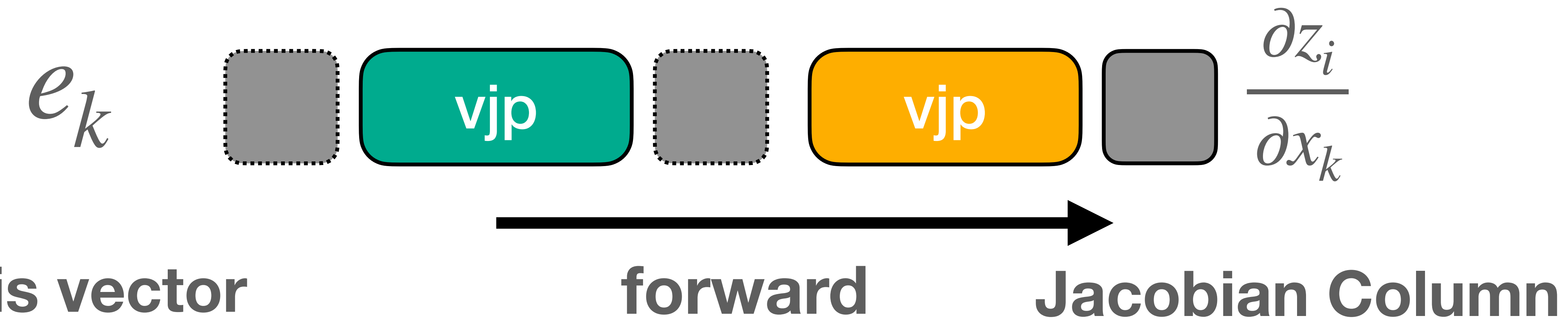


$$a_i = (Jb)_i = \sum_{p \in \text{parents}(y)} \frac{\partial u_i}{\partial v_p} b_p$$

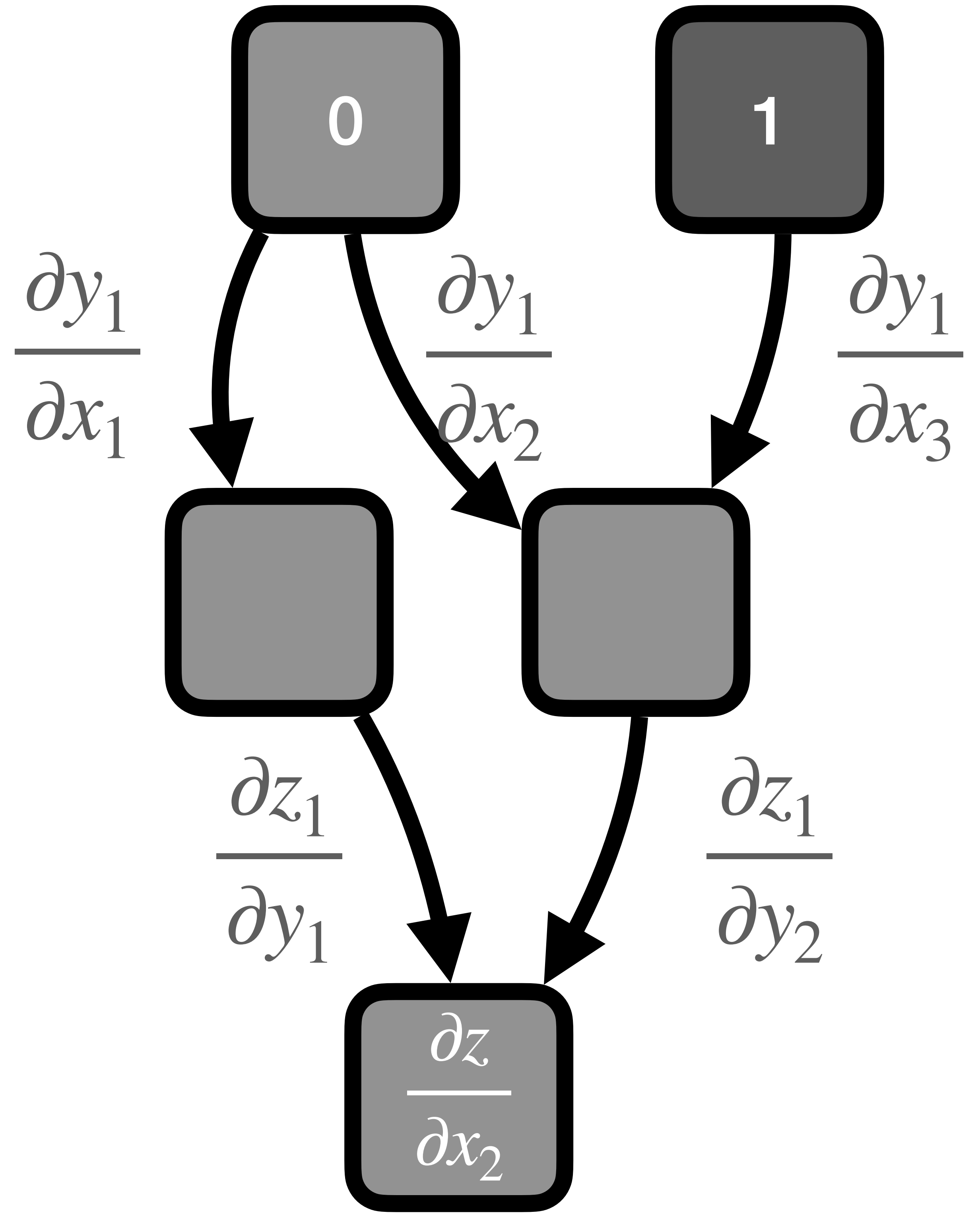


JVP/VJP in graph

Compute derivatives by propagating basis vectors through the graph

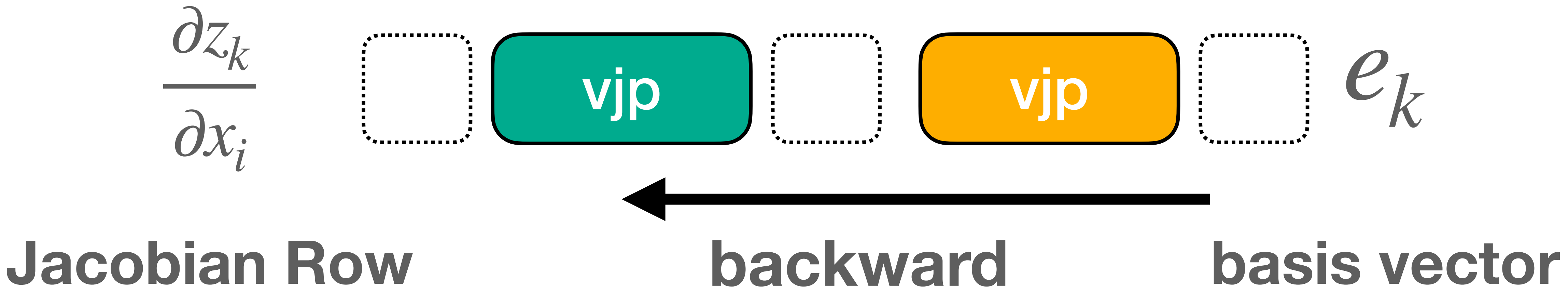


$$a_i = (Jb)_i = \sum_{p \in \text{parents}(y)} \frac{\partial u_i}{\partial v_p} b_p$$

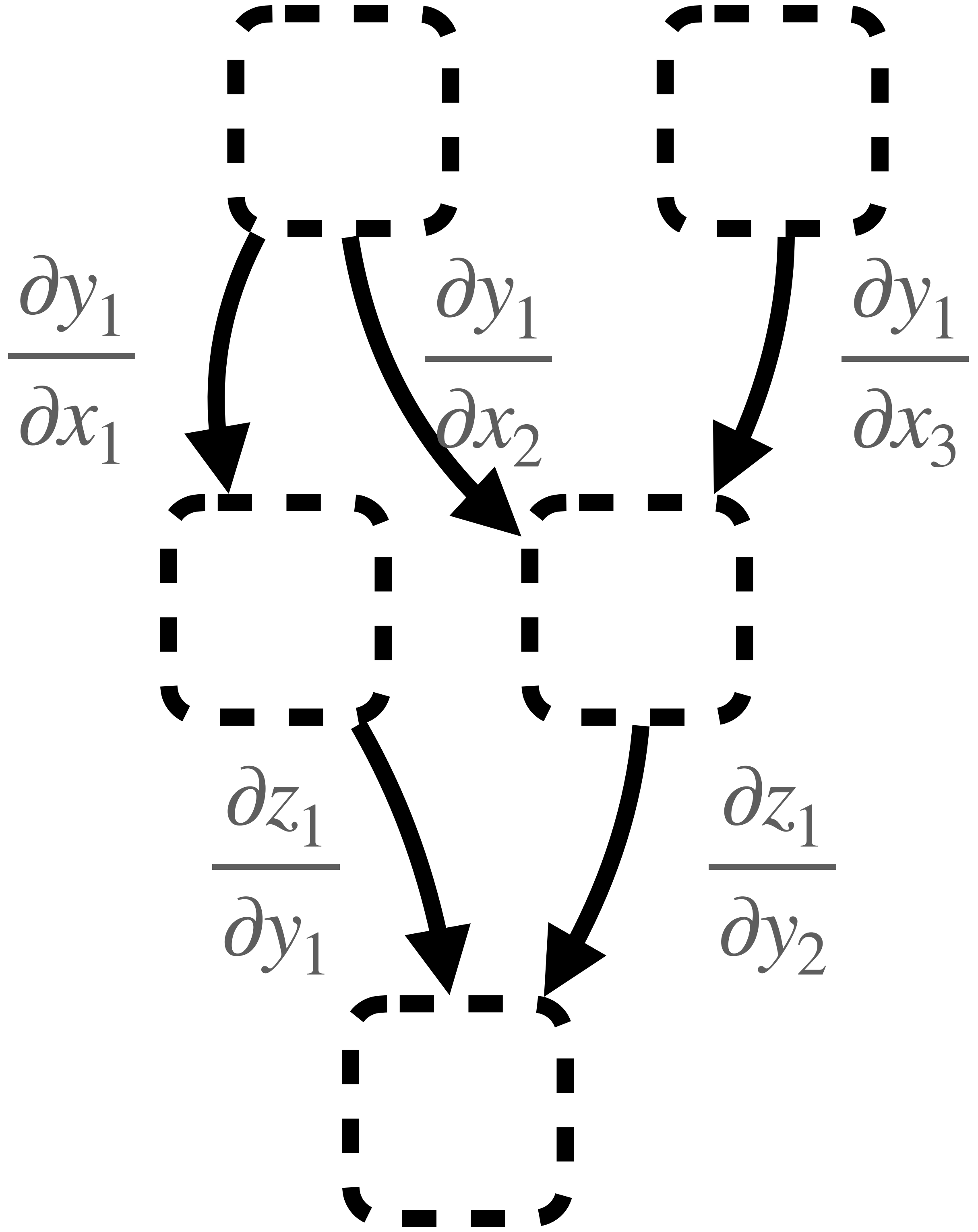


JVP/VJP in graph

Compute derivatives by propagating basis vectors through the graph

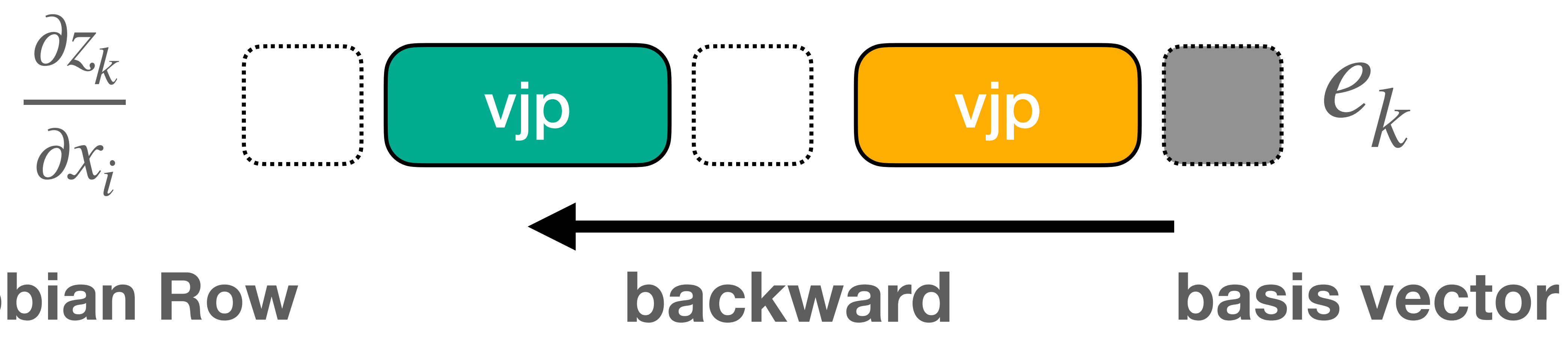


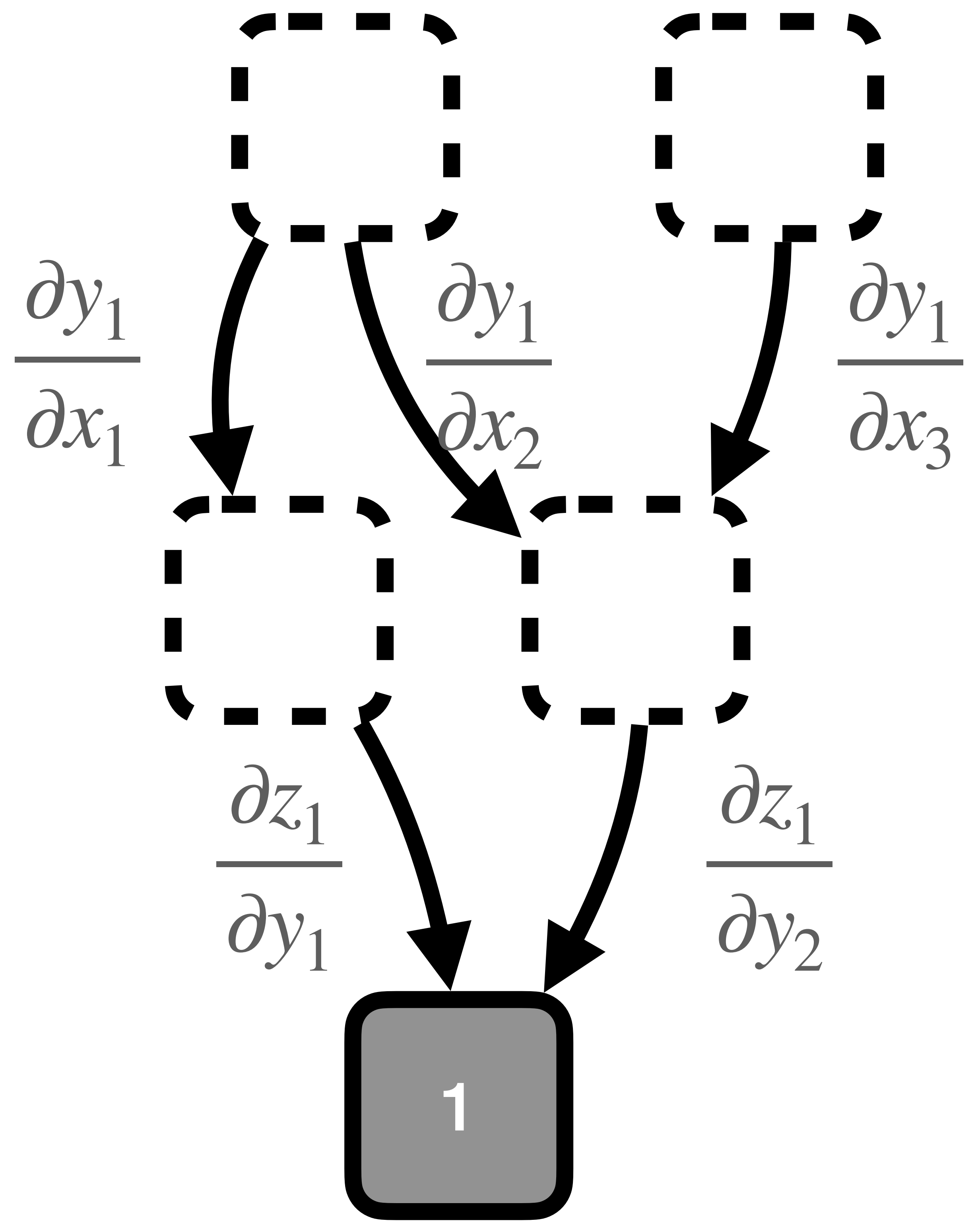
$$a_i = (b^T J)_i = \sum_{b \in \text{children}(z)} z_p \frac{\partial u_p}{\partial v_i}$$



JVP/VJP in graph

Compute derivatives by propagating basis vectors through the graph

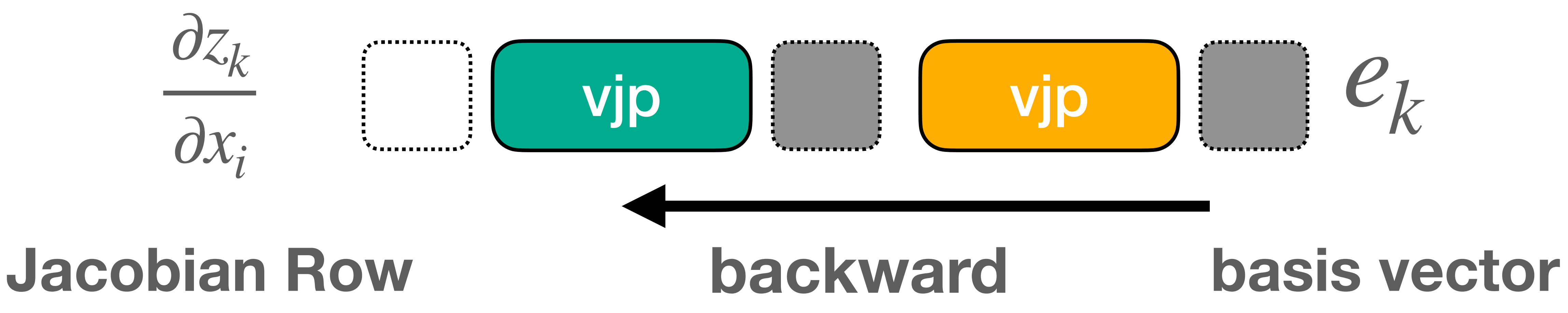


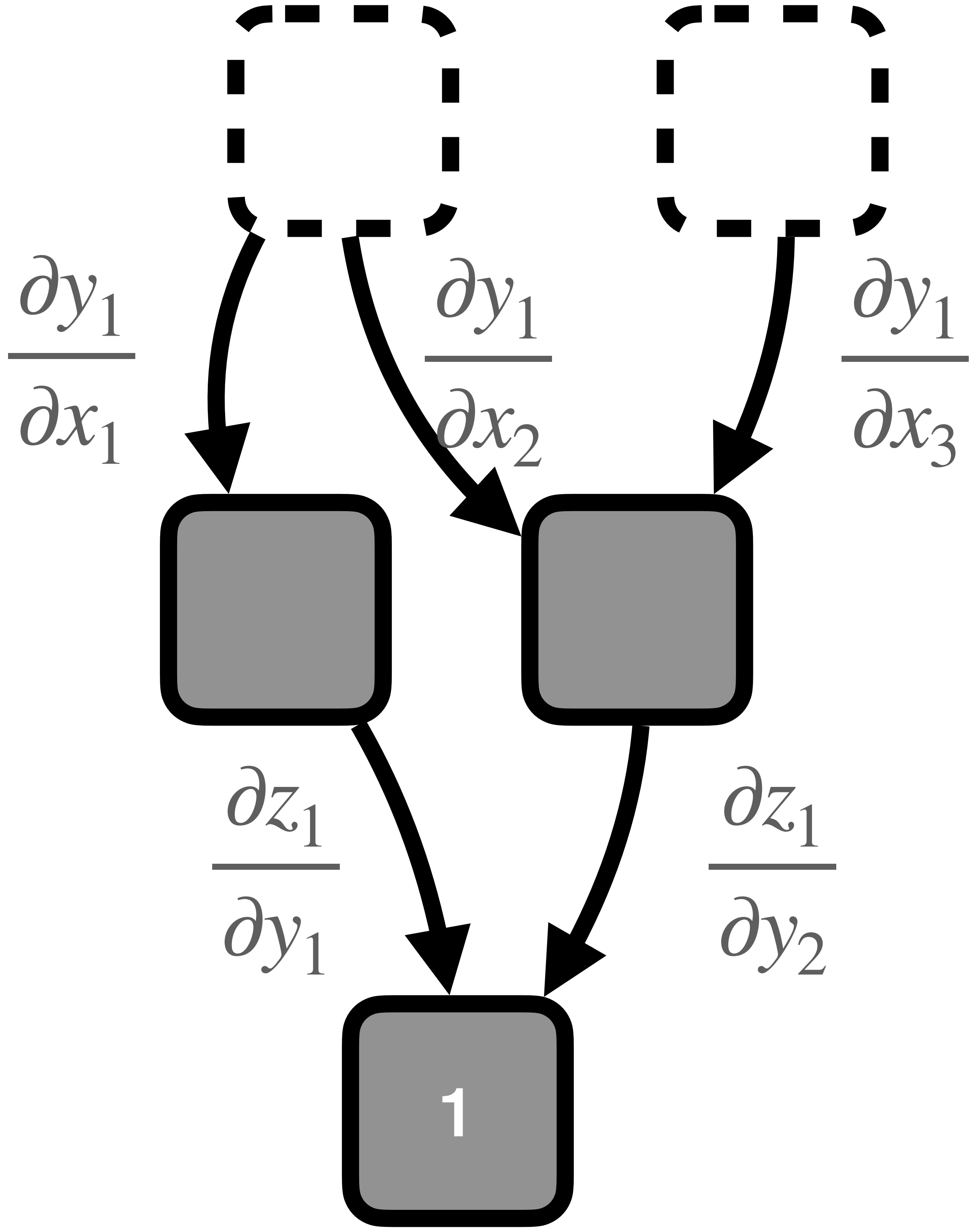


$$a_i = (b^T J)_i = \sum_{b \in \text{children}(z)} z_p \frac{\partial u_p}{\partial v_i}$$

JVP/VJP in graph

Compute derivatives by propagating basis vectors through the graph

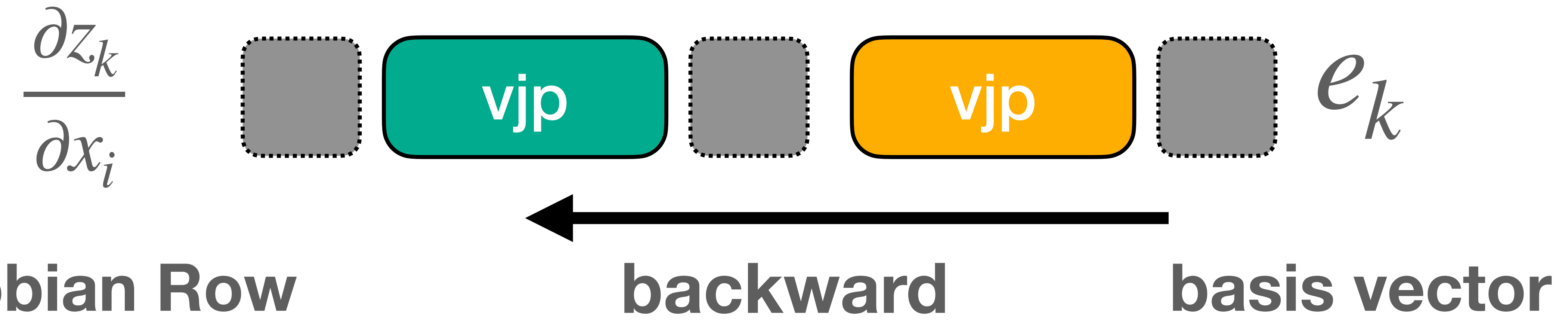


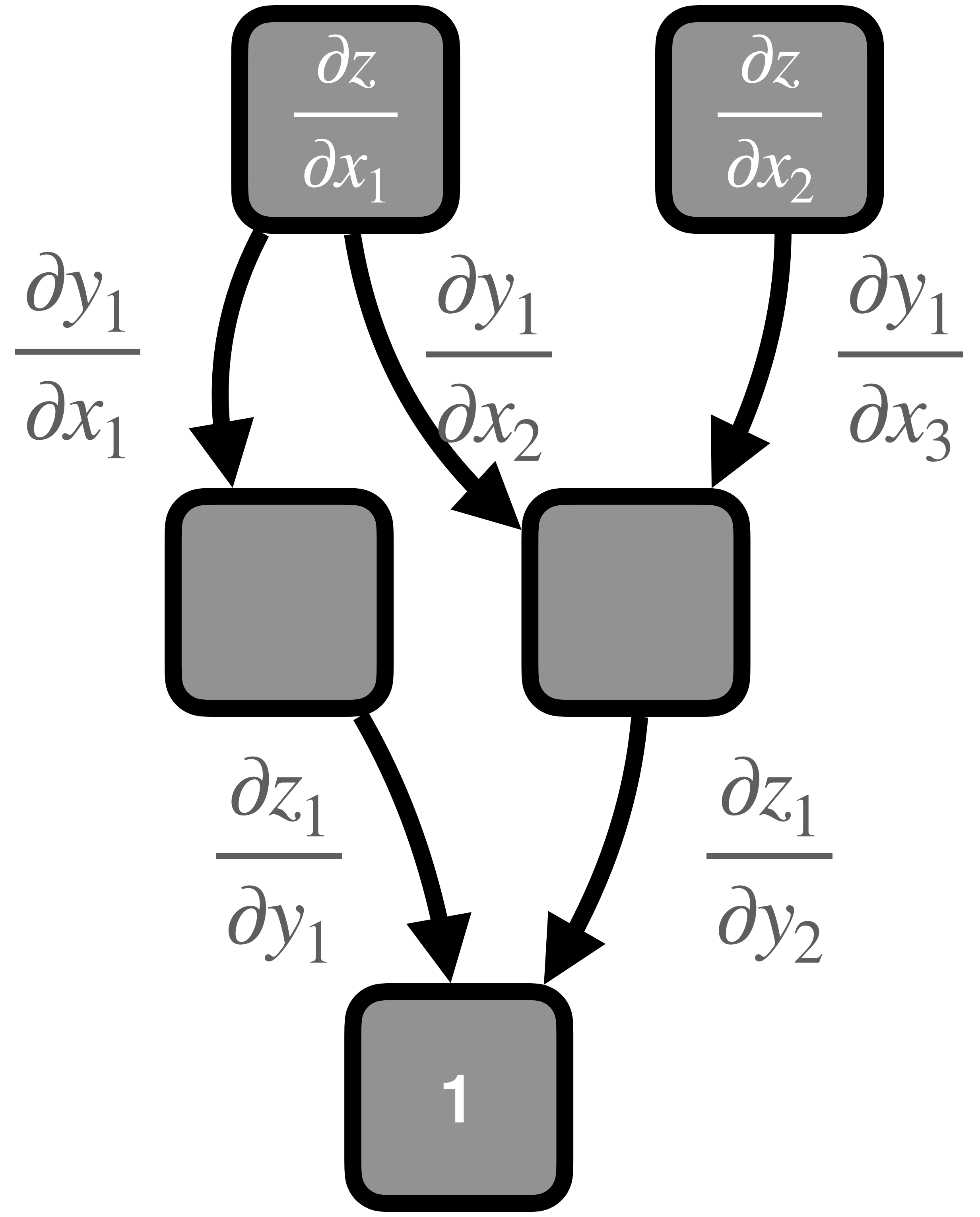


$$a_i = (b^T J)_i = \sum_{b \in \text{children}(z)} z_p \frac{\partial u_p}{\partial v_i}$$

JVP/VJP in graph

Compute derivatives by propagating basis vectors through the graph





$$a_i = (b^T J)_i = \sum_{b \in \text{children}(z)} z_p \frac{\partial u_p}{\partial v_i}$$

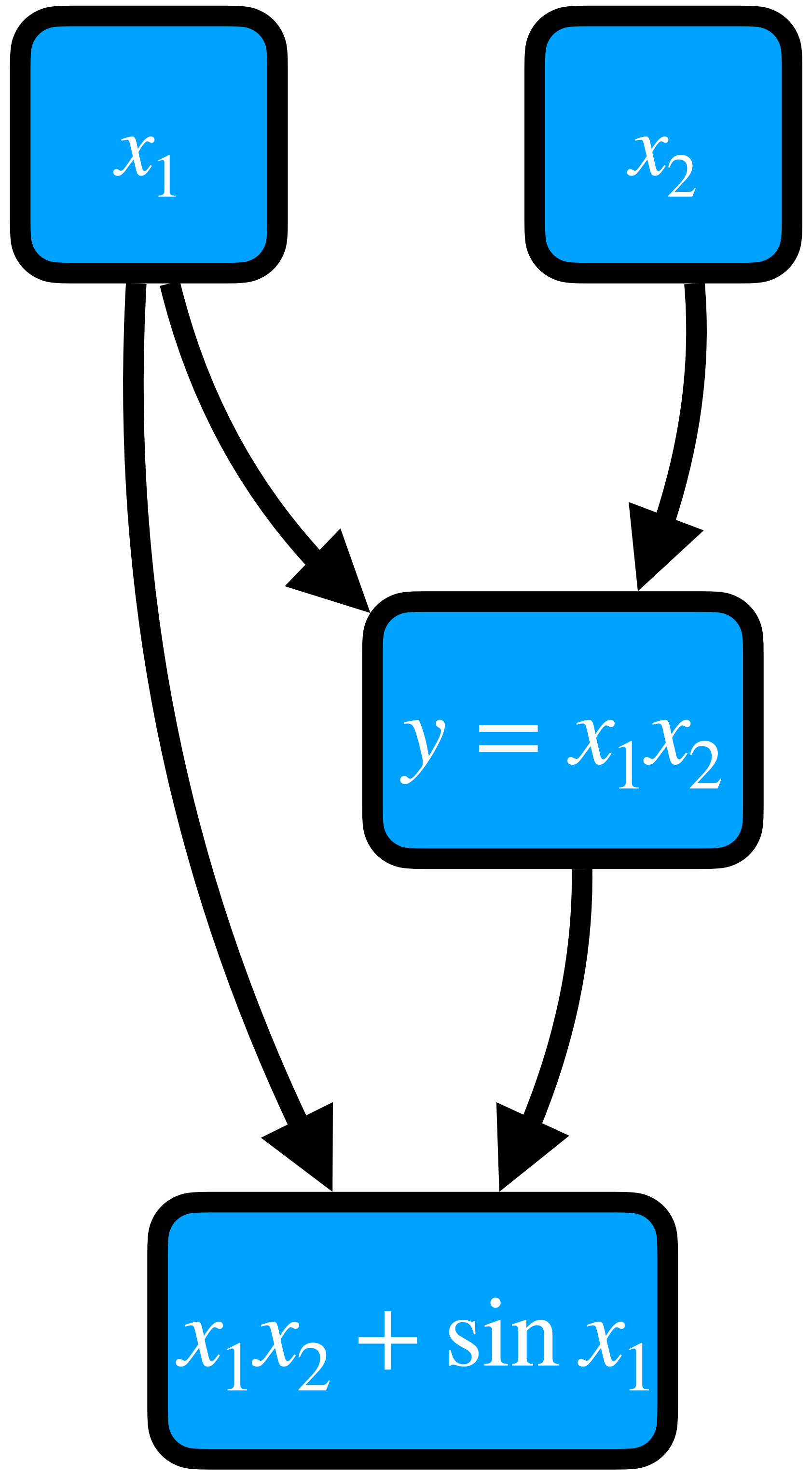
Example

Let's take a simple example:

$$z = x_1 x_2 + \sin x_1 = y + \sin x_1$$

Example

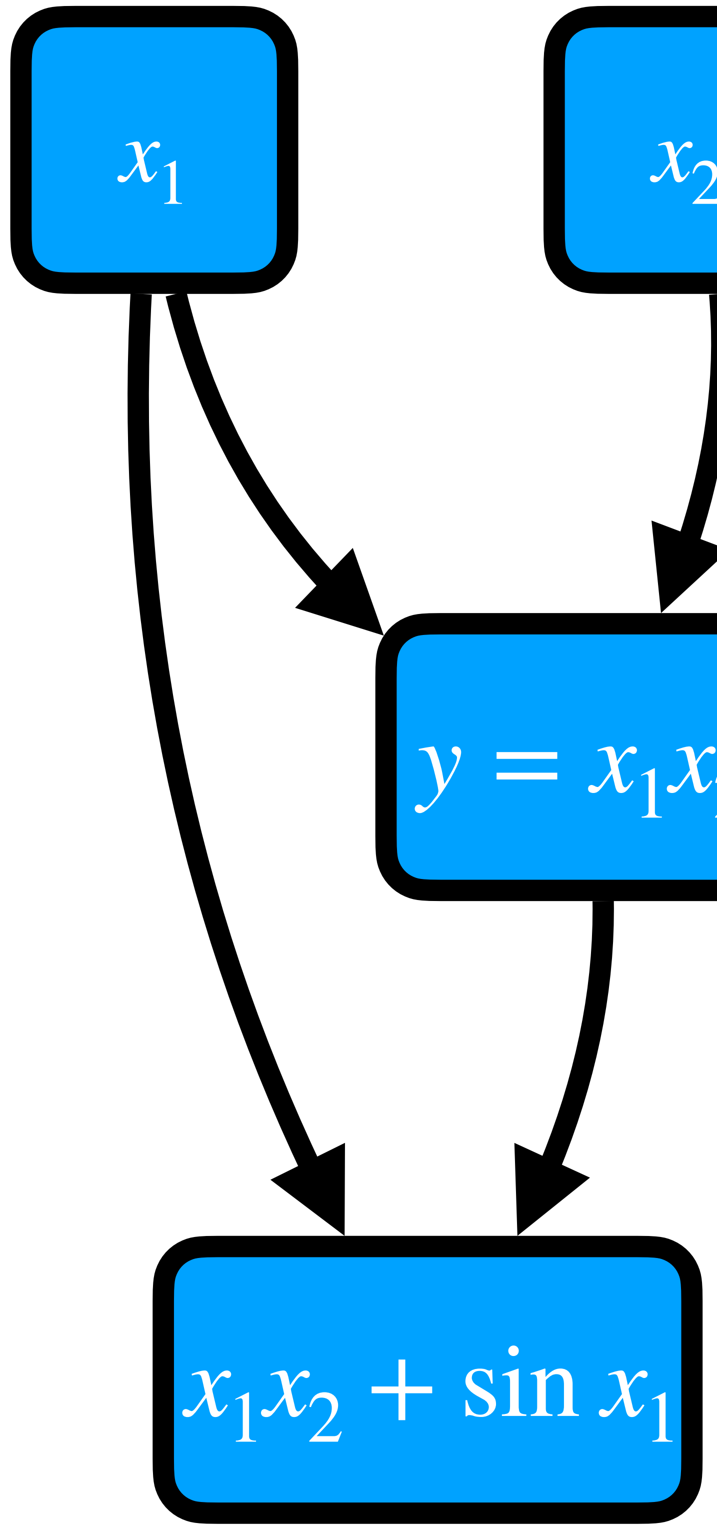
Express as a graph:

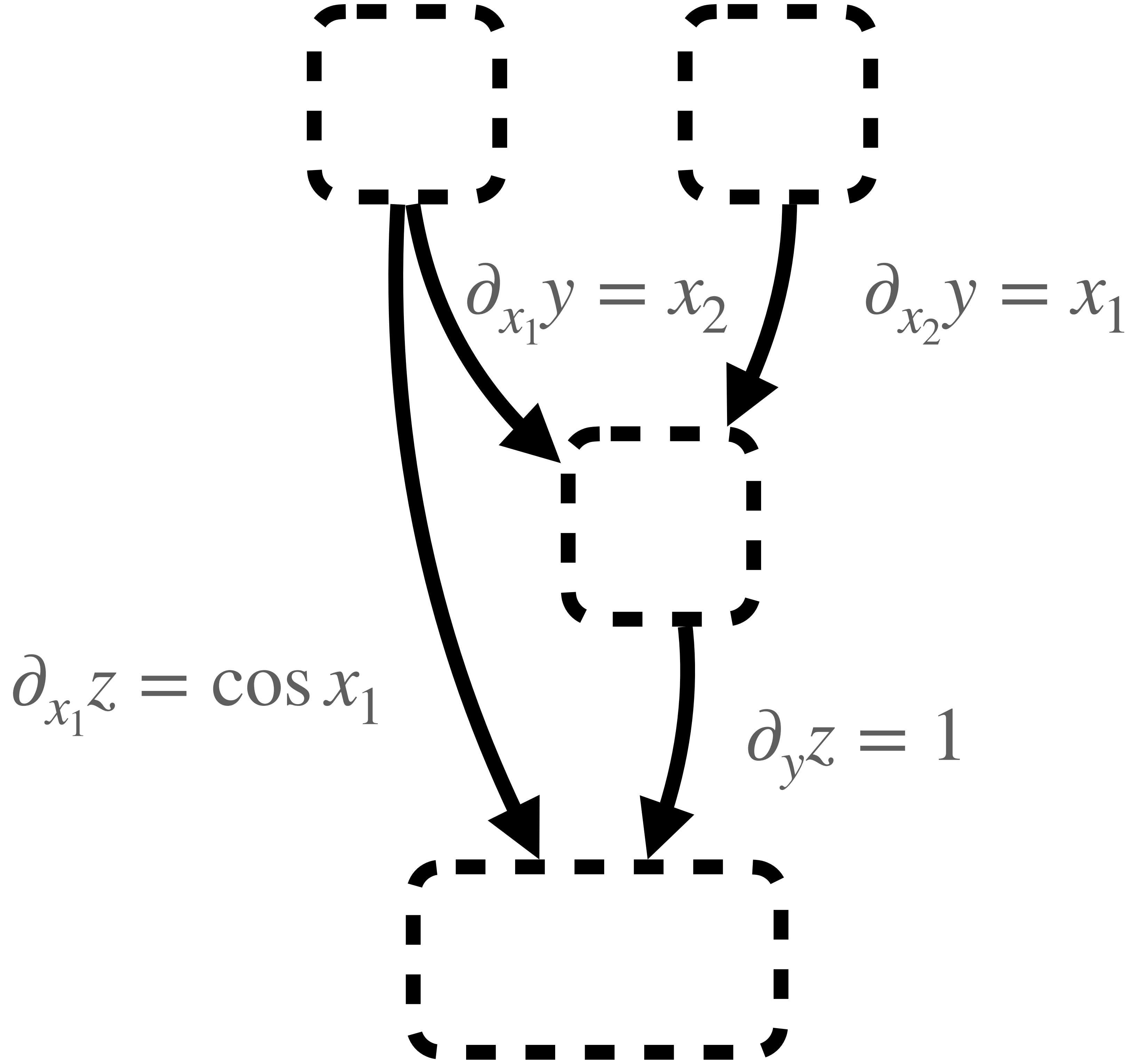


$$z = x_1 x_2 + \sin x_1 = y + \sin x_1$$

Example

Annotate Edges: (add JVP/JVP information)

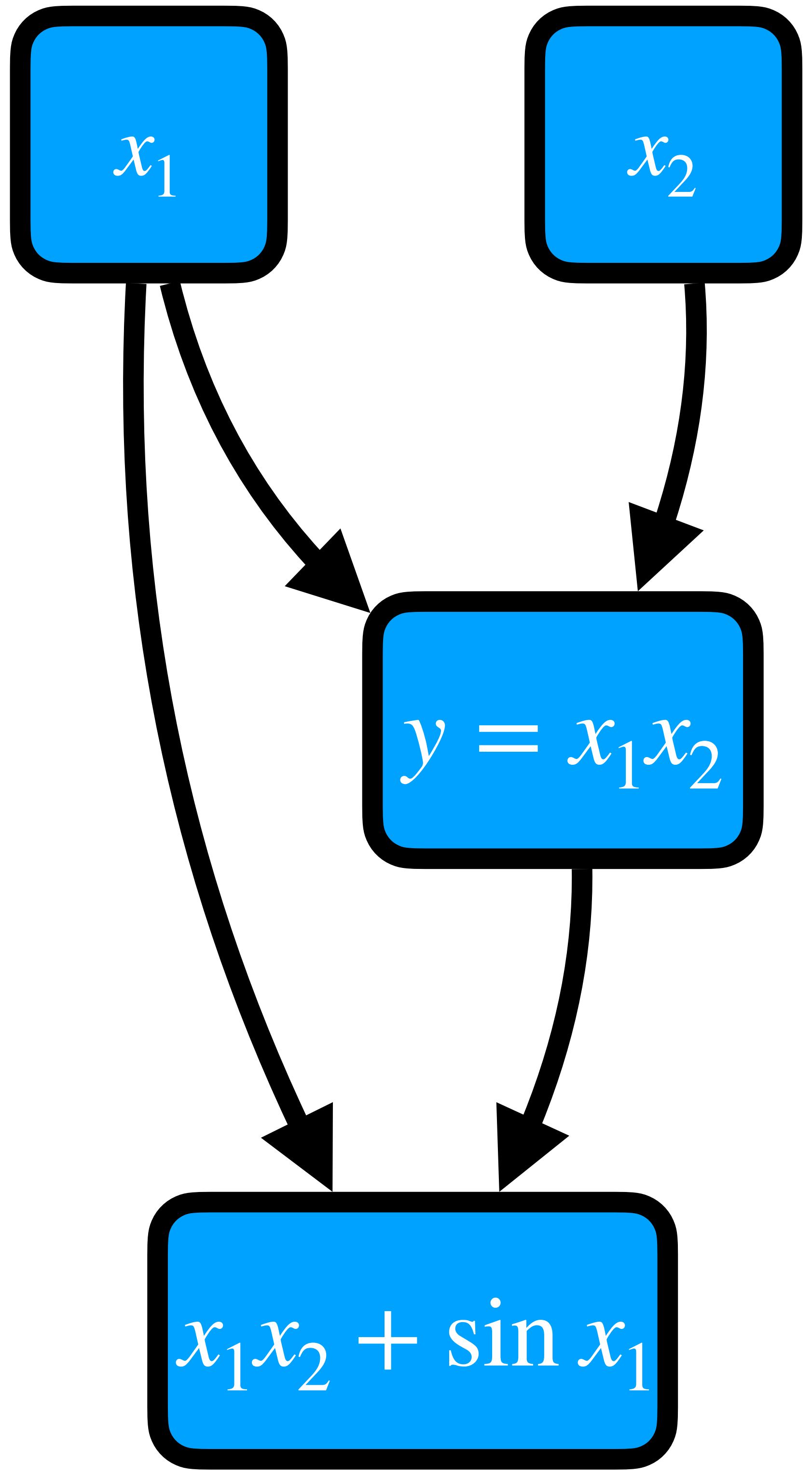




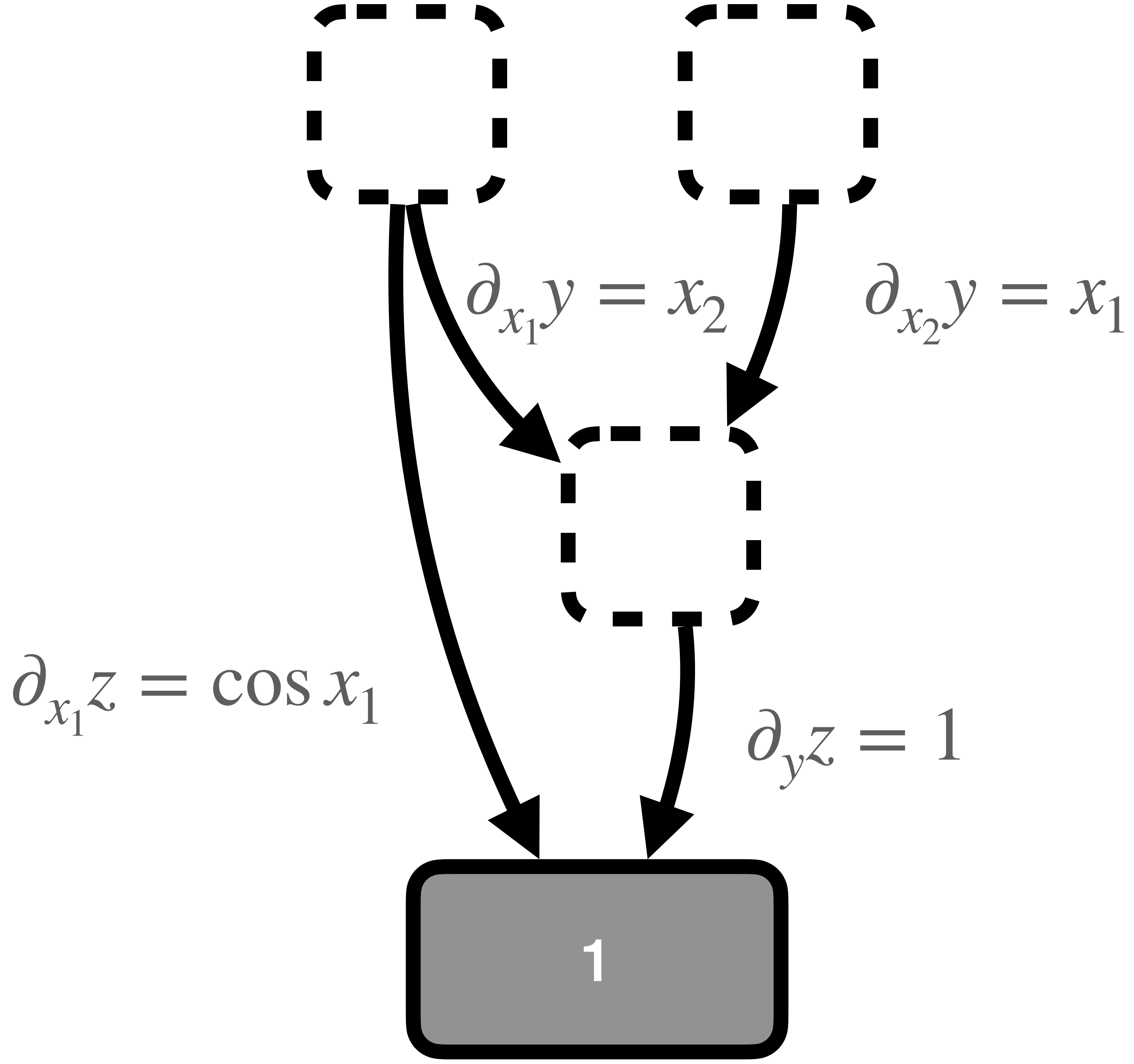
$$z = x_1 x_2 + \sin x_1 = y + \sin x_1$$

Example

Run backwards

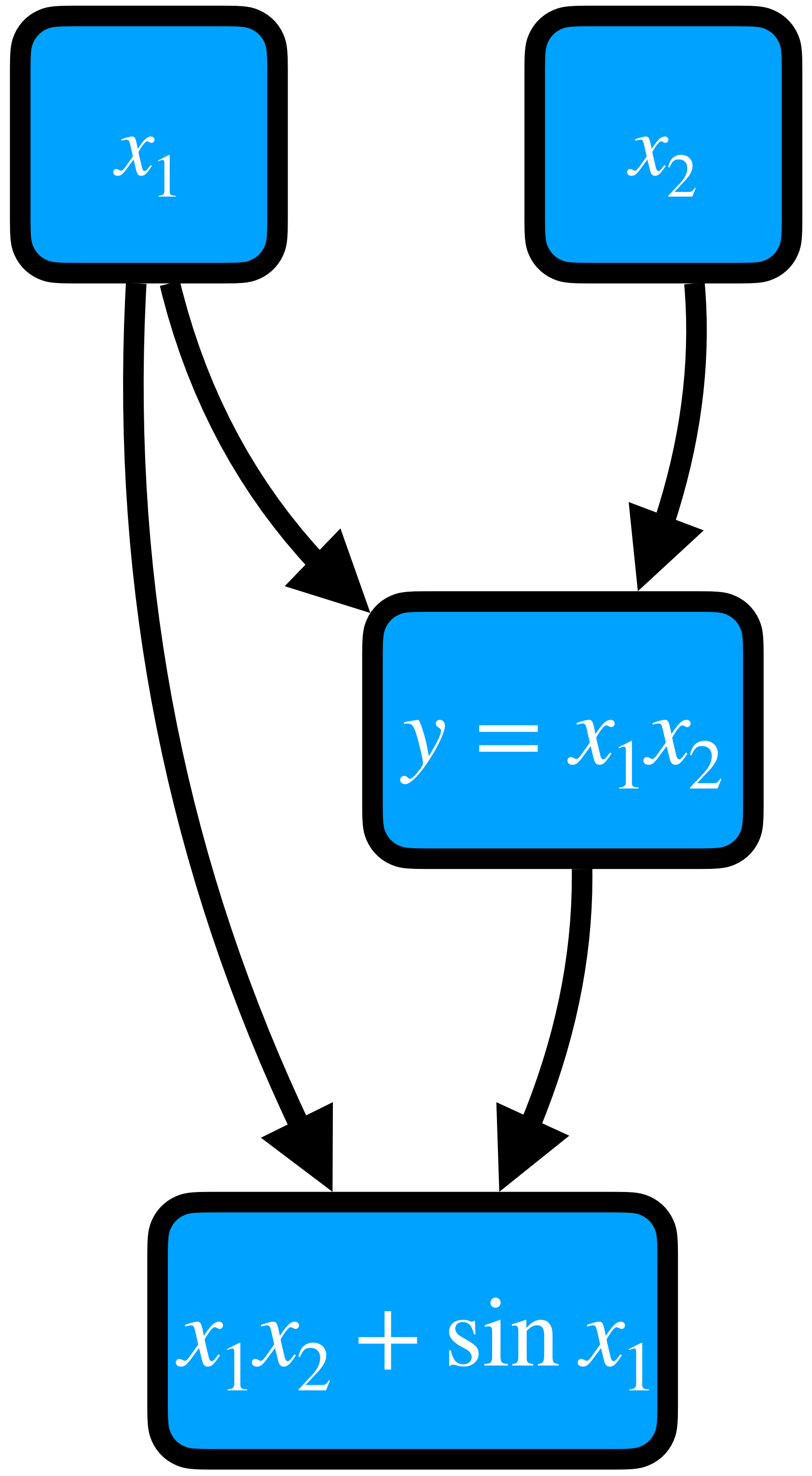


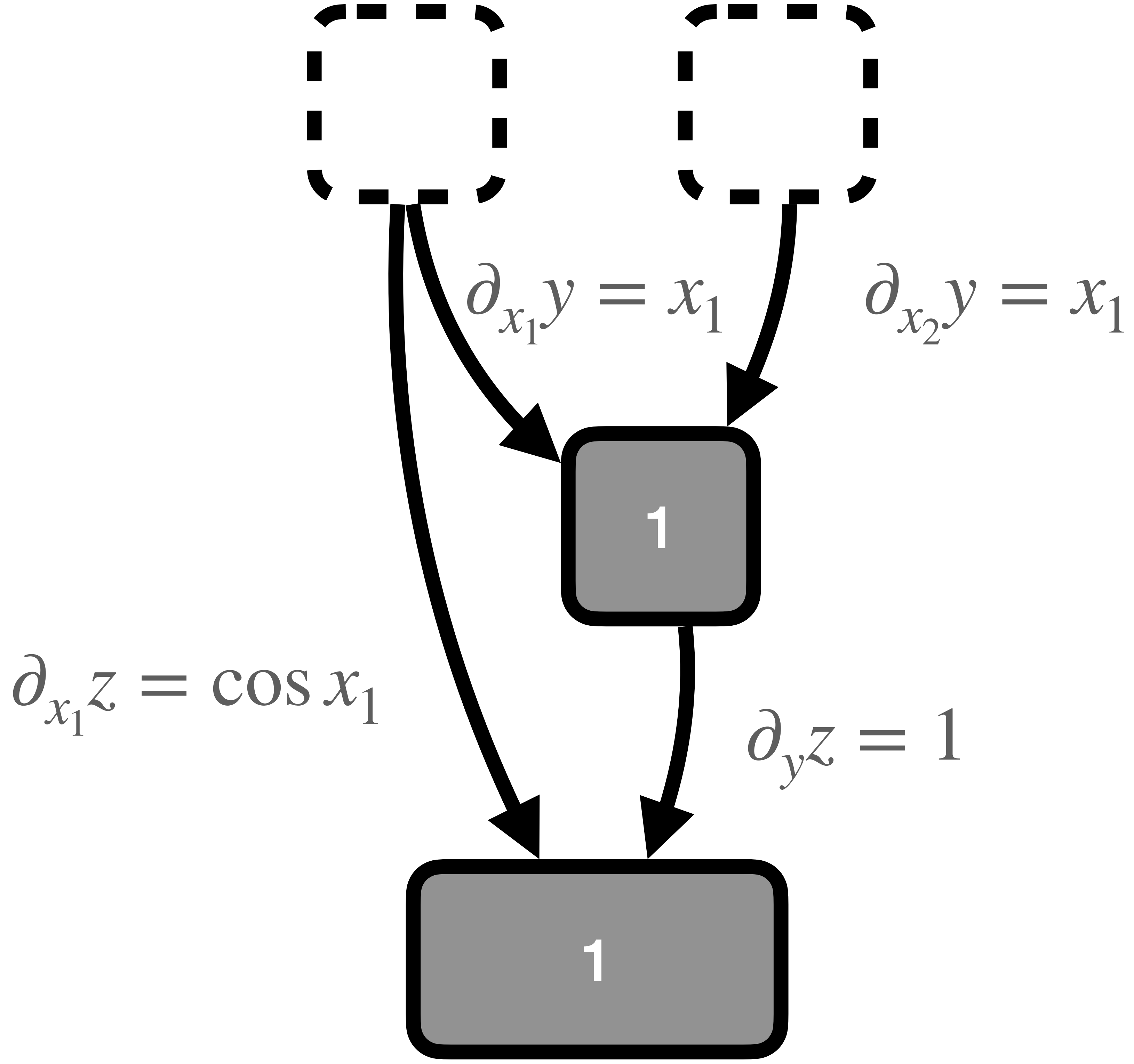
$$z = x_1 x_2 + \sin x_1 = y + \sin x_1$$



Example

Run backwards

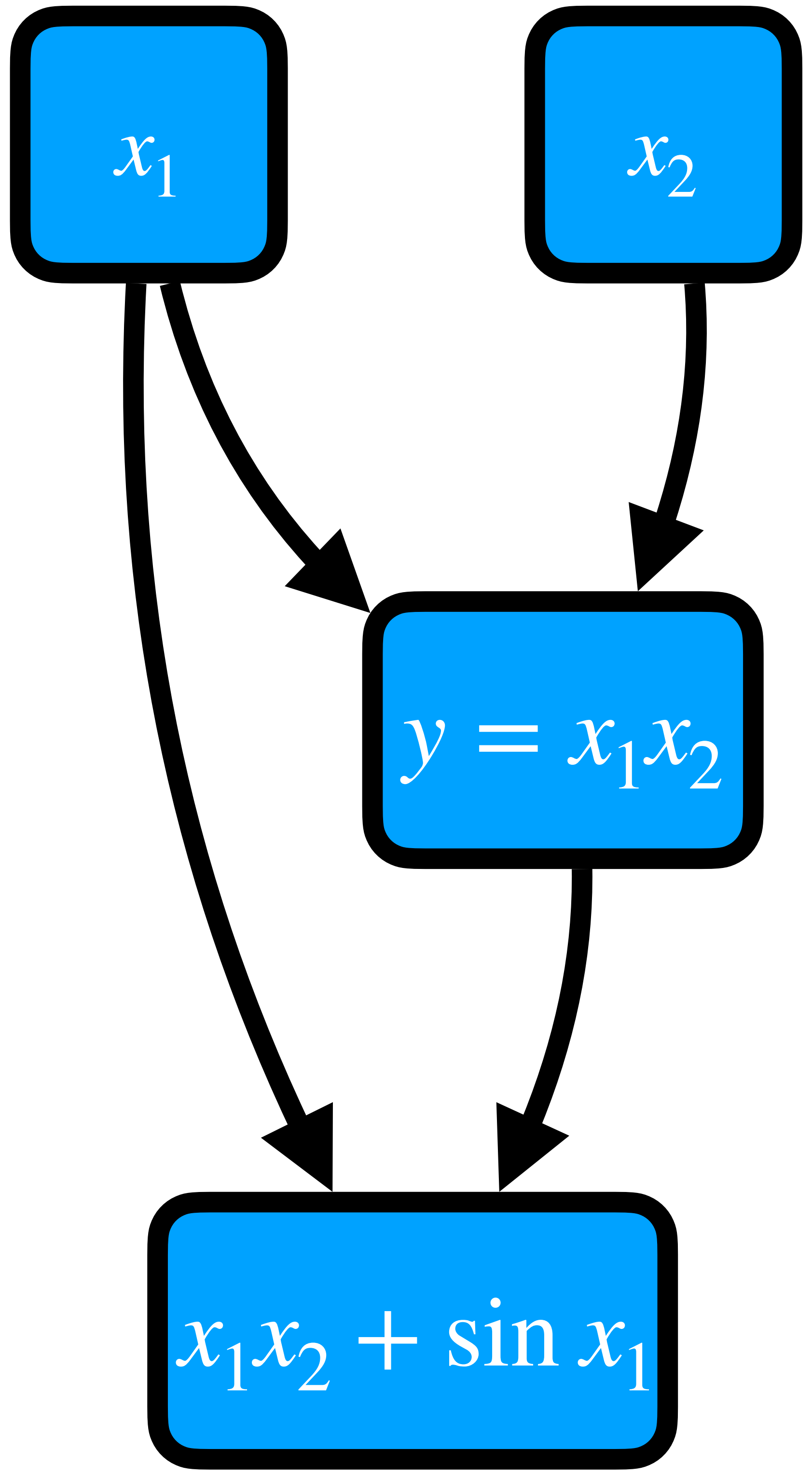




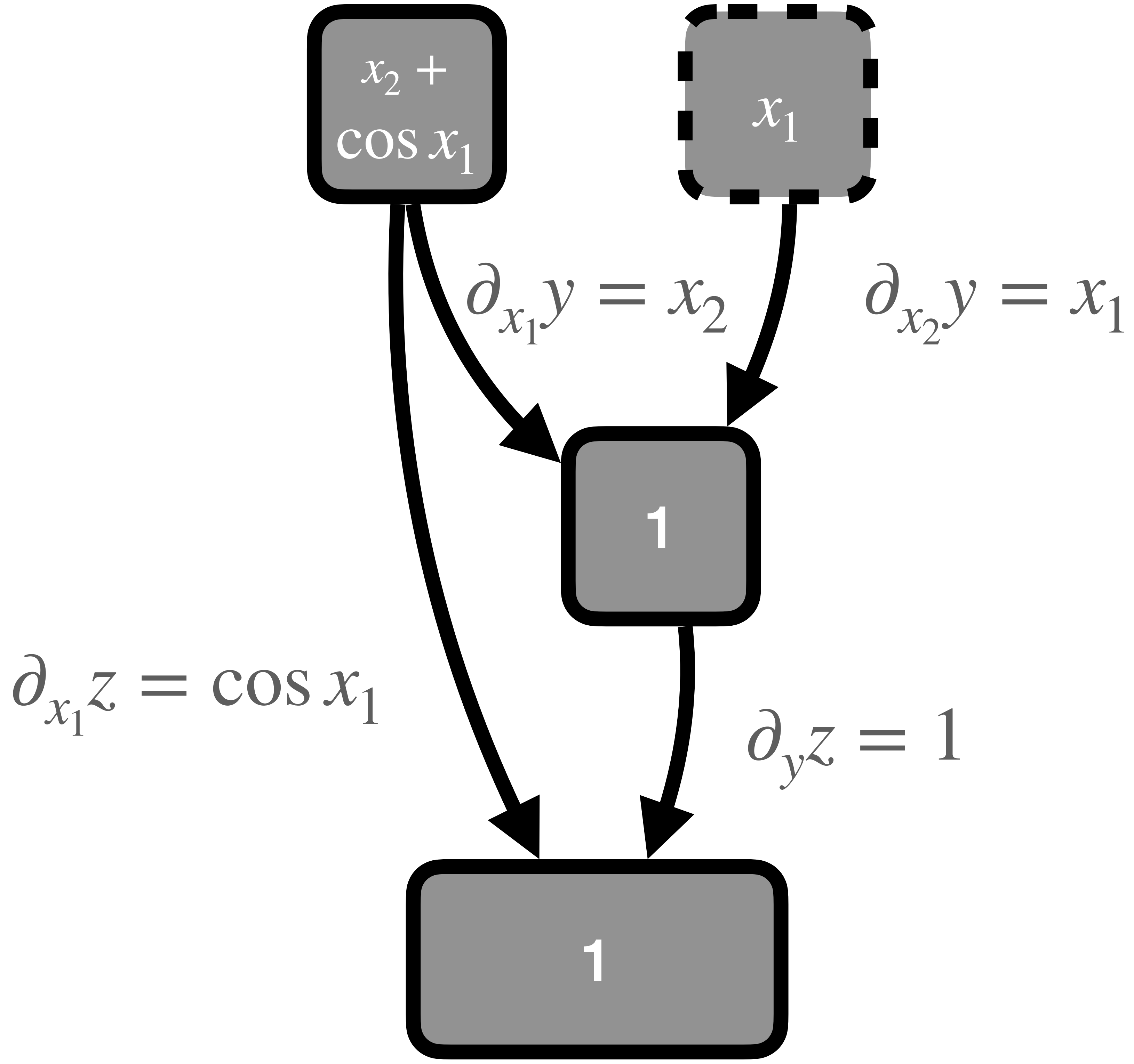
$$z = x_1 x_2 + \sin x_1 = y + \sin x_1$$

Example

Run backwards



$$z = x_1 x_2 + \sin x_1 = y + \sin x_1$$



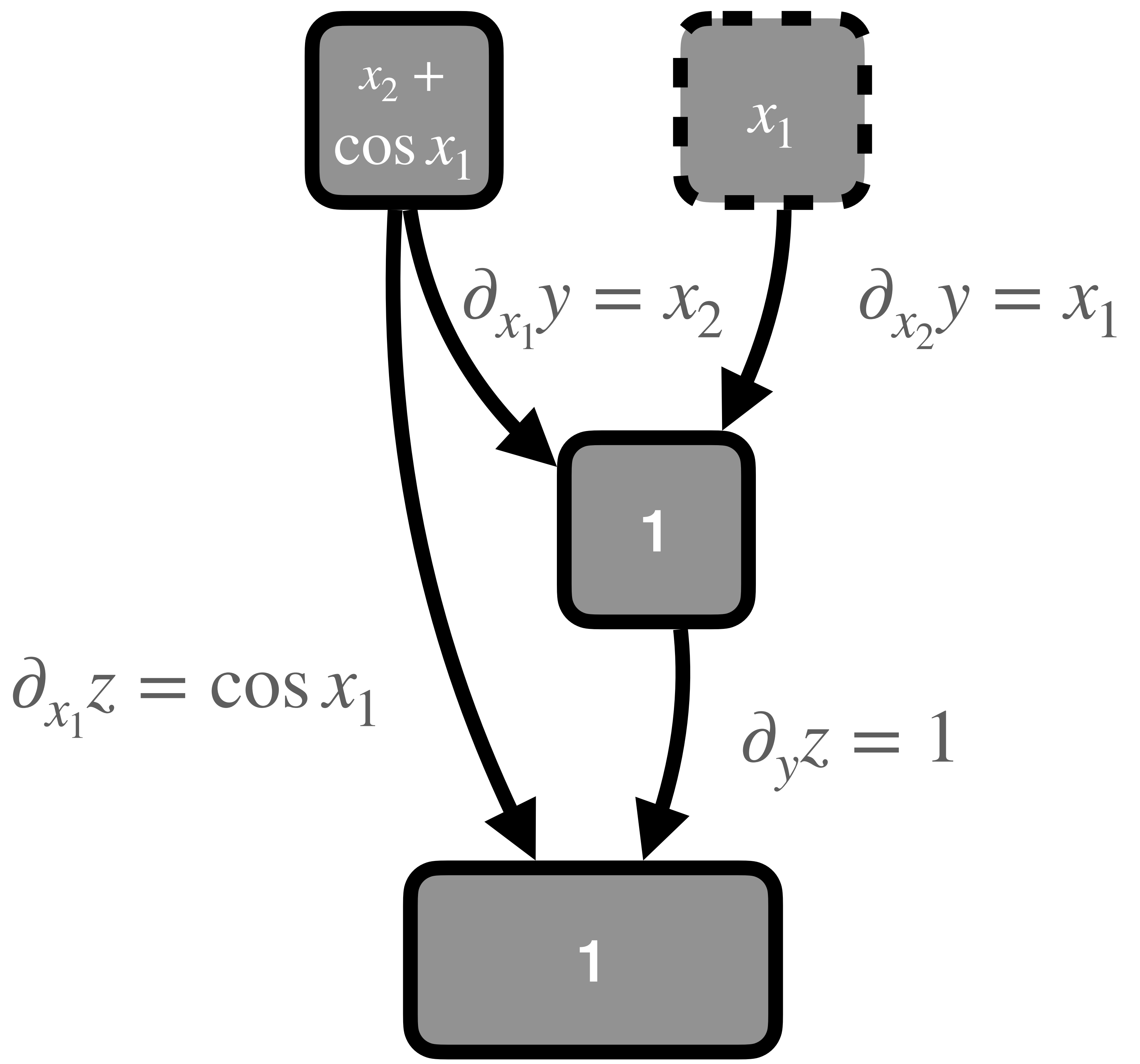
Example

Voilà!

$$z = x_1 x_2 + \sin x_1 = y + \sin x_1$$

$$\frac{\partial z}{\partial x_1} = x_2 + \cos x_1$$

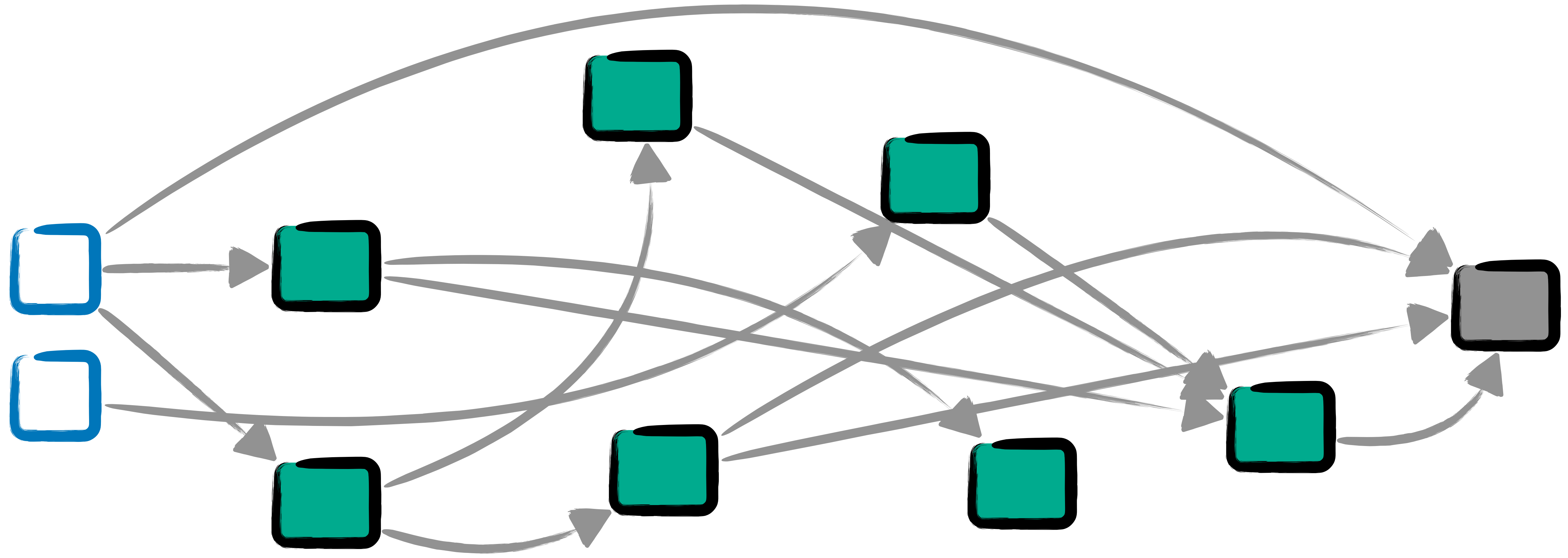
$$\frac{\partial z}{\partial x_2} = x_1$$



A Graph Picture of Matrix Multiplication

With the graph picture, we can generalize differentiation to arbitrary computation graphs, i.e. arbitrary programs!

∇_ϕ



$$a_i = (Jb)_i = \sum_{p \in \text{parents}(y)} \frac{\partial u_i}{\partial v_p} b_p$$

$$a_i = (b^T J)_i = \sum_{b \in \text{children}(z)} z_p \frac{\partial u_p}{\partial v_i}$$

Forward Propagation

Backward Propagation

Let's automate this!

We derived a strategy to efficiently compute the gradient of any given well-defined program

But of course, we don't want to do this ourselves

- in any case this is just mechanically putting together some JVP, VJP functions
- it's something a computer can do!

Automatic Differentiation (AD)

Automatic Differentiation Systems

Most Deep Learning Framework are at their core Autodiff systems

- Differentiable Programming Languages first, ML Stuff second

TensorFlow

PYTORCH

Beyond Deep Learning

But there is a long list of non-DL focused AD frameworks as well

- idea exist in many language (C++, Julia, Fortran, ...)

autodiff

automatic differentiation in C++ couldn't be simpler

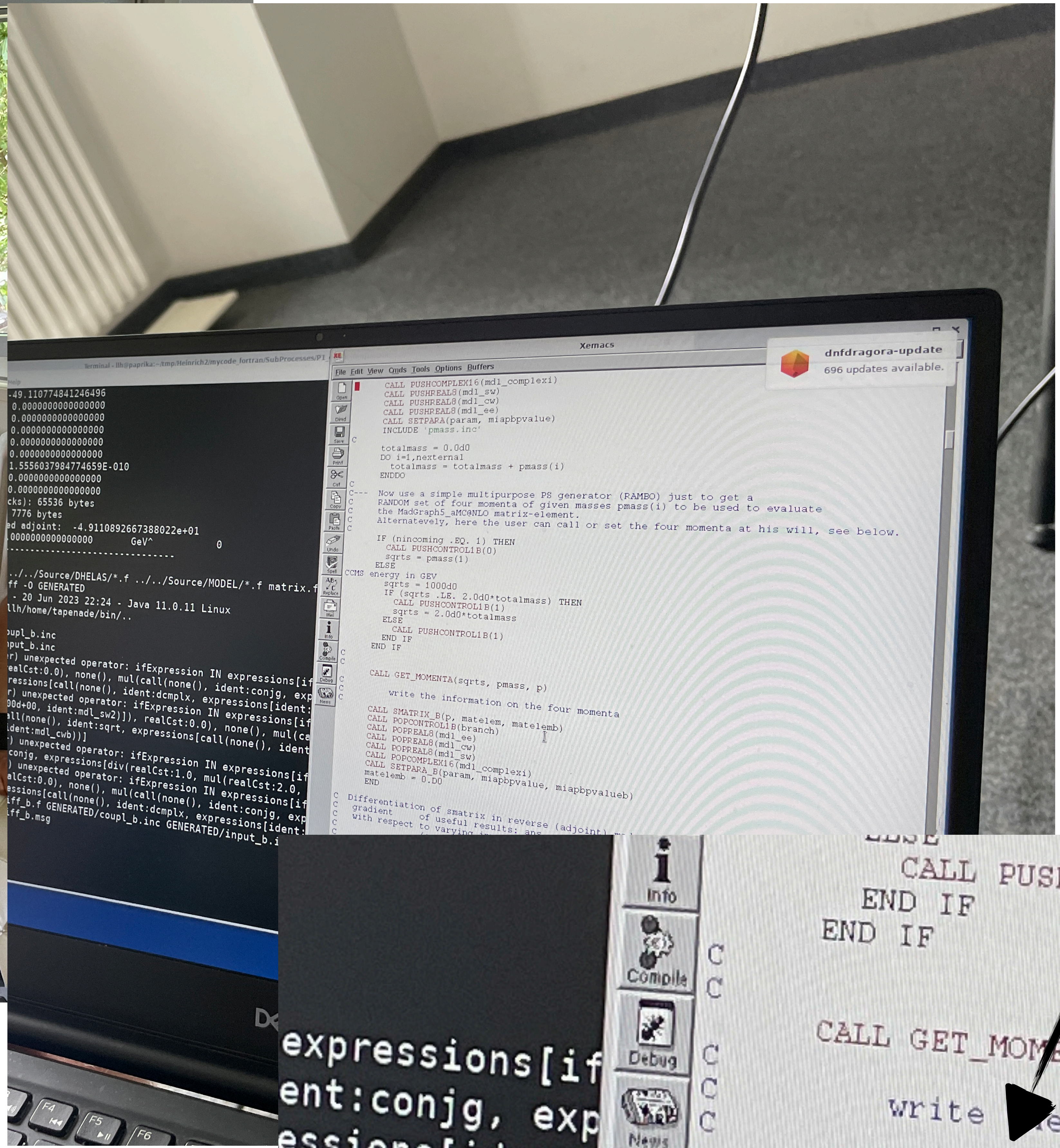
autodiff (C++)

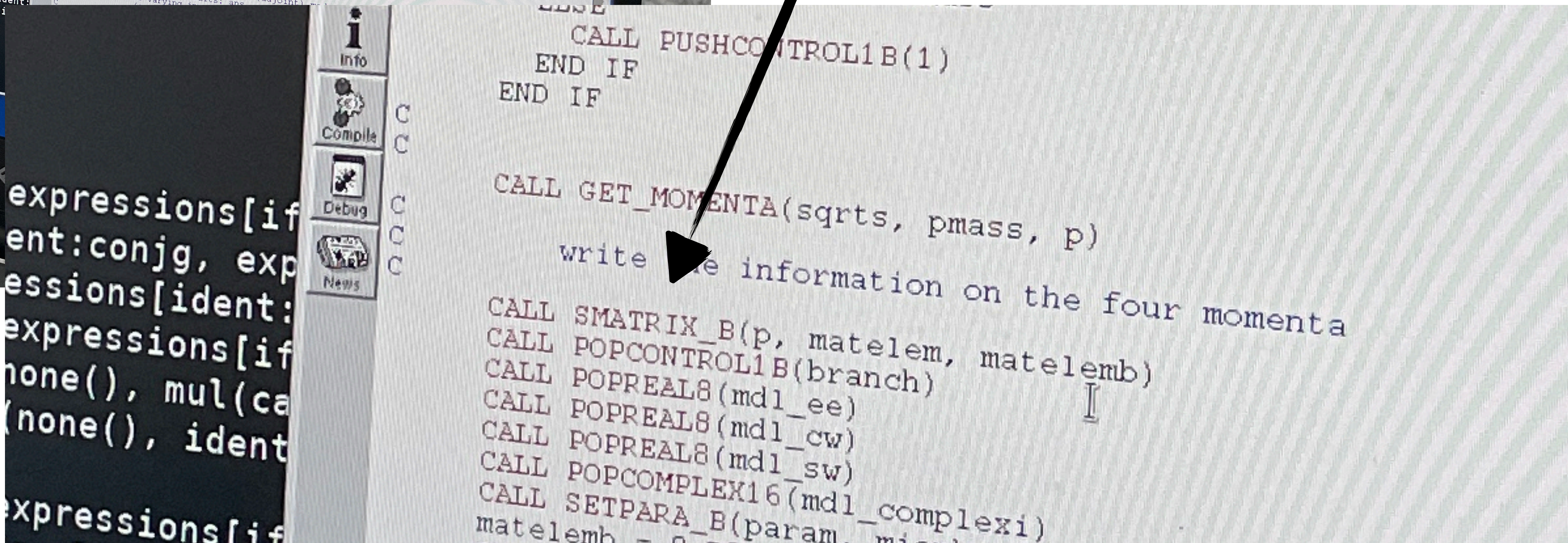
Enzyme.jl (Julia)

Remember: ML kind of rediscovered AD on its own

Beyond Deep Learning

Munich 2023





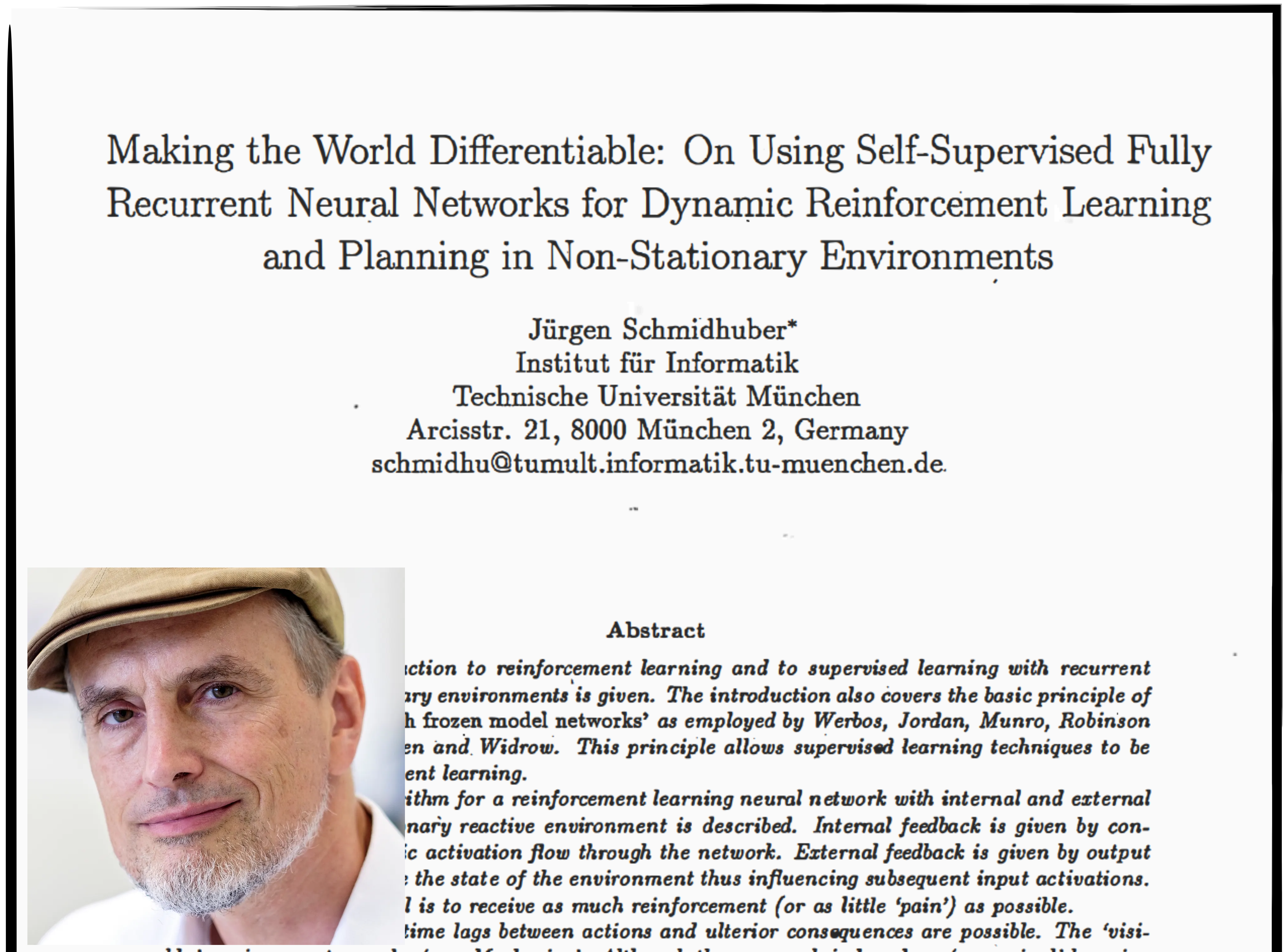
Differentiable MadGraph in FORTRAN

A general pattern

Differentiable Programming is a generic way to introduce physics structure into ML without losing learnability

But most of our code is not (yet) differentiable

A program for the next few years



Making the World Differentiable: On Using Self-Supervised Fully Recurrent Neural Networks for Dynamic Reinforcement Learning and Planning in Non-Stationary Environments

Jürgen Schmidhuber*
Institut für Informatik
Technische Universität München
Arcisstr. 21, 8000 München 2, Germany
schmidhu@tumult.informatik.tu-muenchen.de

Abstract

...ction to reinforcement learning and to supervised learning with recurrent ...ary environments is given. The introduction also covers the basic principle of ...h frozen model networks' as employed by Werbos, Jordan, Munro, Robinson ...en and Widrow. This principle allows supervised learning techniques to be ...ent learning.
...ithm for a reinforcement learning neural network with internal and external ...nary reactive environment is described. Internal feedback is given by con-...ic activation flow through the network. External feedback is given by output ...the state of the environment thus influencing subsequent input activations. ...l is to receive as much reinforcement (or as little 'pain') as possible. ...time lags between actions and ulterior consequences are possible. The 'visi-

End.