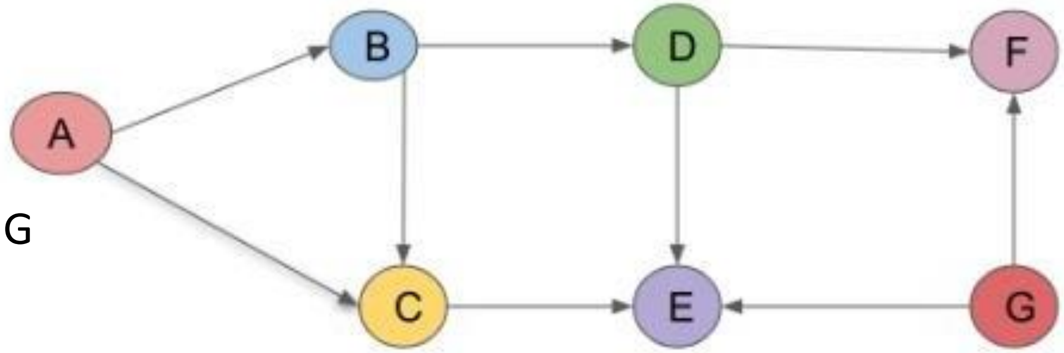
Topological Sort : backprop the correct way

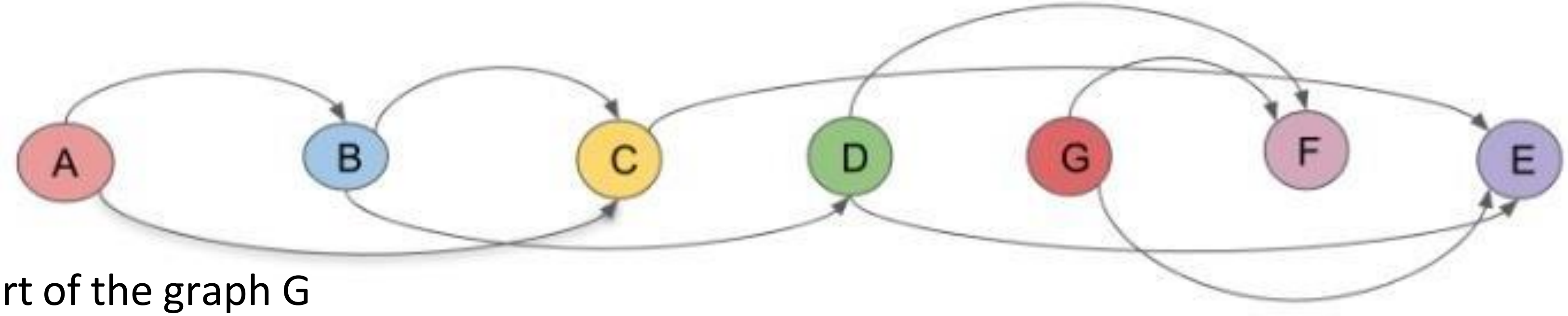
Mustafa Hajij

Topological Sort :

Of a directed graph is a linear ordering of its vertices such that for every directed edge uv from vertex u to vertex v, u comes before v in the ordering.



G



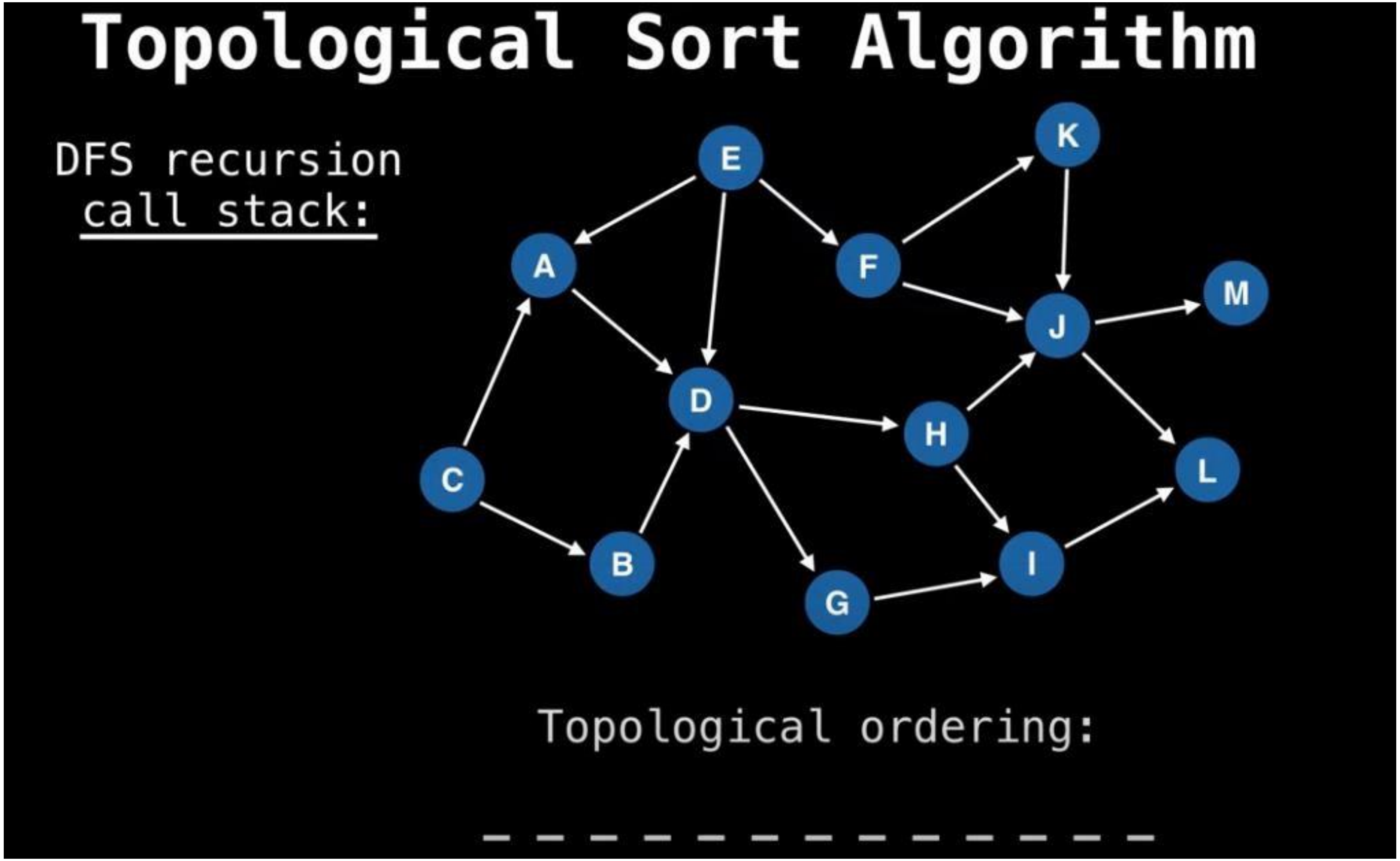
A topological sort of the graph G

Application :

1-the vertices of the graph may represent tasks to be performed, and the edges may represent constraints that one task must be performed before another.

2-backprob (today)

Topological Sort :

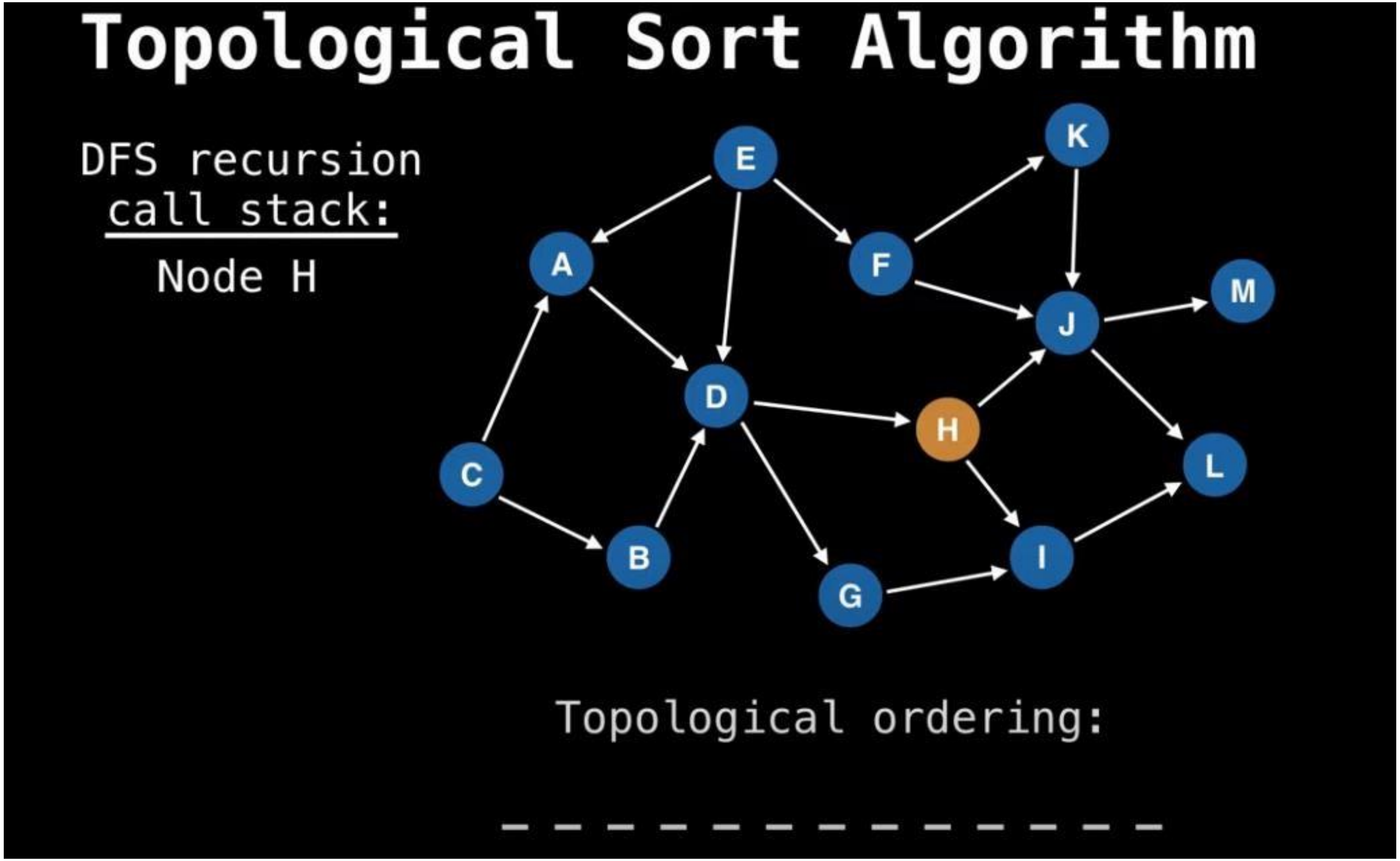


Pick any node (say H)

Then do DFS

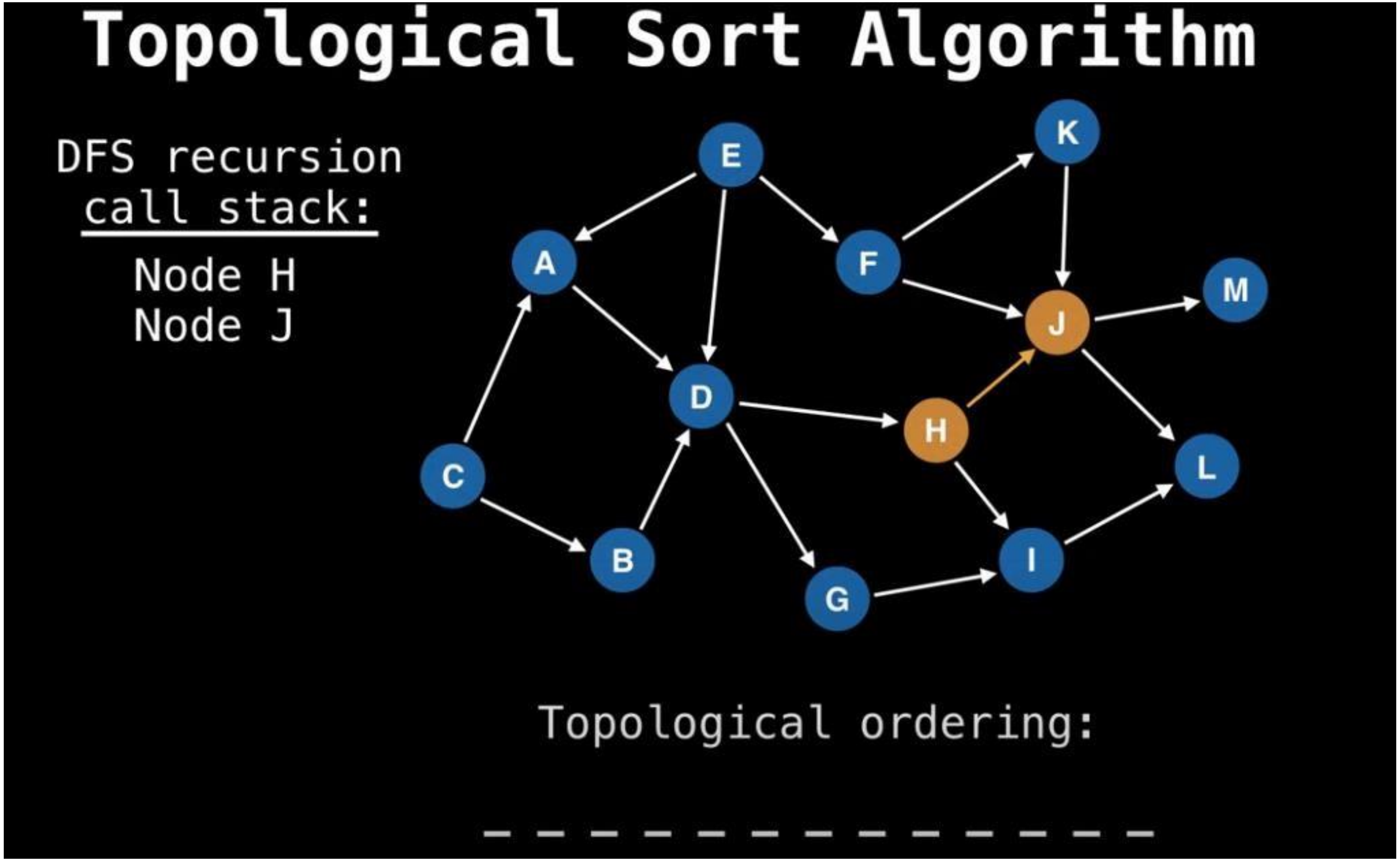
Example ref: https://www.youtube.com/watch?v=eL-KzMXSXXI

Topological Sort :



Pick any node
(say H)

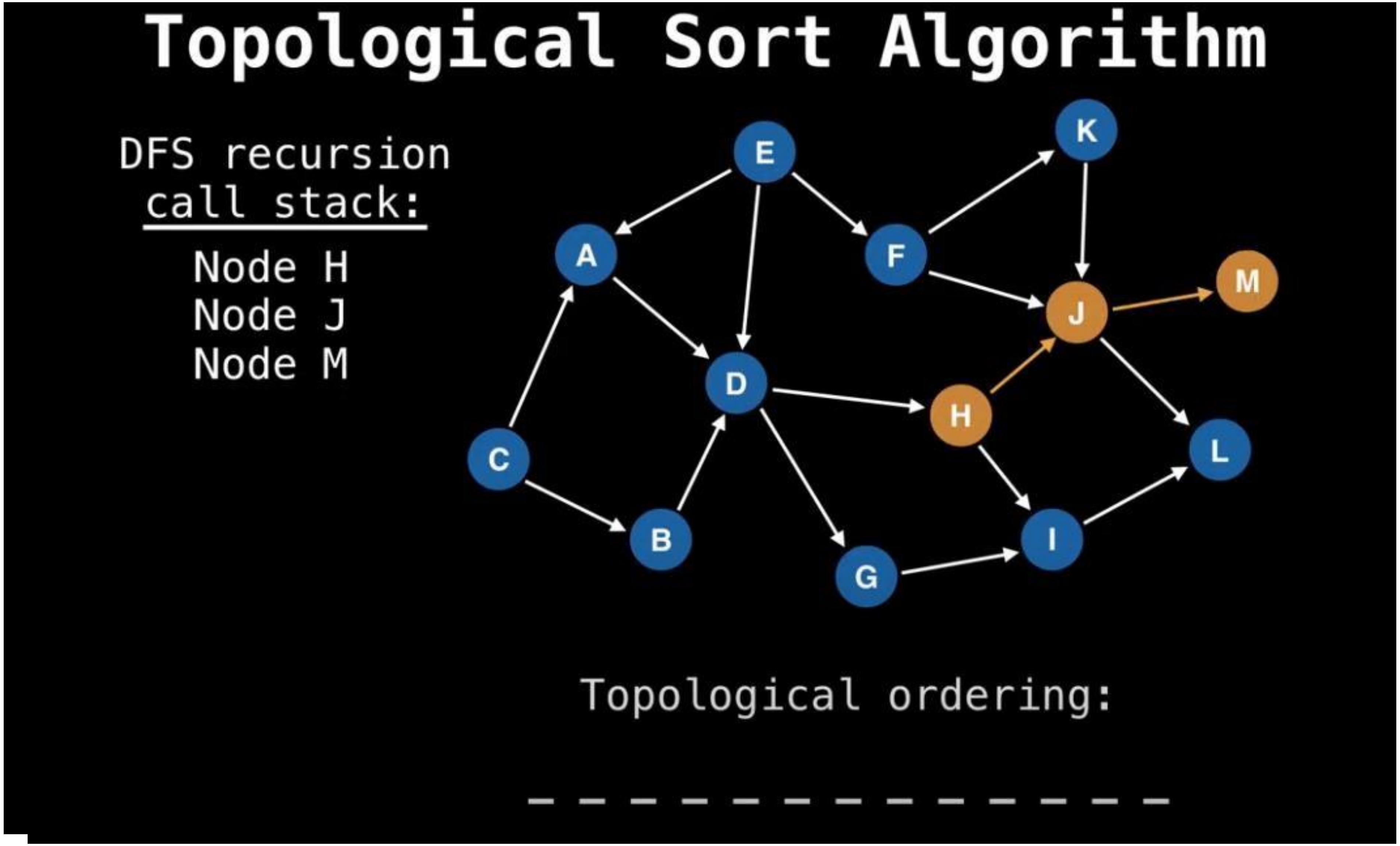
Topological Sort :



Pick any node (say H)

Then do DFS

Topological Sort :



Pick any node (say H)

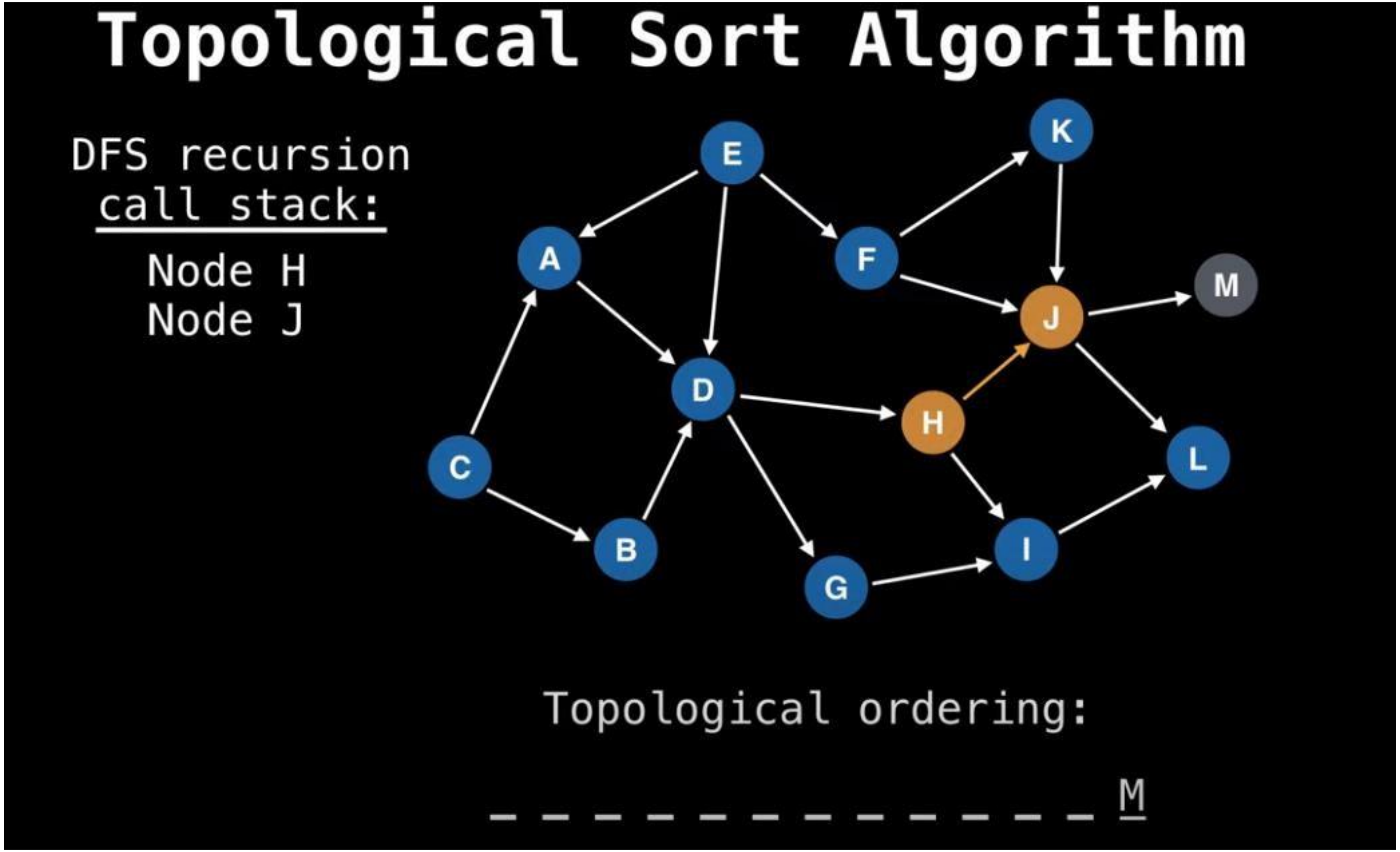
Then do DFS

Arrive at the end

of a path,

backtrack..

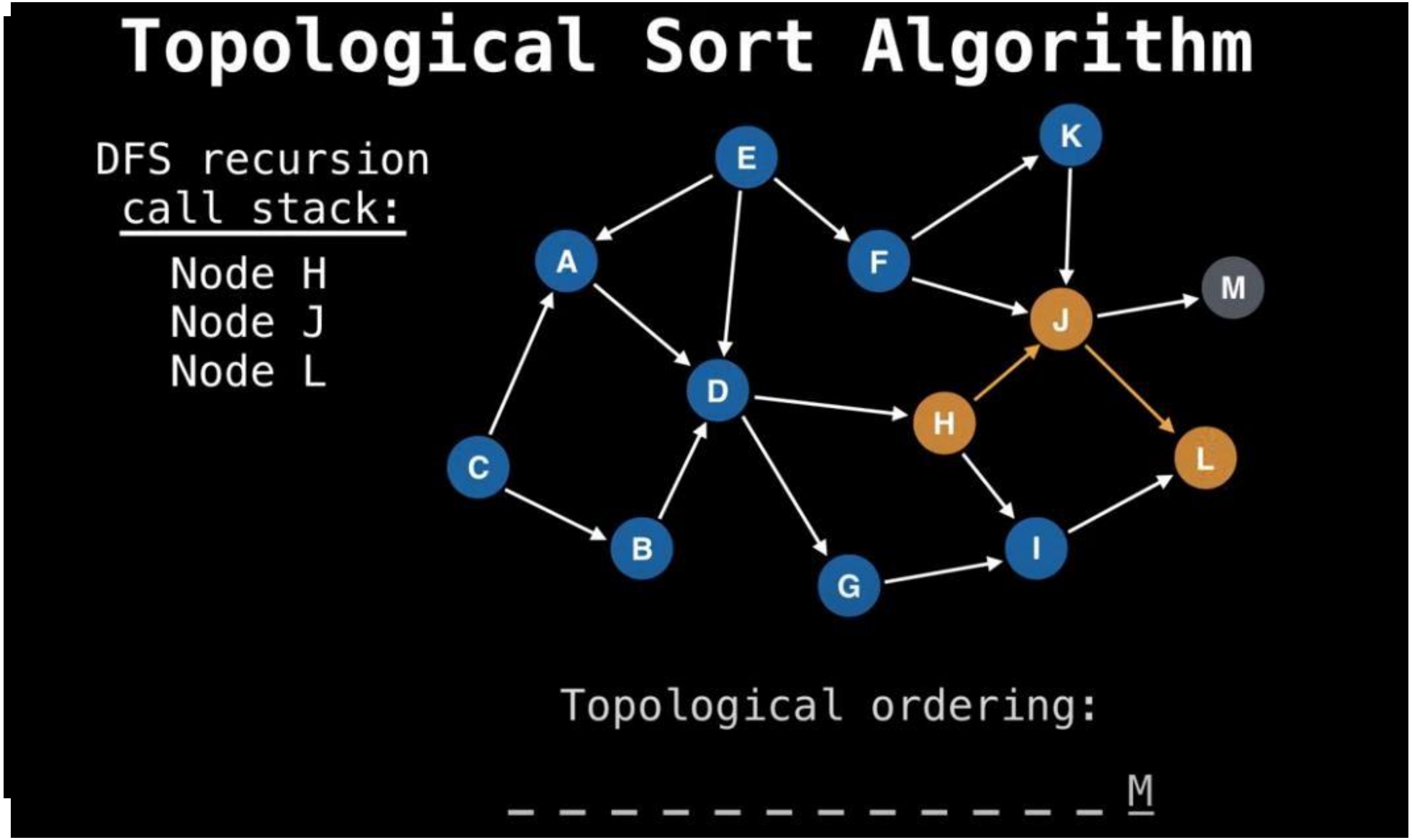
Topological Sort :



Pick any node (say H)

Then do DFS

Topological Sort :



Pick any node (say H)

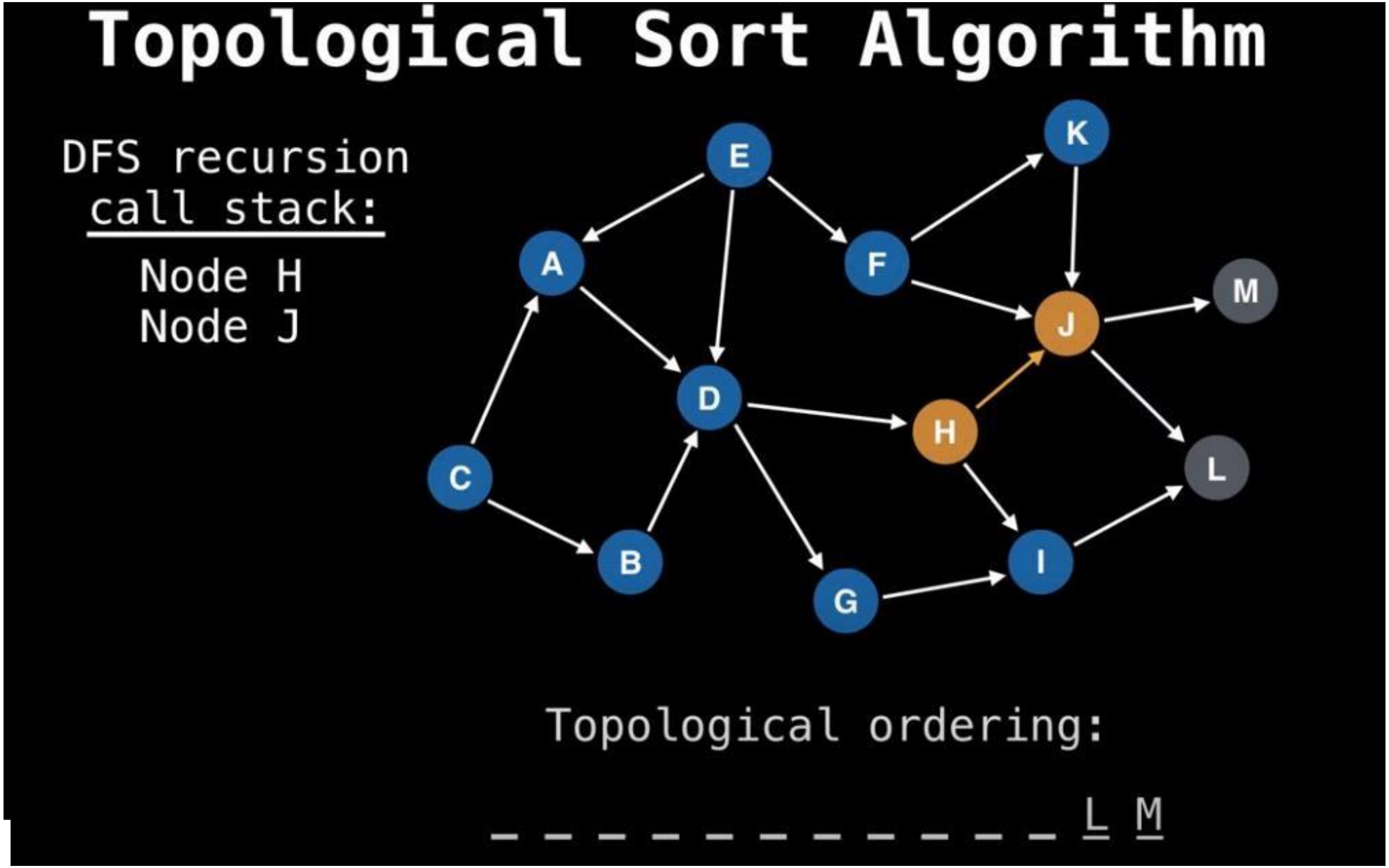
Then do DFS

Arrive at the end

of a path,

backtrack..

Topological Sort :

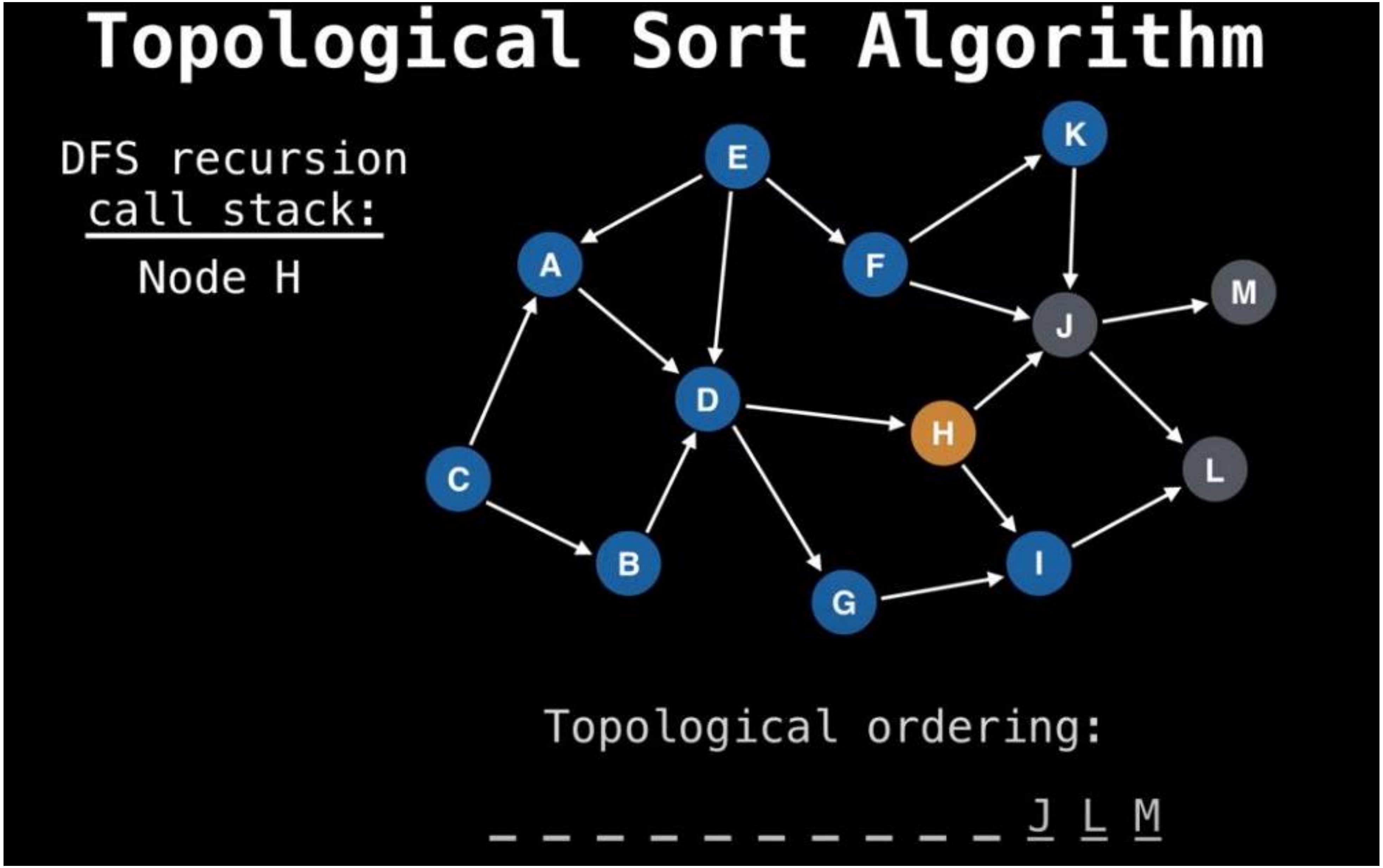


Pick any node (say H)

Then do DFS

There are no other nodes to visit to backtrack

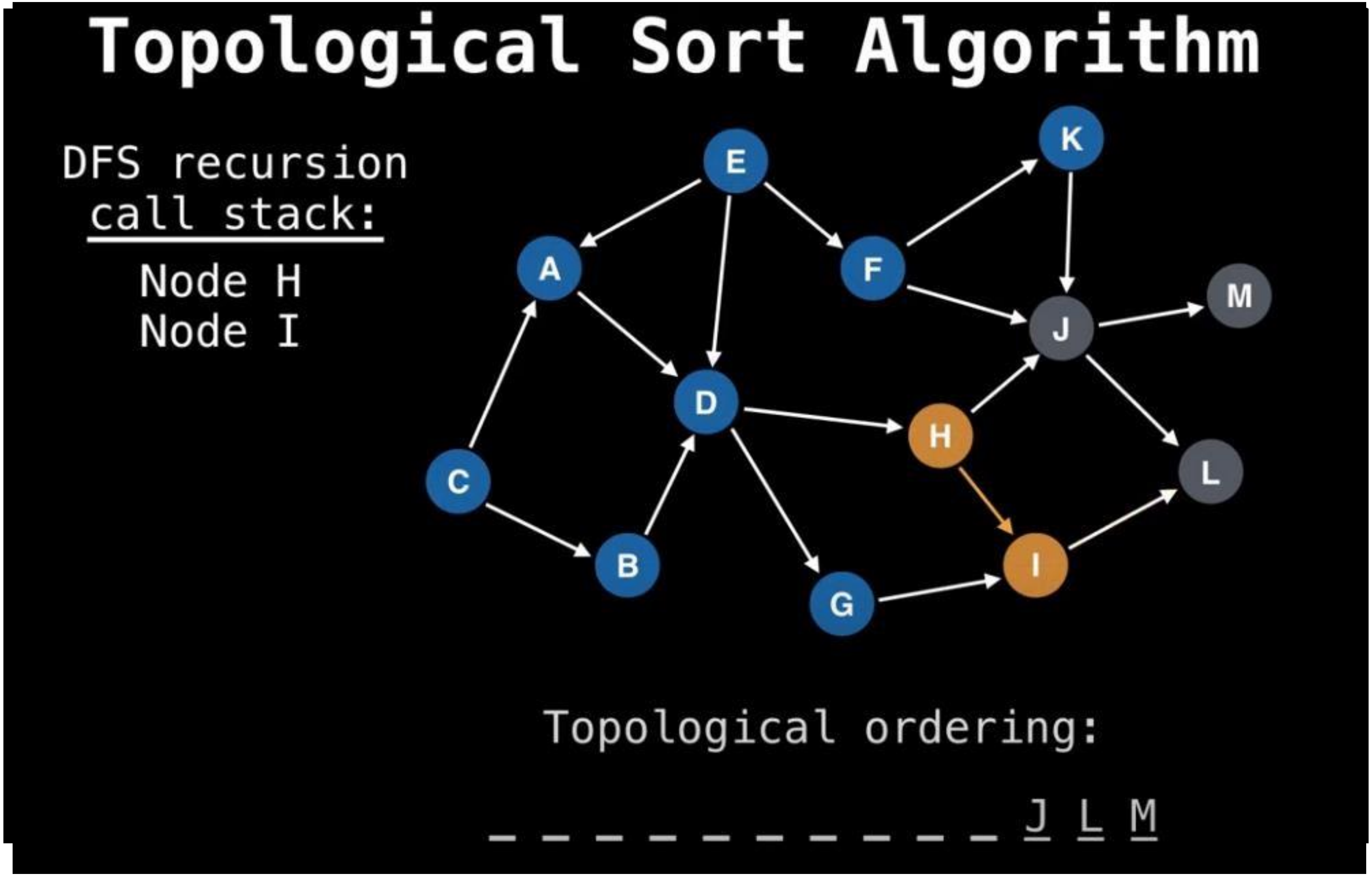
Topological Sort :



Pick any node (say H)

Then do DFS

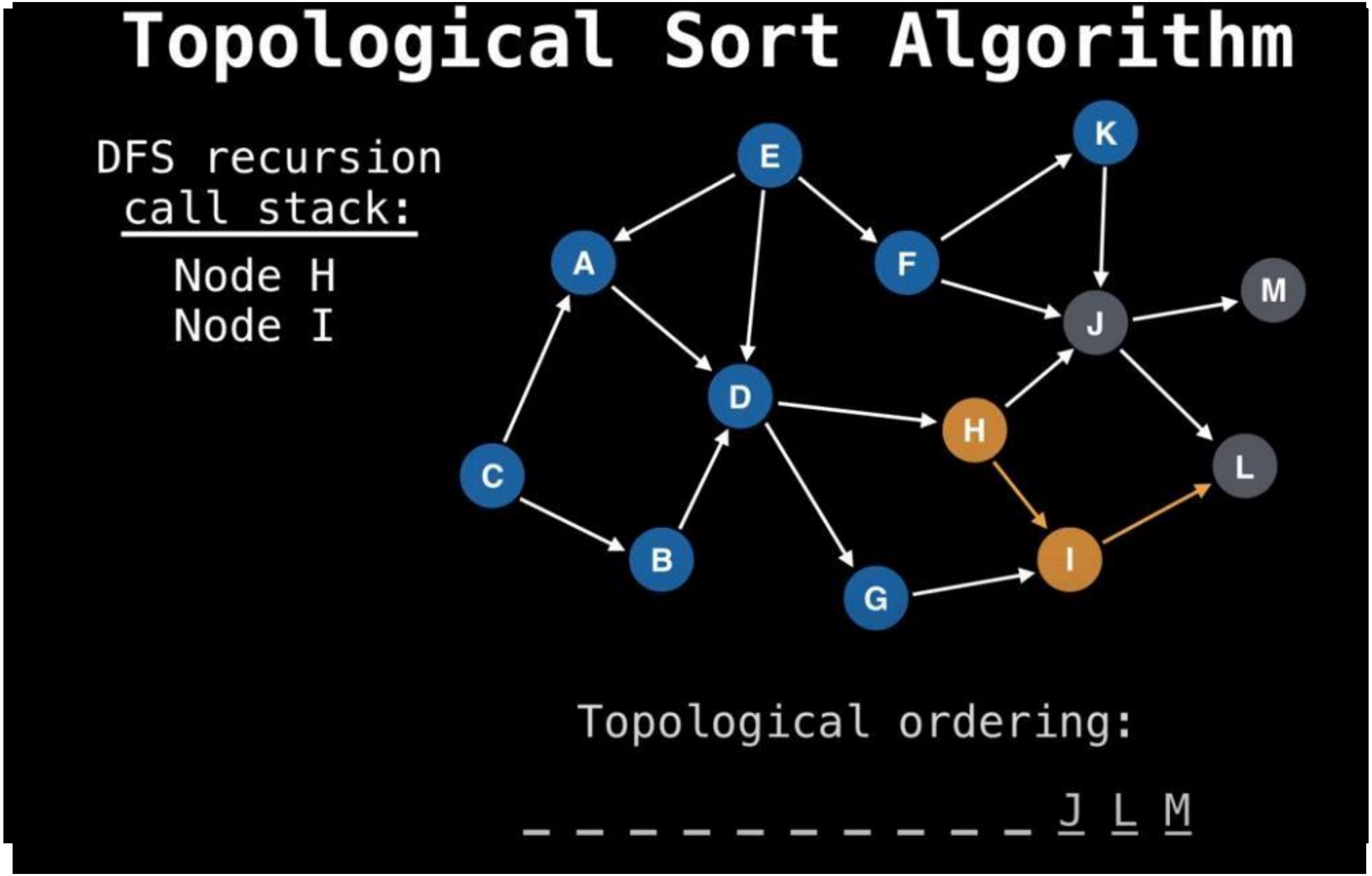
Topological Sort :



Pick any node (say H)

Then do DFS

Topological Sort :

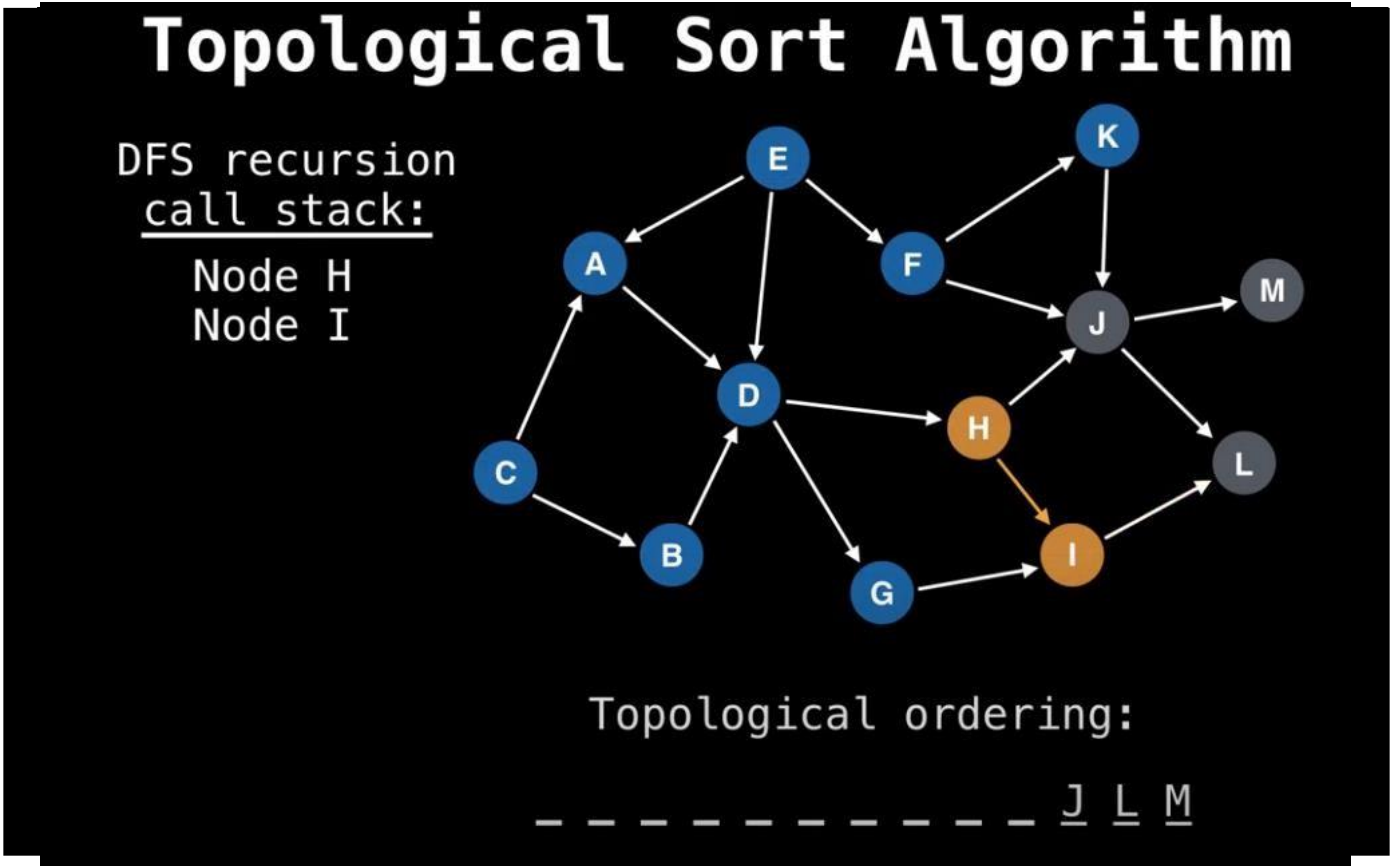


Pick any node (say H)

Then do DFS

We already visited L

Topological Sort :



Pick any node (say H)

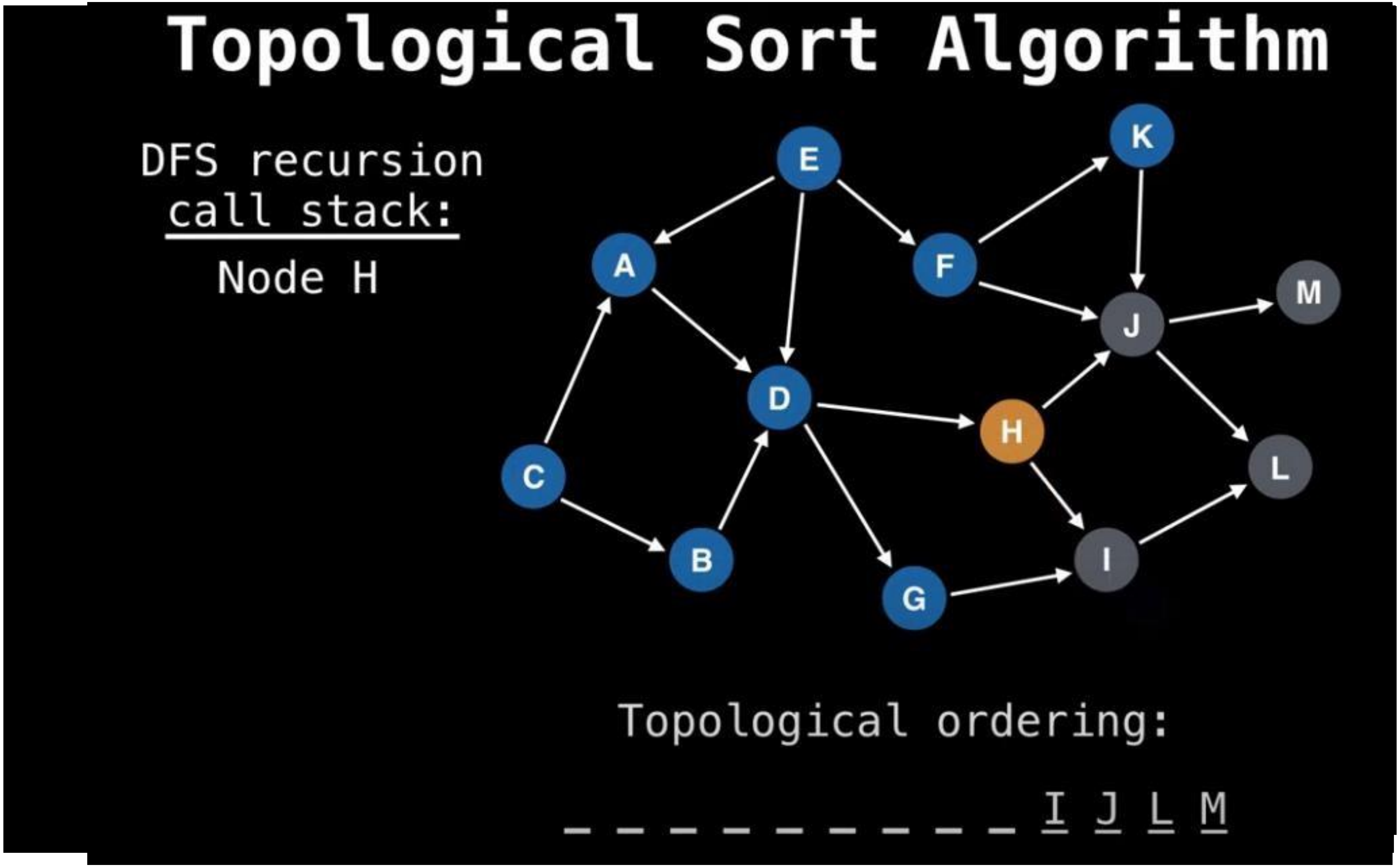
Then do DFS

Arrive at the end

of a path,

backtrack..

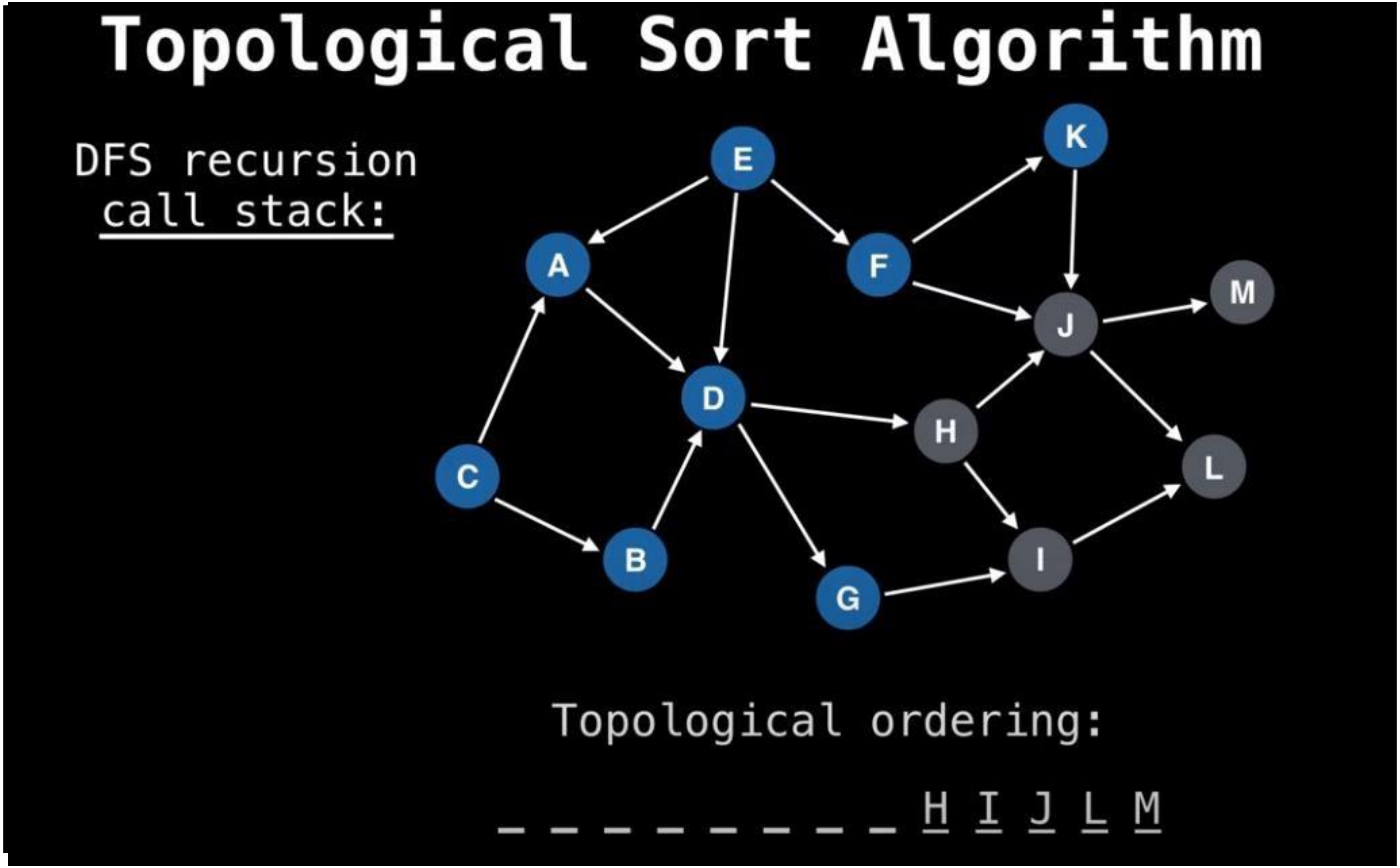
Topological Sort :



Pick any node (say H)

Then do DFS

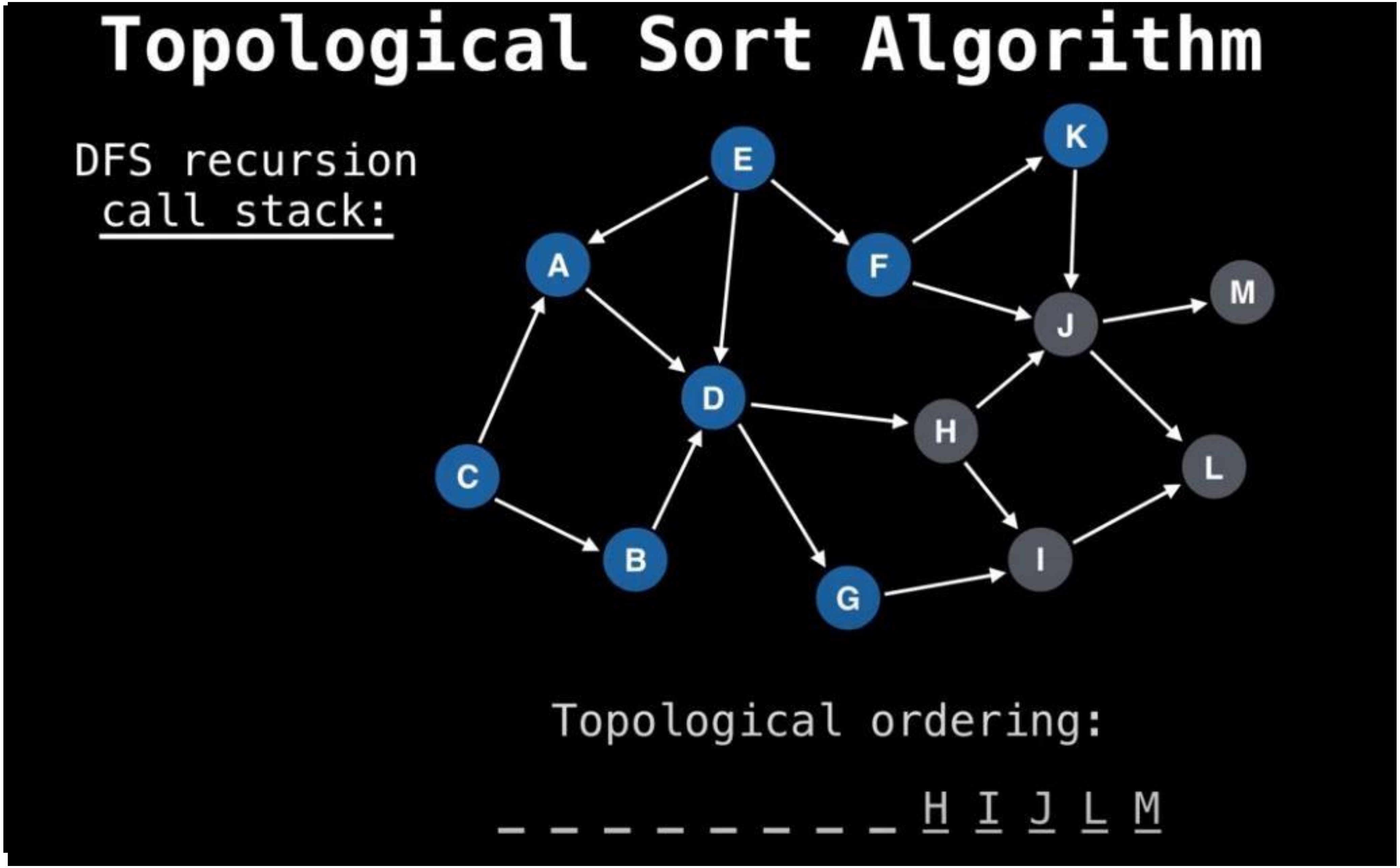
Topological Sort :



Pick any node (say H)

Then do DFS

Topological Sort :



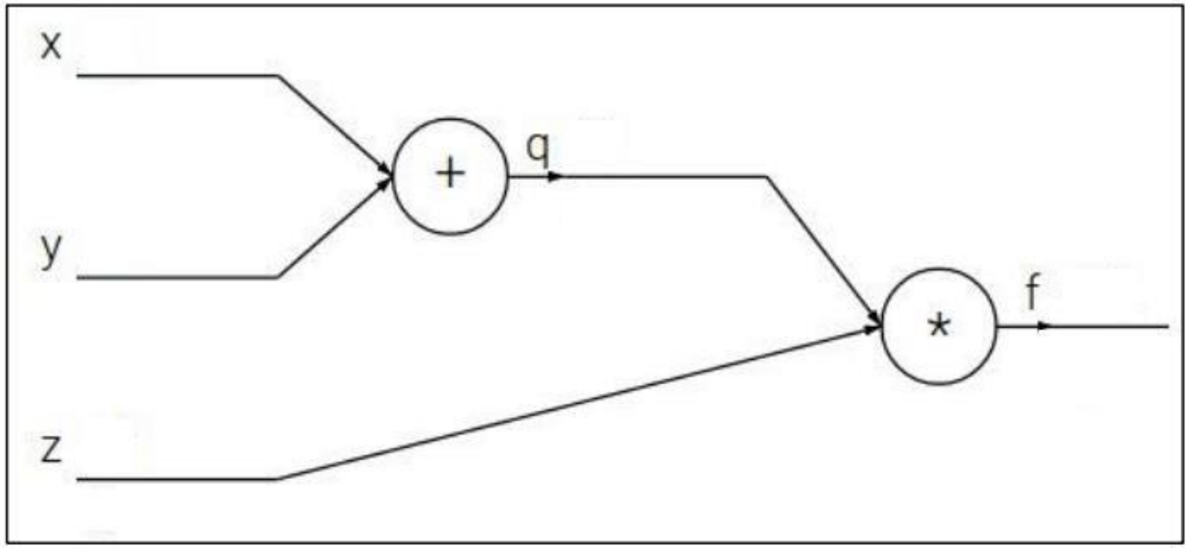
Pick any node (say H)

Then do DFS

Repeat the process until you visit all nodes

Computation graph

$$f(x, y, z) = (x + y)z$$

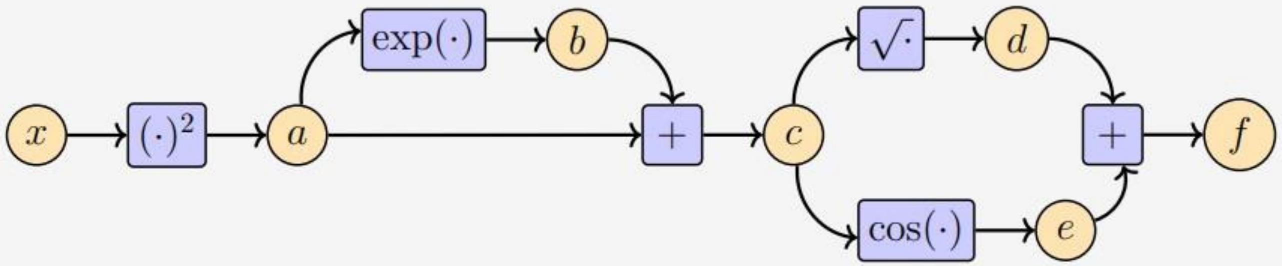


Computation graph of f

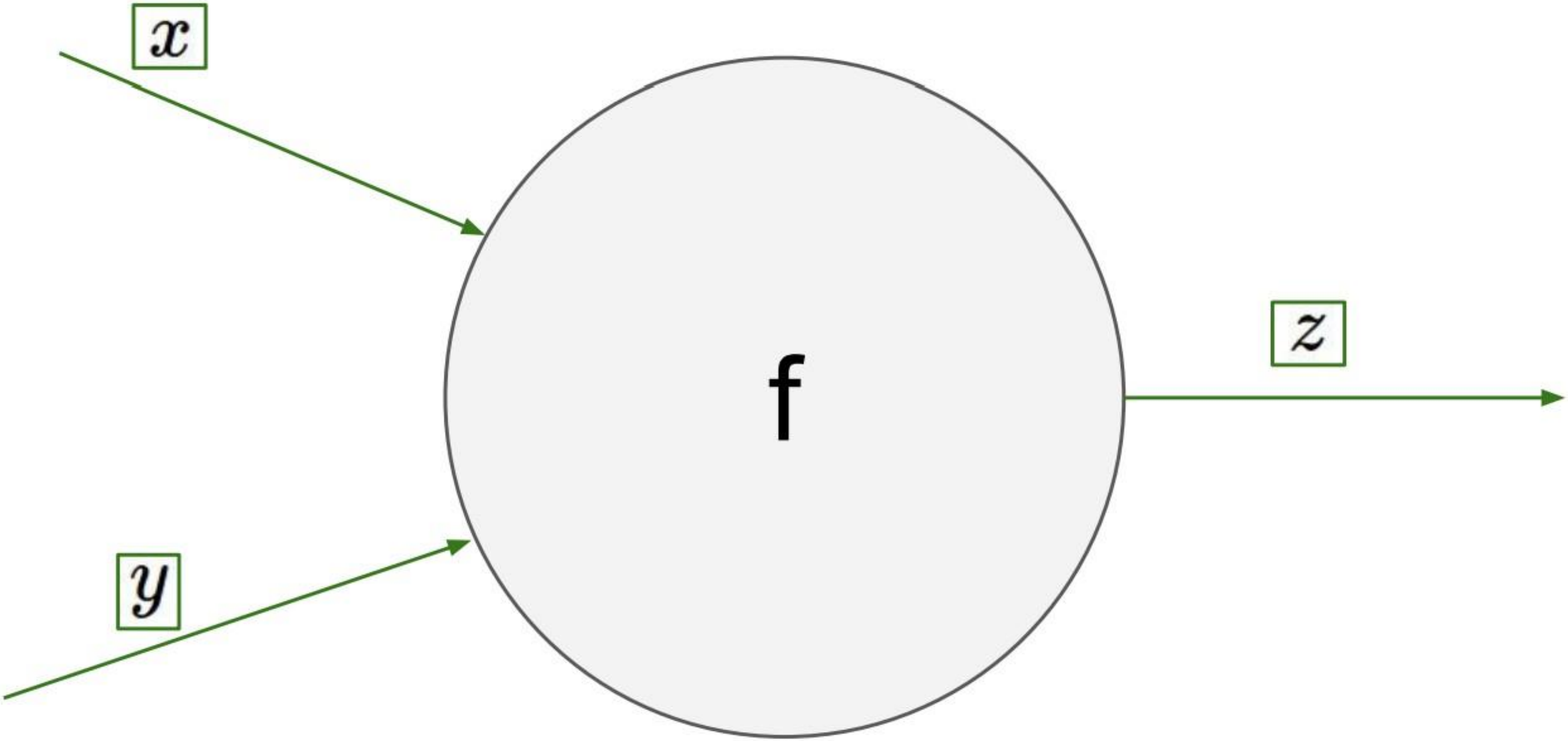
Computation graph

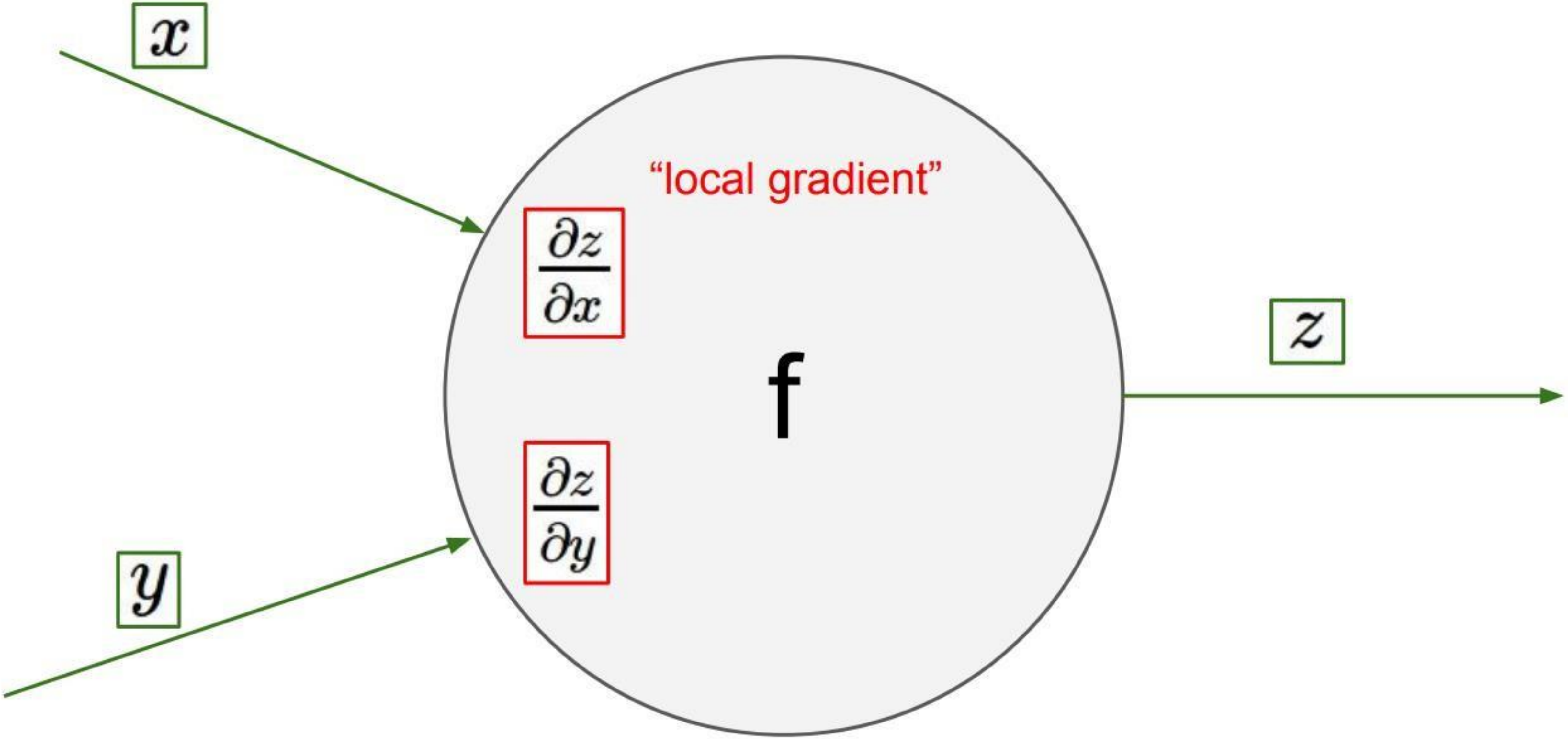
$$f(x) = \sqrt{x^2 + \exp(x^2)} + \cos\left(x^2 + \exp(x^2)\right)$$

$$\begin{aligned}
a &= x^2\,, \\
b &= \exp(a)\,, \\
c &= a + b\,, \\
d &= \sqrt{c}\,, \\
e &= \cos(c)\,, \\
f &= d + e\,.
\end{aligned}$$

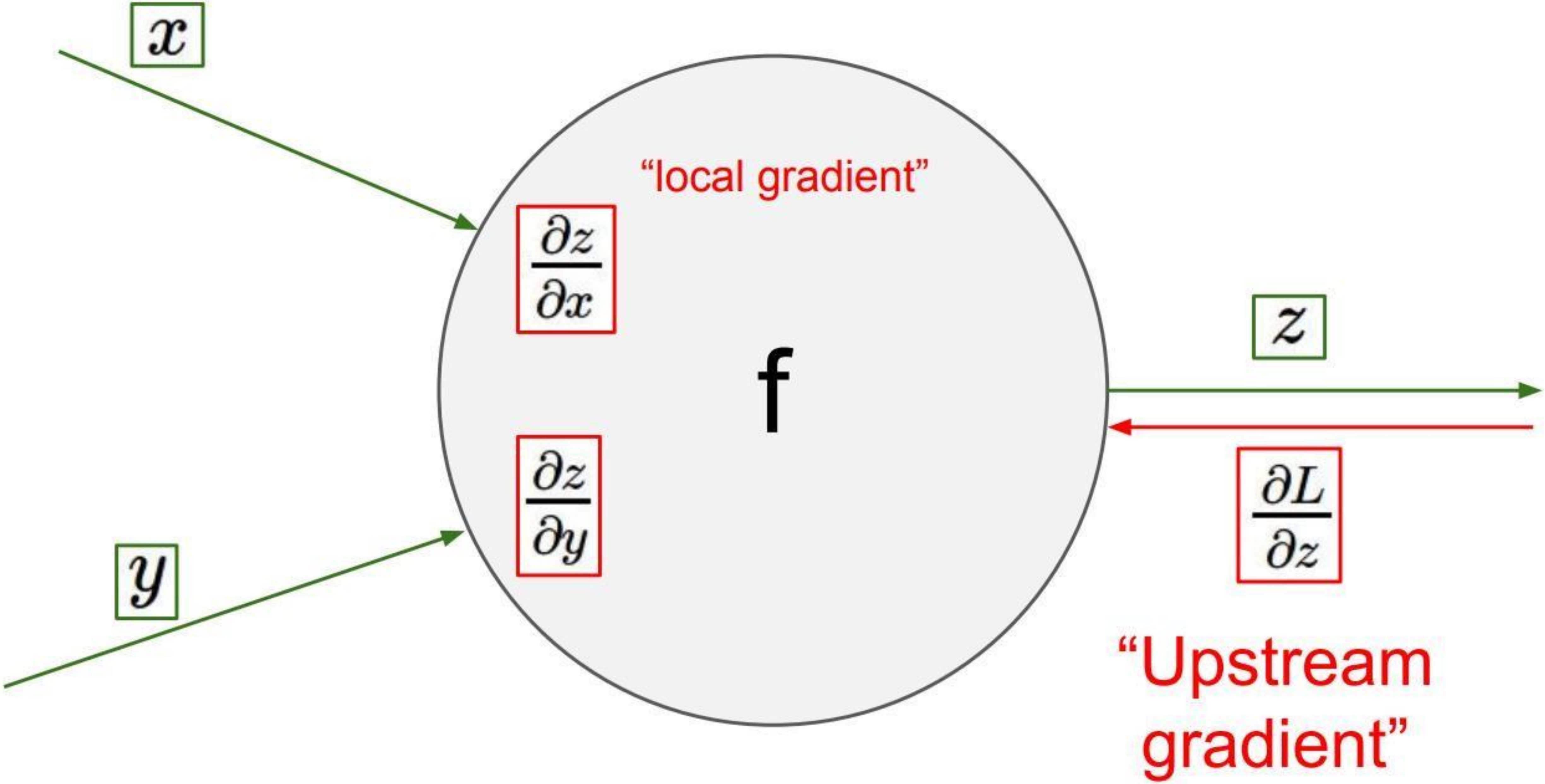


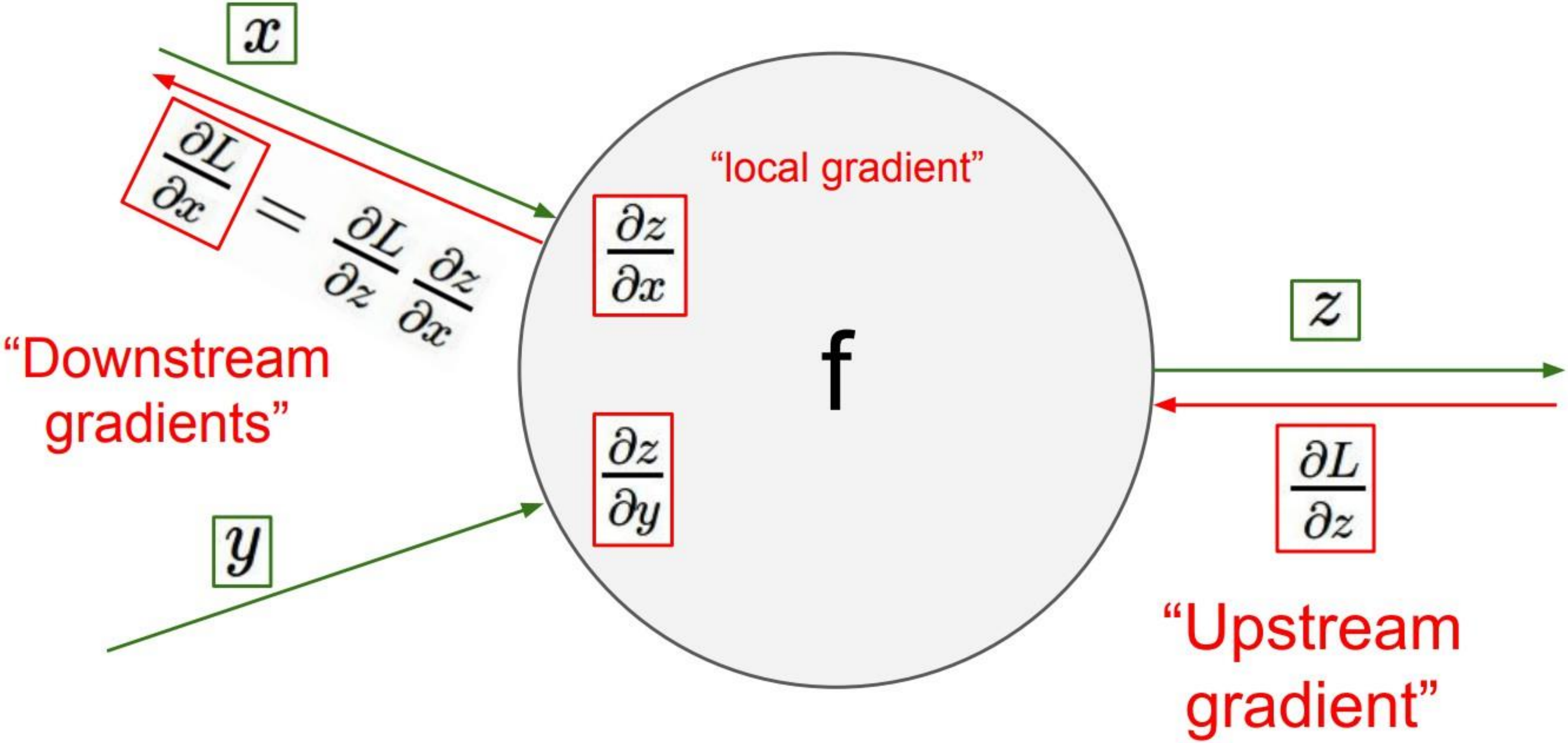
Computation graph of f


x
y
f
z

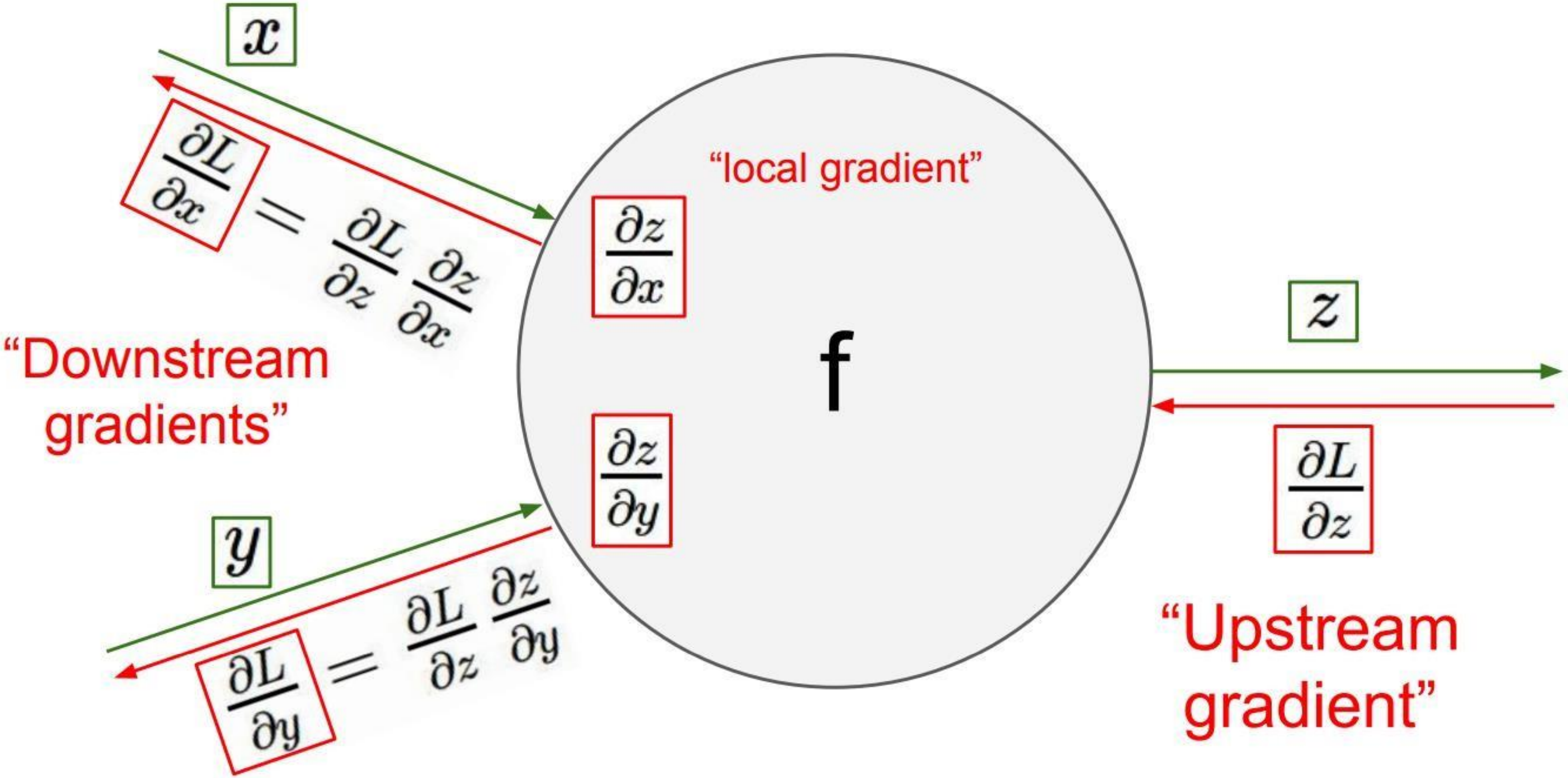


x
"local gradient"
$\frac{\partial z}{\partial x}$
z
f
$\frac{\partial z}{\partial y}$
y

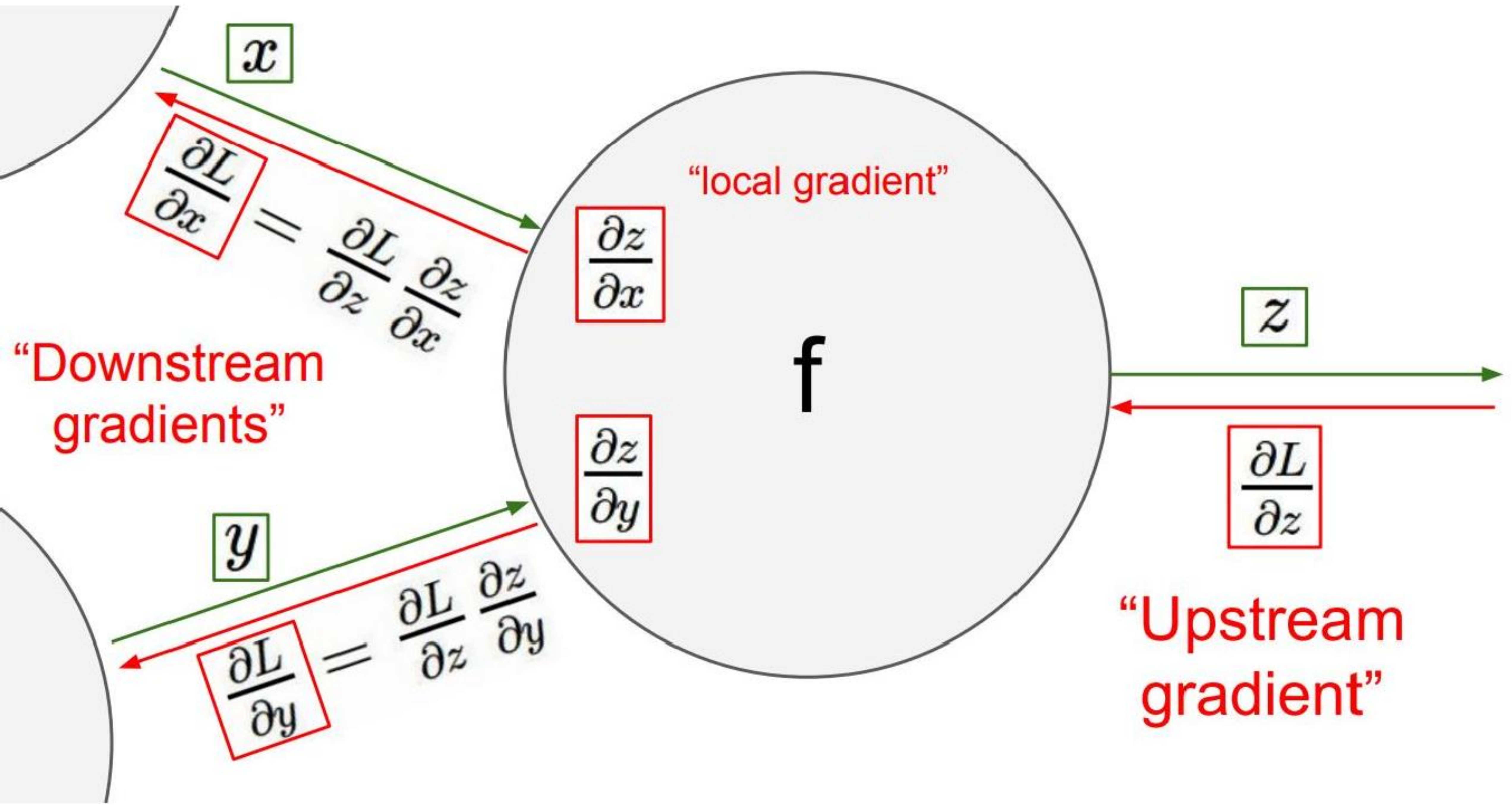

x
y
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
f
z
$\frac{\partial L}{\partial z}$
"Upstream gradient"



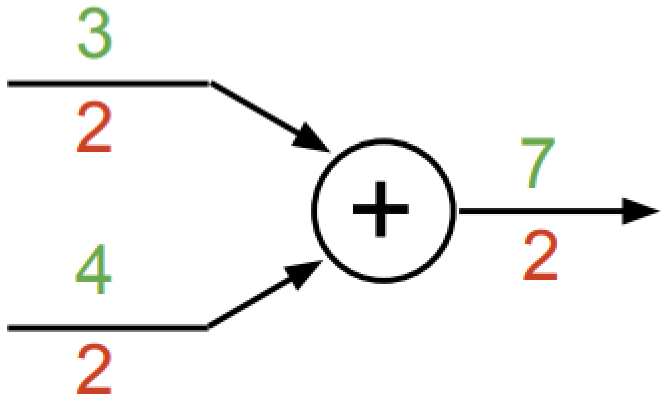
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
"Downstream gradients"
z
f
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
"Upstream gradient"



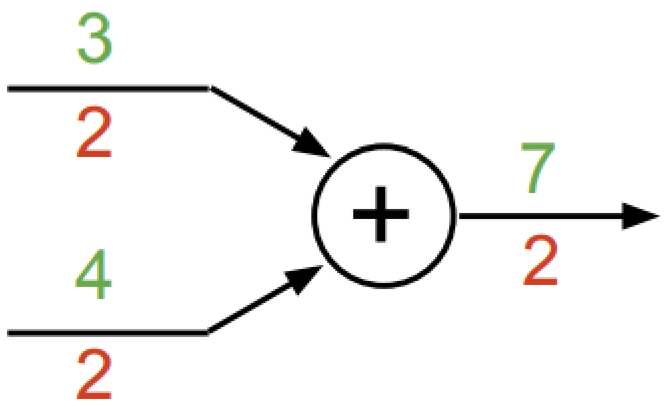
x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
z
f
"Downstream gradients"
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
"Upstream gradient"


x
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
"local gradient"
$\frac{\partial z}{\partial x}$
z
f
"Downstream gradients"
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
y
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
"Upstream gradient"

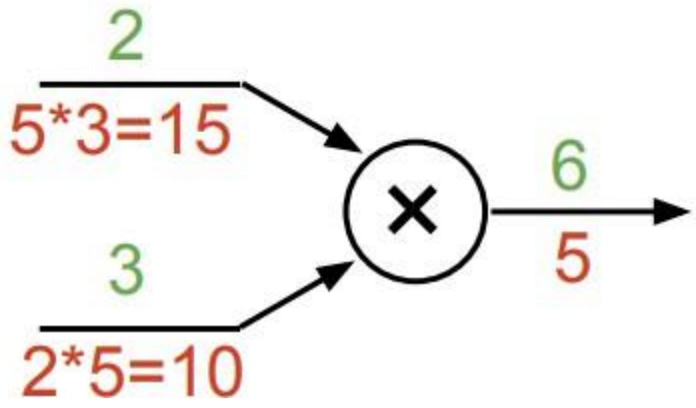
add gate: gradient distributor



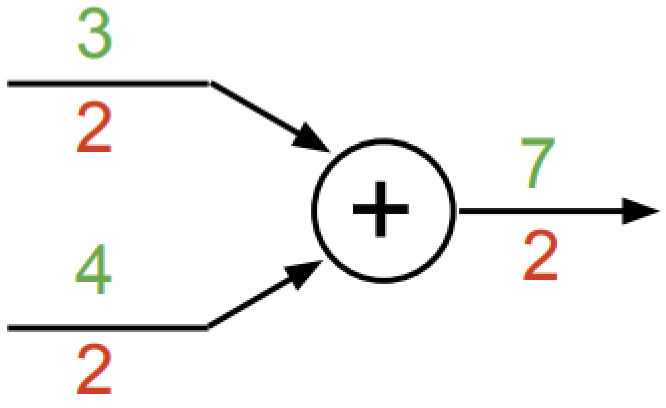
add gate: gradient distributor



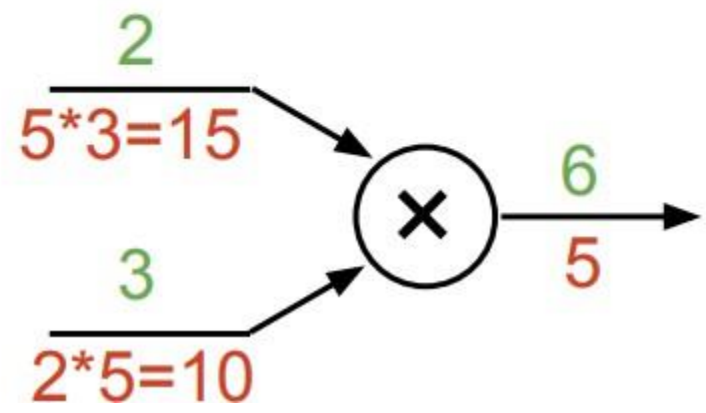
mul gate: “swap multiplier”



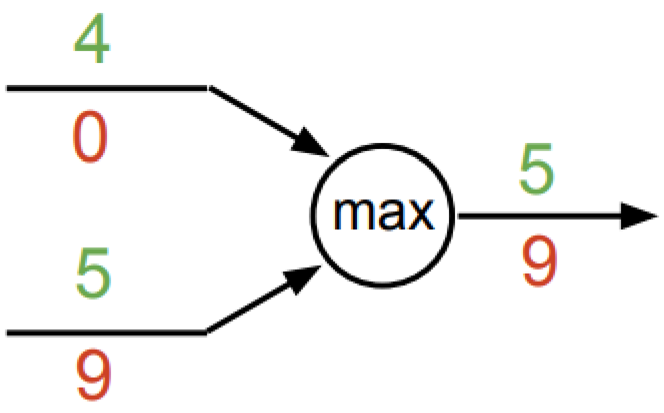
add gate: gradient distributor

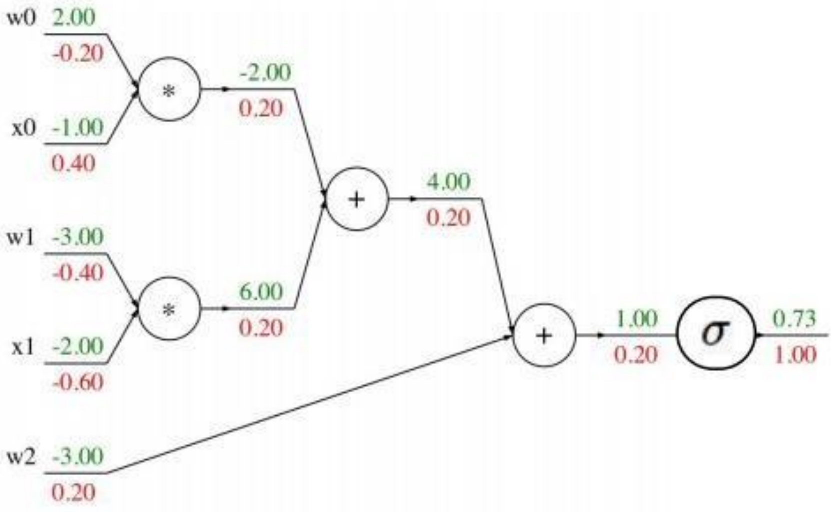


mul gate: “swap multiplier”



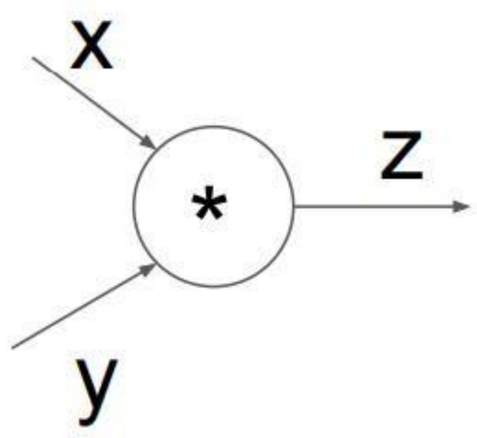
max gate: gradient router






```python
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

Gate / Node / Function object: Actual PyTorch code



(x,y,z are scalars)

```
class Multiply(torch.autograd.Function):
  @staticmethod
  def forward(ctx, x, y):
    ctx.save_for_backward(x, y)
    z = x * y
    return z
  @staticmethod
  def backward(ctx, grad_z):
    x, y = ctx.saved_tensors
    grad_x = y * grad_z   # dz/dx * dL/dz
    grad_y = x * grad_z   # dz/dy * dL/dz
    return grad_x, grad_y
```

Need to stash some values for use in backward

Upstream gradient

Multiply upstream and local gradients

Automatic Differentiation – Reverse Mode (aka. Backpropagation)

Forward Computation

1. Write an **algorithm** for evaluating the function $y = f(\mathbf{x})$. The algorithm defines a **directed acyclic graph**, where each variable is a node (i.e. the "**computation graph**")
2. Visit each node in **topological order**.
   For variable u_i with inputs $v_1, \ldots, v_N$
   a. Compute $u_i = g_i(v_1, \ldots, v_N)$
   b. Store the result at the node

Backward Computation

1. **Initialize** all partial derivatives dy/du_j to 0 and $dy/dy = 1$.
2. Visit each node in **reverse topological order**.
   For variable $u_i = g_i(v_1, \ldots, v_N)$
   a. We already know dy/du_i
   b. Increment dy/dv_j by $(dy/du_i)(du_i/dv_j)$
      (Choice of algorithm ensures computing (du_i/dv_j) is easy)

Return partial derivatives dy/du_i for all variables

Refs

http://cs231n.stanford.edu/slides/2020/lecture_4.pdf

http://www.cs.cmu.edu/~mgormley/courses/10601bd-f18/slides/lecture12-backprop.pdf

https://www.youtube.com/watch?v=eL-KzMXSXXI