Understanding Topological Sort

Janani Ravi
CO-FOUNDER, LOONYCORN
www.loonycorn.com

Three Common Graph Problems

Establishing precedence

Getting from point A to point B

Covering all nodes in a graph

Three Common Graph Problems

Establishing precedence

Topological sort

Getting from point A to point B

Covering all nodes in a graph

Three Common Graph Problems

Establishing precedence

Topological sort

Getting from point A to point B

Shortest path algorithms

Covering all nodes in a graph

Three Common Graph Problems

Establishing precedence

Topological sort

Getting from point A to point B

Shortest path algorithms

Covering all nodes in a graph

Minimum spanning tree algorithms

Three Common Graph Operations

Topological sort

Computation graphs in neural networks

Getting from point A to point B

Shortest path algorithms

Covering all nodes in a graph

Minimum spanning tree algorithms

Three Common Graph Operations

Topological sort

Computation graphs in neural networks

Shortest path

Deliveries from warehouses to customers

Covering all nodes in a graph

Minimum spanning tree algorithms

Three Common Graph Operations

Topological sort

Computation graphs in neural networks

Shortest path

Deliveries from warehouses to customers

Minimum spanning tree

Planning railway lines

Three Common Graph Operations

Topological sort

Computation graphs in neural networks

Shortest path

Deliveries from warehouses to customers

Minimum spanning tree

Planning railway lines

Overview

Directed Acyclic Graphs (DAGs) are extremely versatile constructs

Applications of DAGs include building neural network models

DAGs specify precedence relationships between nodes

Any ordering of all nodes that satisfies all relationships is a topological sort

Topological sort can be implemented via a simple iterative algorithm

The Power of Directed Acyclic Graphs

A directed graph with no cycle is called a Directed Acyclic Graph

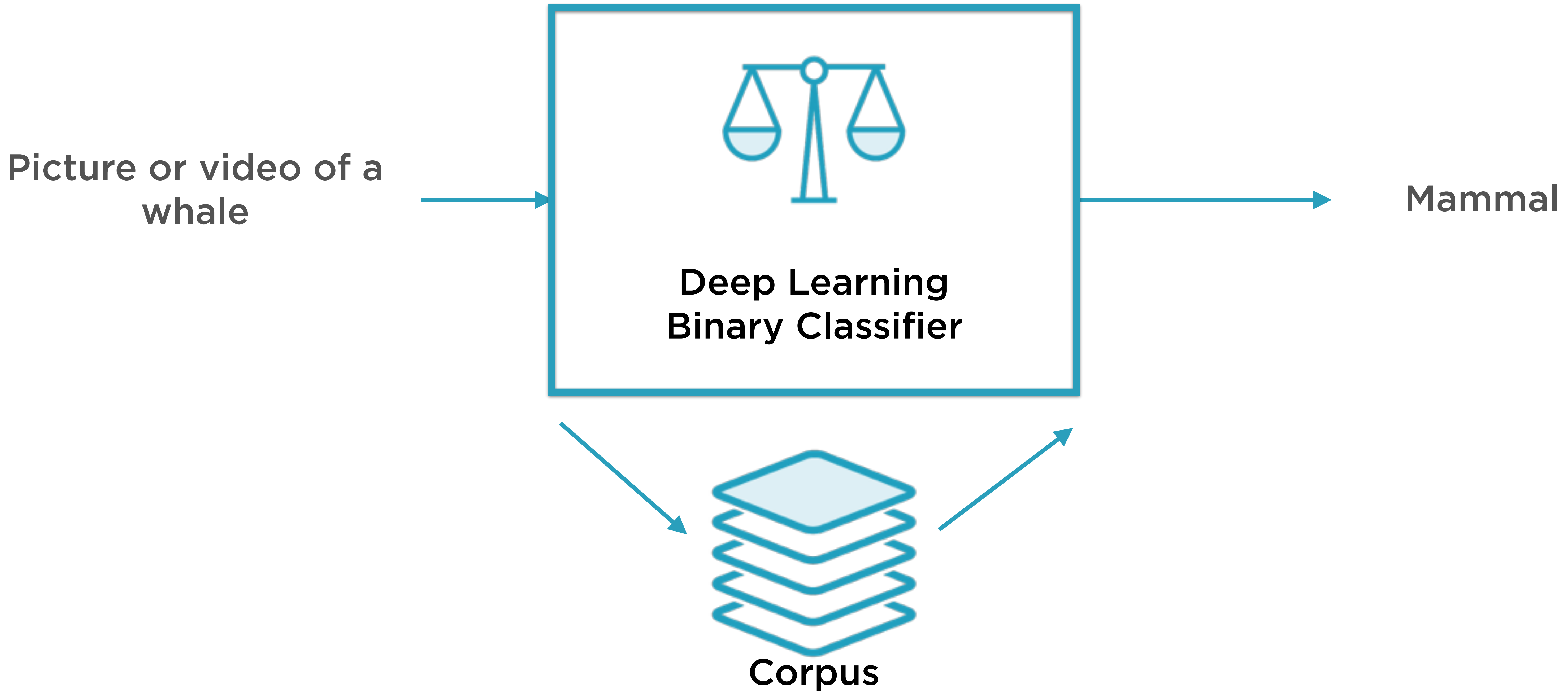
Directed Acyclic Graphs (DAGs)

Especially important type of graph

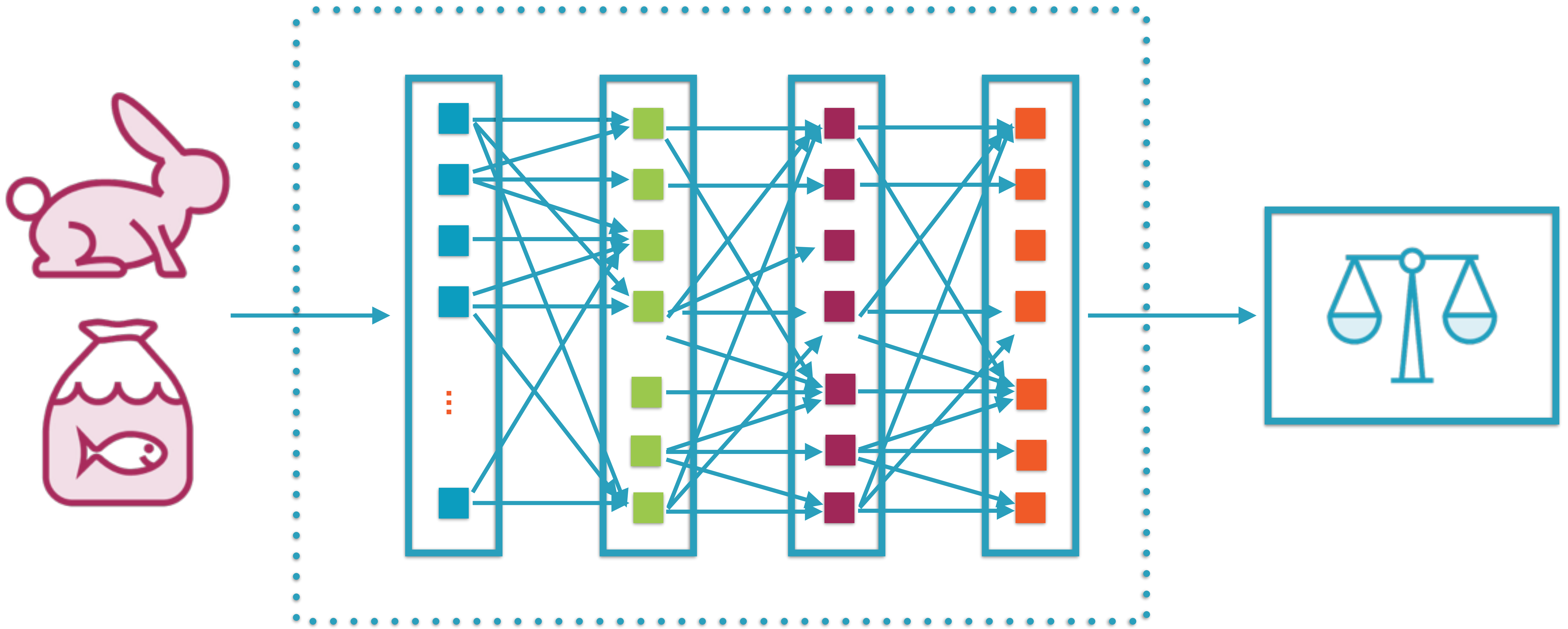
Common applications

- Scheduling tasks
- Evaluating expressions
- Building neural network models

"Deep Learning" Binary Classifier



Neural Network Computation Graph

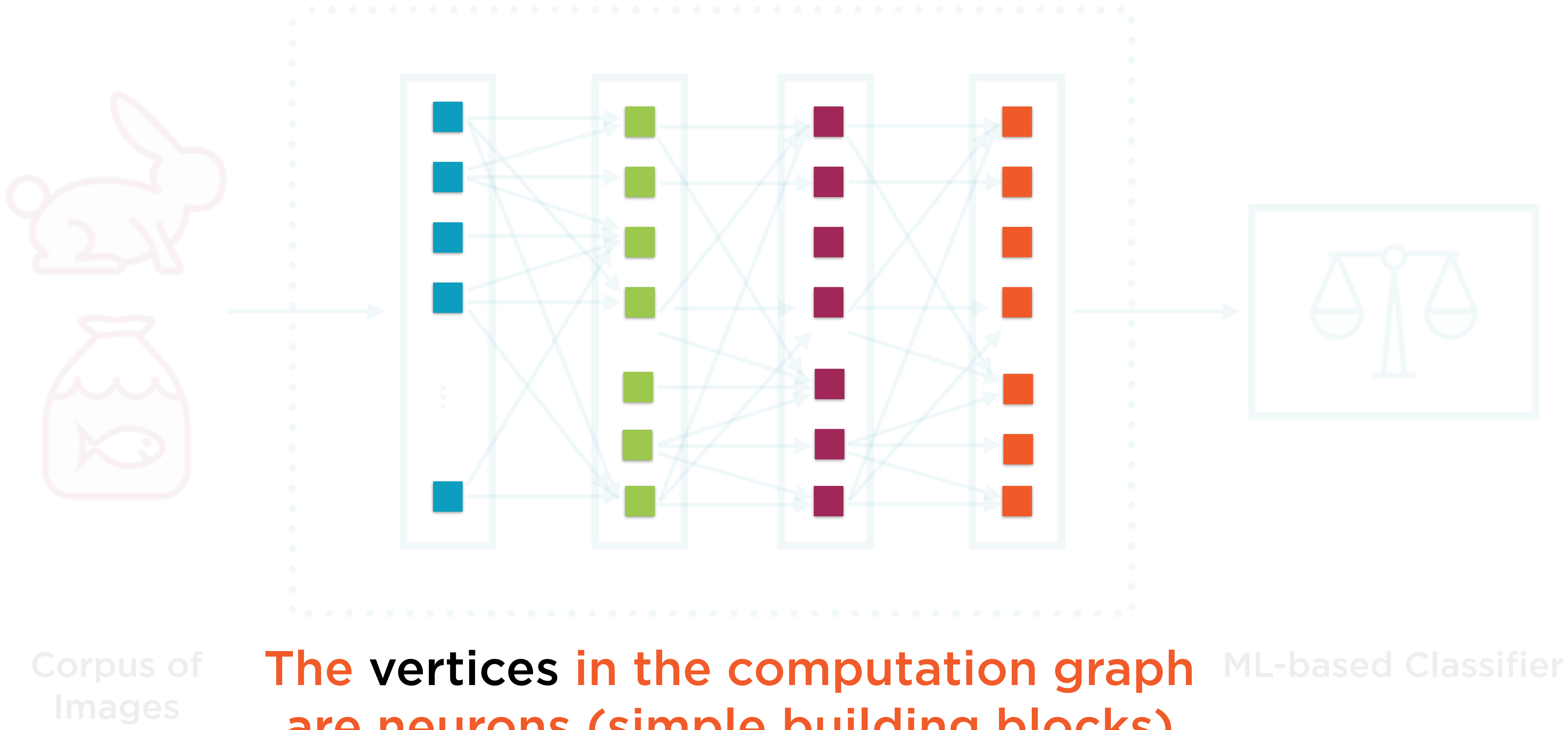


Corpus of Images

Operations (nodes) on data (edges)

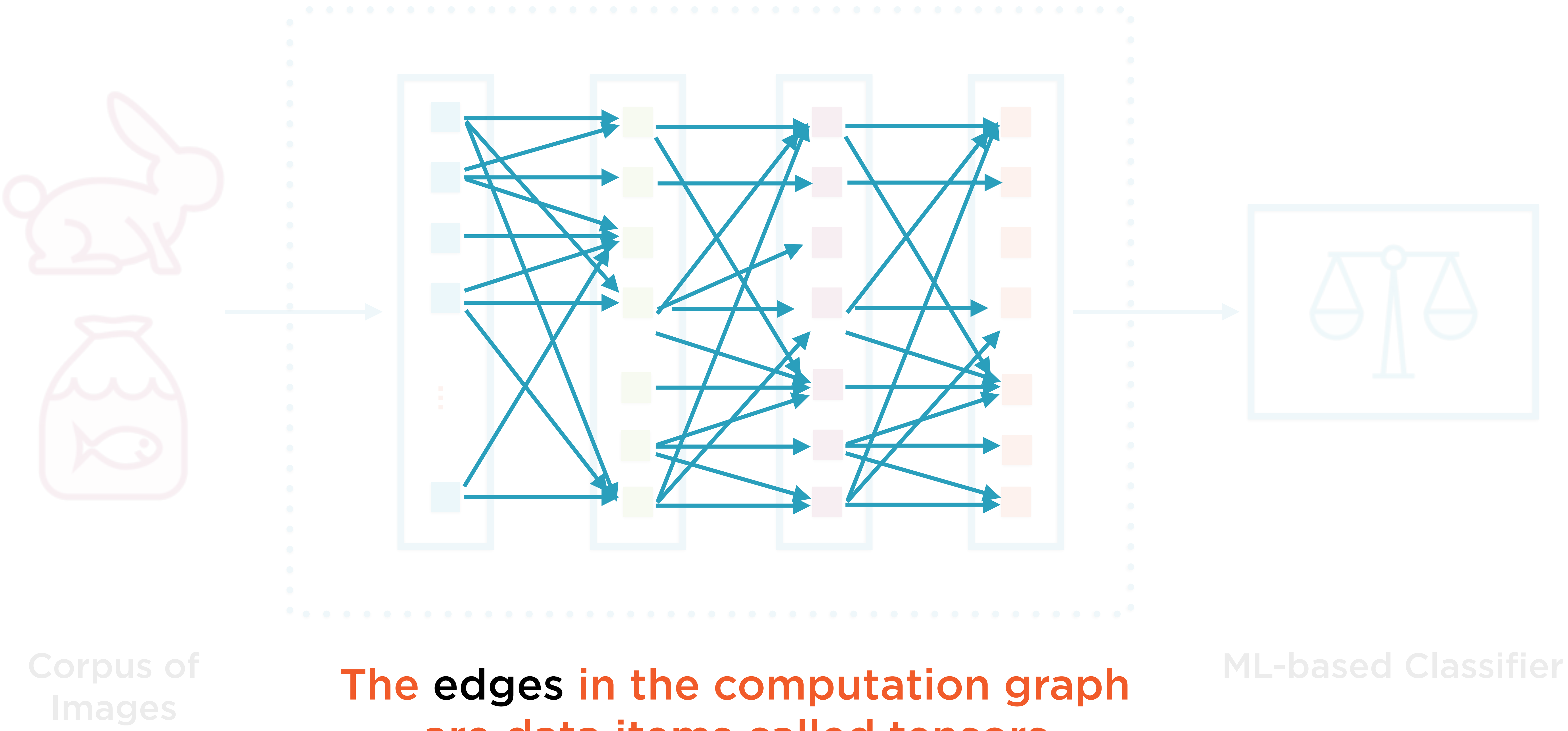
ML-based Classifier

Neural Network Computation Graph



The **vertices** in the computation graph are neurons (simple building blocks)

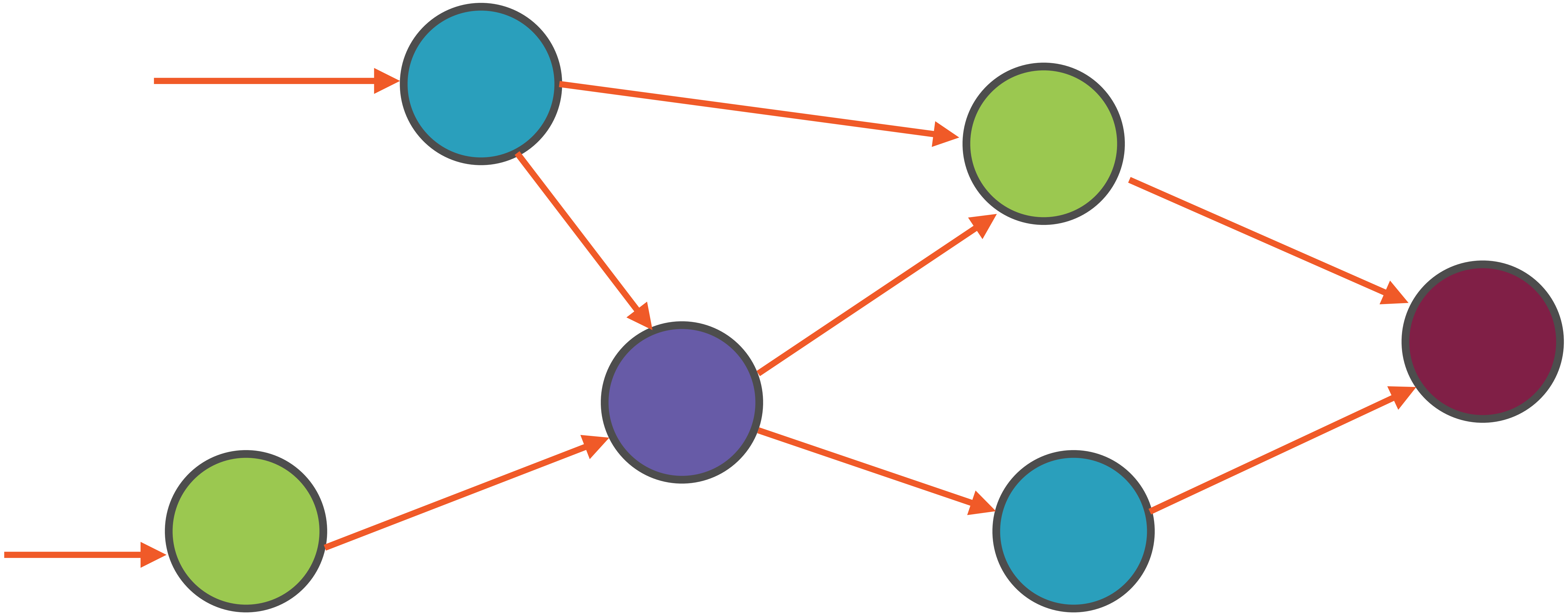
Neural Network Computation Graph



The **edges** in the computation graph are data items called tensors

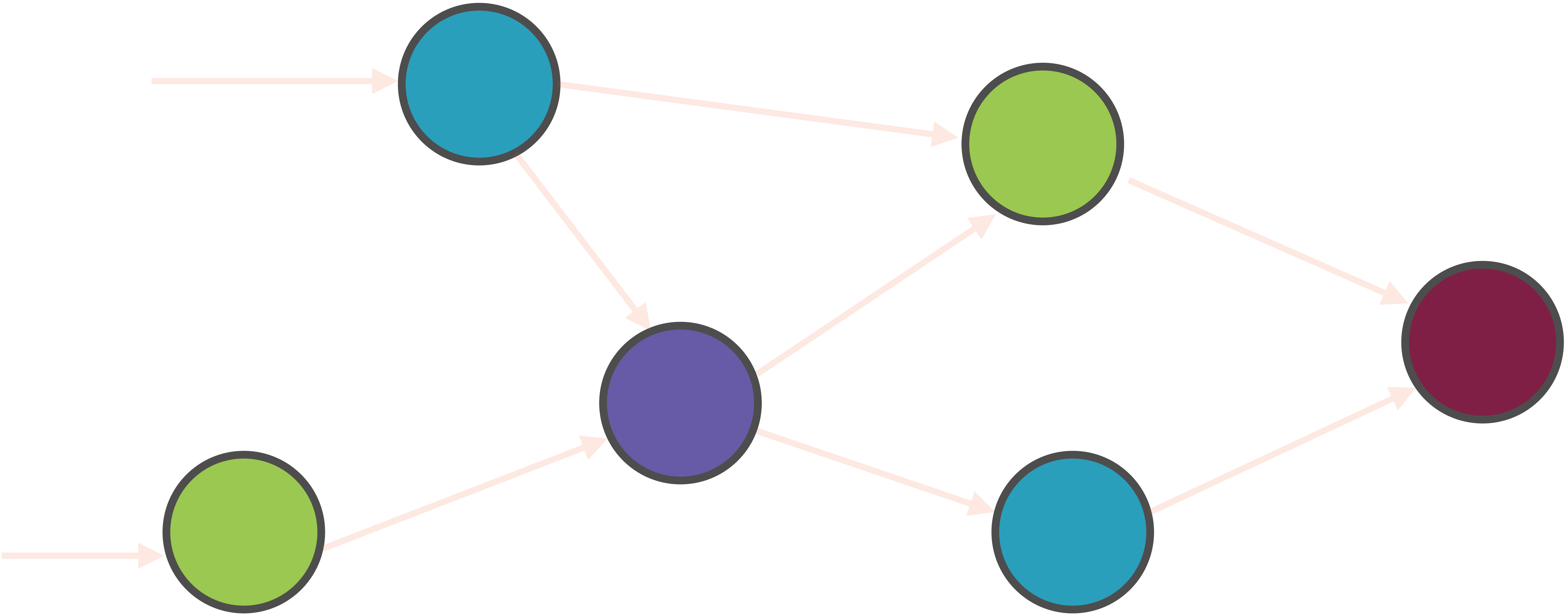
TensorFlow is a language for machine learning which is optimized for building neural networks

Everything is a Graph in TensorFlow



A network

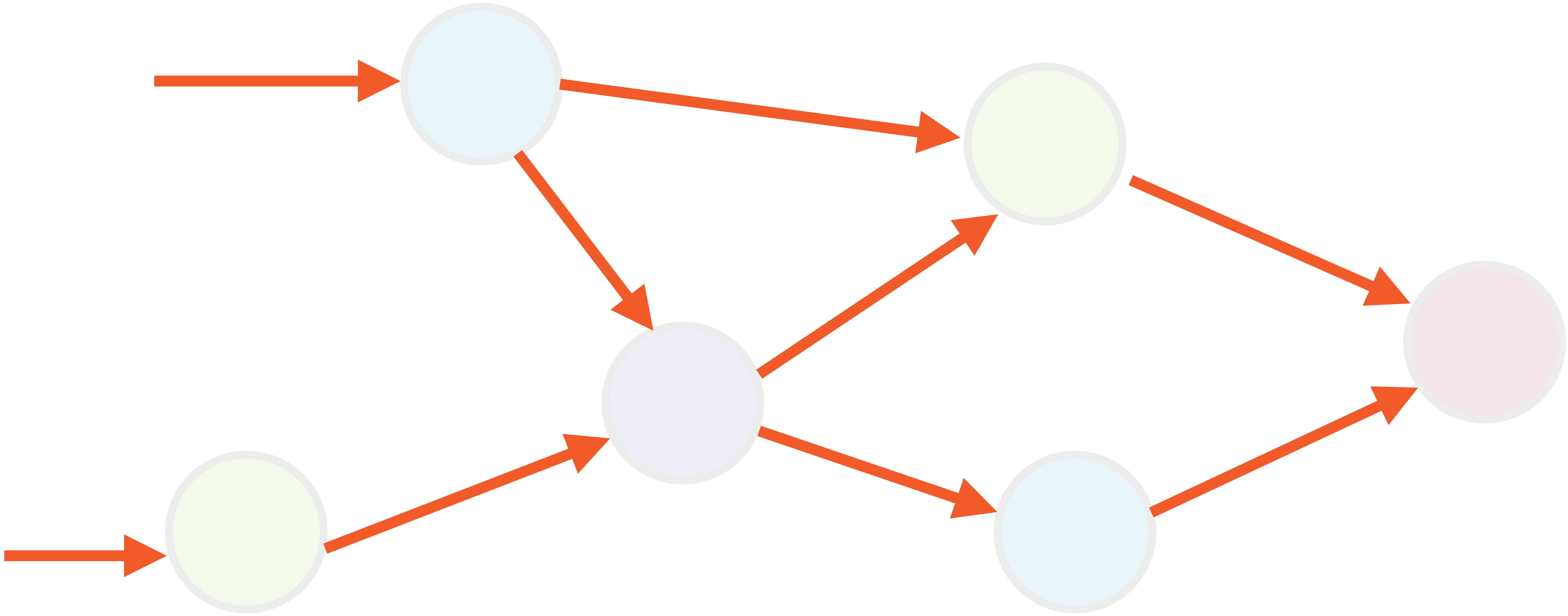
Everything is a Graph in TensorFlow



Computations

Operators

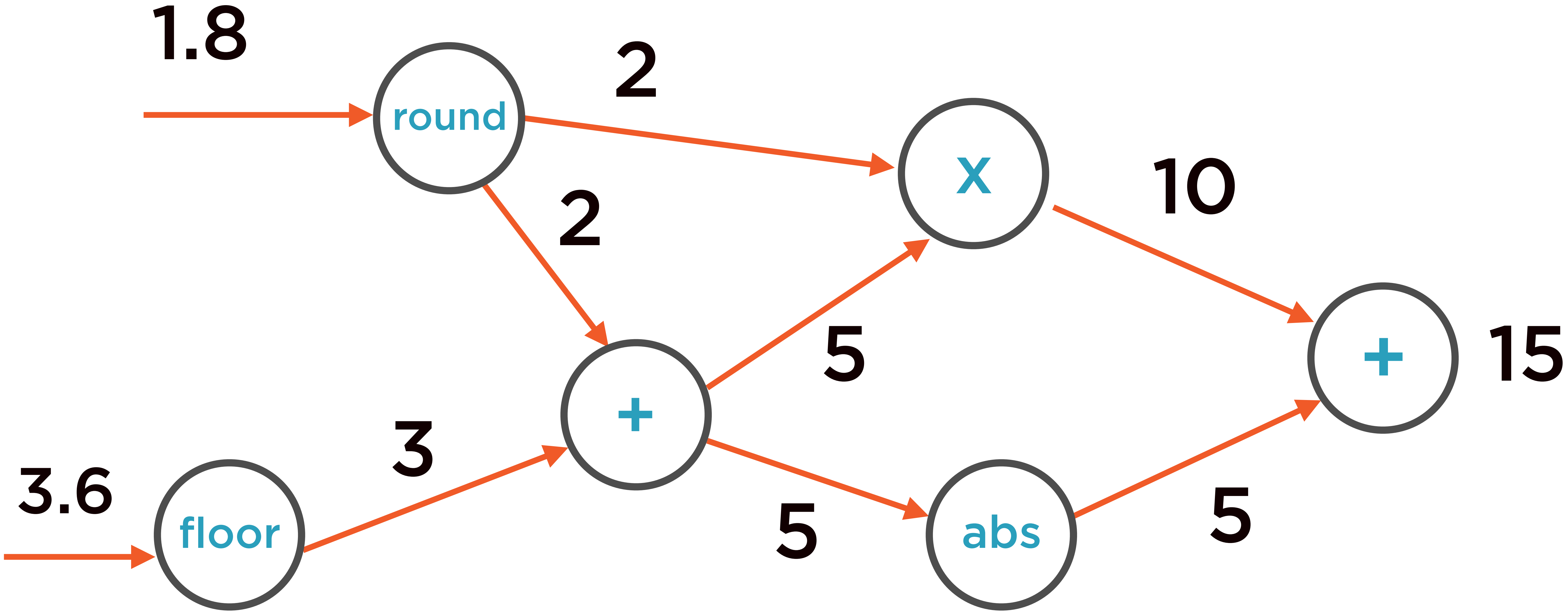
Everything is a Graph in TensorFlow



Data

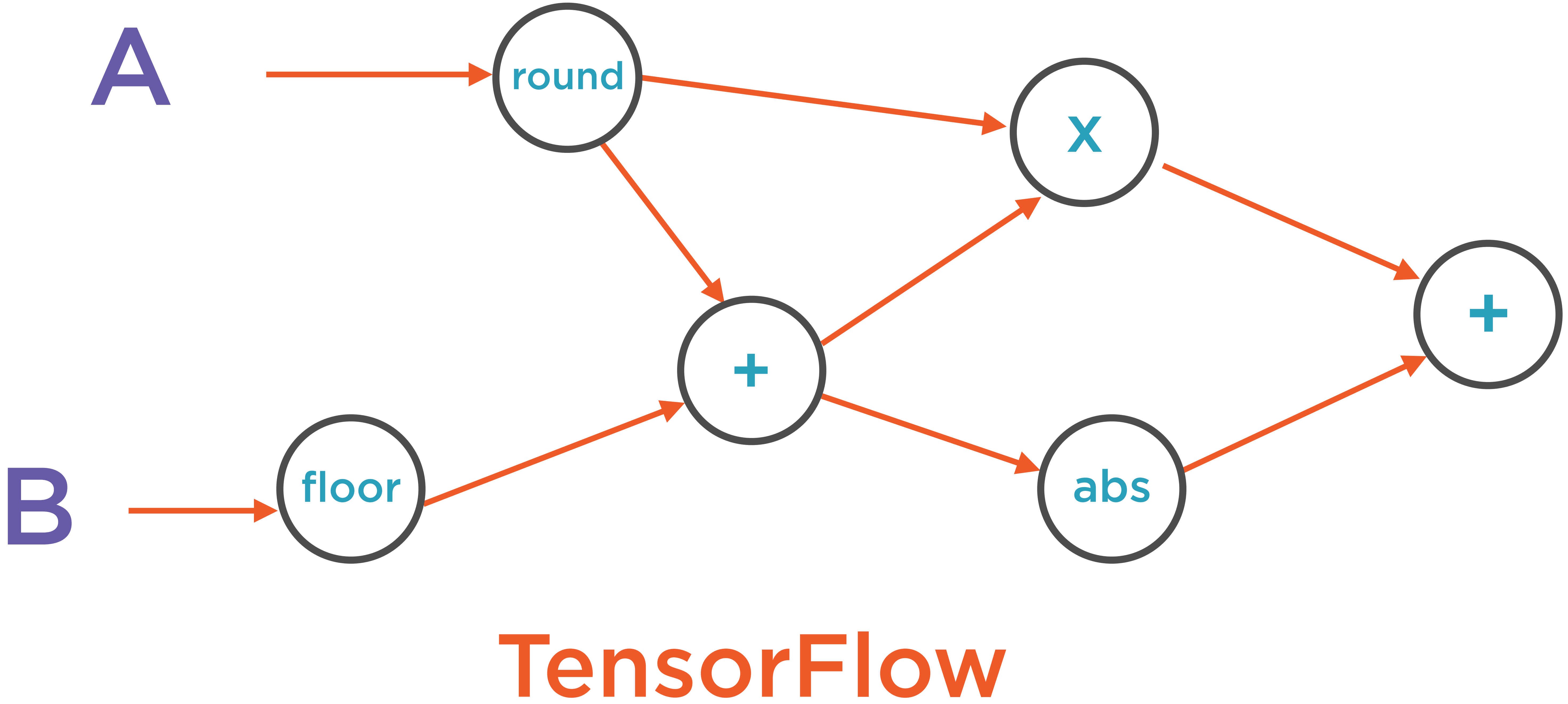
Tensors

Tensors Flow Through the Graph

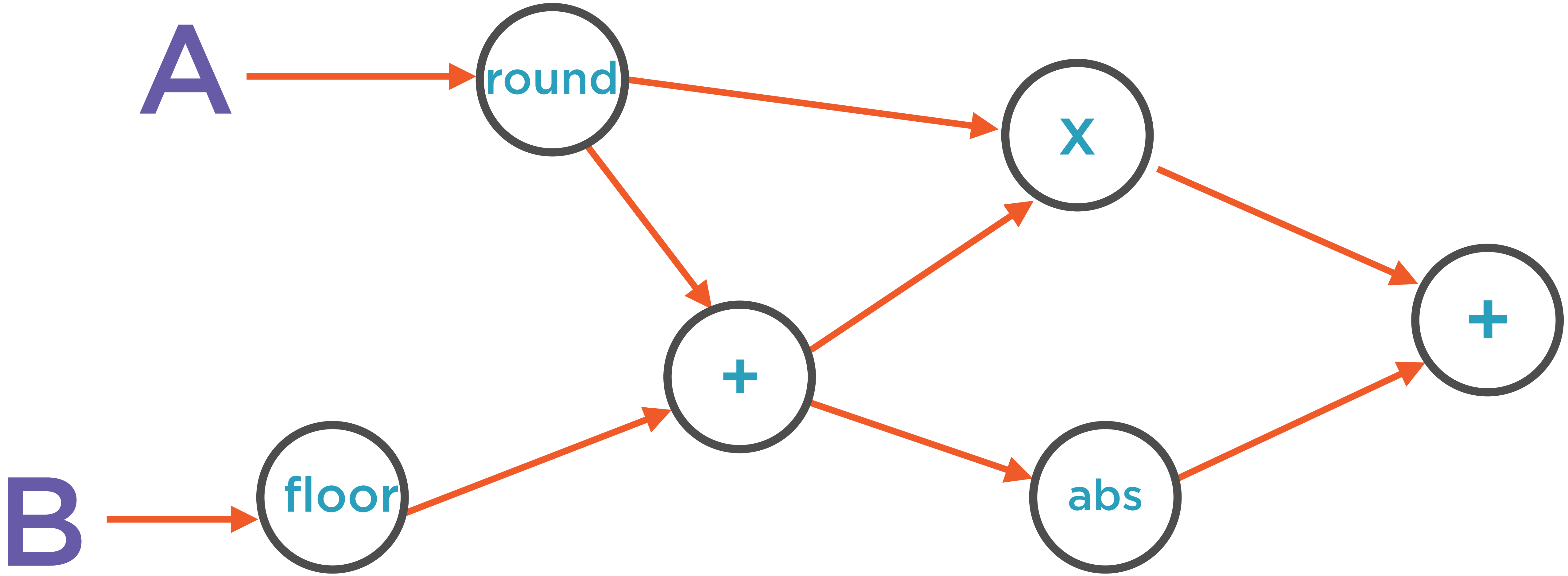


TensorFlow

Tensors Flow Through the Graph

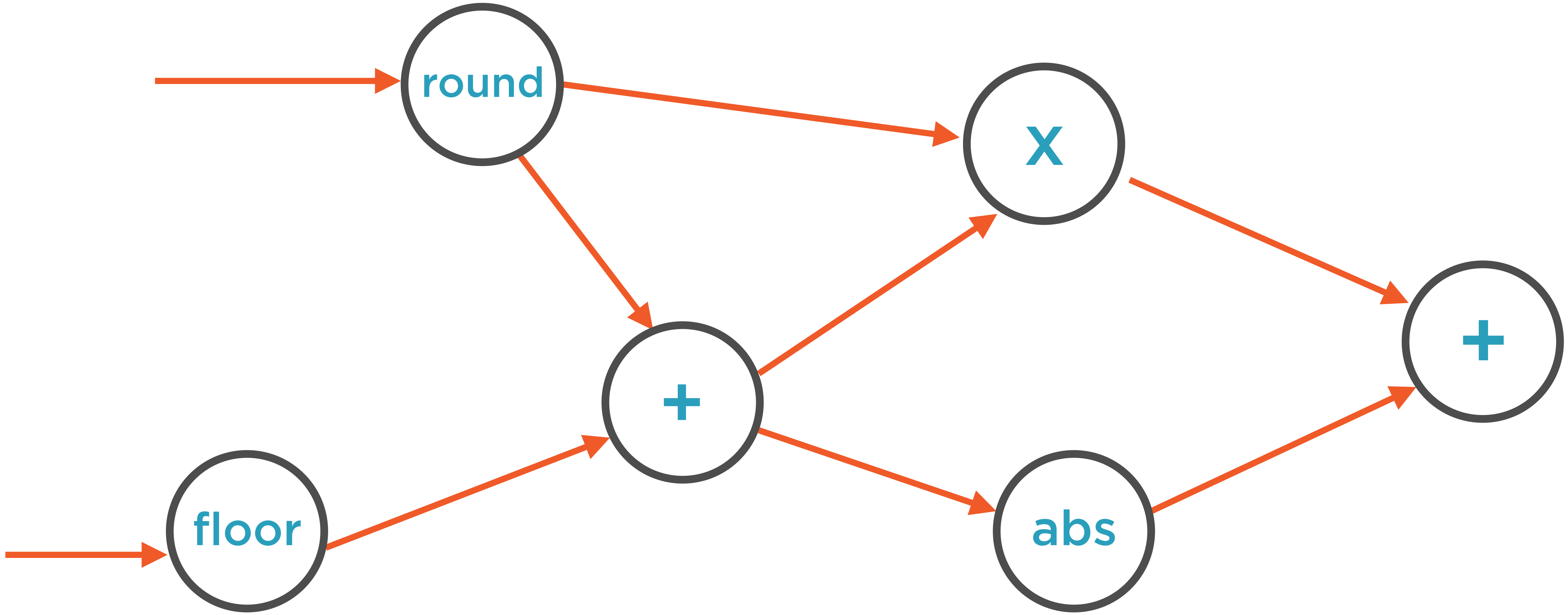


Tensors Flow Through the Graph

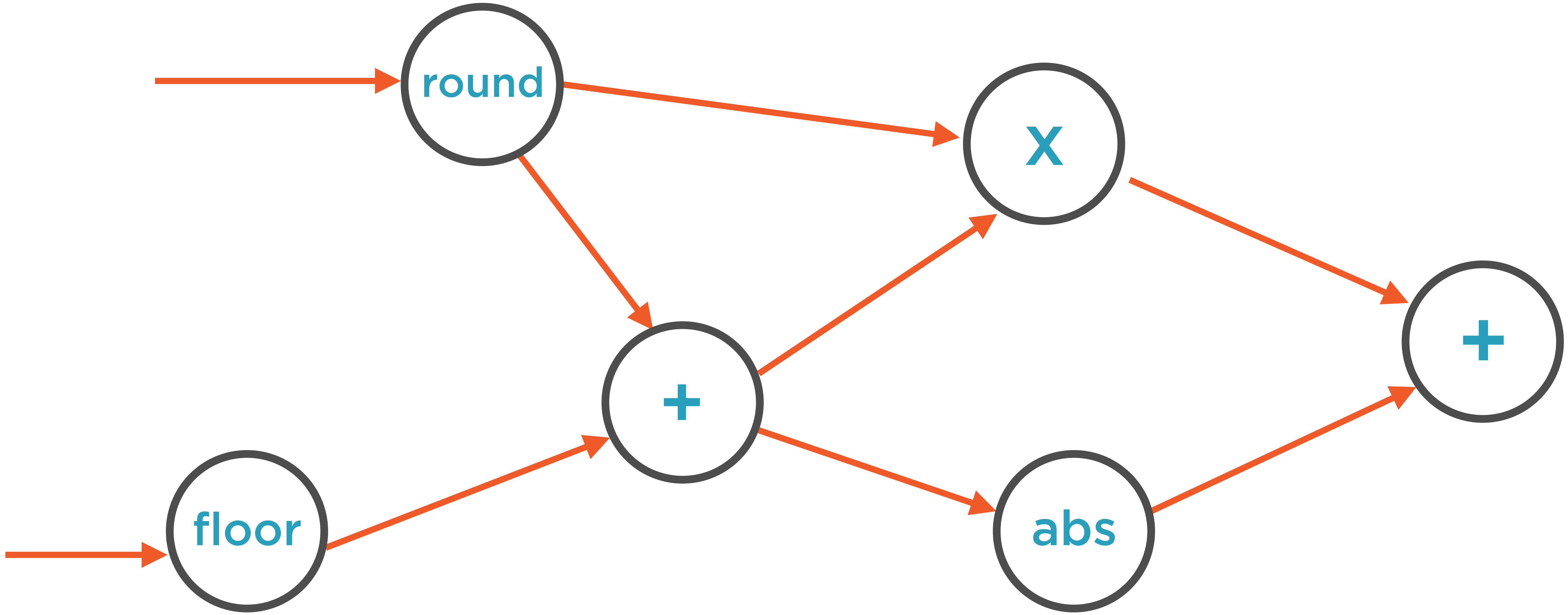



```
Y = (round(A) + floor(B)) *
round(A) + abs(round(A) + floor(B))
```

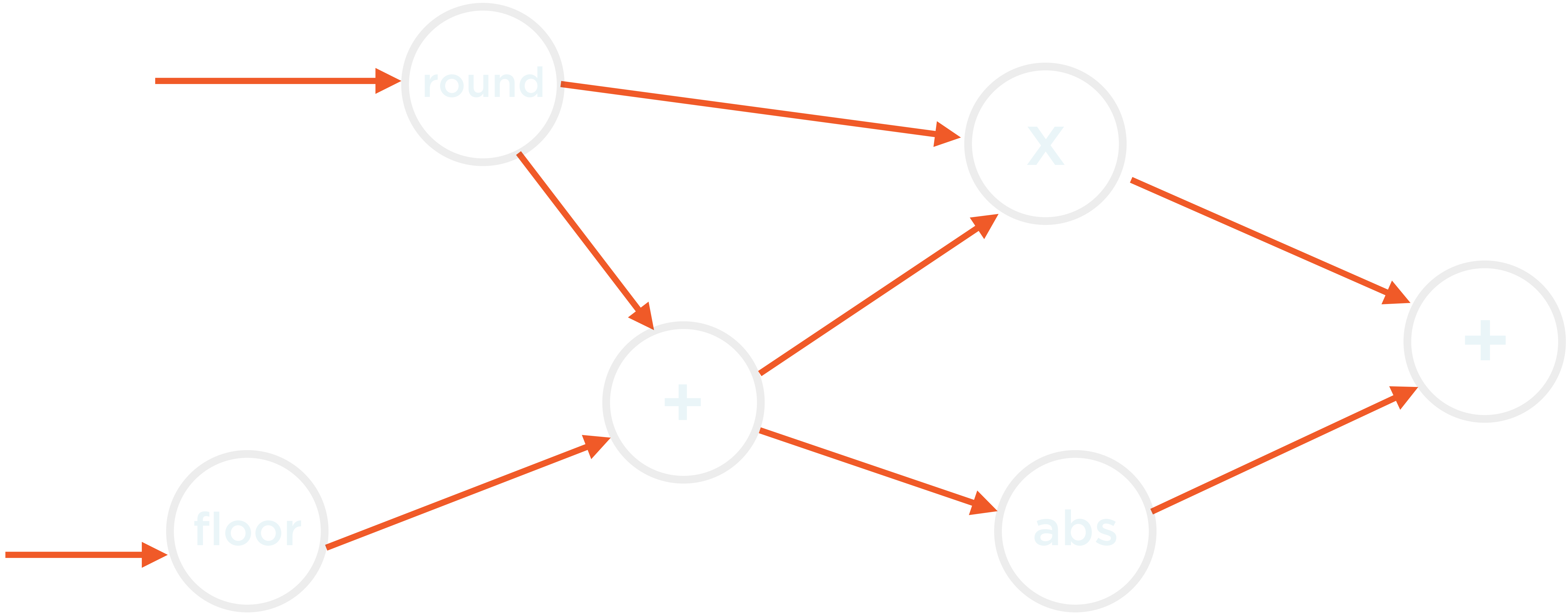
Directed-acyclic Graph



Directed-acyclic Graph

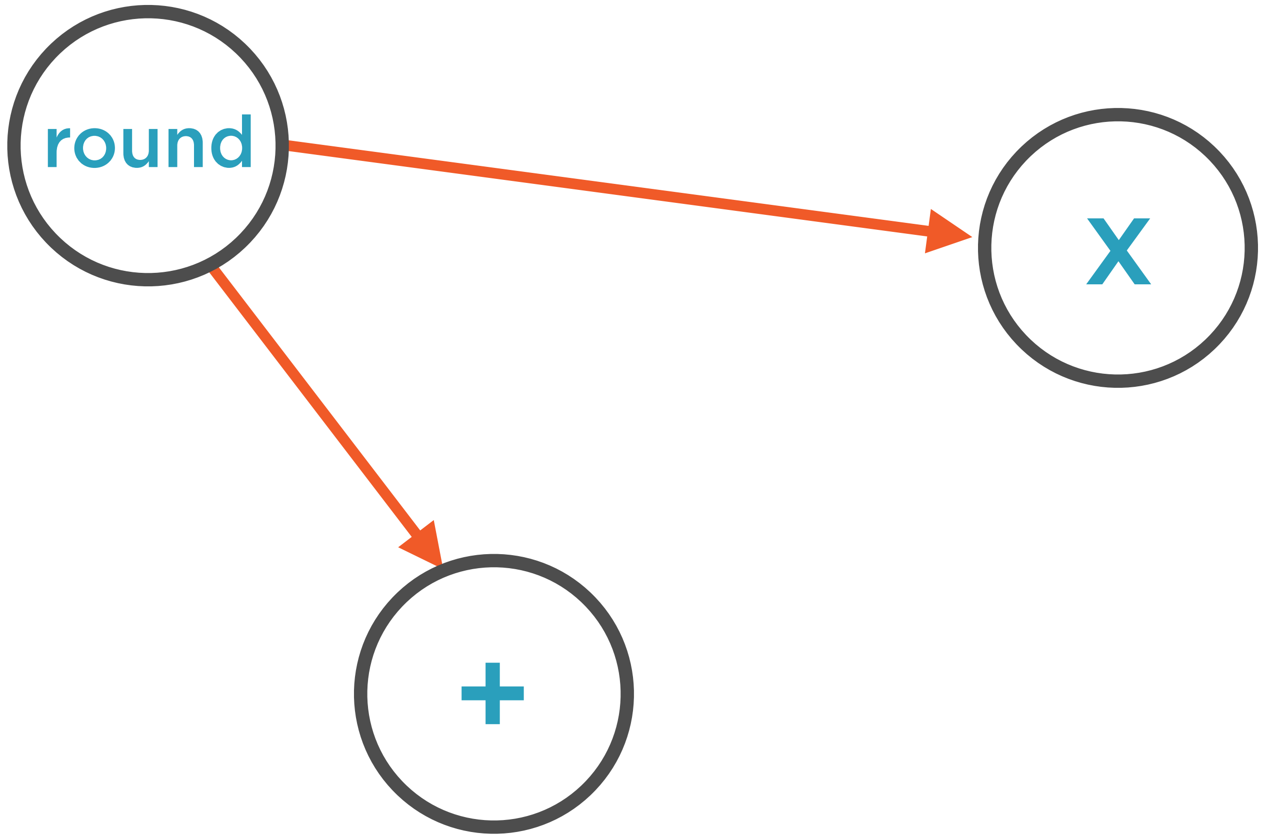


Directed-acyclic Graph



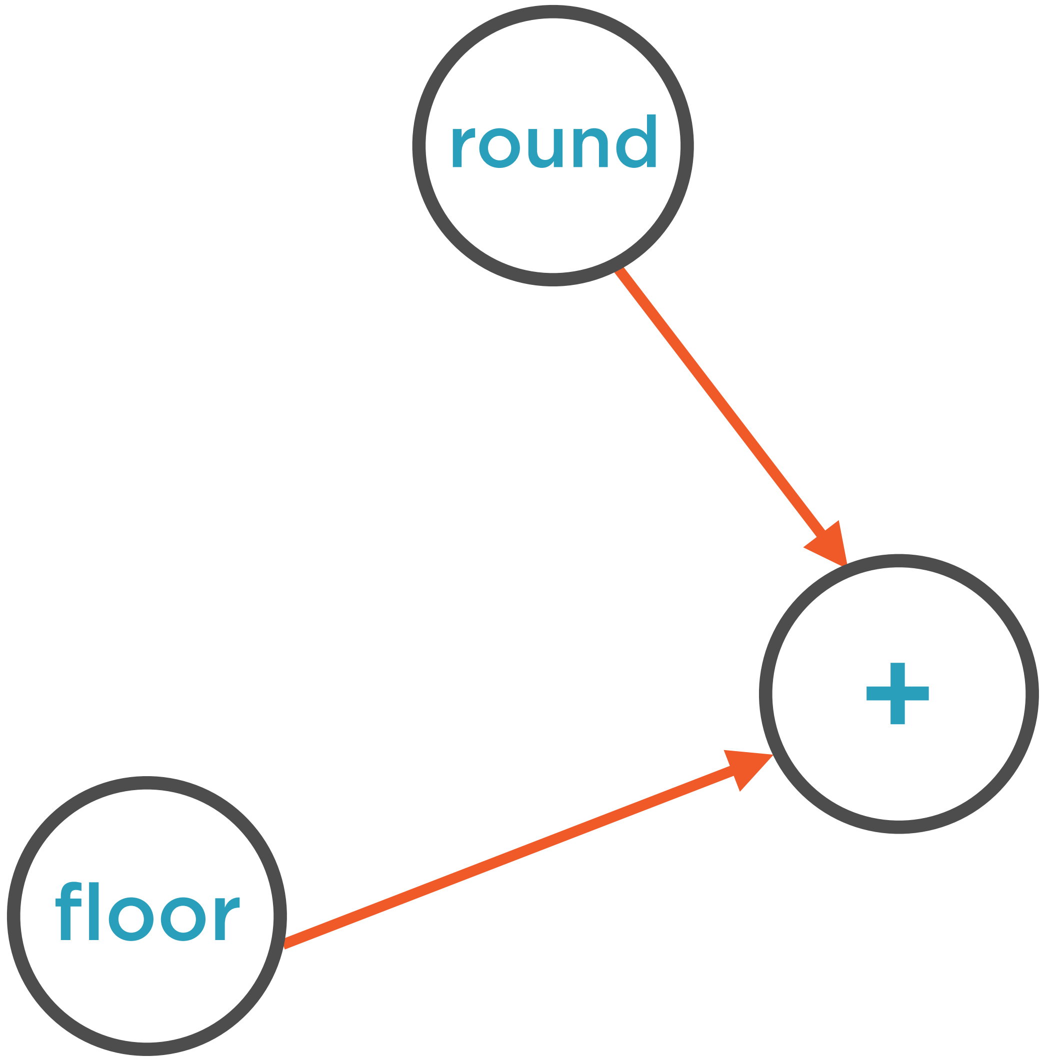
Edges point forward towards a result i.e. directed

Dependencies



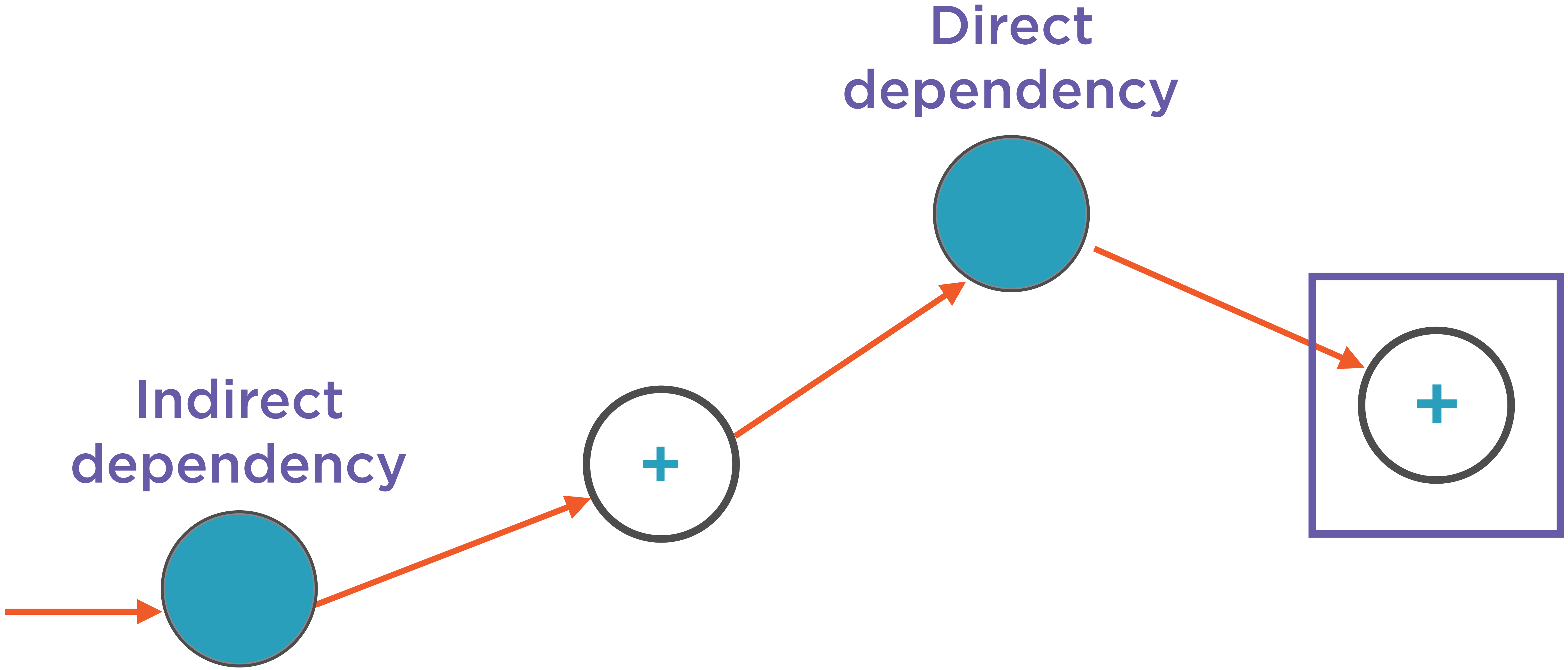
One node can send its output to multiple nodes

Dependencies

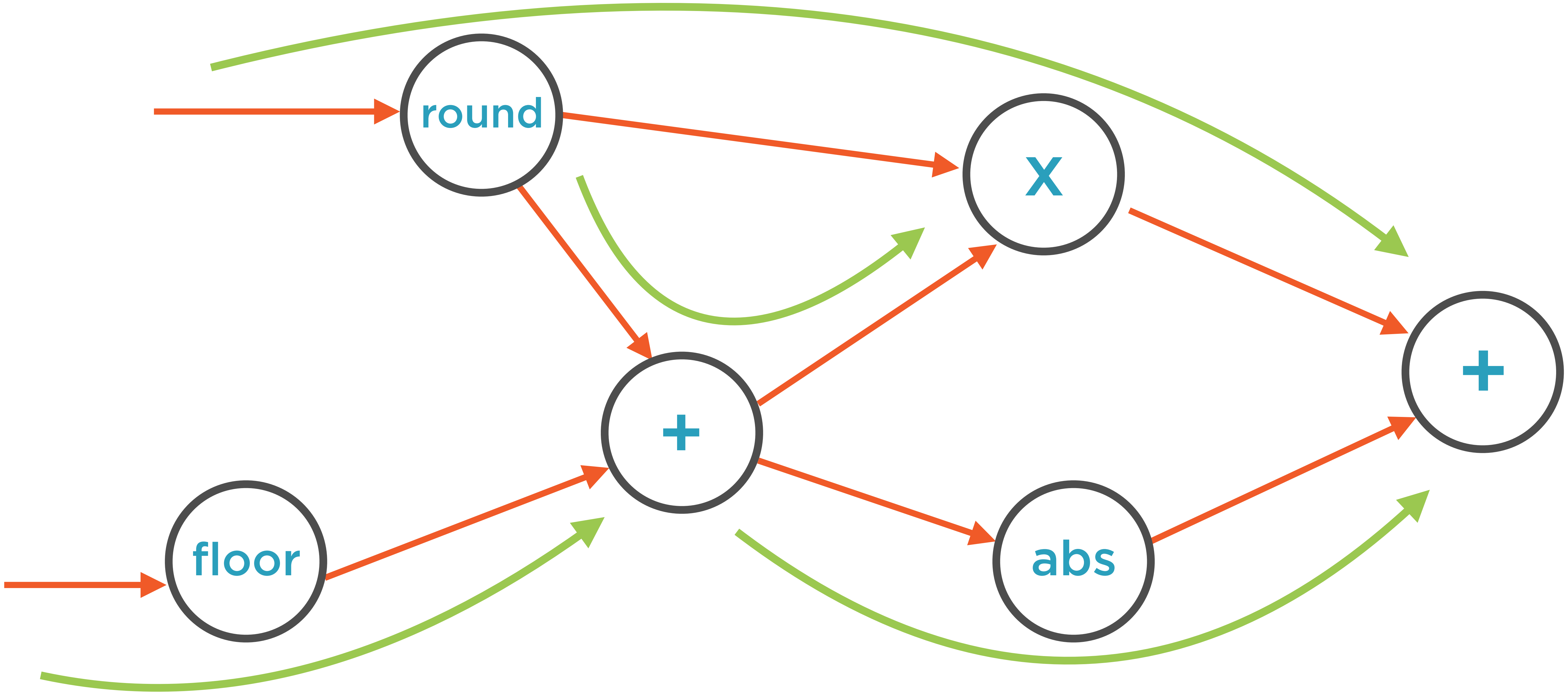


Or receive inputs from multiple nodes

Dependencies

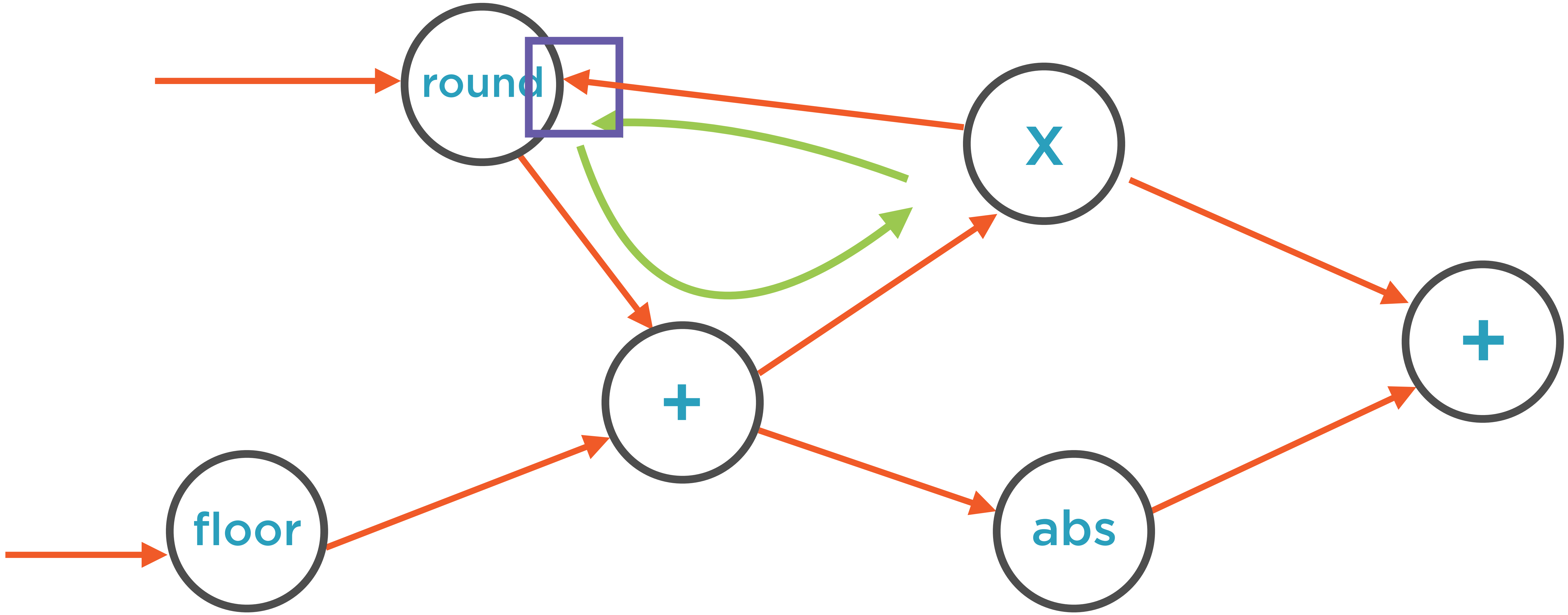


Directed-acyclic Graph



There are no cycles in the graph i.e. acyclic

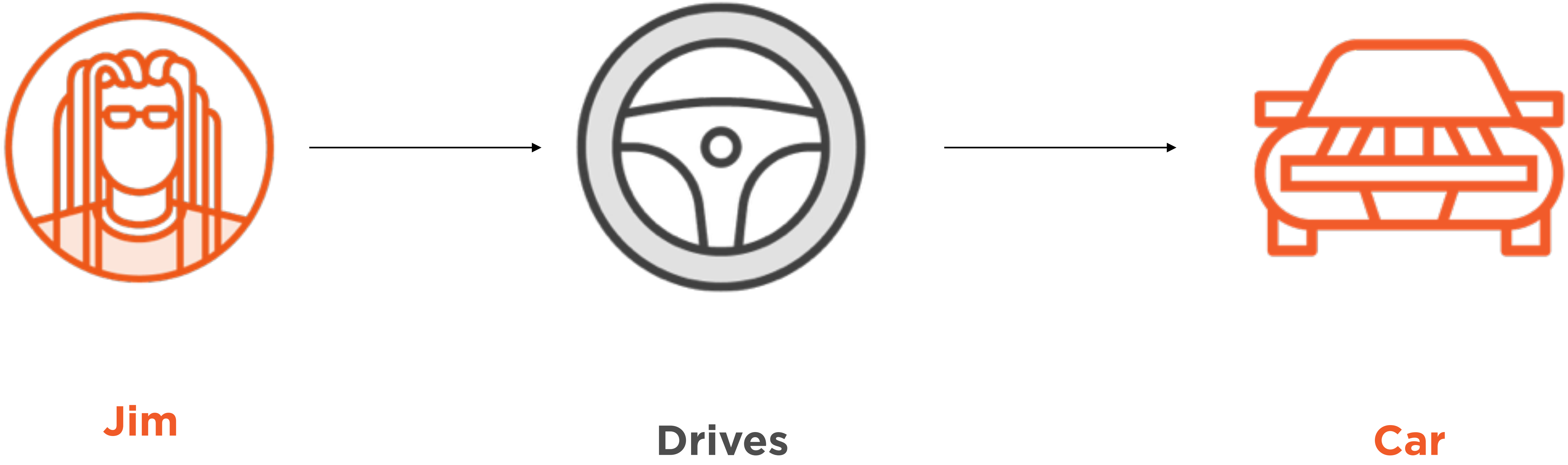
Directed-acyclic Graph



A graph with cycles will never finish computation

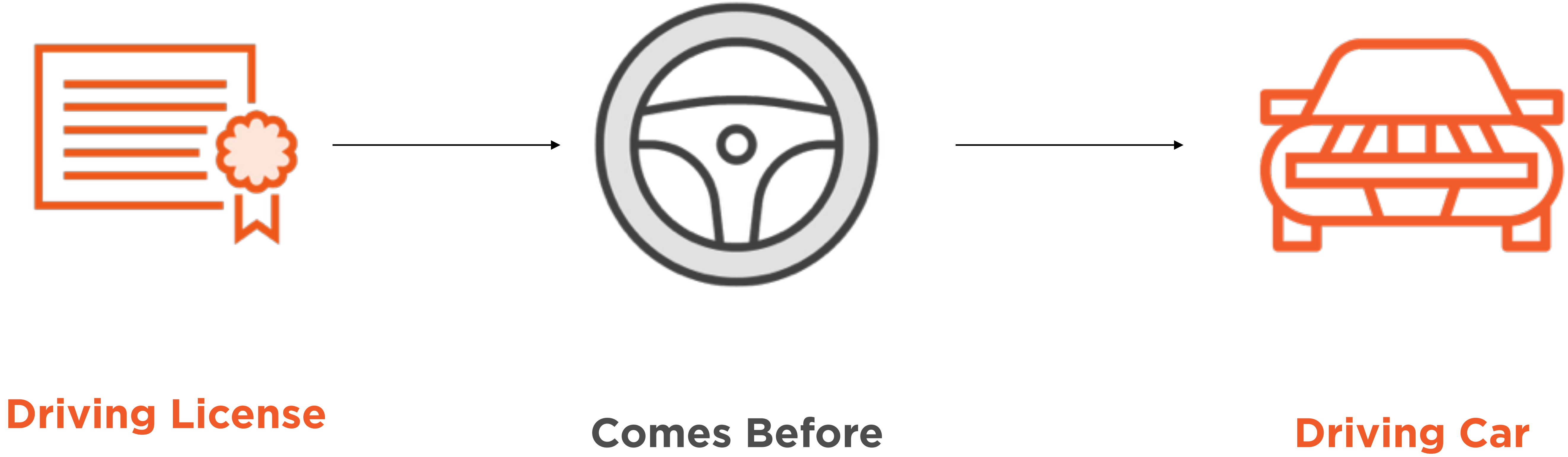
The Topological Sort Algorithm

Interconnections



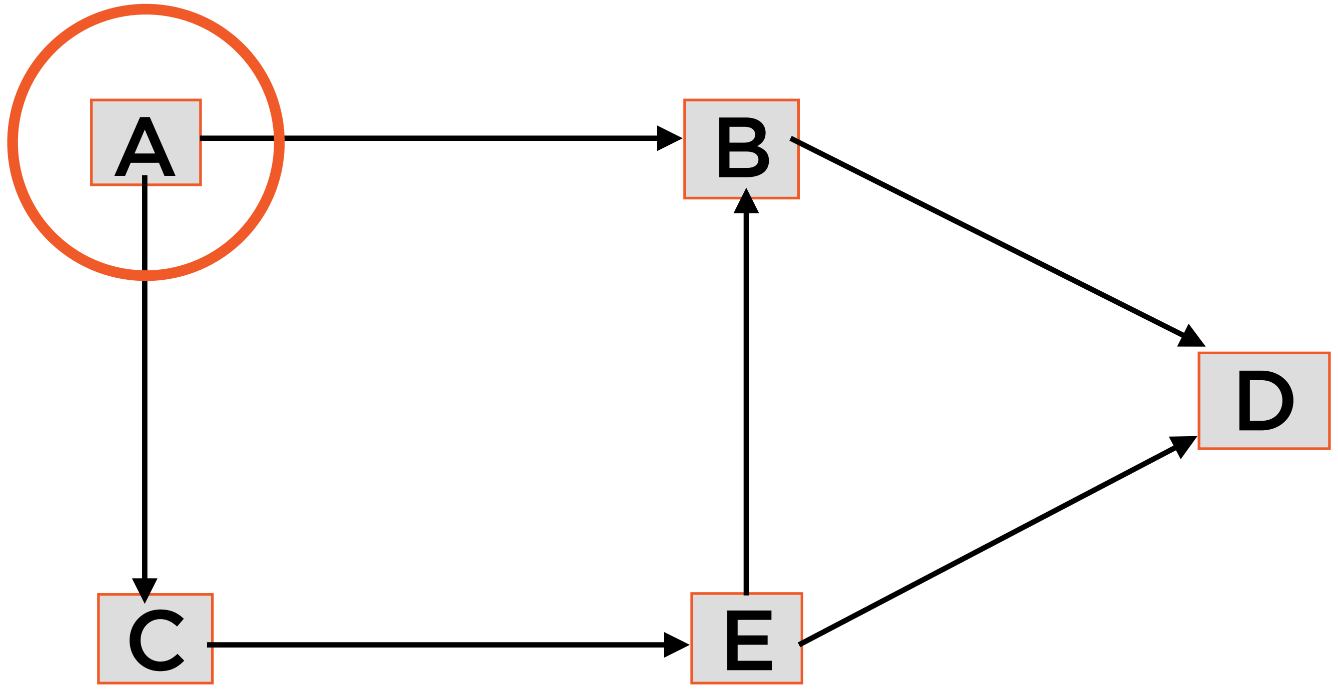
Graphs model relationships

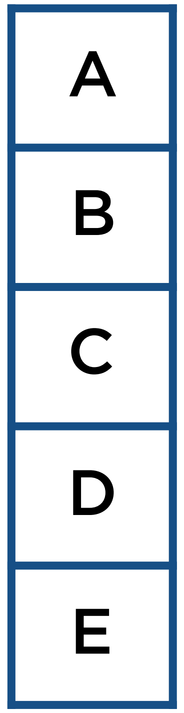
Precedence



DAGs model precedence relationships

Topological Sort





"Comes Before"

B and C

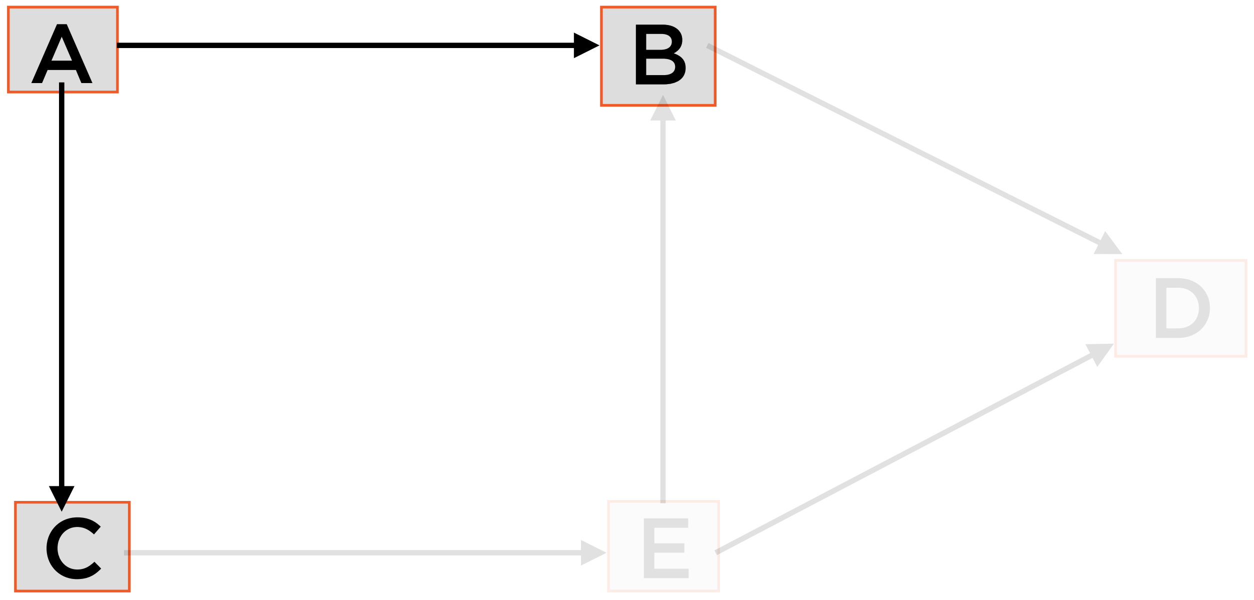
D

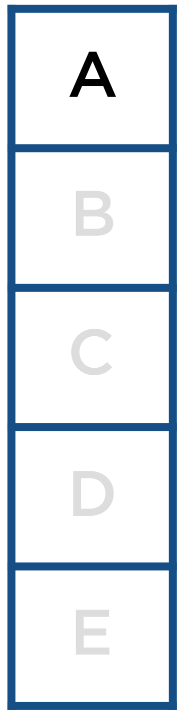
E

B and D

DAGs model precedence relationships

Topological Sort





"Comes Before"

B and C

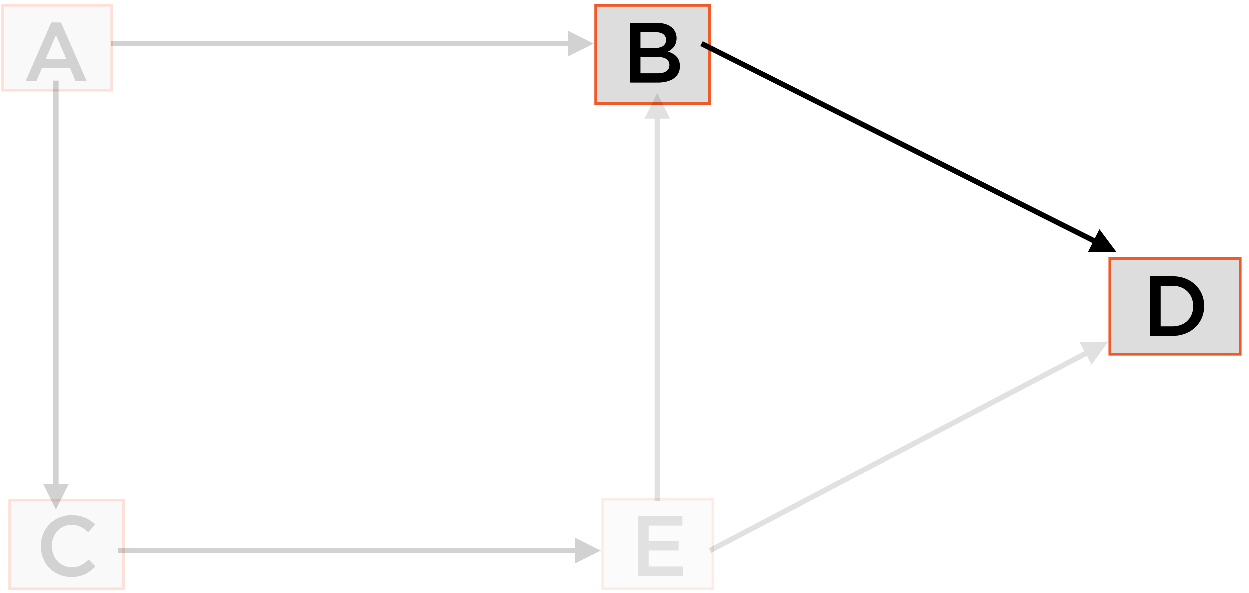
D

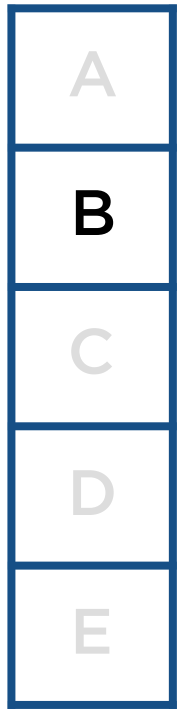
E

B and D

Two edges emanate from A, A->B and A->C

Topological Sort





"Comes Before"

B and C

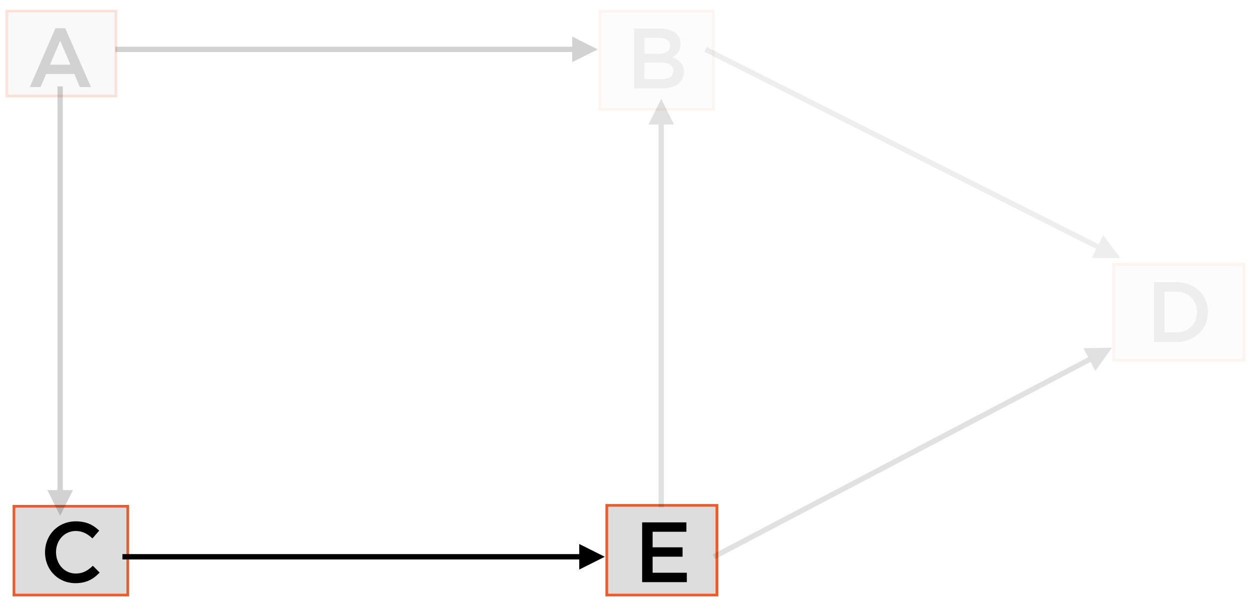
D

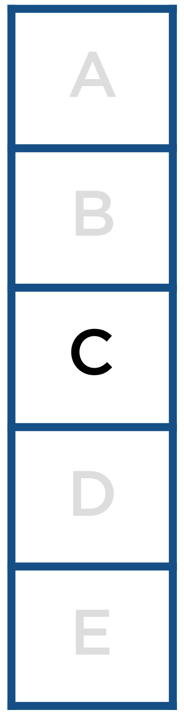
E

B and D

One edge emanates from B, B->D

Topological Sort





"Comes Before"

B and C

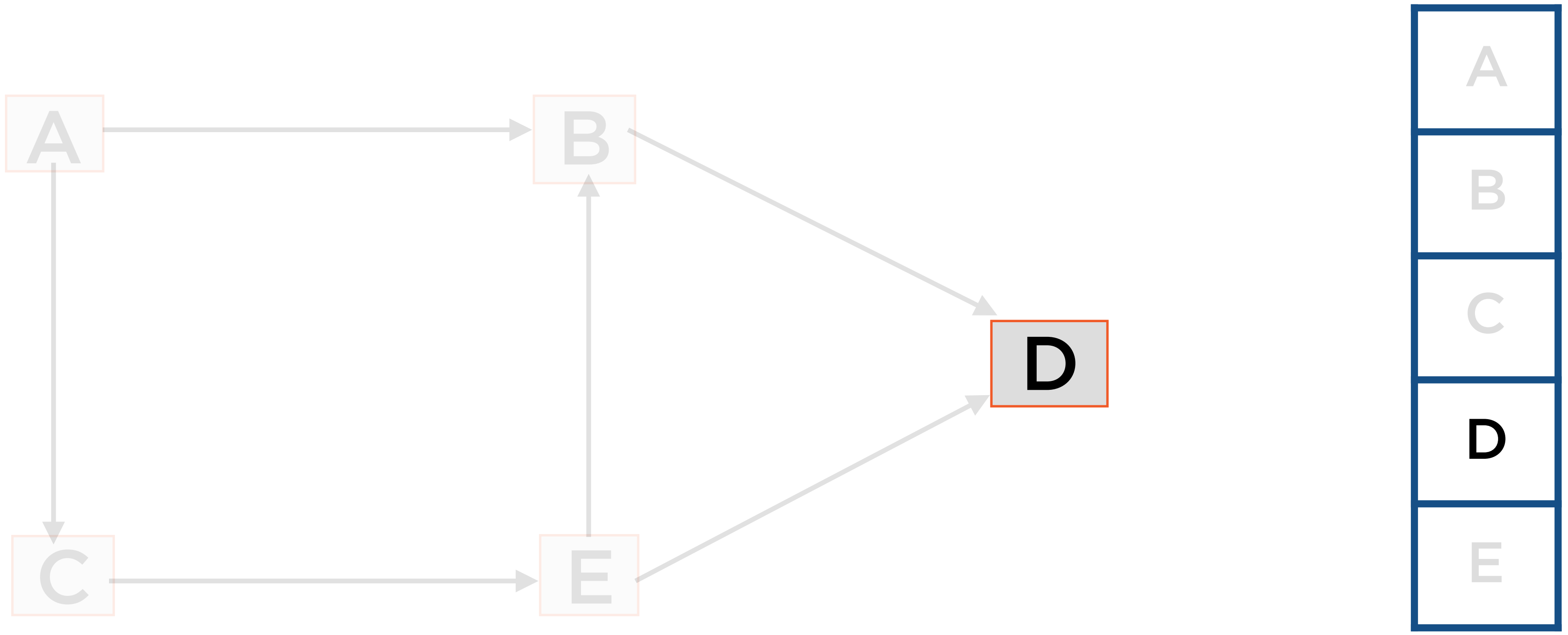
D

E

B and D

One edge emanates from C, C->E

Topological Sort



"Comes Before"

B and C

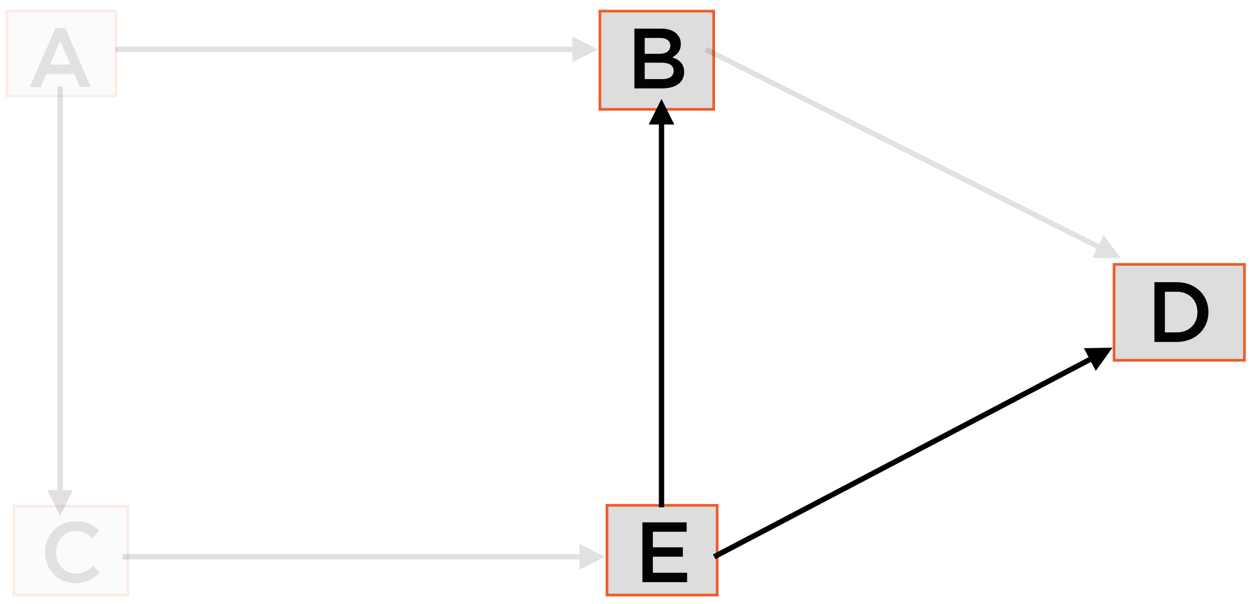
D

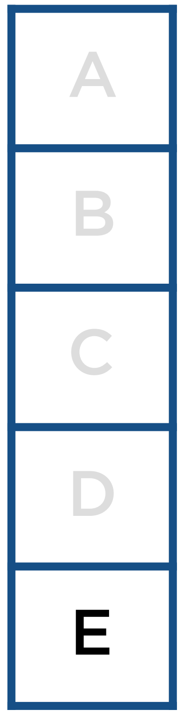
E

B and D

No edges emanate from D

Topological Sort





"Comes Before"

B and C

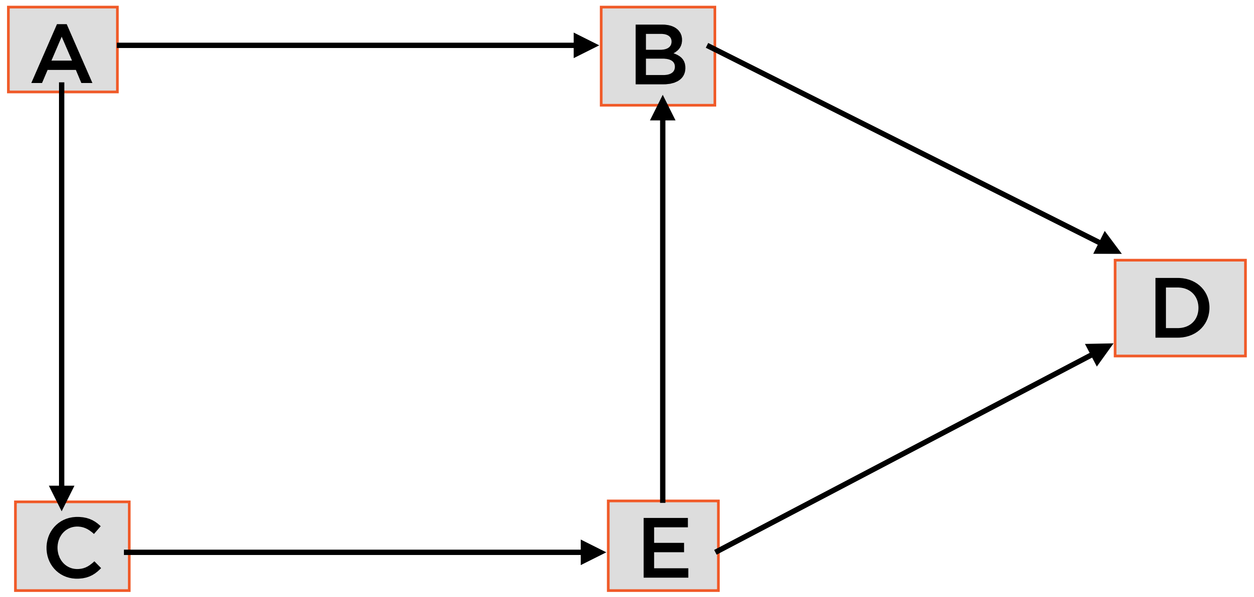
D

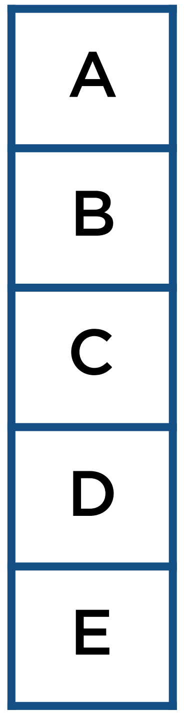
E

B and D

Two edges emanate from E, E->B and E->D

Topological Sort





“Comes Before”

B and C

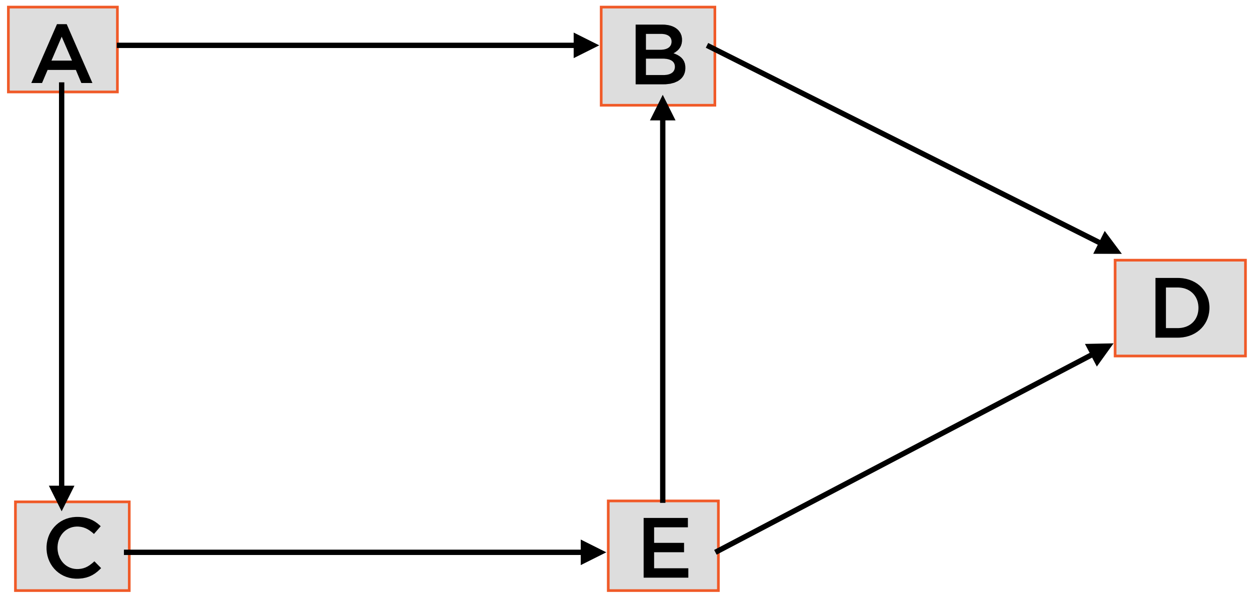
D

E

B and D

DAGs specify precedence relationships

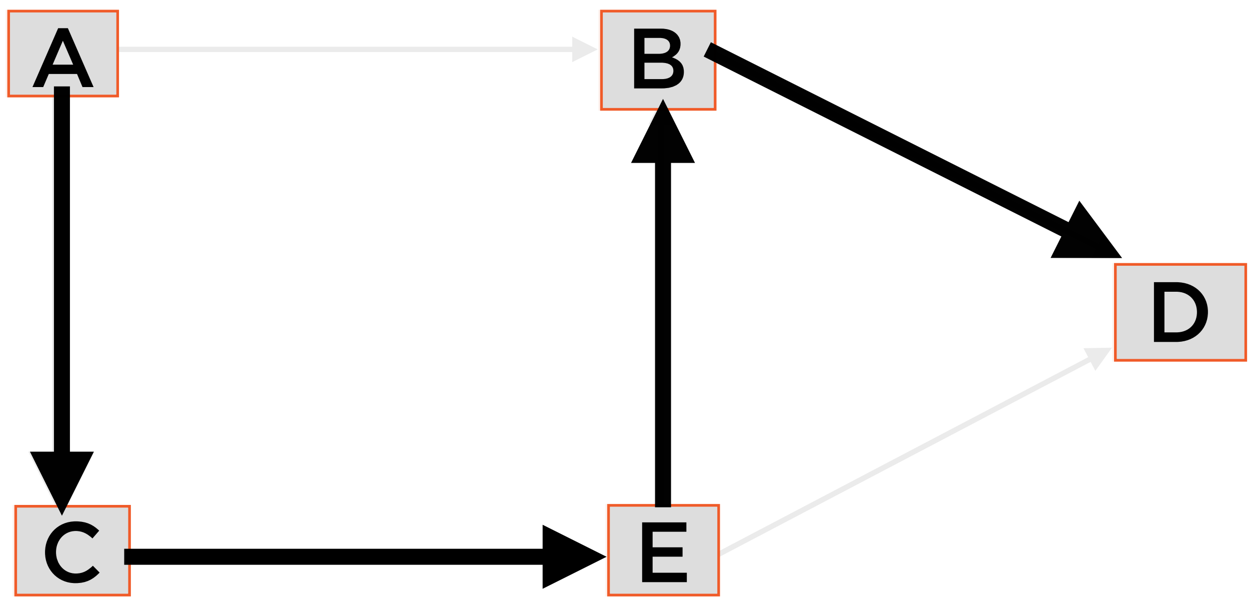
Topological Sort



A -> C -> E -> B -> D

Here, only 1 acceptable ordering of vertices

Topological Sort



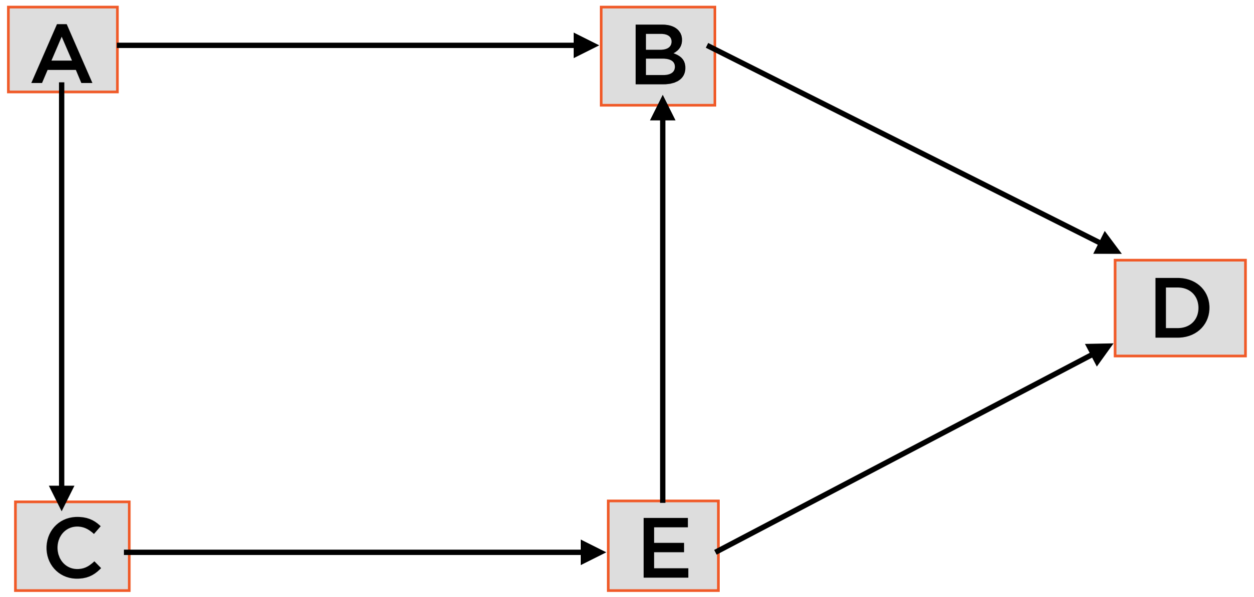
A -> C -> E -> B -> D

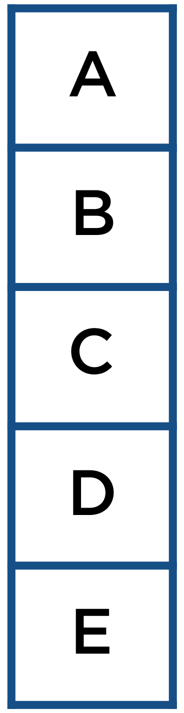
Topological sort

A Topological Sort is any ordering of all the graph's vertices that satisfies all precedence relationships

The Topological Sort Implementation

Topological Sort





"Comes **Before**"

B and C

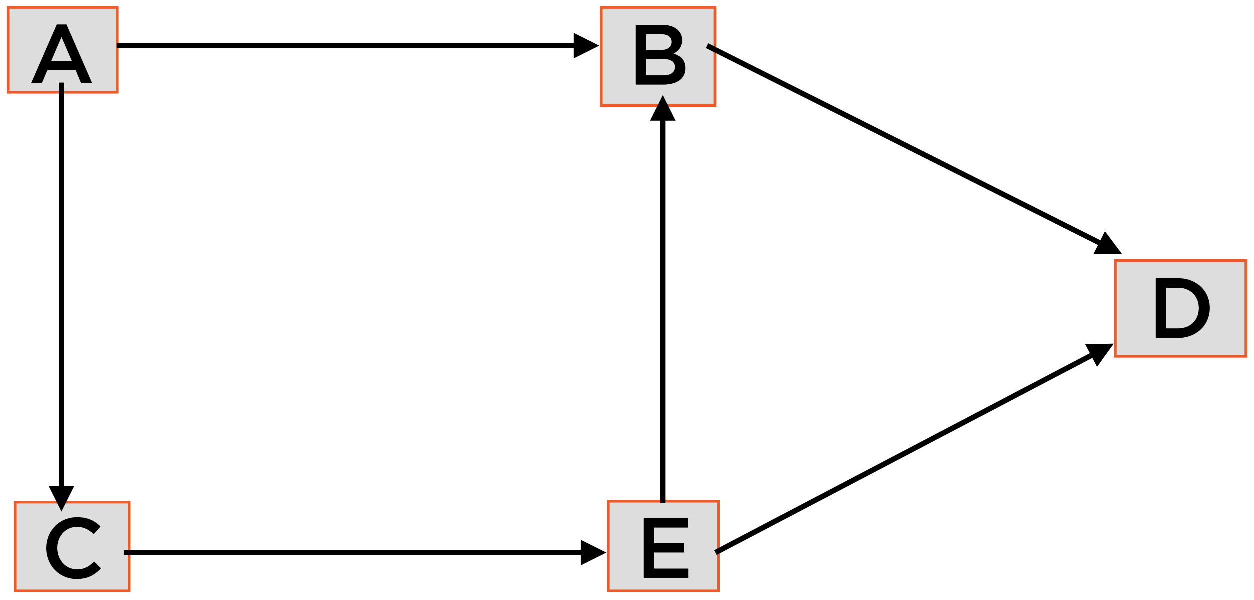
D

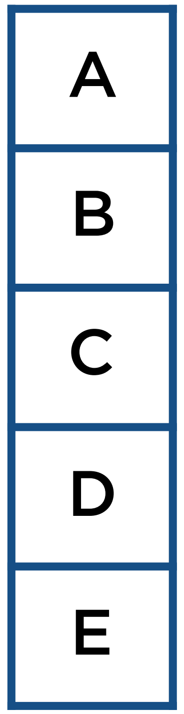
E

B and D

DAGs specify precedence relationships

Topological Sort





“Comes After”

A and E

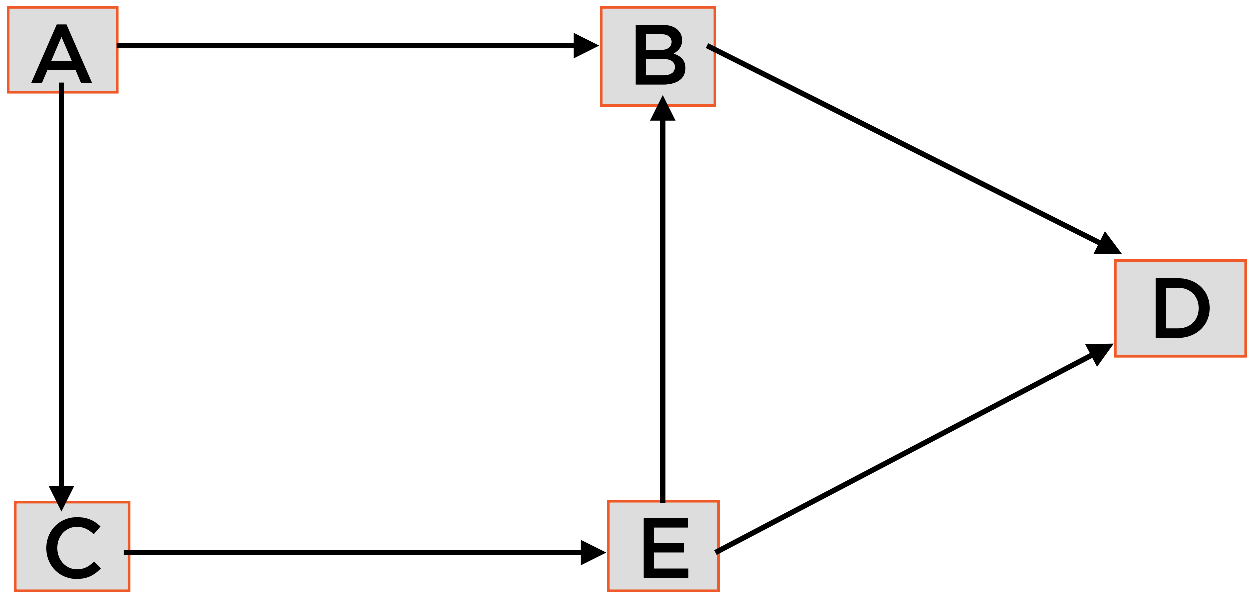
A

B and E

C

DAGs specify precedence relationships

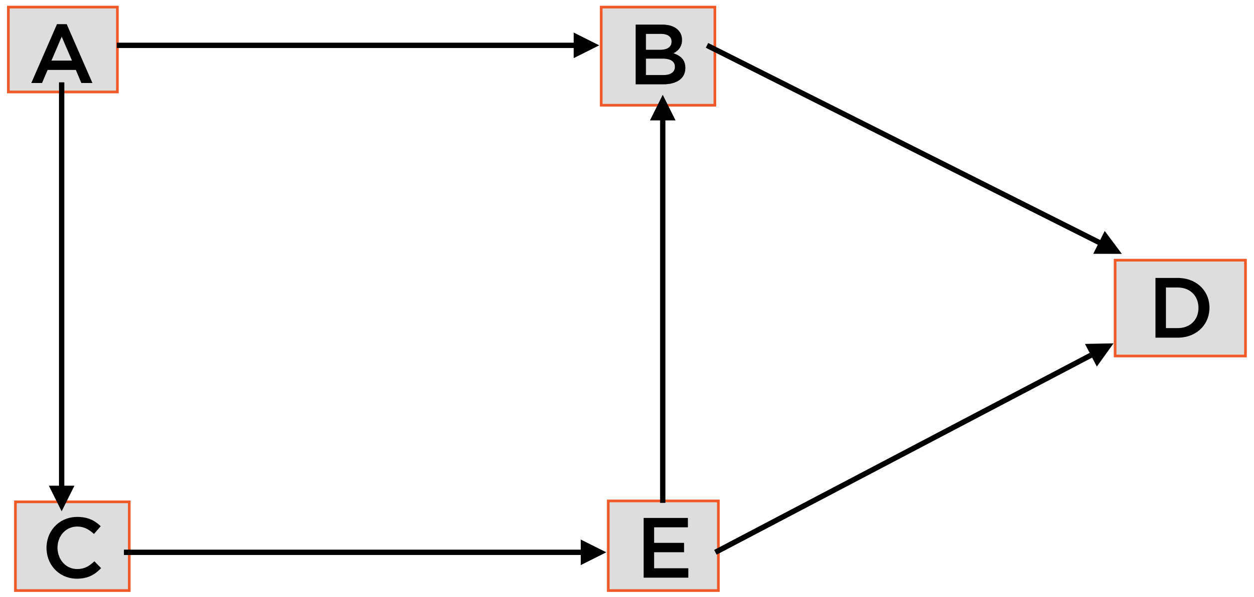
In-degree

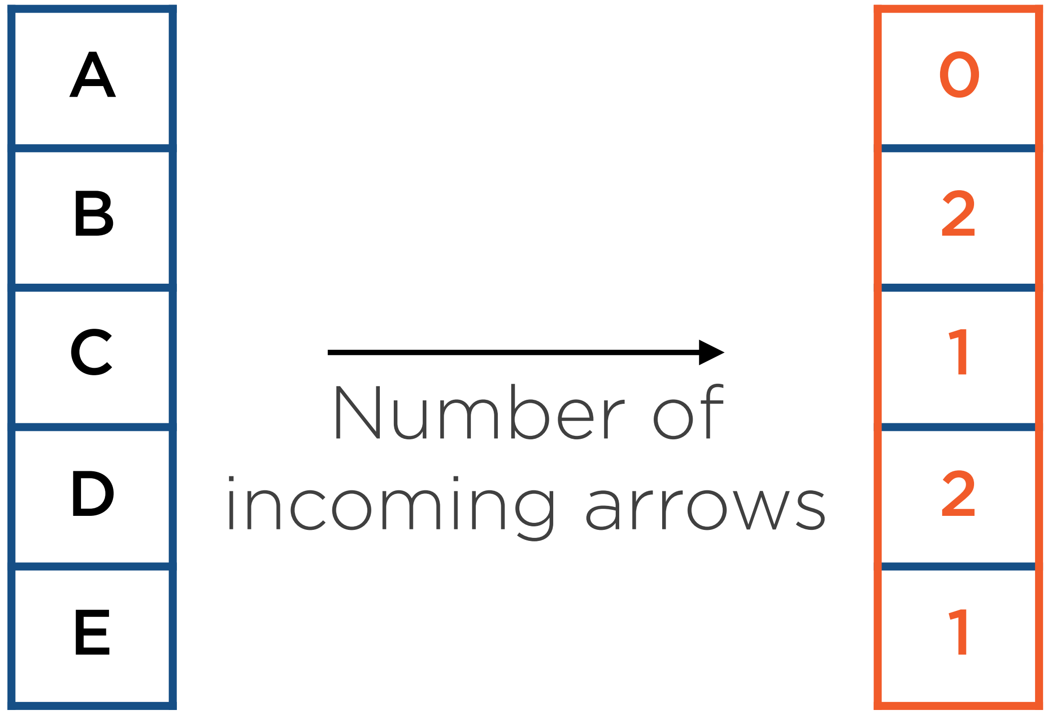


	"Comes After"	"In-degree"
A		0
B	A and E	2
C	A	1
D	B and E	2
E	C	1

The in-degree of a node is the number of directed edges that directly flow into that node

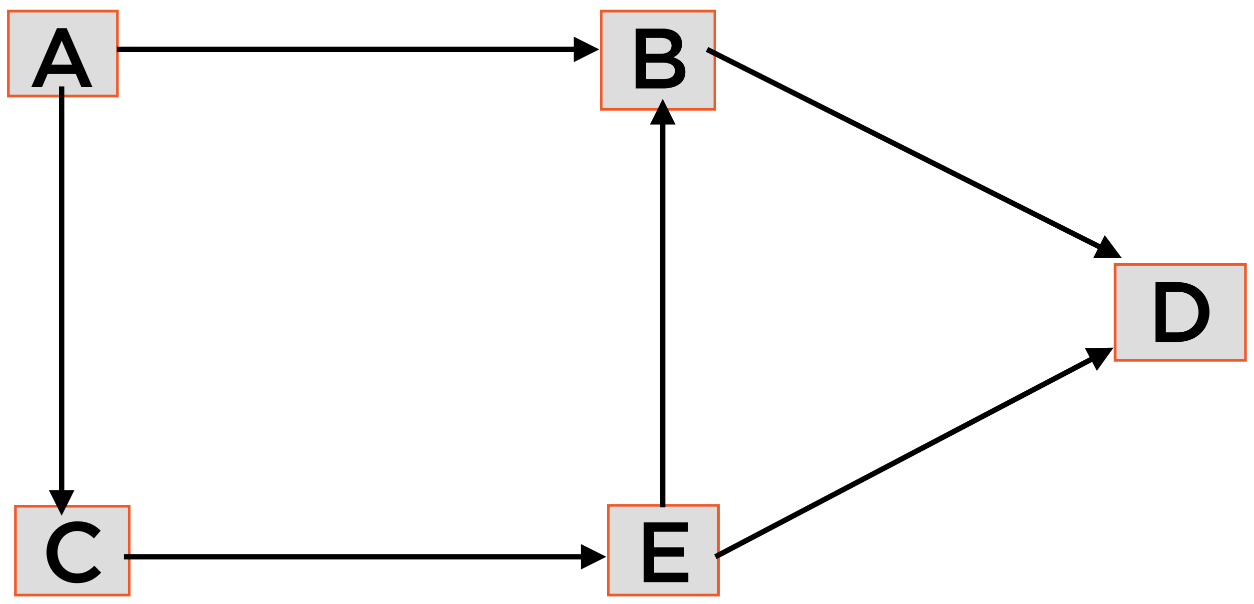
In-degree

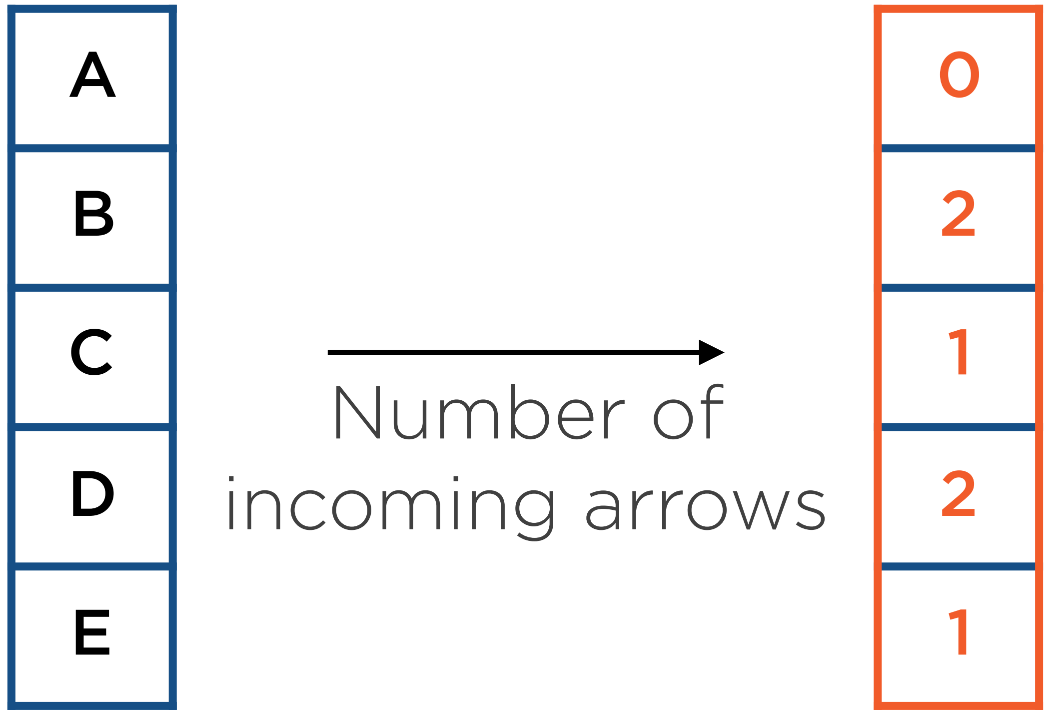




The in-degree of a node is the number of directed edges that directly flow into that node

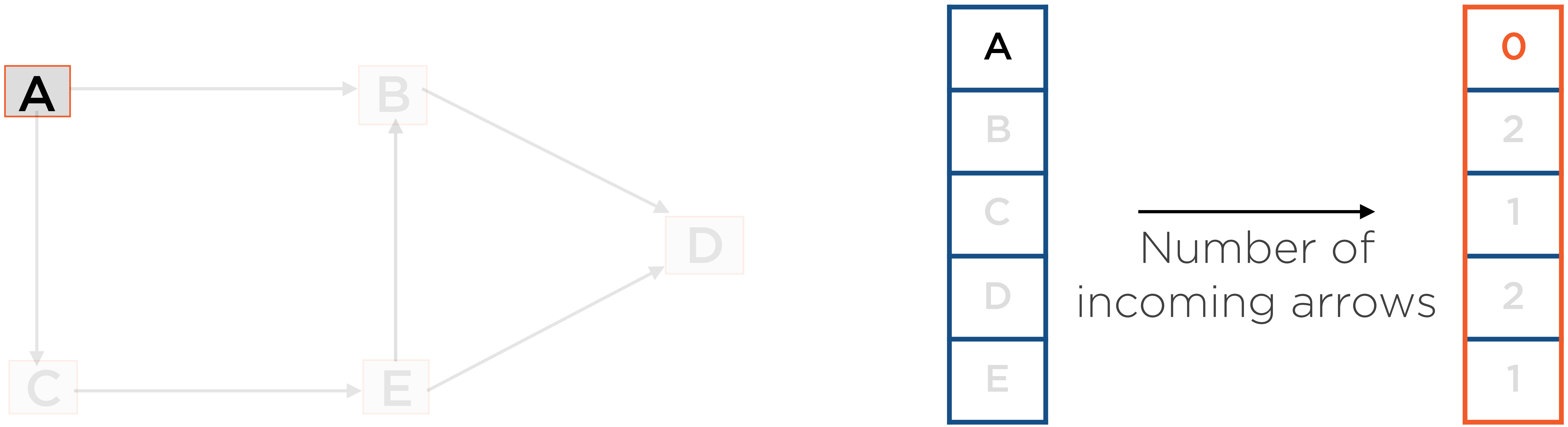
Starting Node: In-degree = 0





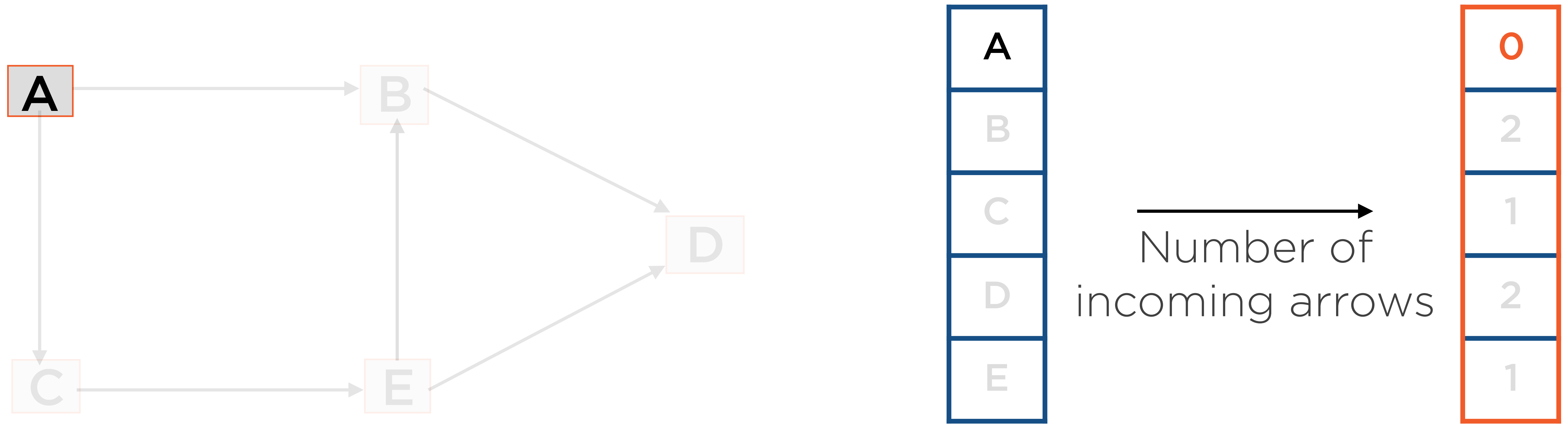
Start the topological sort procedure by visiting any node that has in-degree = 0

Starting Node: In-degree = 0



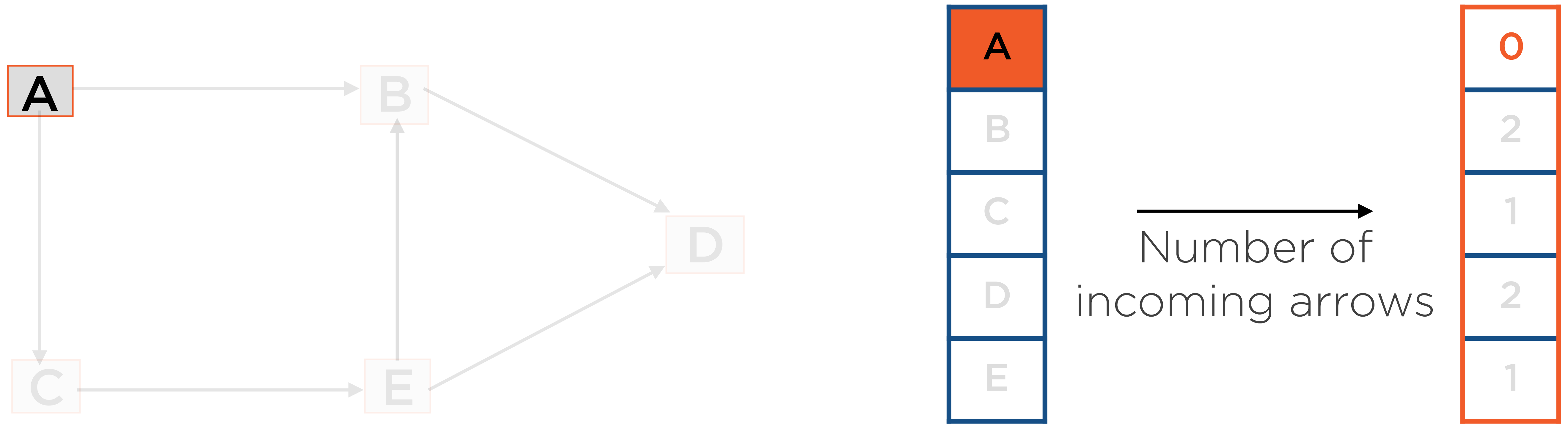
Start the topological sort procedure by visiting any node that has in-degree = 0

Starting Node: In-degree = 0



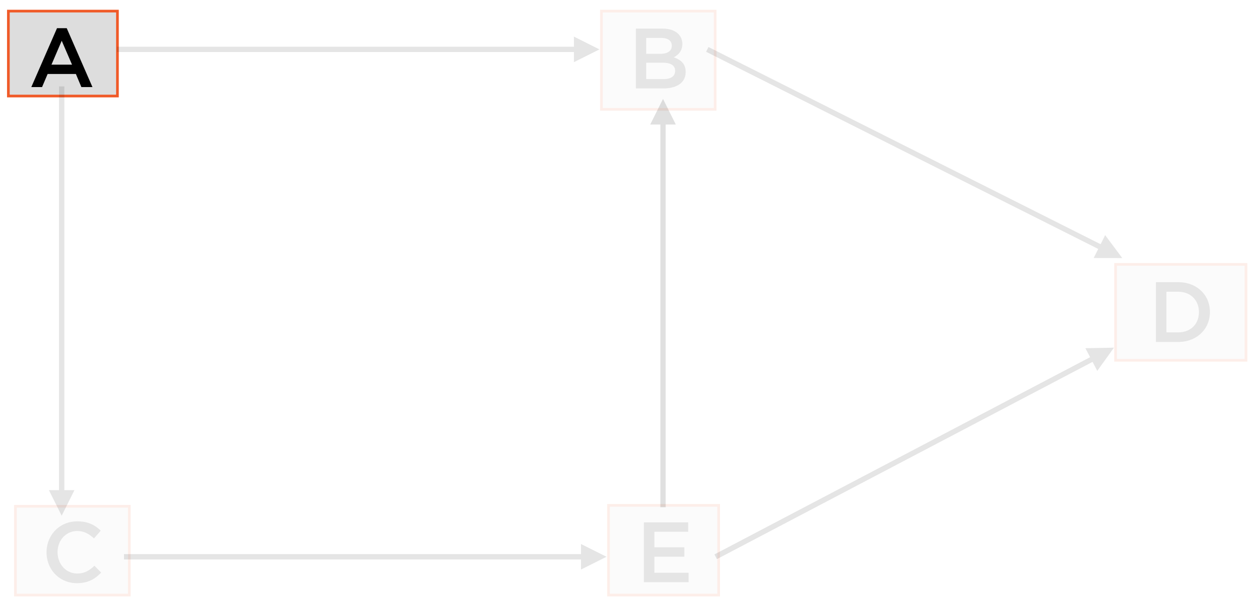
If no node has in-degree = 0, the graph has a cycle (i.e. it is not a DAG, topological sort not defined)

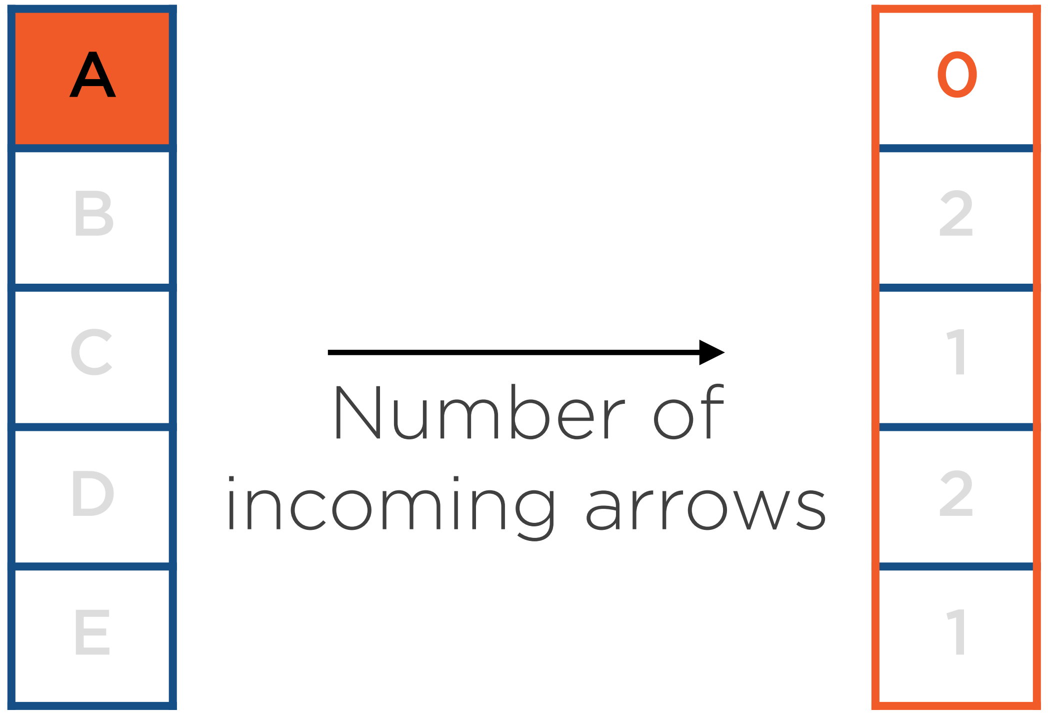
Starting Node: In-degree = 0



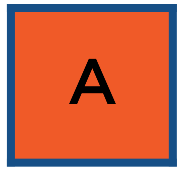
Add A to the result of our topological sort

Starting Node: In-degree = 0

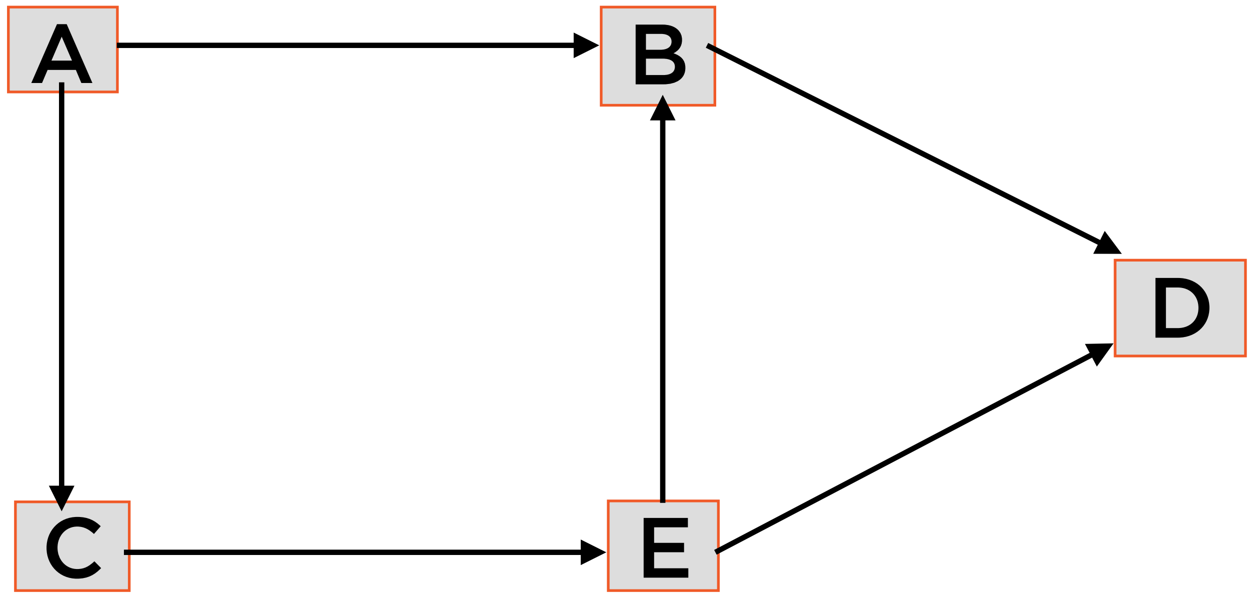


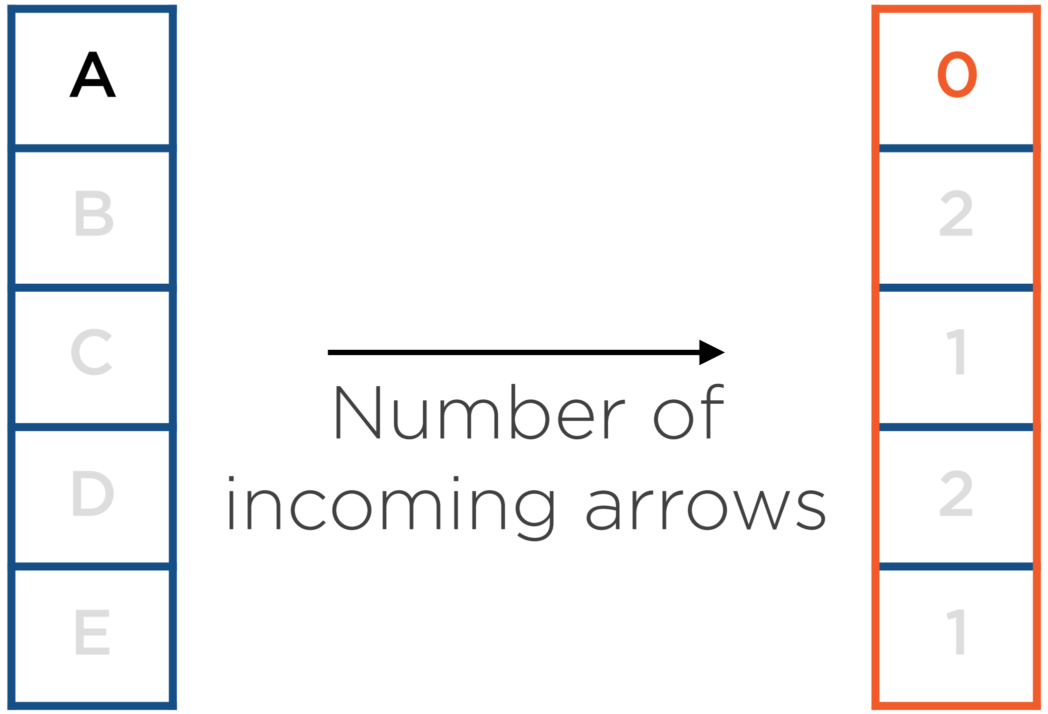


Result



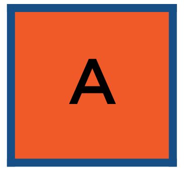
Starting Node: In-degree = 0



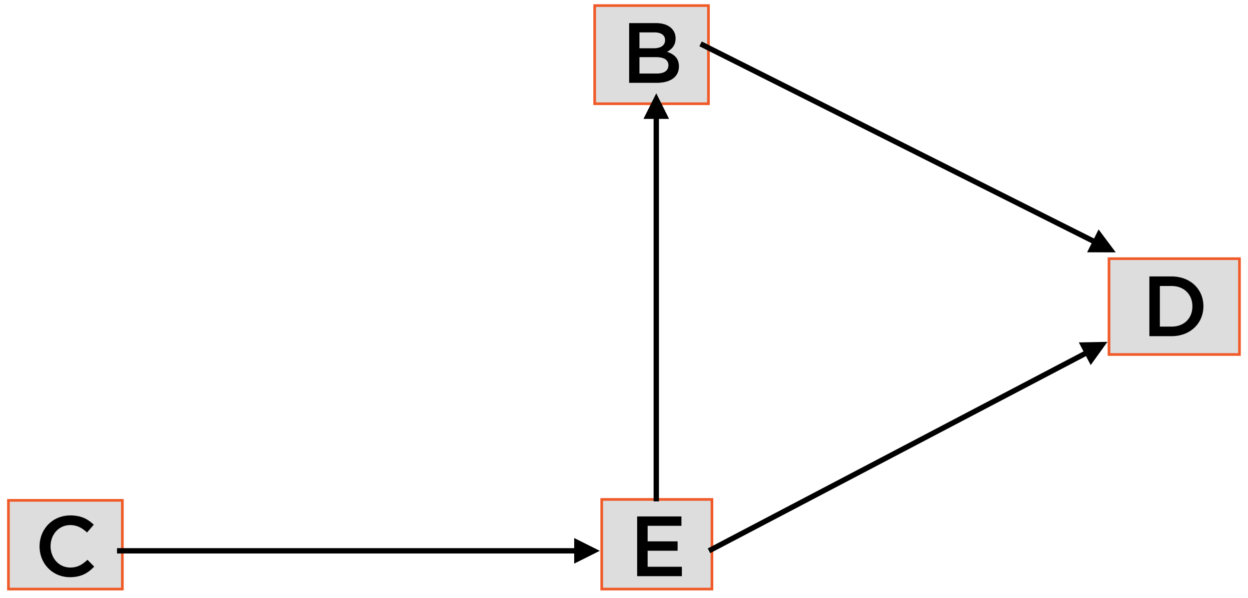


Remove A from our original graph

Result



Starting Node: In-degree = 0

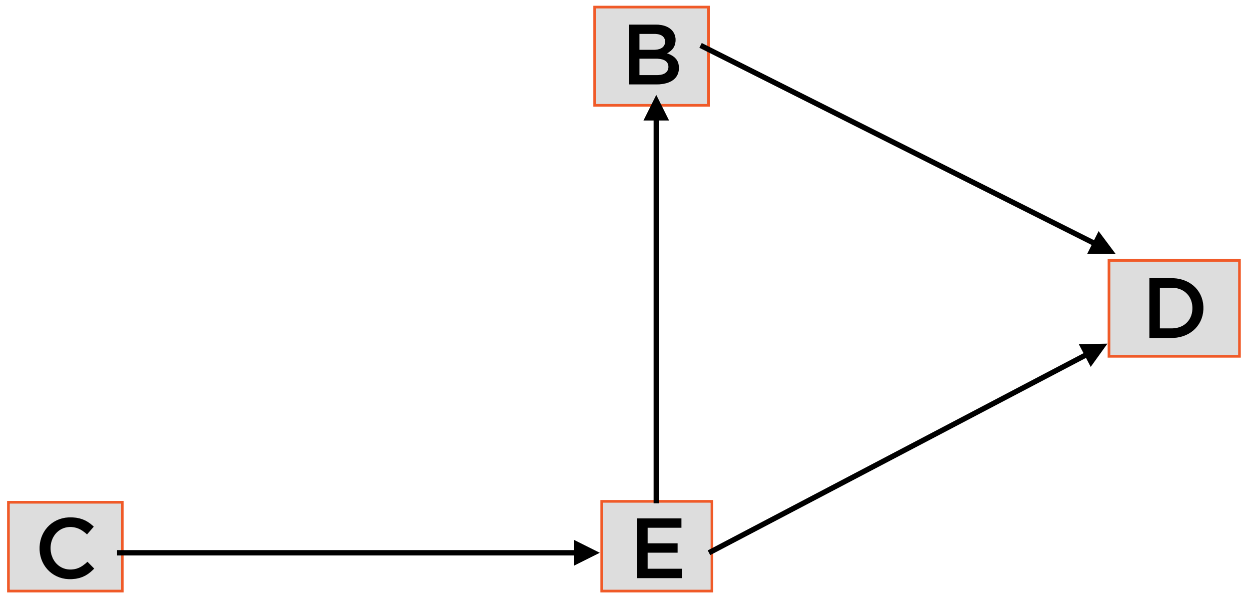


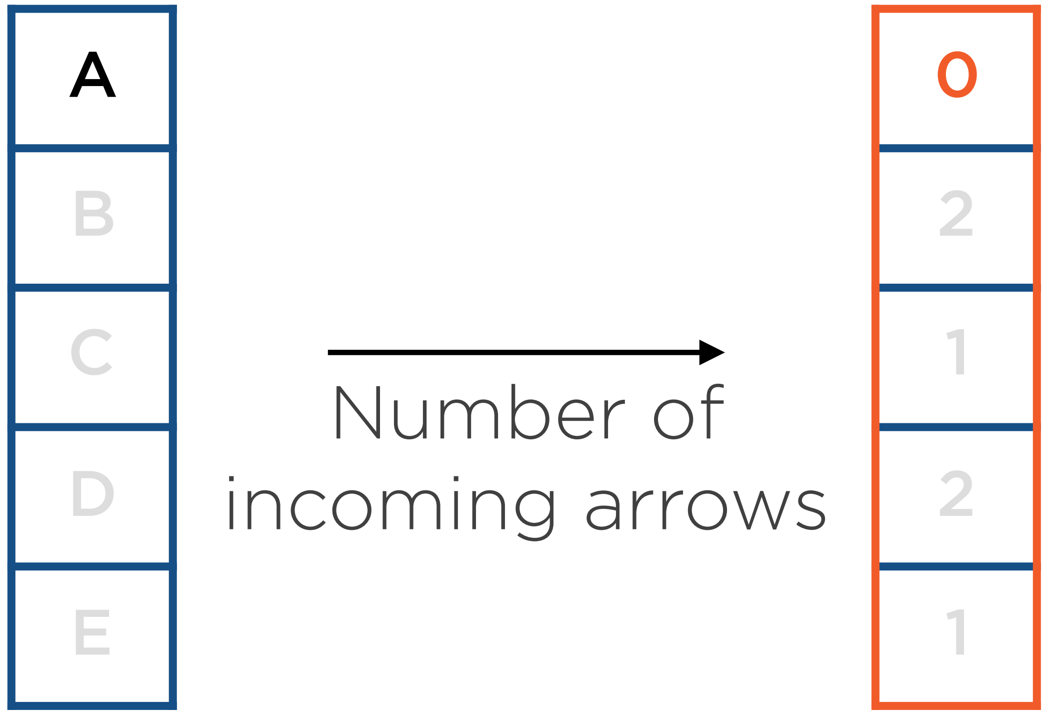
A	0
B	2
C	1
D	2
E	1

Number of incoming arrows

Result **A**

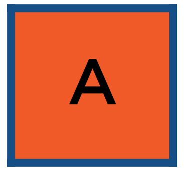
Starting Node: In-degree = 0



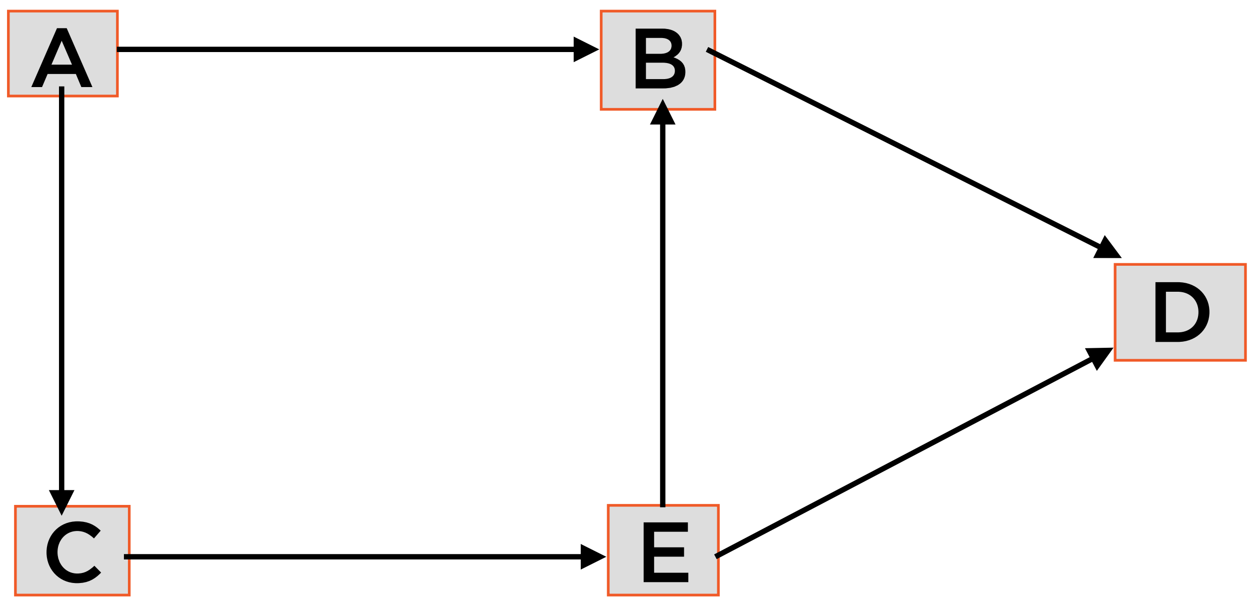


Decrement the in-degree of all nodes that "come after" A

Result

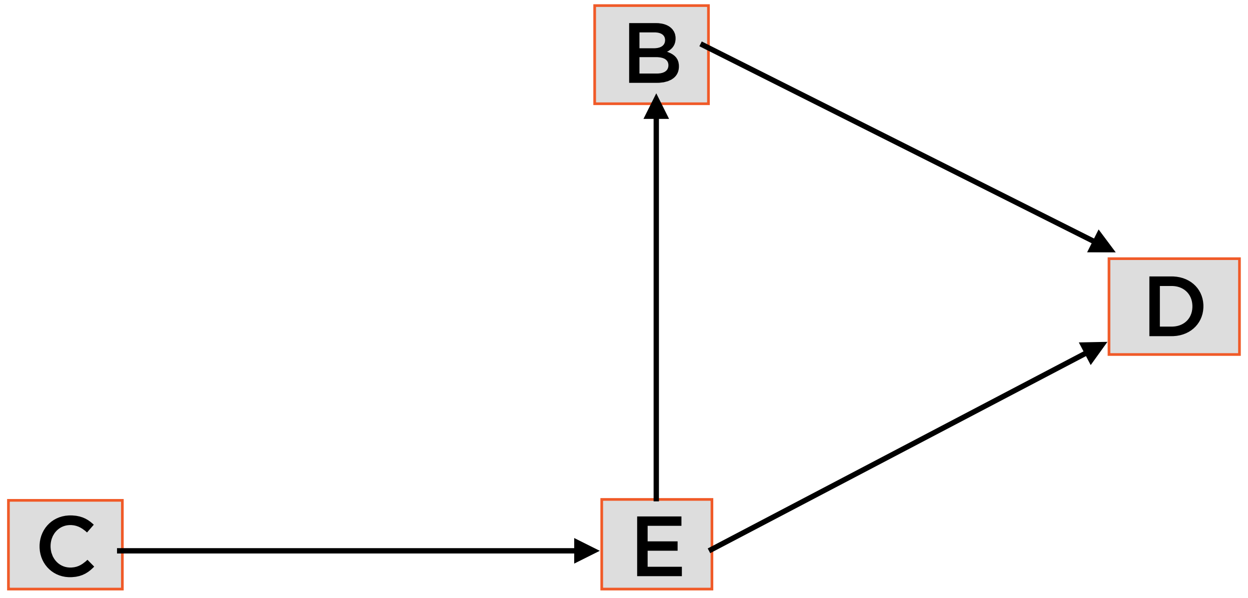


Before Removing A



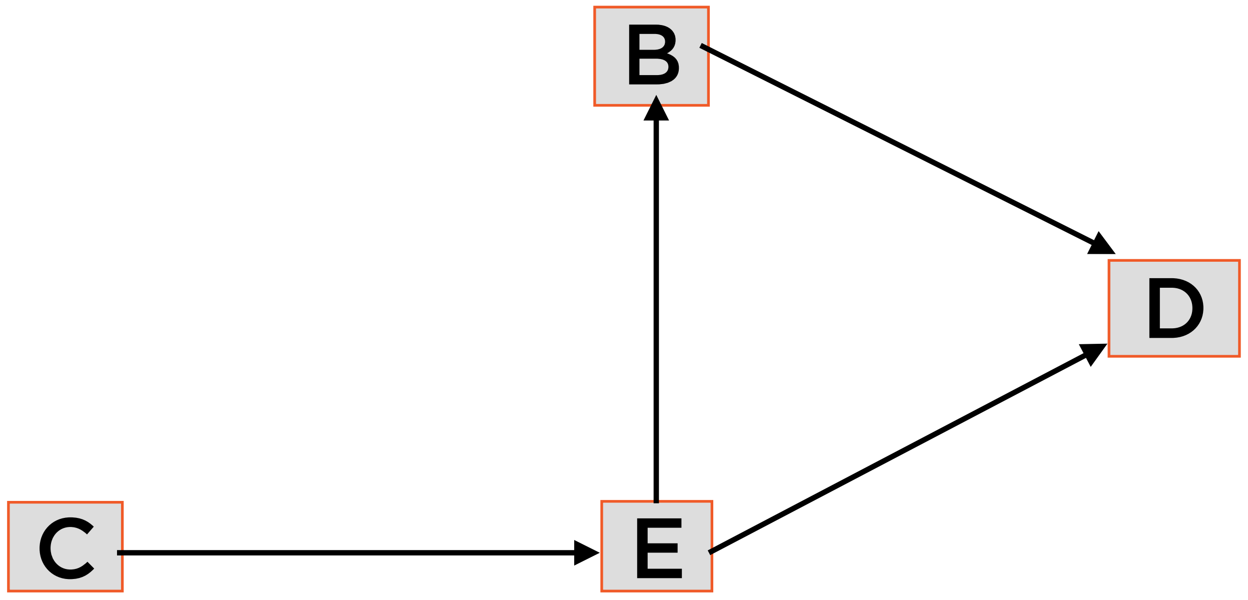
	"Comes After"	"In-degree"
A		0
B	A and E	2
C	A	1
D	B and E	2
E	C	1

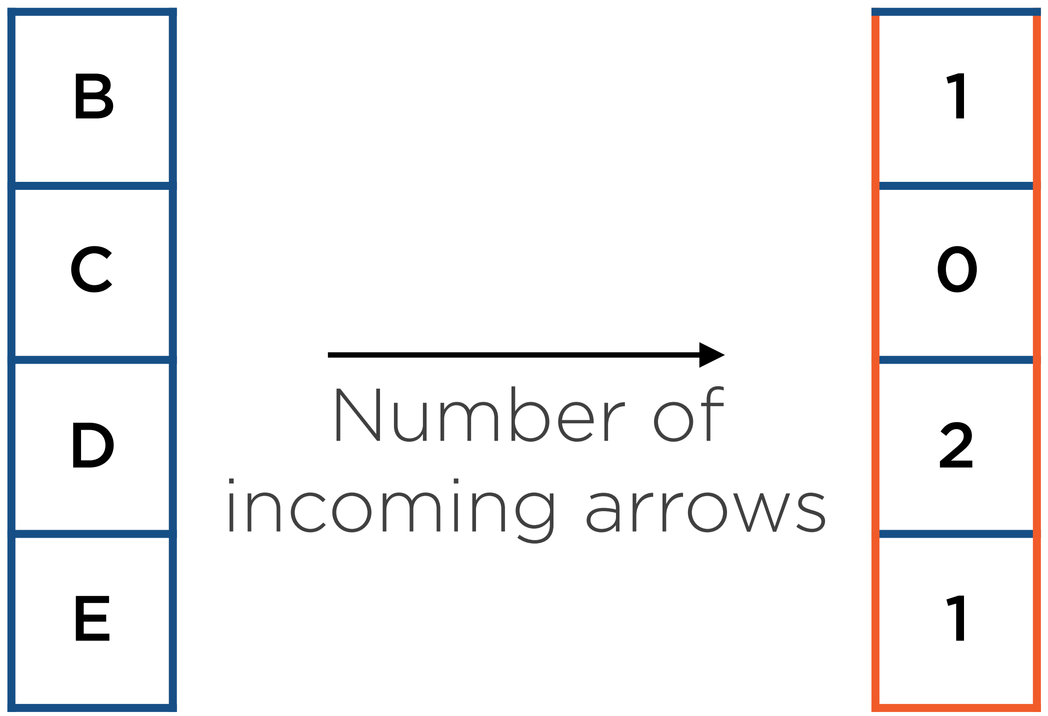
After Removing A



	"Comes After"	"In-degree"
B	E	1
C		0
D	B and E	2
E	C	1

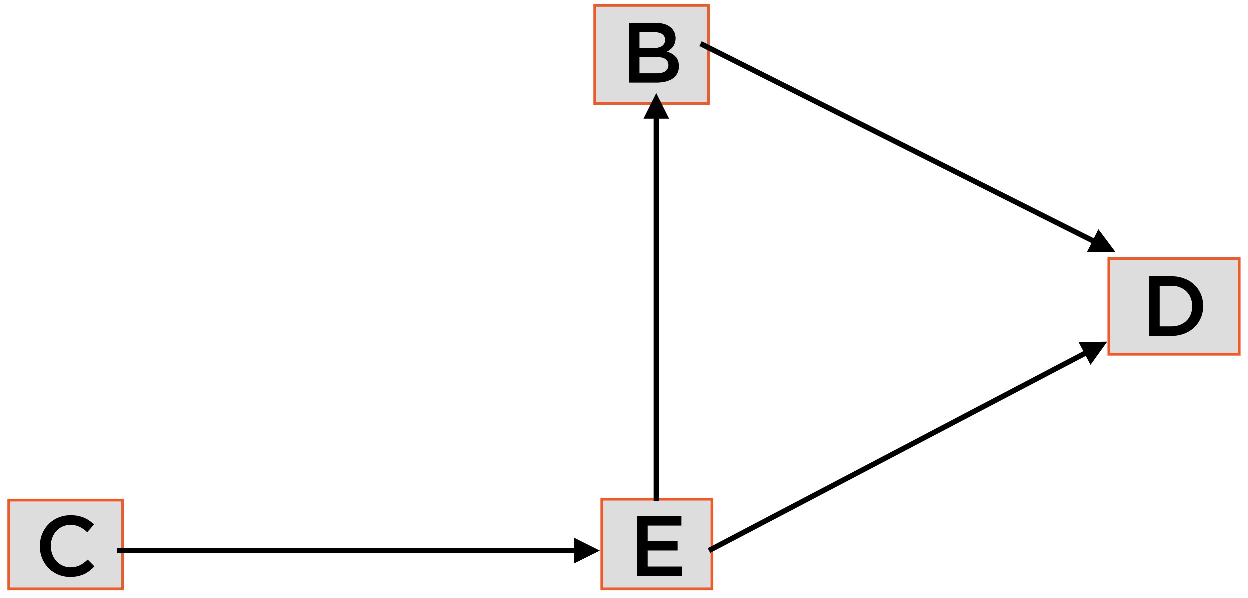
Updated In-degrees

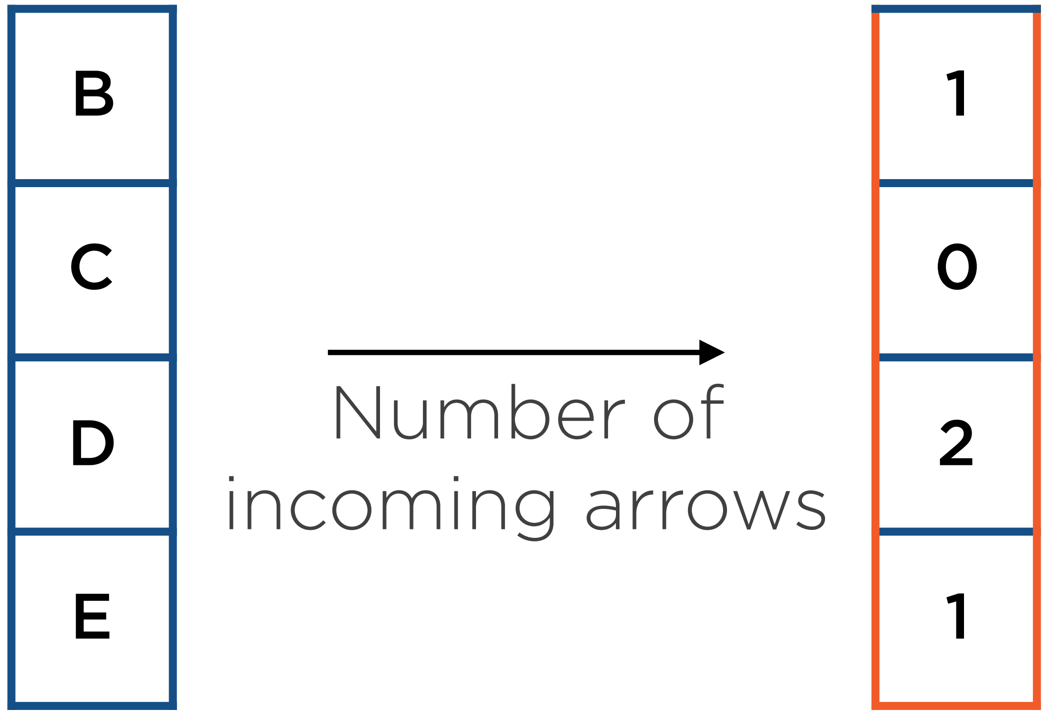




Result A

Rinse-and-Repeat

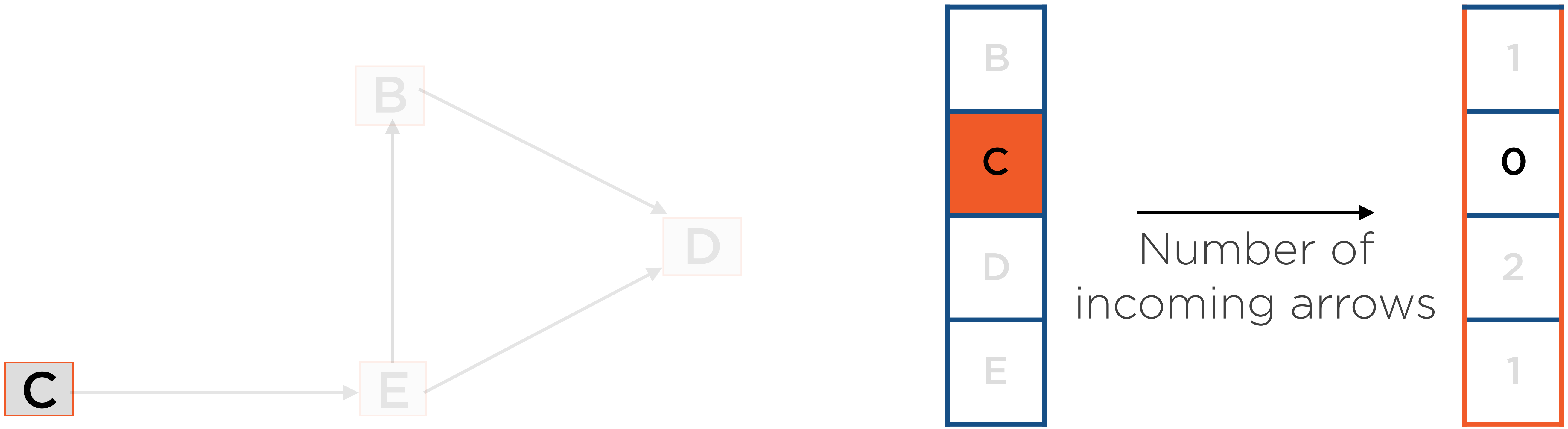




Next, visit any node that has in-degree = 0

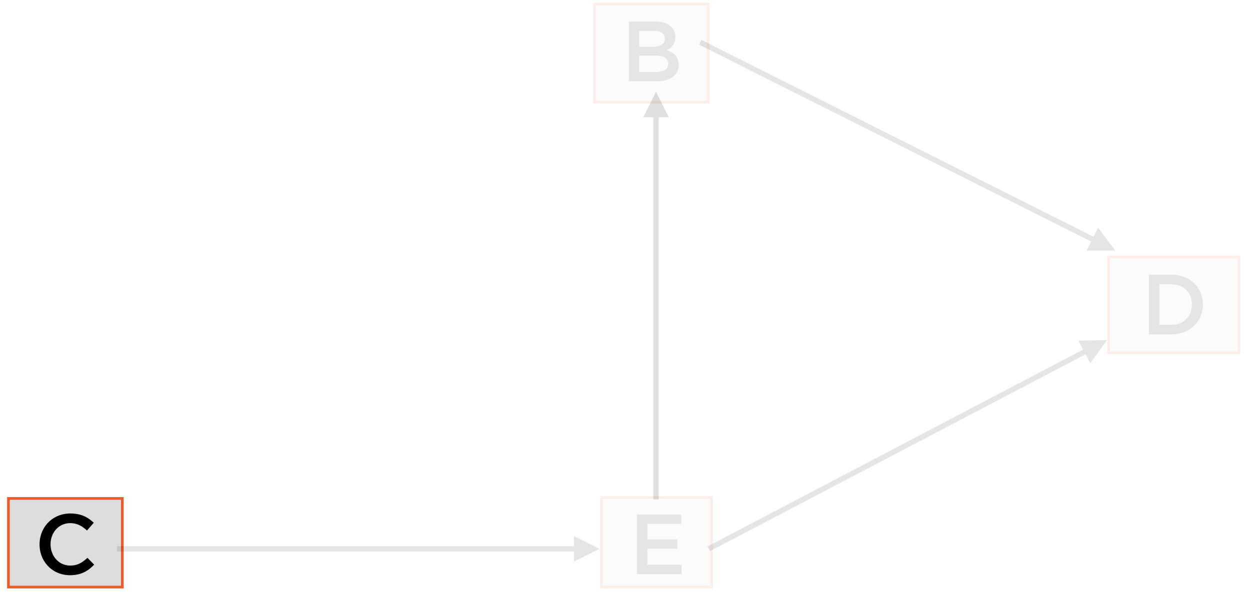
Result A

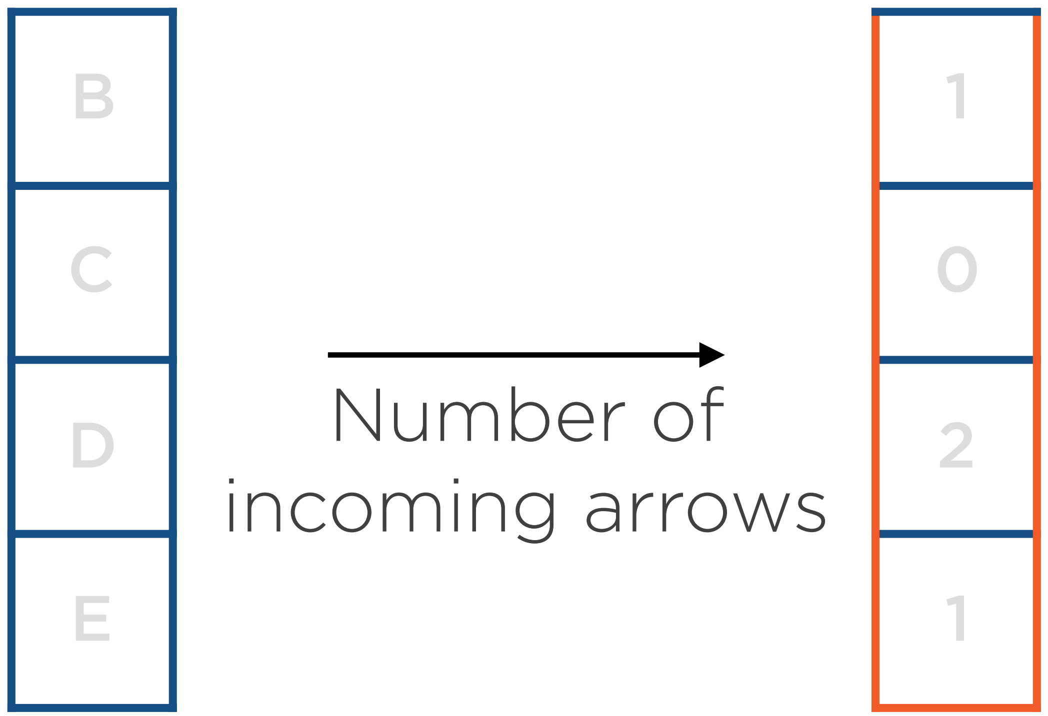
Rinse-and-Repeat



Next, visit any node that has in-degree = 0

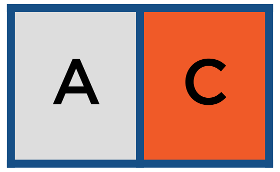
New Starting Node: C



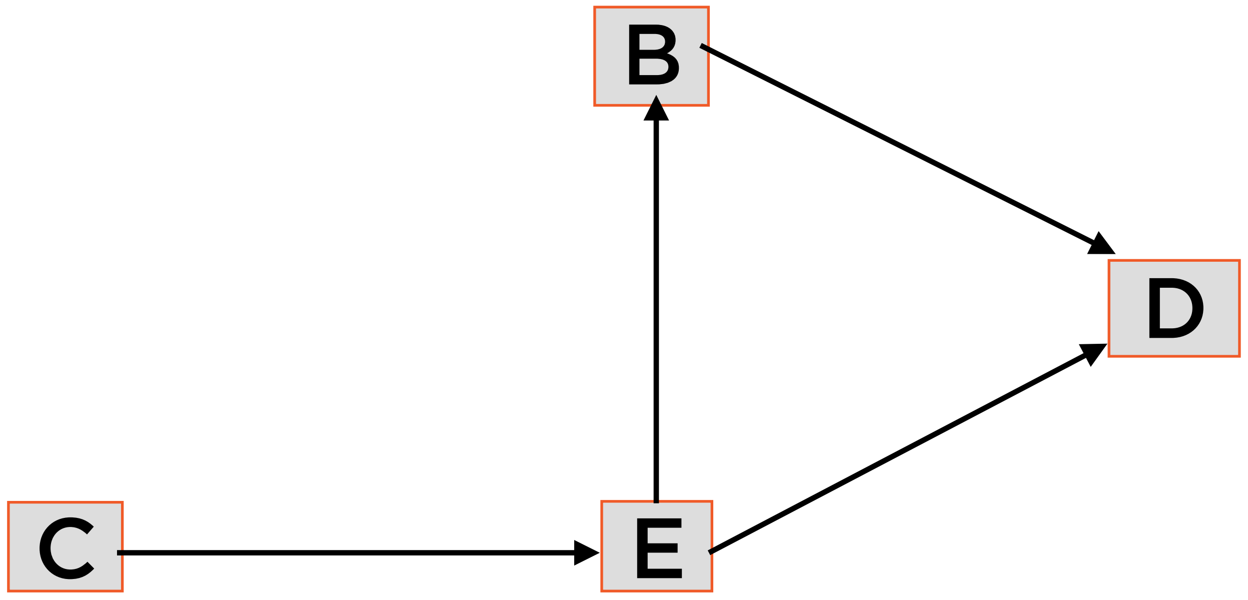


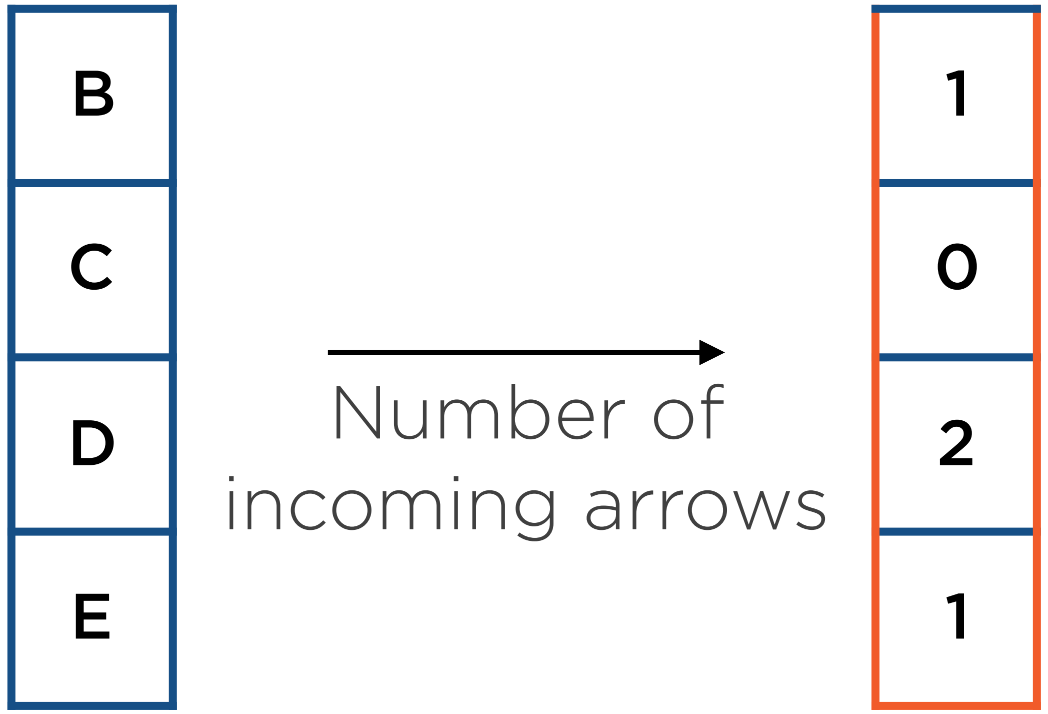
Add C to the result of our topological sort

Result



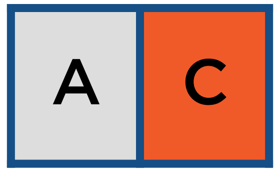
New Starting Node: C



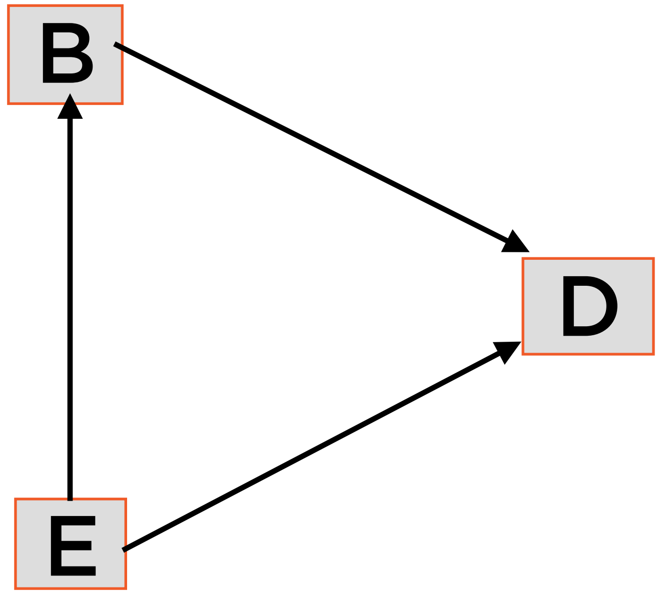


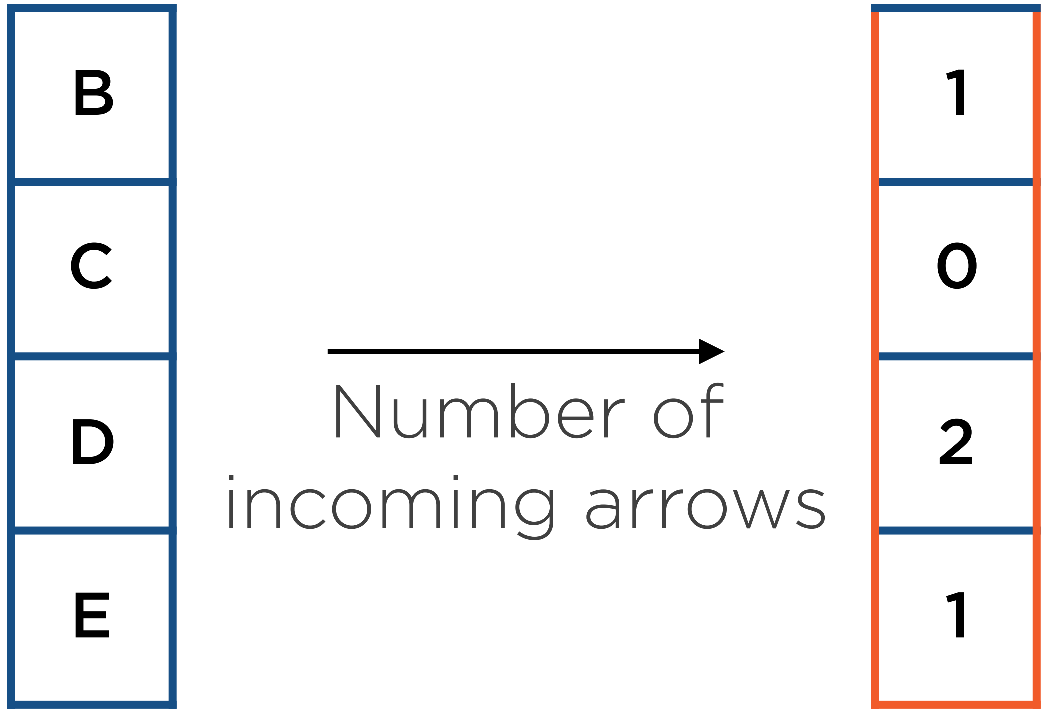
Remove C from our original graph

Result



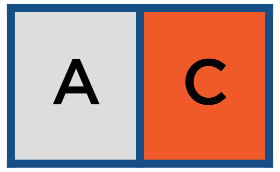
New Starting Node: C



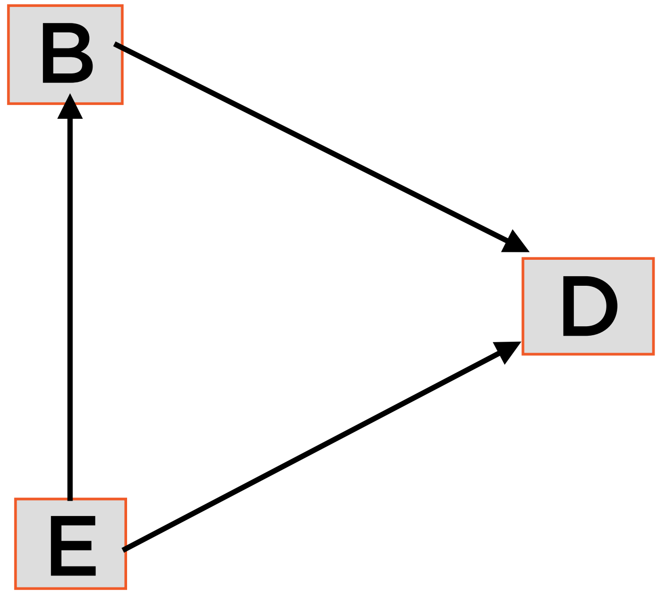


Remove C from our original graph

Result



New Starting Node: C



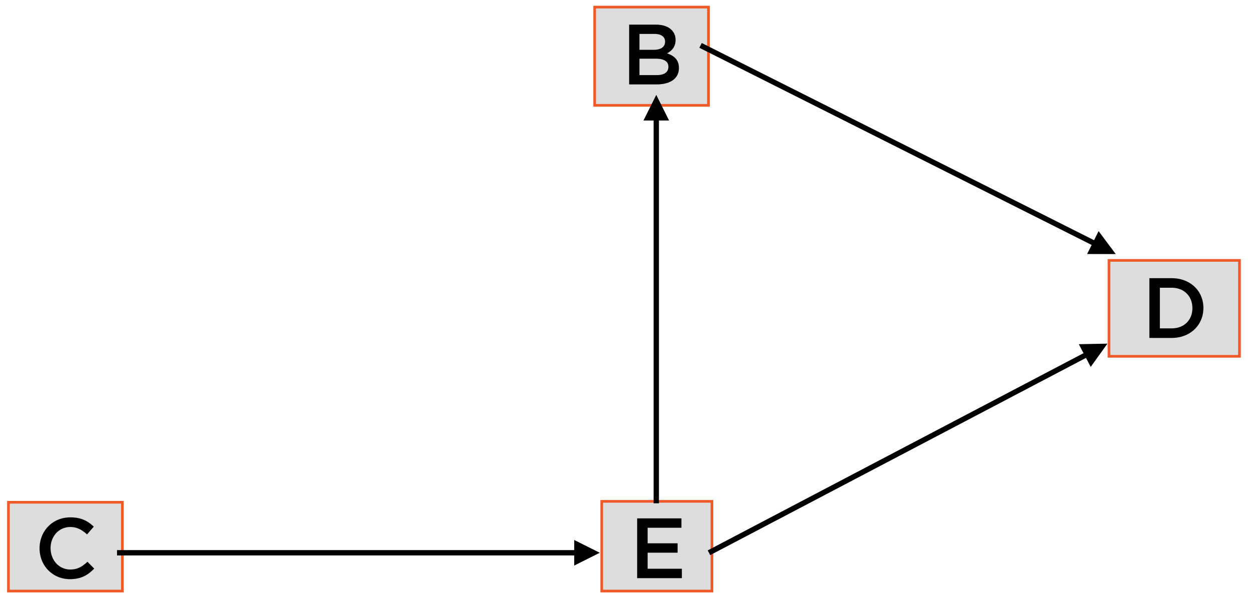
	Number of incoming arrows
B	1
C	0
D	2
E	1

Decrement the in-degree of all nodes that “come after” C

Result

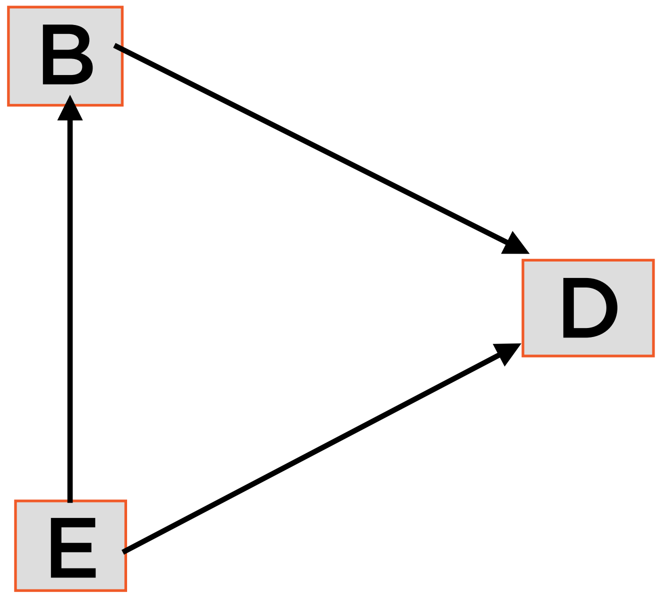
A	C

Before Removing C



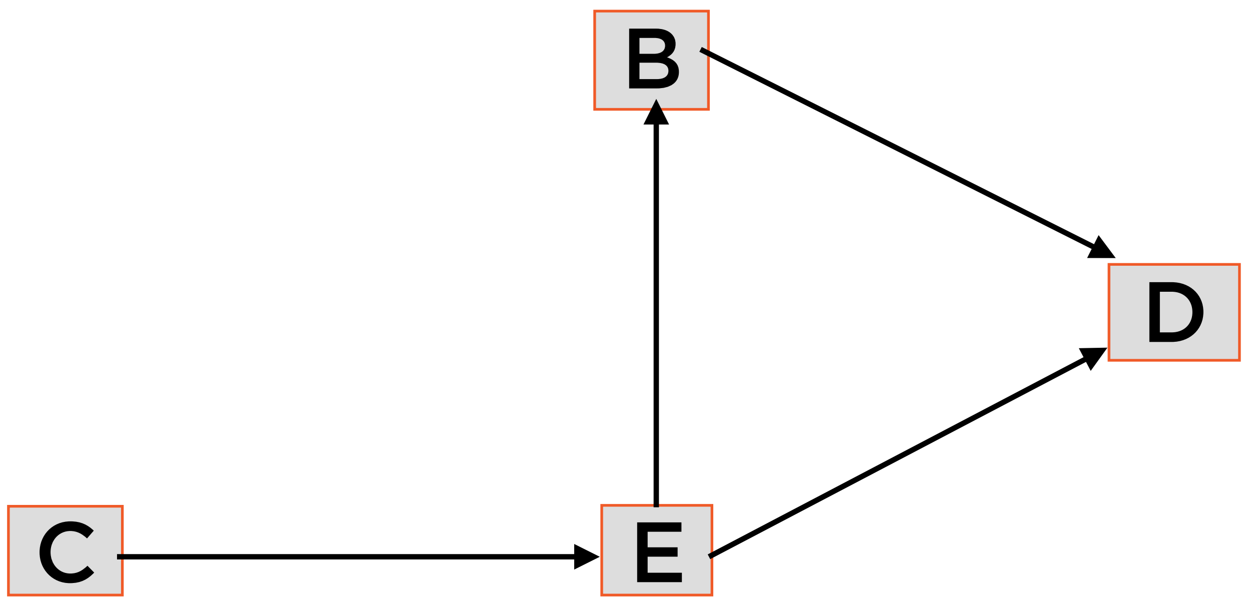
	"Comes **After**"	"In-degree"
B	E	1
C		0
D	B and E	2
E	C	1

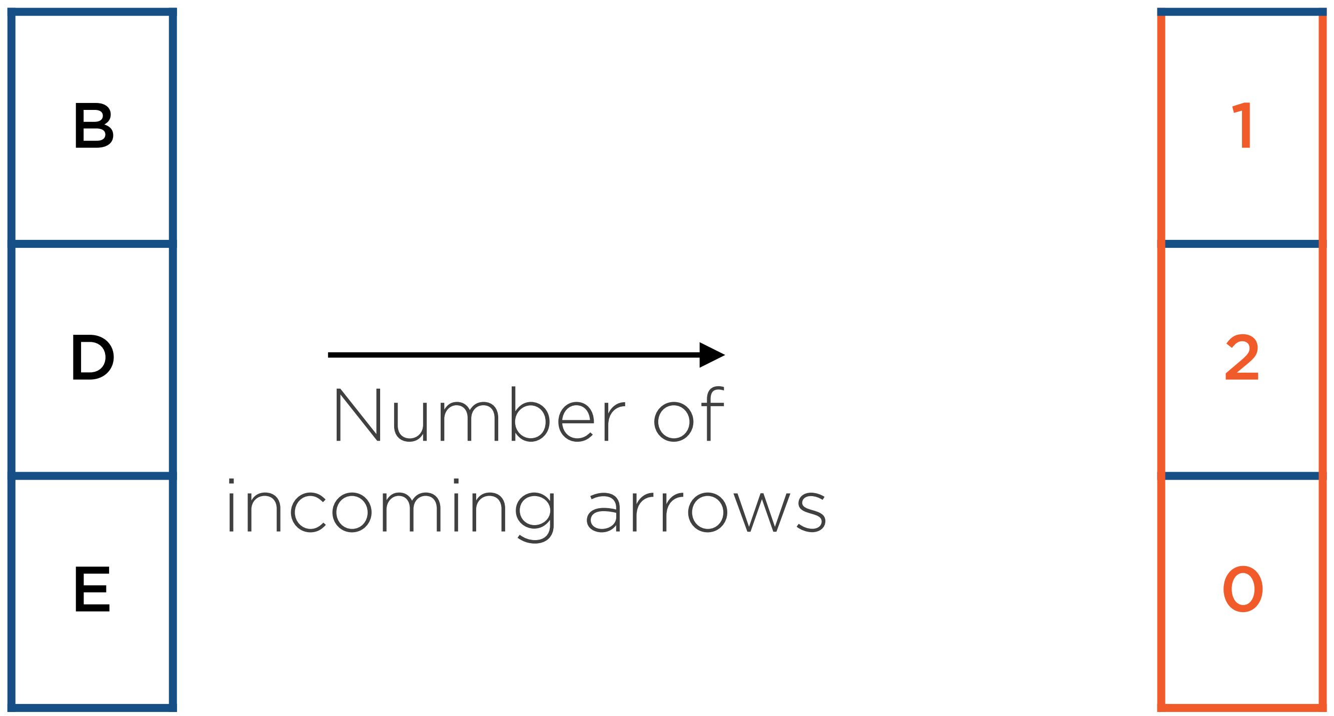
After Removing C



	"Comes After"	"In-degree"
B	E	1
D	B and E	2
E		0

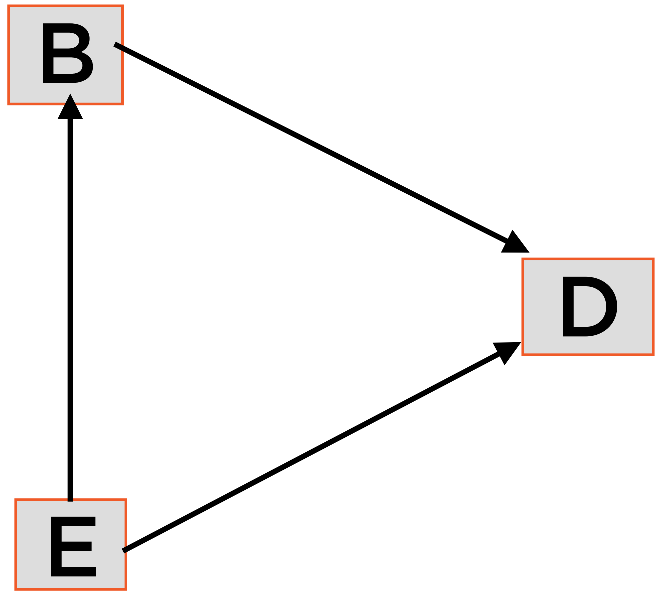
Updated In-degrees

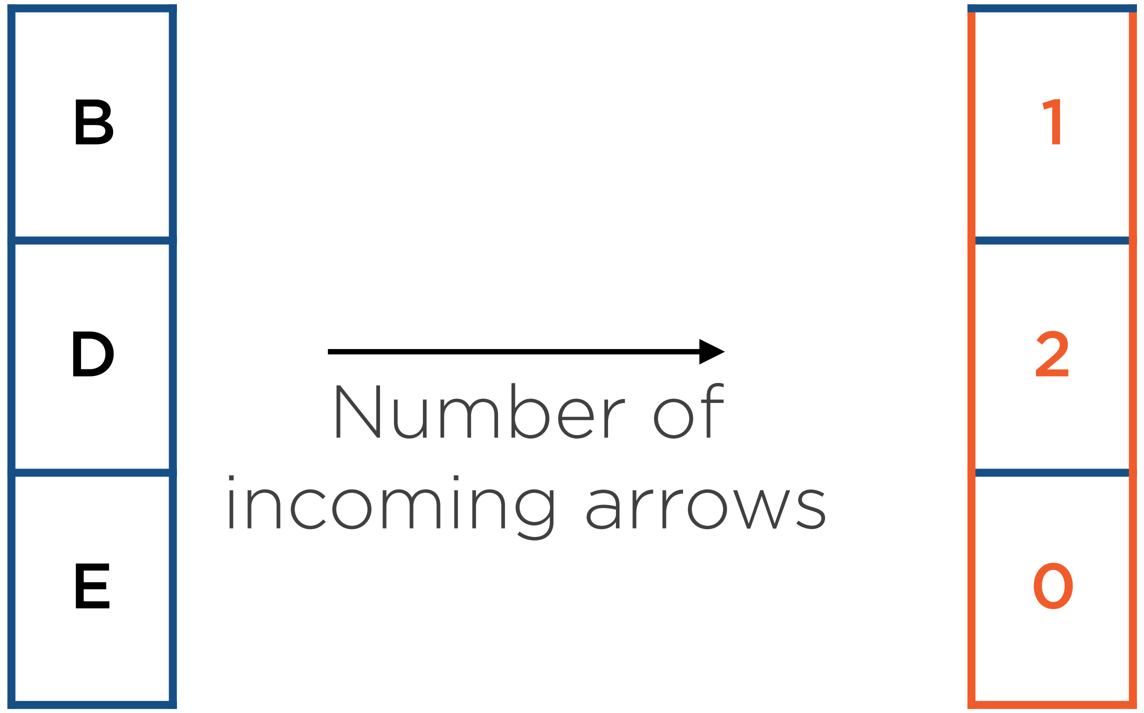




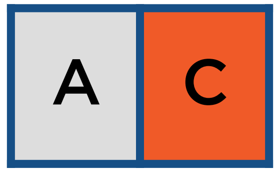
Remove C from our original graph

Updated In-degrees

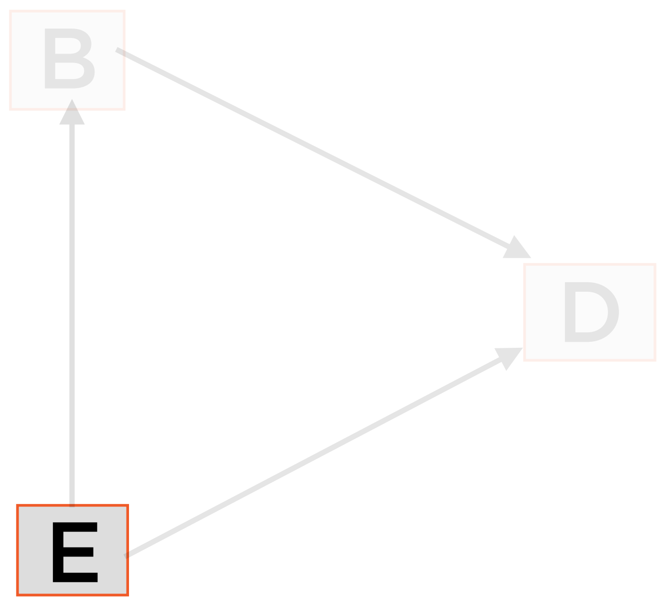


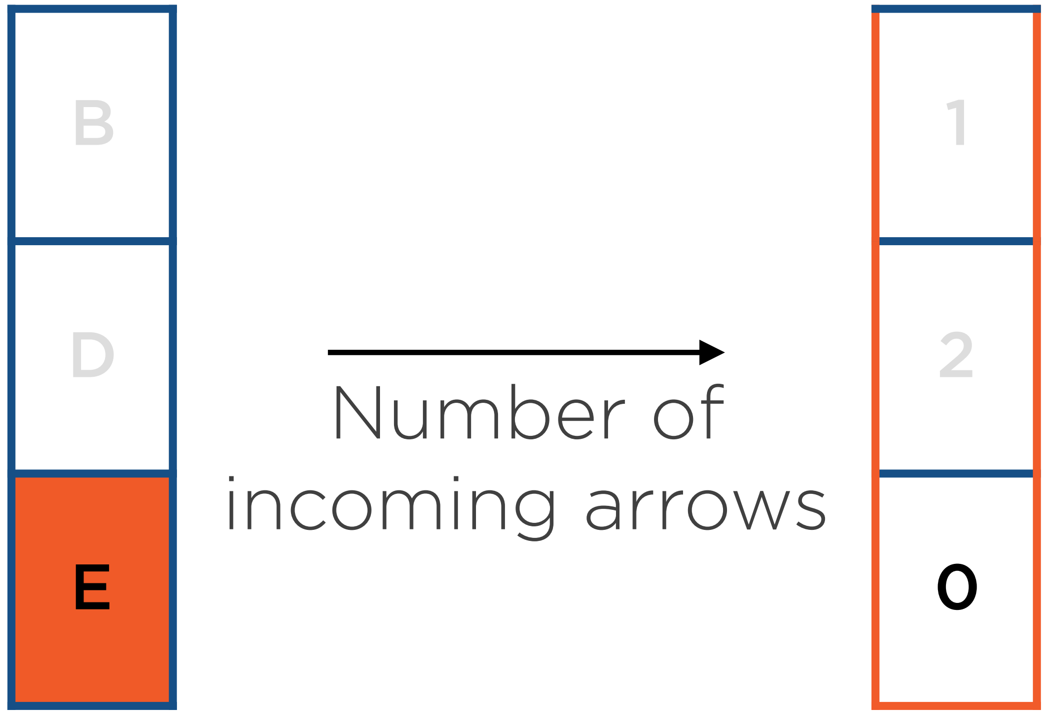


Result



Rinse-and-Repeat

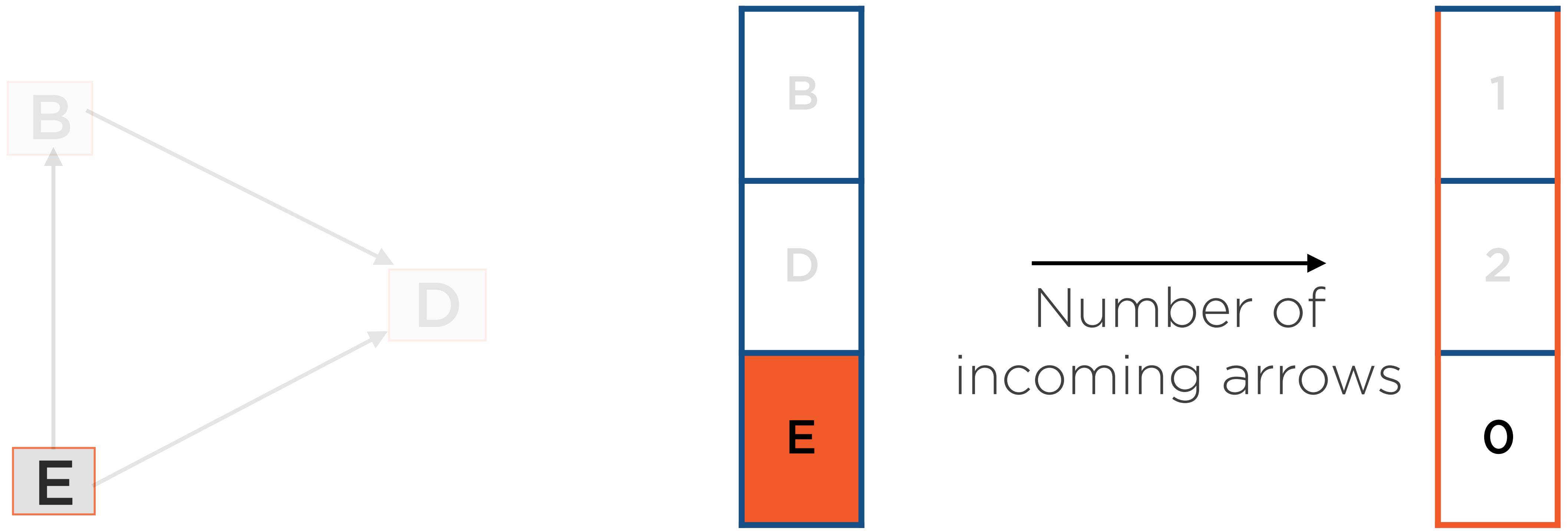




Next, visit any node that has in-degree = 0

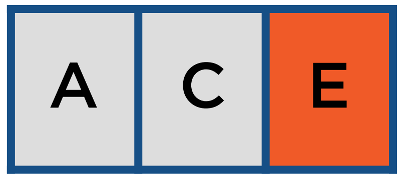
Result | A | C

New Starting Node: E

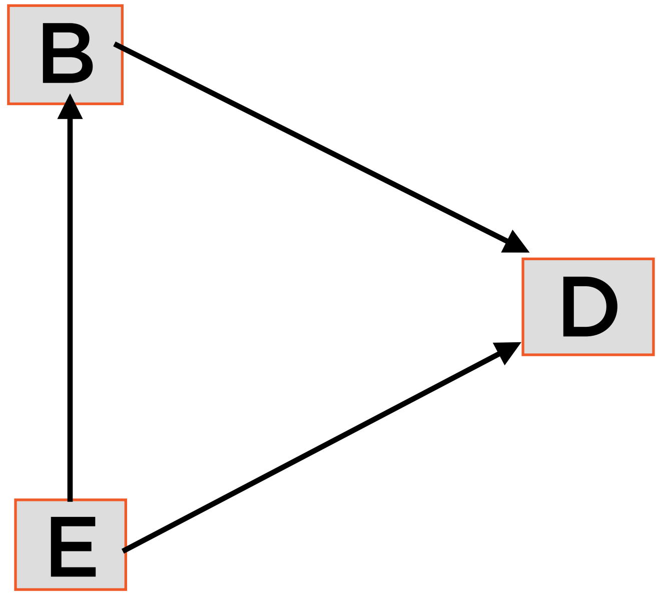


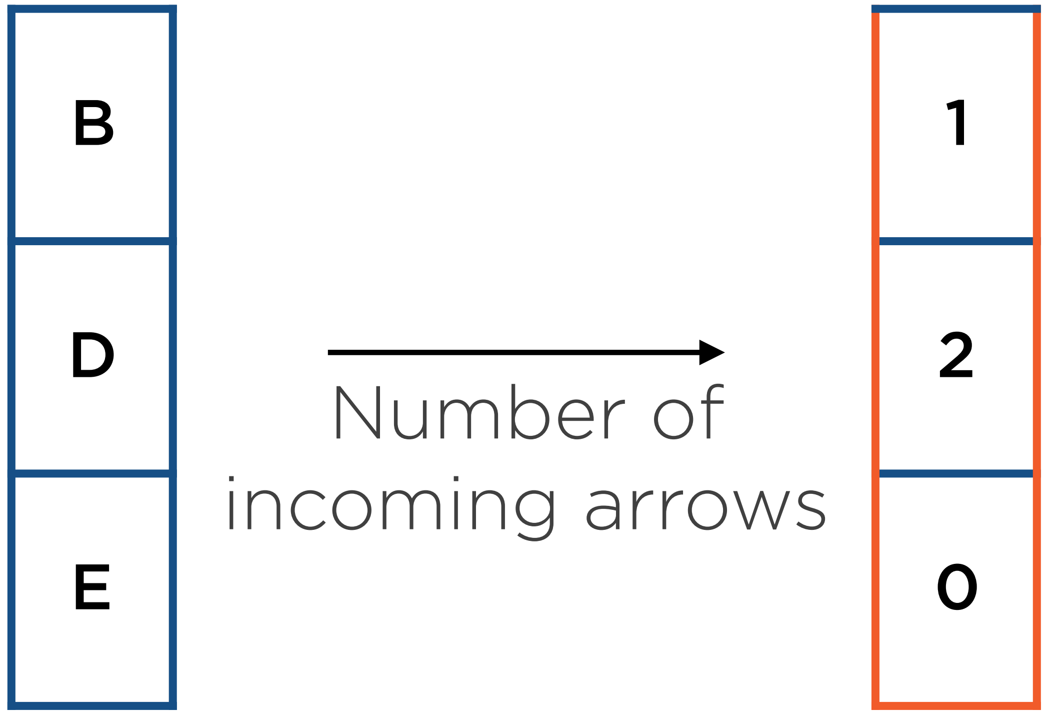
Add E to the result of our topological sort

Result



New Starting Node: E



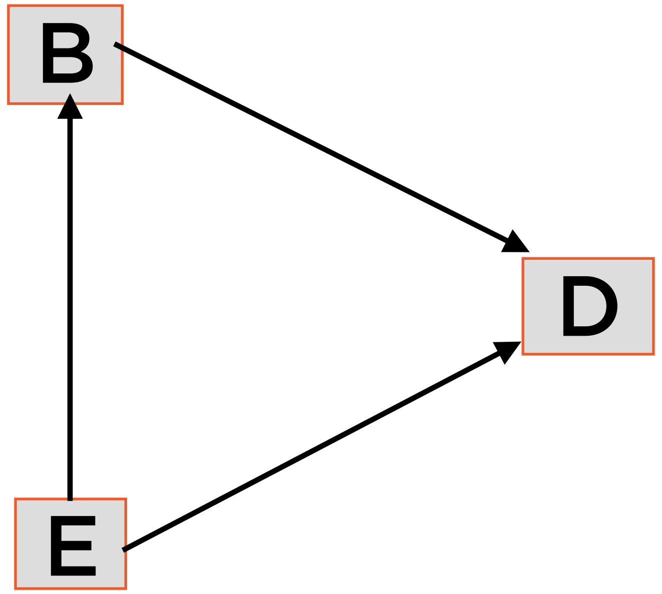


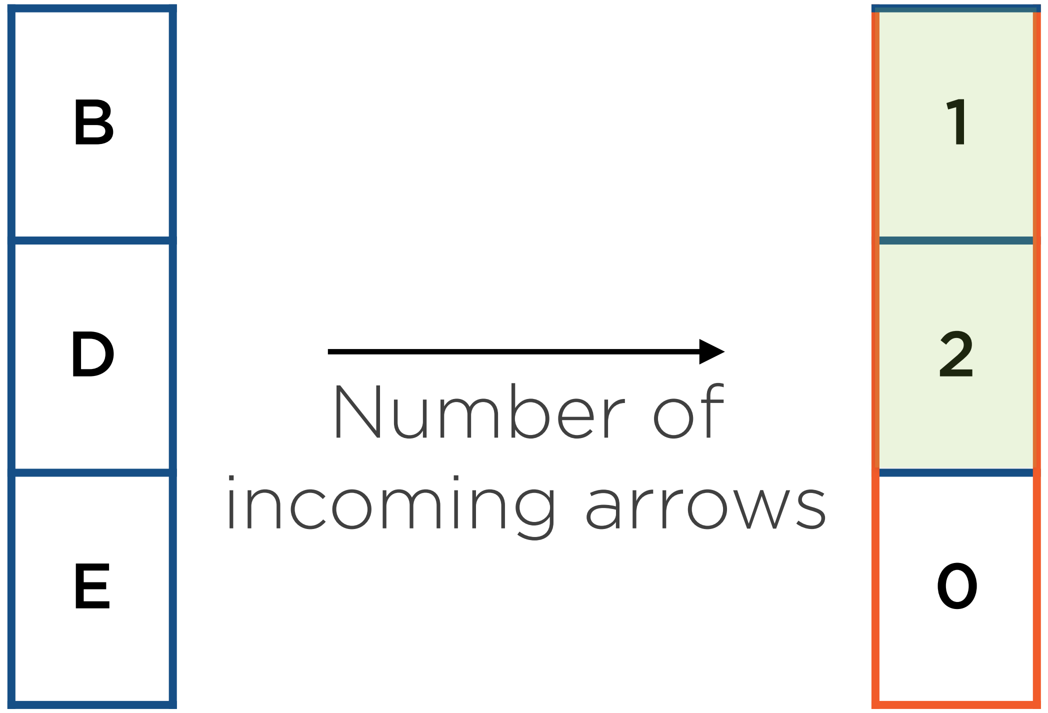
Decrement the in-degree of all nodes that “come after” E

Result

A	C	E

Before Removing E



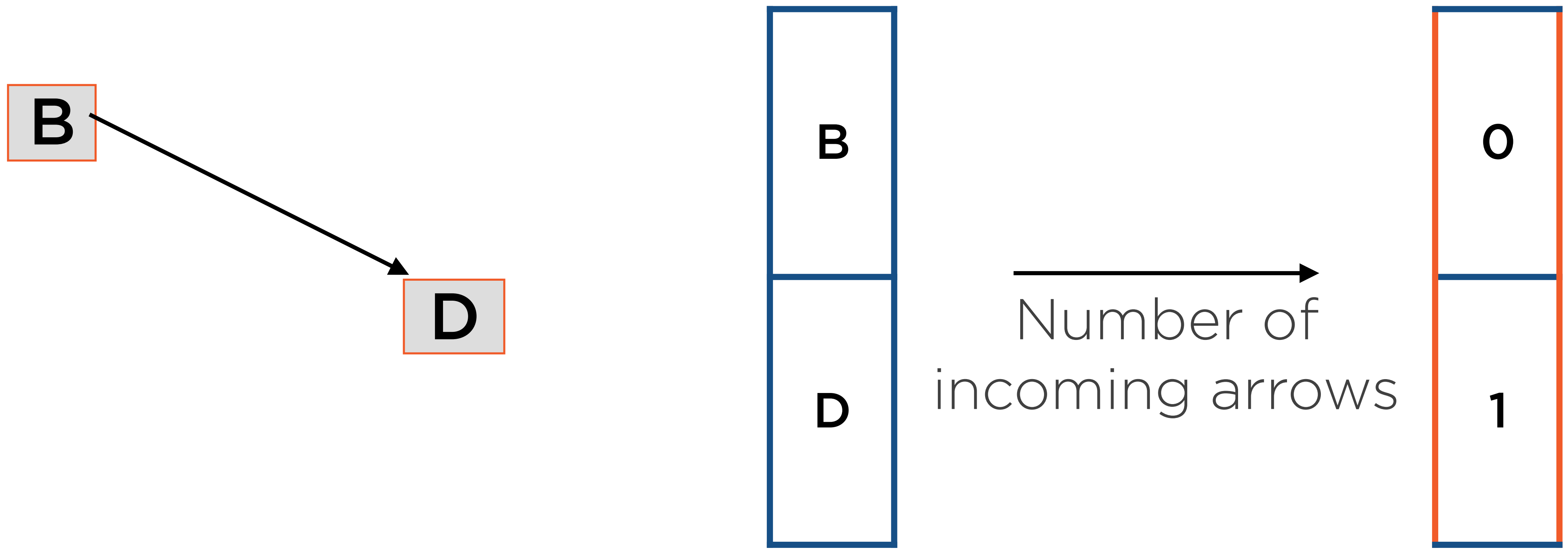


Decrement the in-degree of all nodes that "come after" E

Result

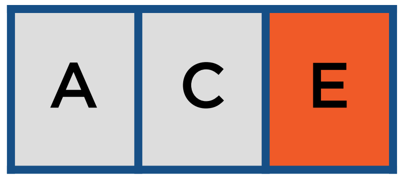
A	C	E

After Removing E

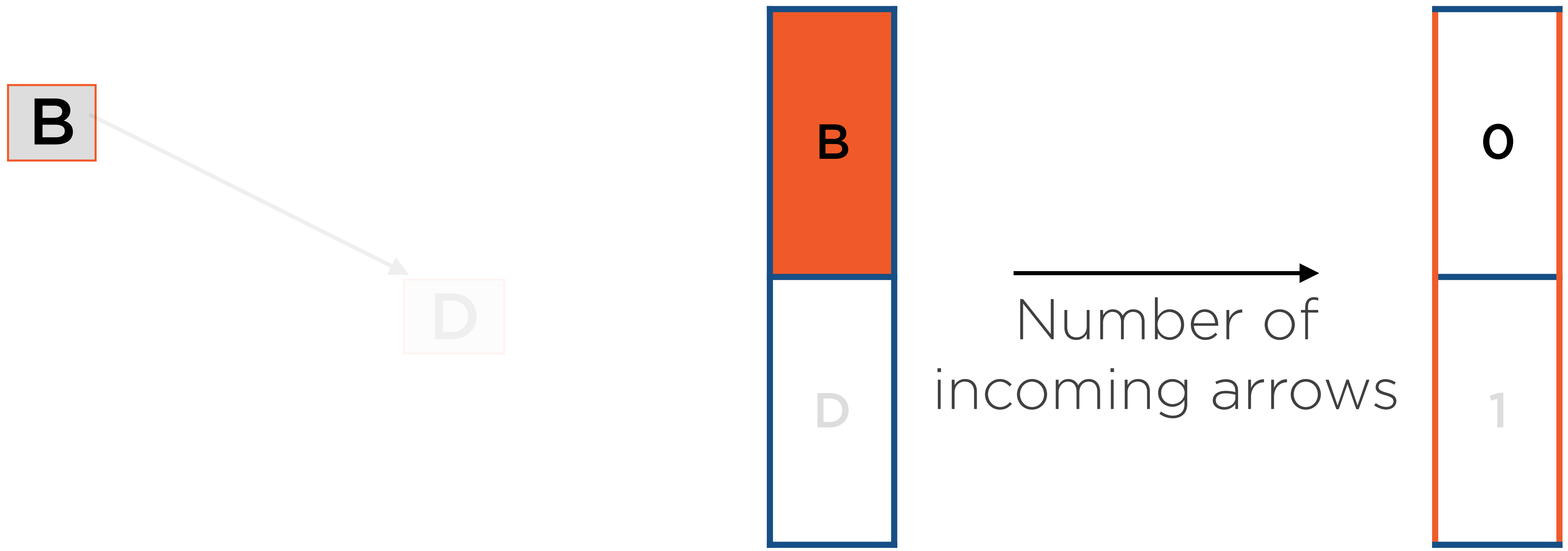


Decrement the in-degree of all nodes that “come after” E

Result



Rinse-and-Repeat

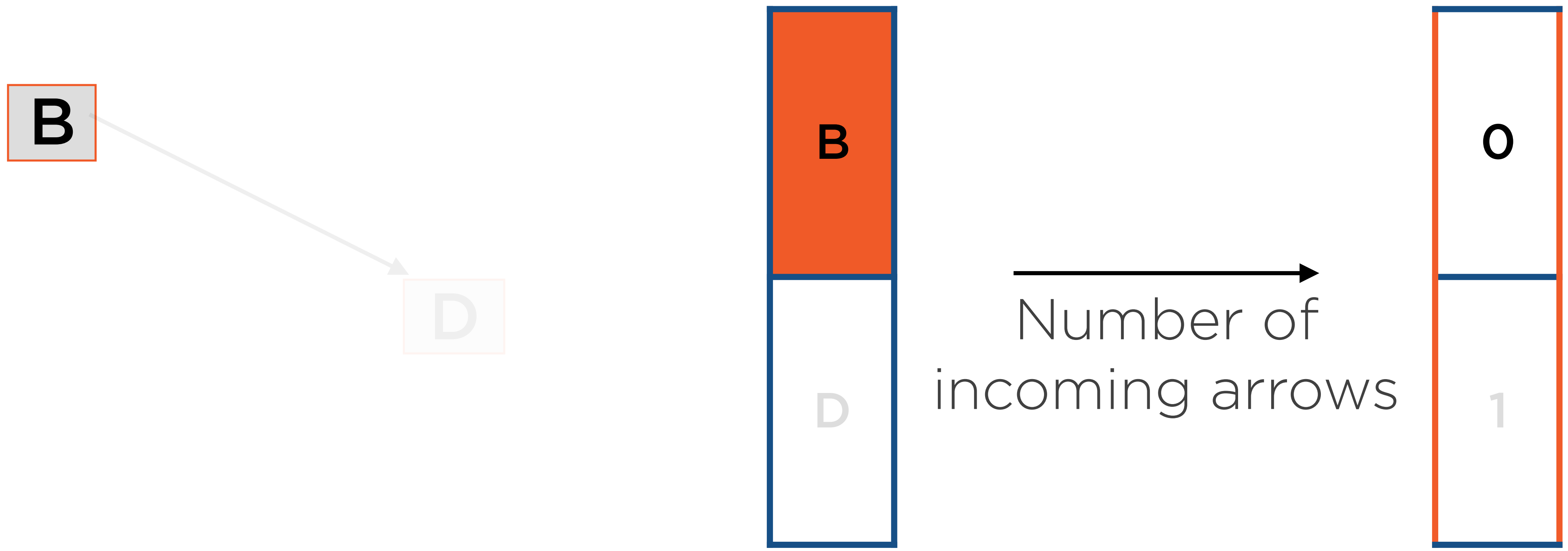


Next, visit any node that has in-degree = 0

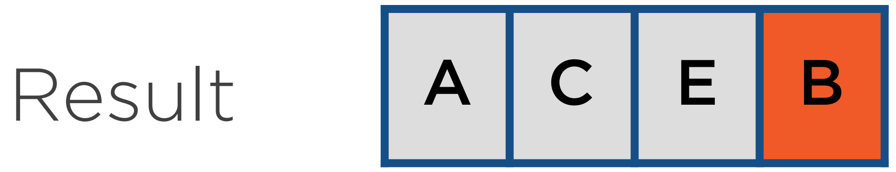
Result

A	C	E

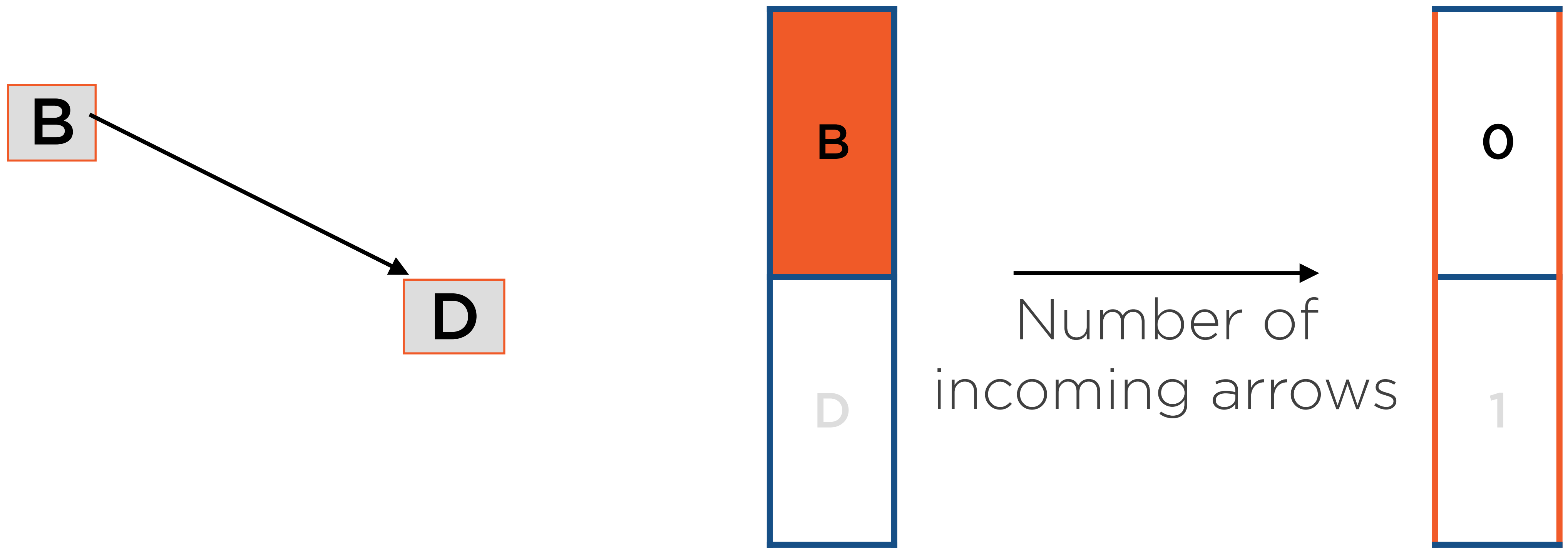
New Starting Node: B



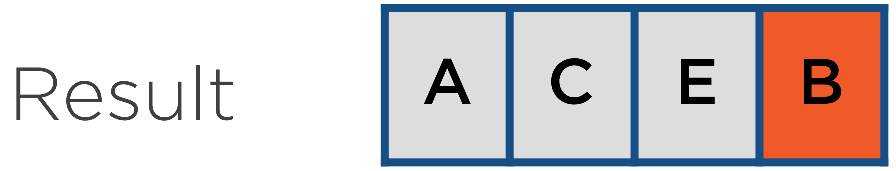
Add this node to the result our topological sort



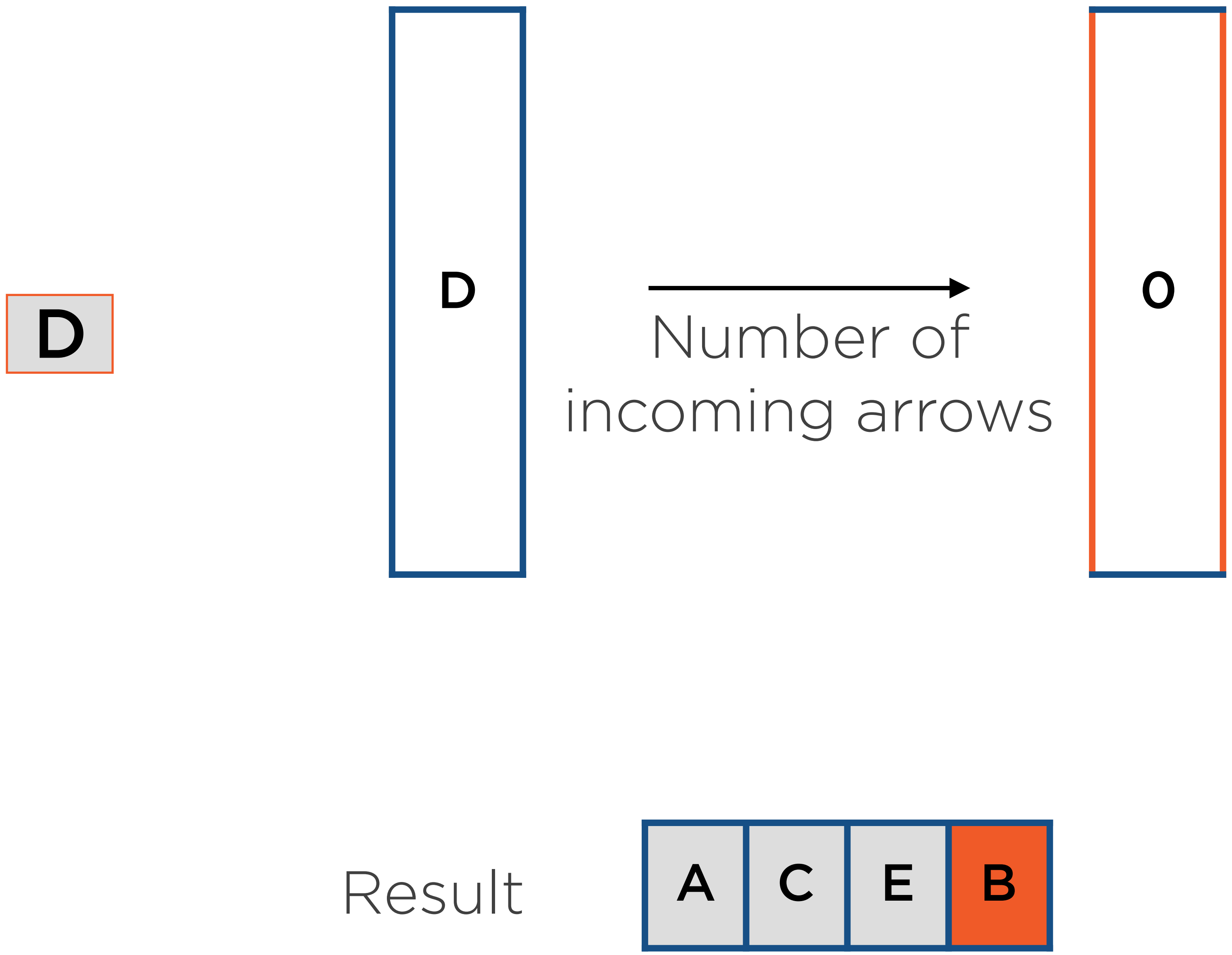
Before Removing B



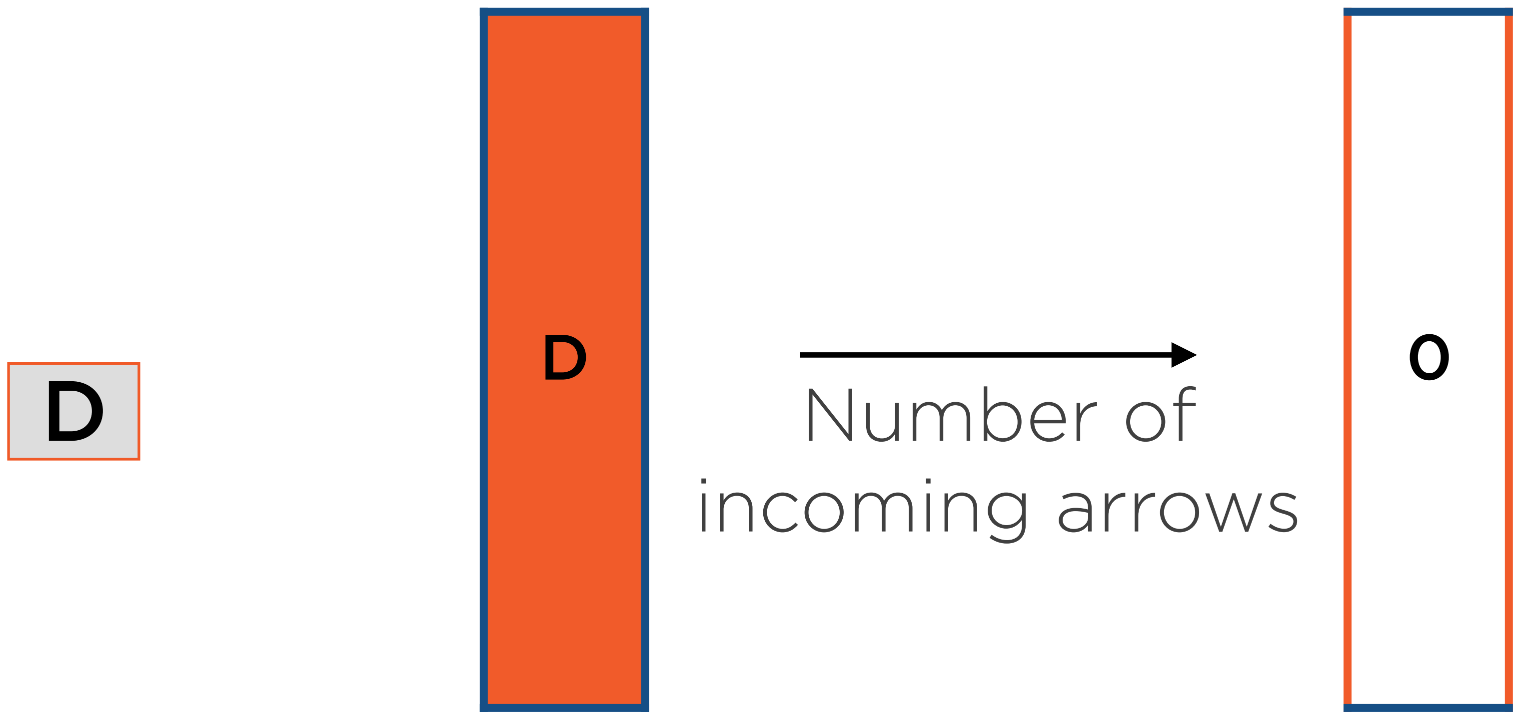
Remove B from our original graph



After Removing B



Rinse-and-Repeat

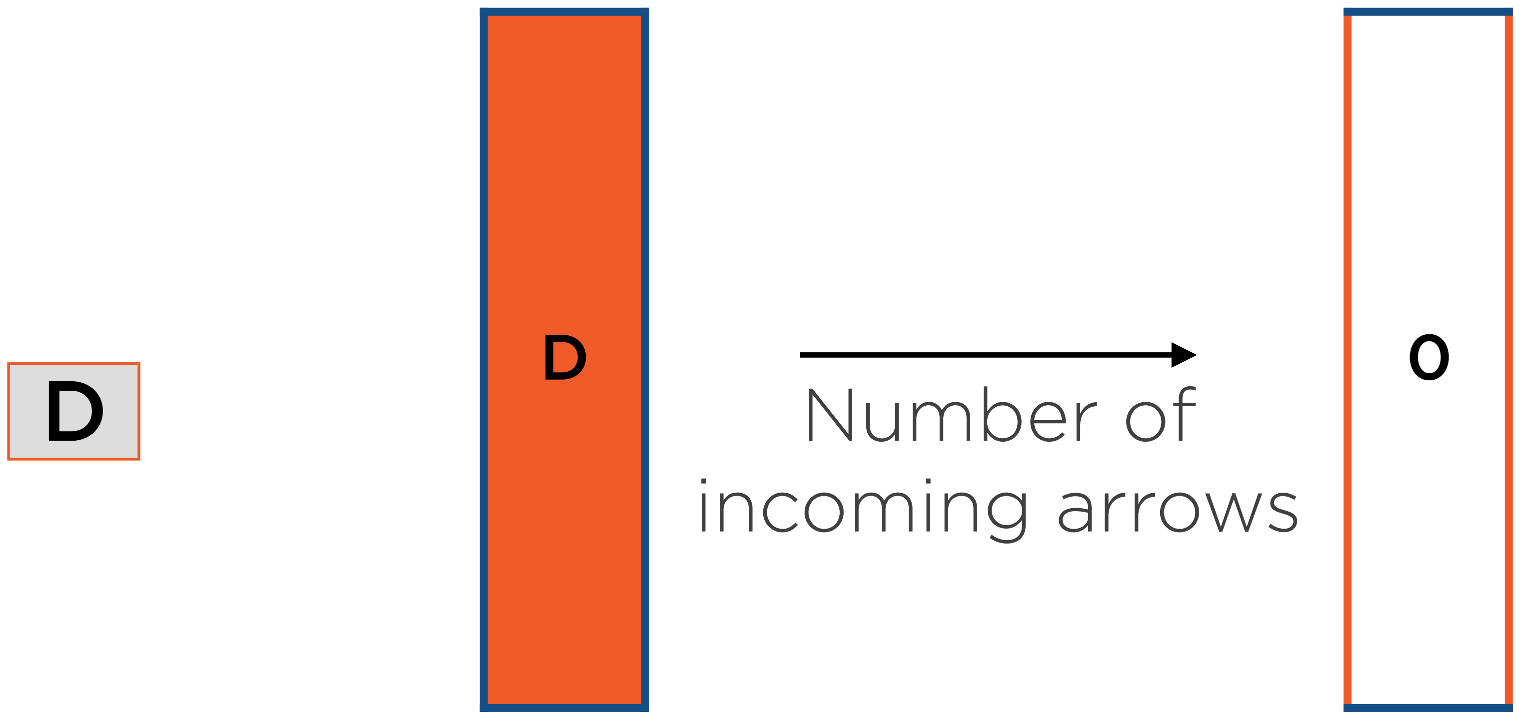


Find node with in-degree = 0

Result

A	C	E	B

Rinse-and-Repeat



Only one node left, add to result

Result	A	C	E	B	D

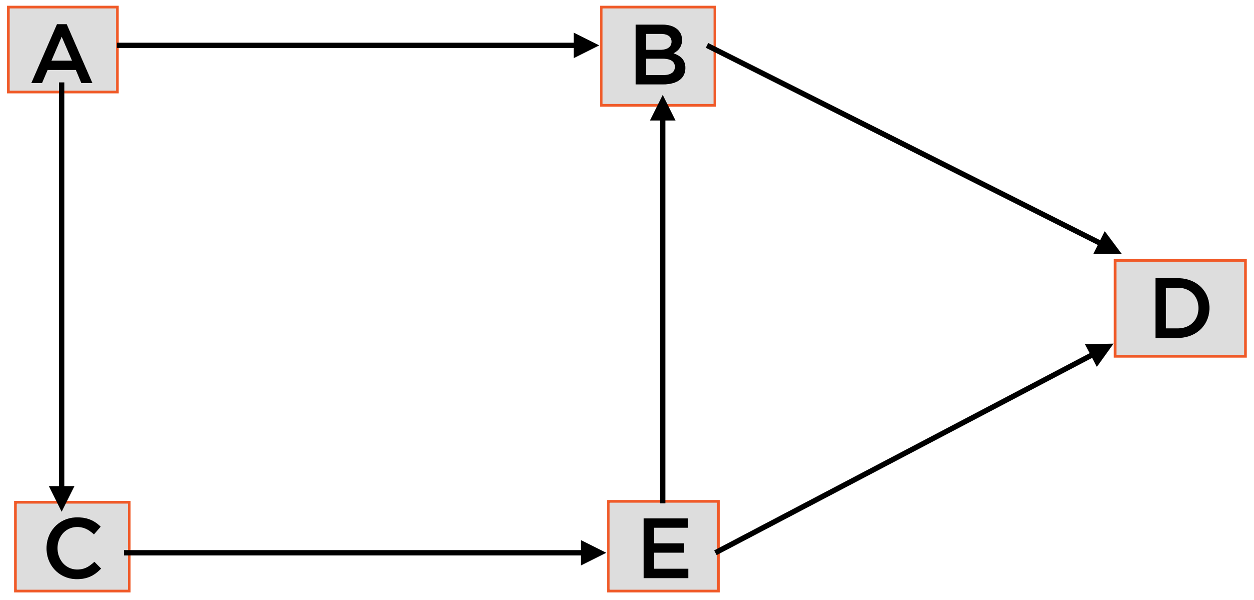
Rinse-and-Repeat

No nodes left to process, the algorithm terminates

Result	A	C	E	B	D

All precedence relationships satisfied

Topological Sort



A -> C -> E -> B -> D

A	C	E	B	D

Here, only 1 acceptable ordering of vertices

A Topological Sort is any ordering of all the DAG's vertices that satisfies all precedence relationships

Topological Sort

O(V+E)

Each edge visited exactly once

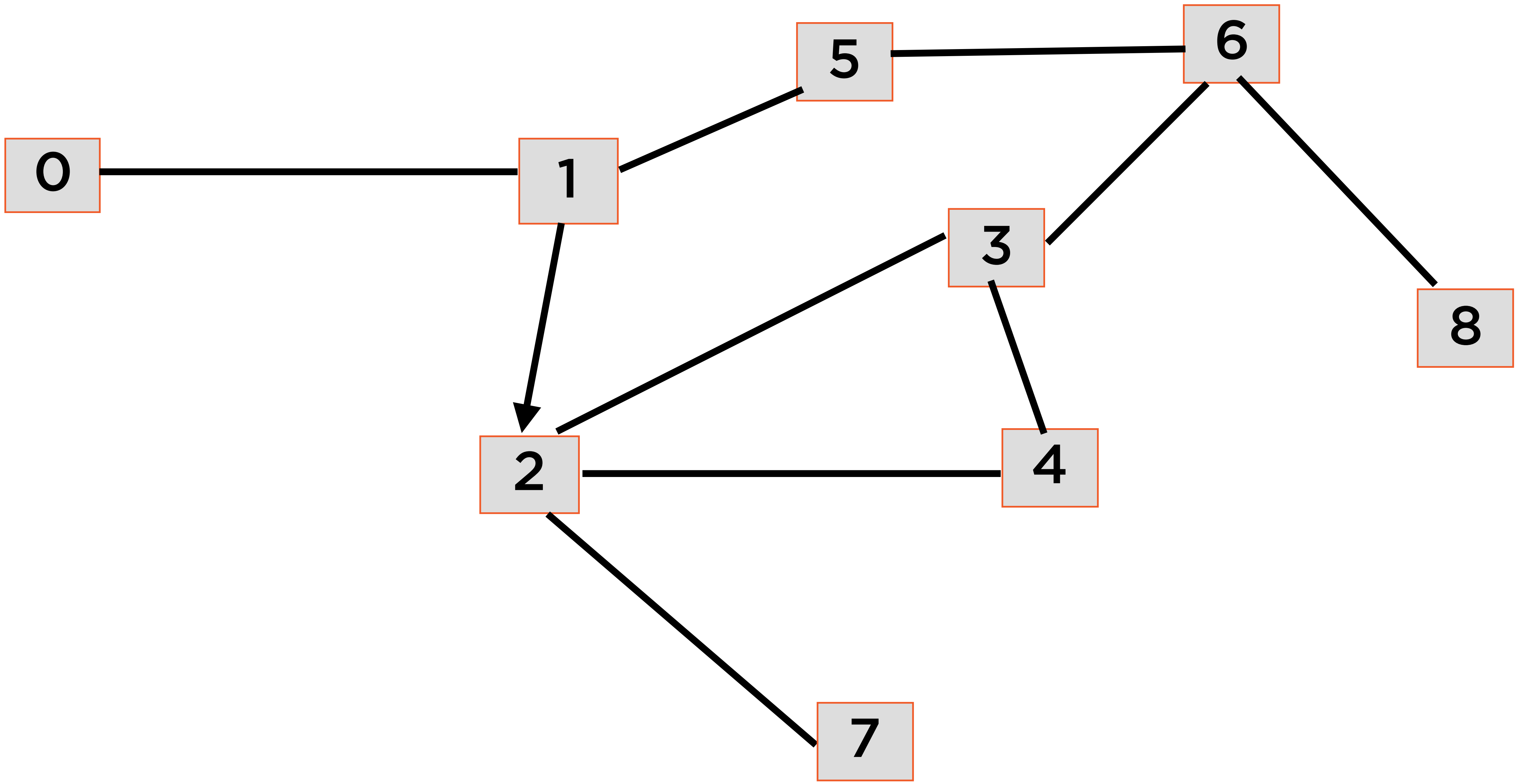
Each vertex visited exactly once

Multiple solutions possible

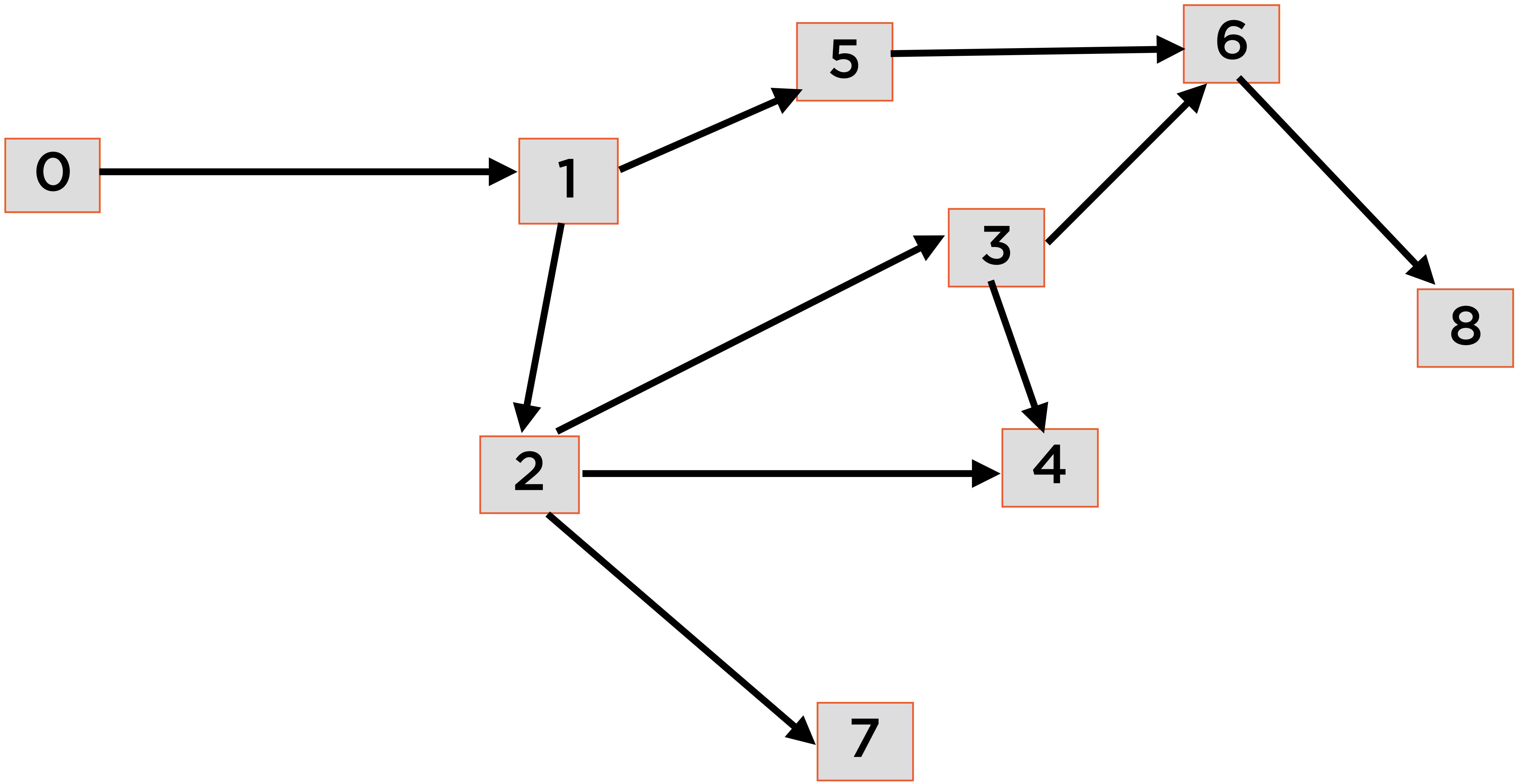
Demo

Implement topological sort in a directed-acyclic graph

A Sample Undirected Graph



A Sample Directed Graph



Summary

Directed Acyclic Graphs (DAGs) are extremely versatile constructs

Applications of DAGs include building neural network models

DAGs specify precedence relationships between nodes

Any ordering of all nodes that satisfies all relationships is a topological sort

Topological sort can be implemented via a simple iterative algorithm