CS 61BL Lab 16

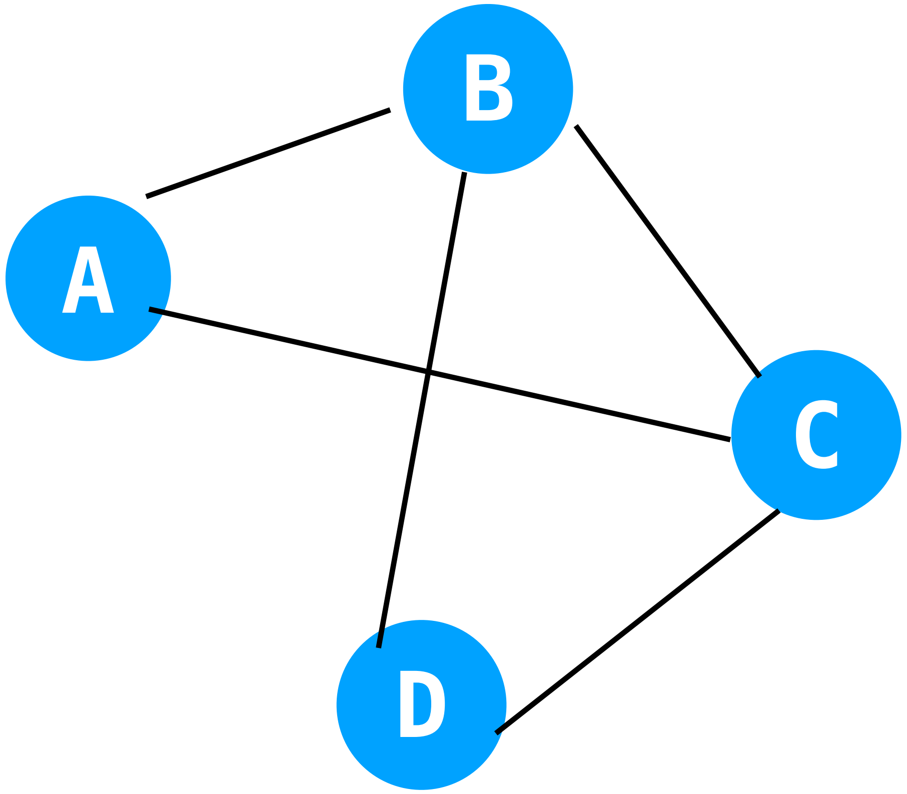
Ryan Purpura

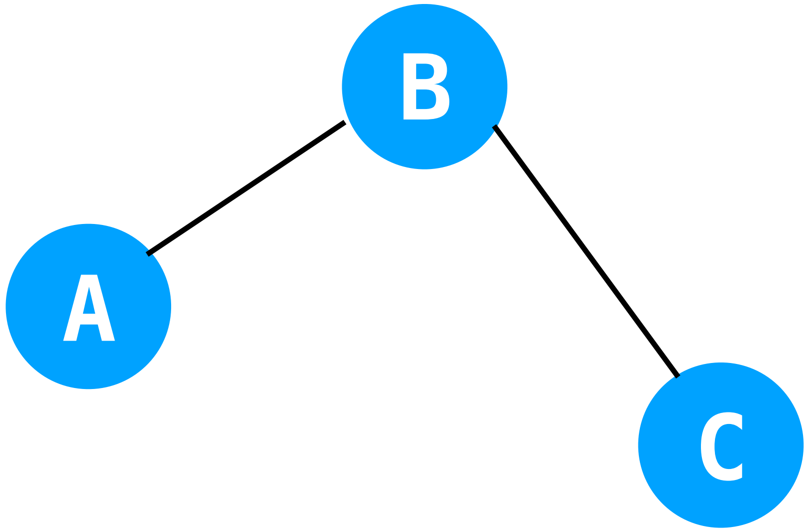
Take a moment to reflect...

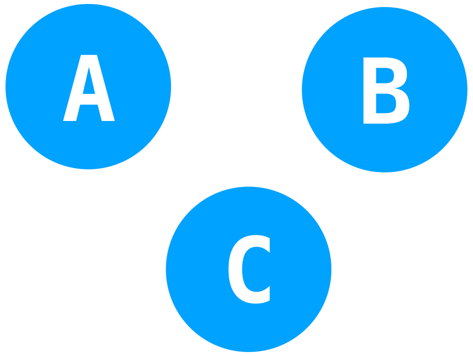
- Midterm 2 is all said and done.
- We've covered an incredible amount of material in 6 short weeks so far.
  - Java, Enigma, Linked Lists, Asymptotics, BSTs, Higher-order functions, B-Trees, LLRBs, Exceptions, Iteration, Hash Tables, Heaps...
- We have two and a half weeks left, don't give up!

Graphs

- Graphs are a set of nodes connected by edges.
- Unlike trees or linked lists, there is no limitation on how nodes can or cannot be connected.

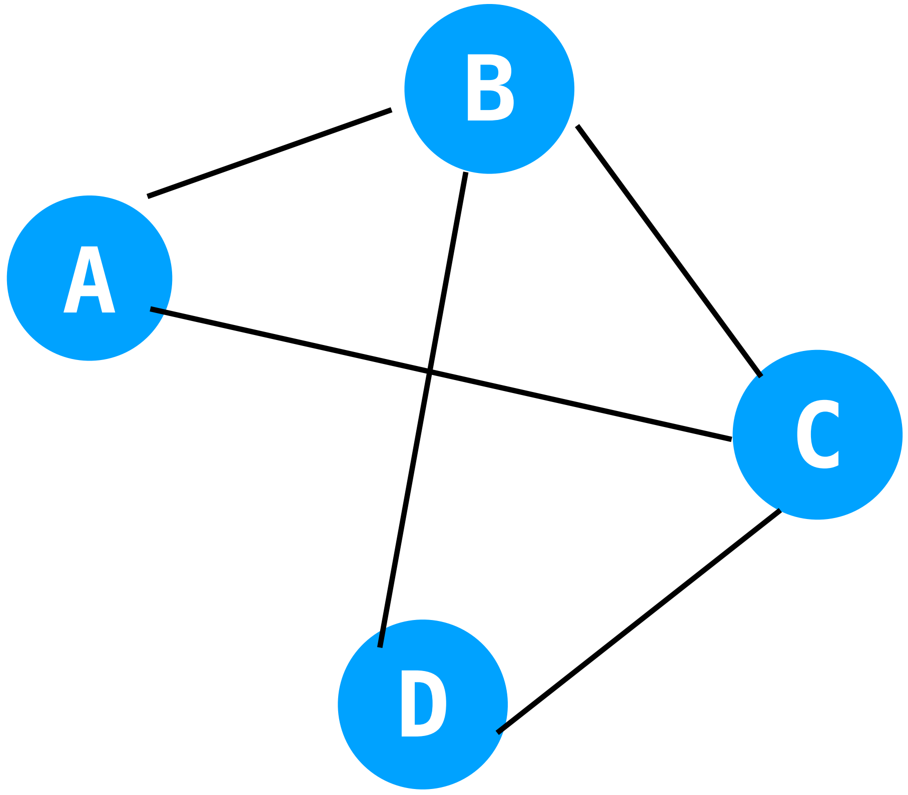






Paths

- A path is a sequence of edges from one vertex to another where no edge or vertex is repeated (except possibly the first and last vertex, as we'll see later)

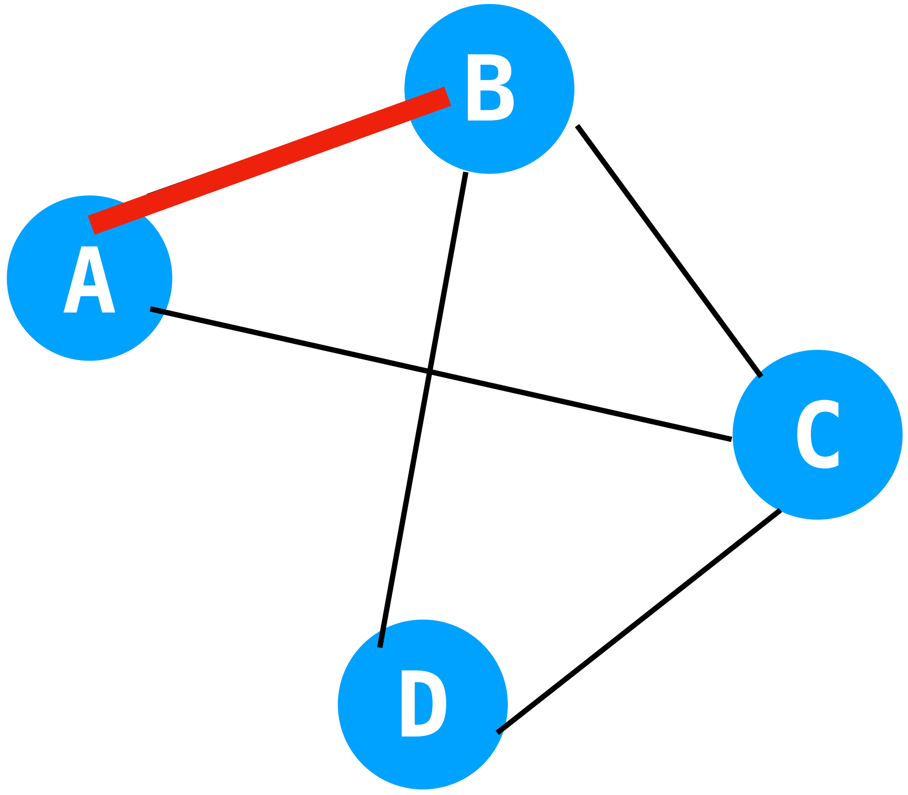


- ***Your definitions may vary based on textbook, instructor, moon phase, astrological sign, etc.)

Paths

- A path is a sequence of edges from one vertex to another where no edge or vertex is repeated (except possibly the first and last vertex, as we'll see later)

(A, B)

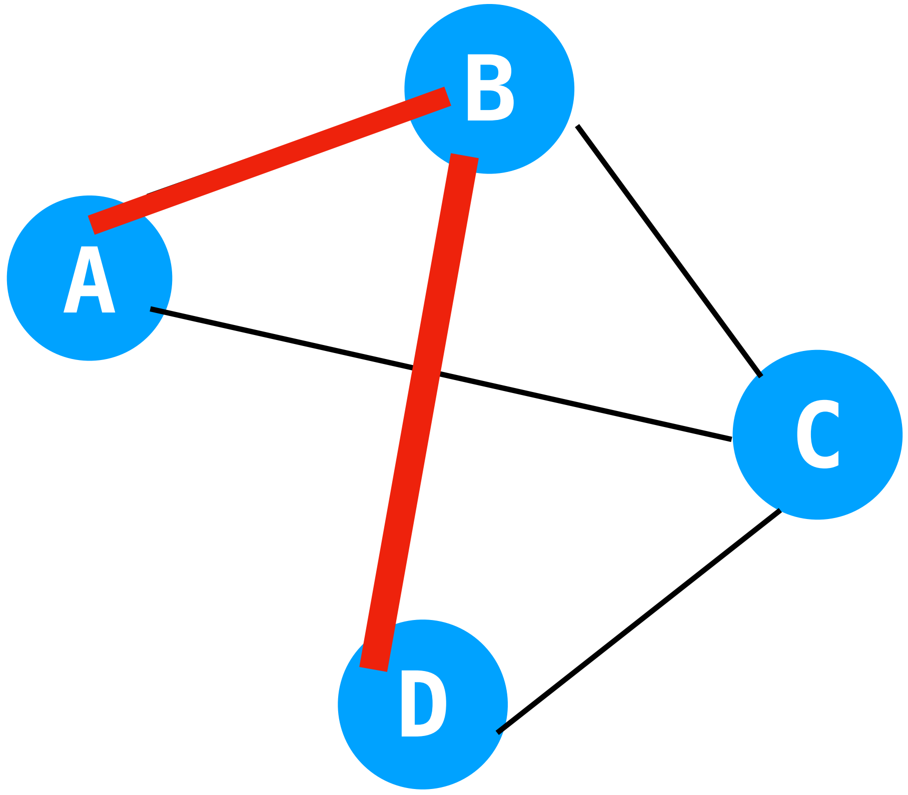


- ***Your definitions may vary based on textbook, instructor, moon phase, astrological sign, etc.)

Paths

- A path is a sequence of edges from one vertex to another where no edge or vertex is repeated (except possibly the first and last vertex, as we'll see later)

(A, B)
(B, D)

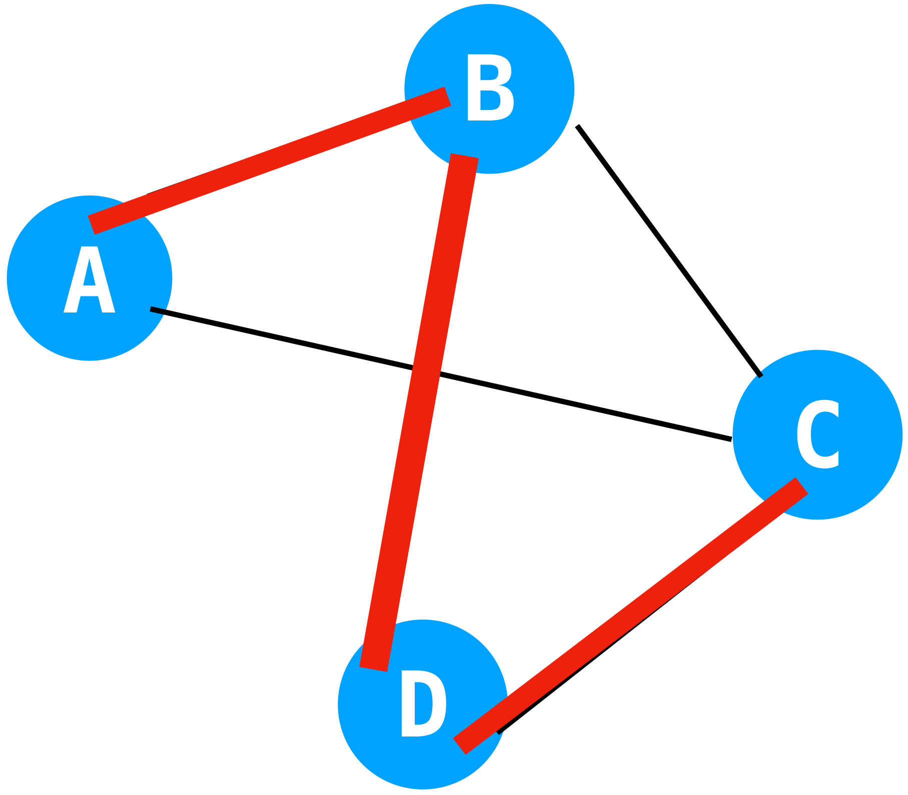


- ***Your definitions may vary based on textbook, instructor, moon phase, astrological sign, etc.)

Paths

- A path is a sequence of edges from one vertex to another where no edge or vertex is repeated (except possibly the first and last vertex, as we'll see later)

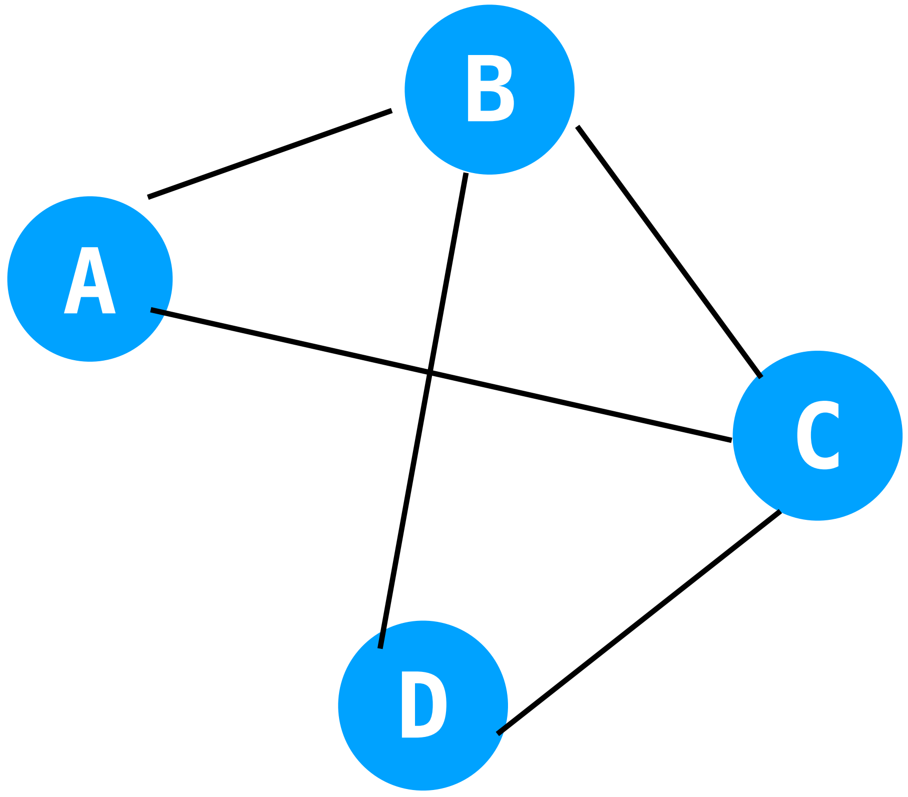
(A, B)
(B, D)
(D, C)



- ***Your definitions may vary based on textbook, instructor, moon phase, astrological sign, etc.)

Cycles

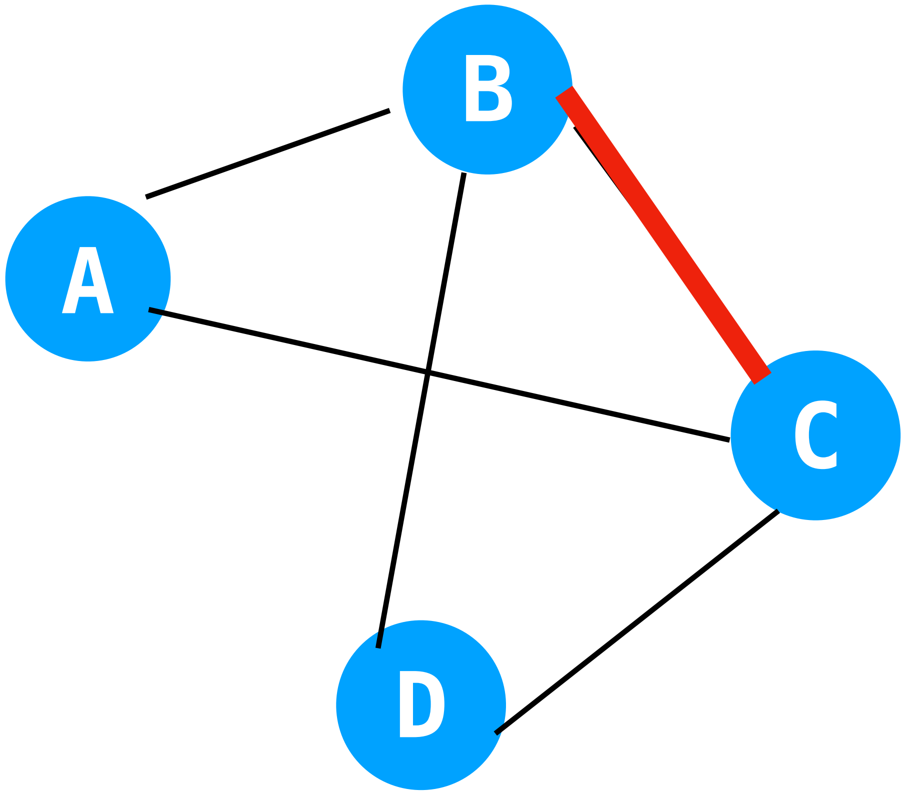
- A cycle is a path that ends at the same vertex where it originally started.



Cycles

- A cycle is a path that ends at the same vertex where it originally started.

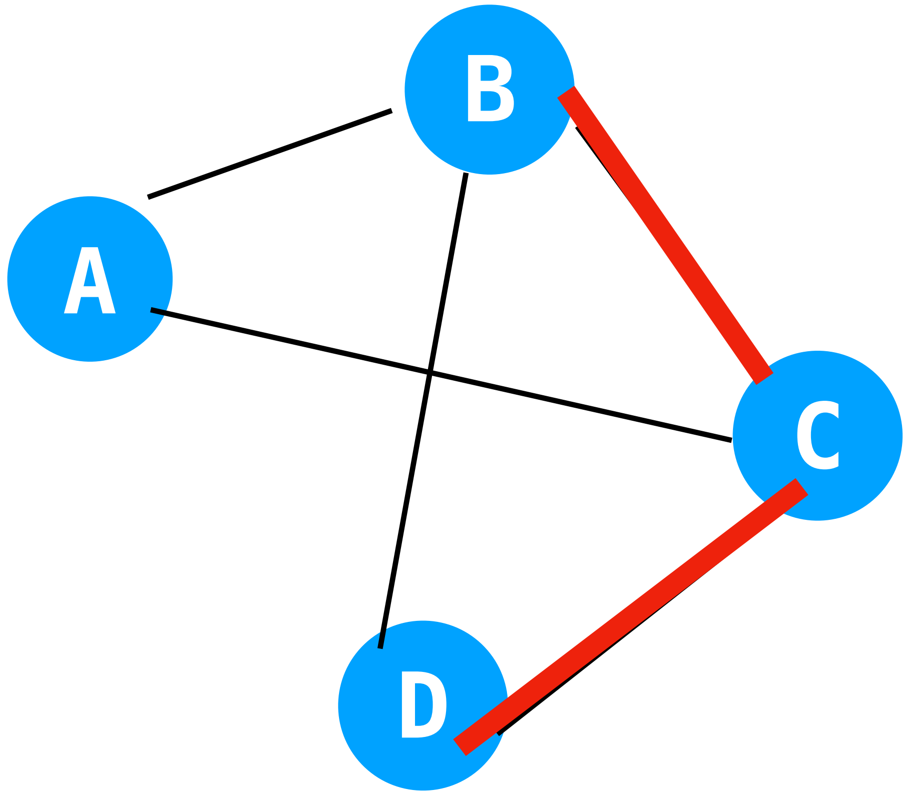
(B, C)



Cycles

- A cycle is a path that ends at the same vertex where it originally started.

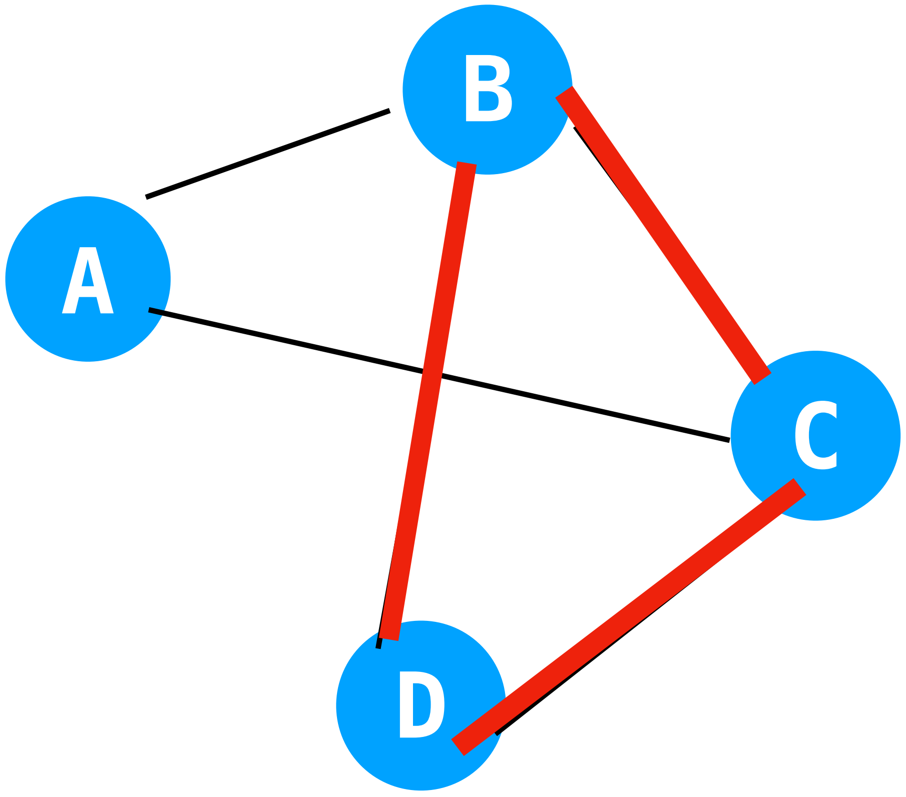
(B, C)
(C, D)



Cycles

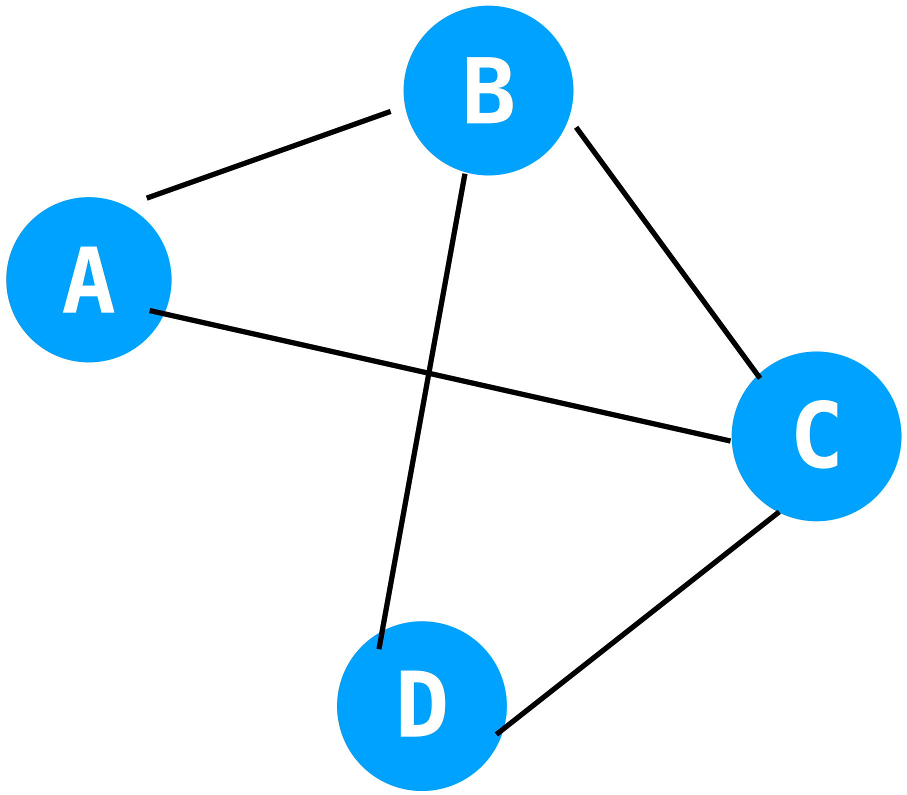
- A cycle is a path that ends at the same vertex where it originally started.

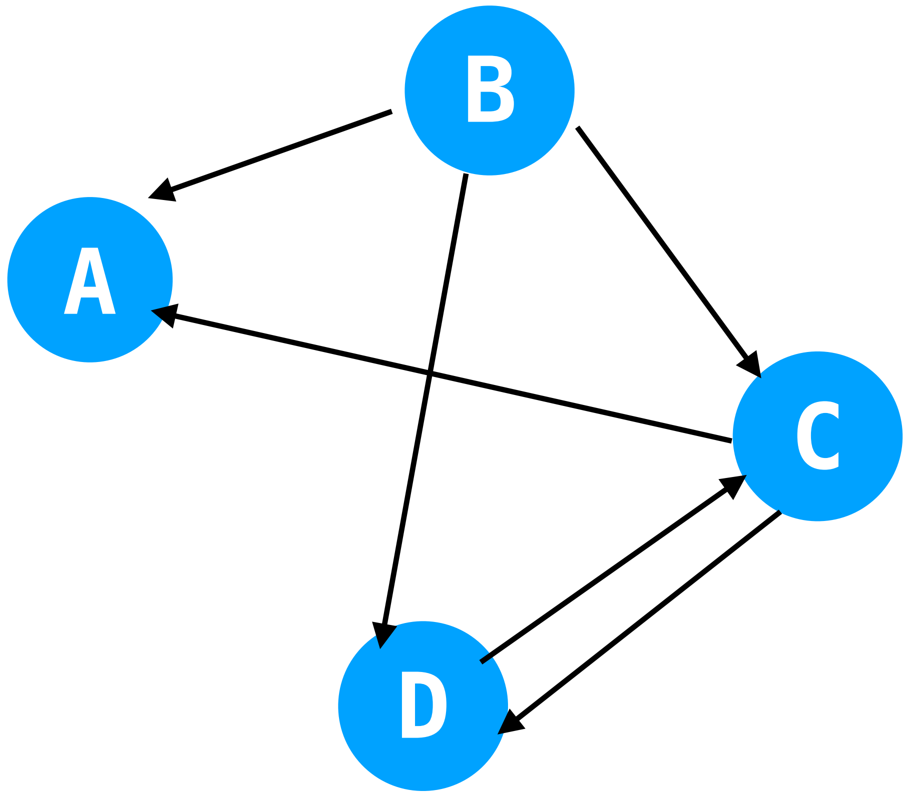
(B, C)
(C, D)
(D, B)



Directed vs. Non-directed

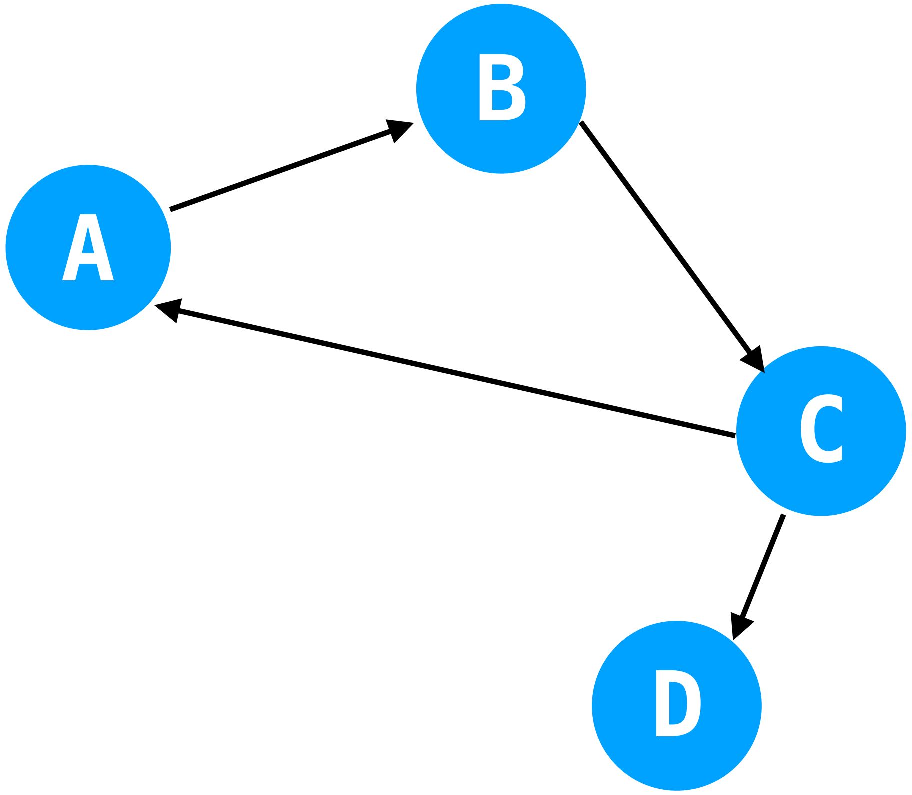
- Directed graphs have one-way edges.





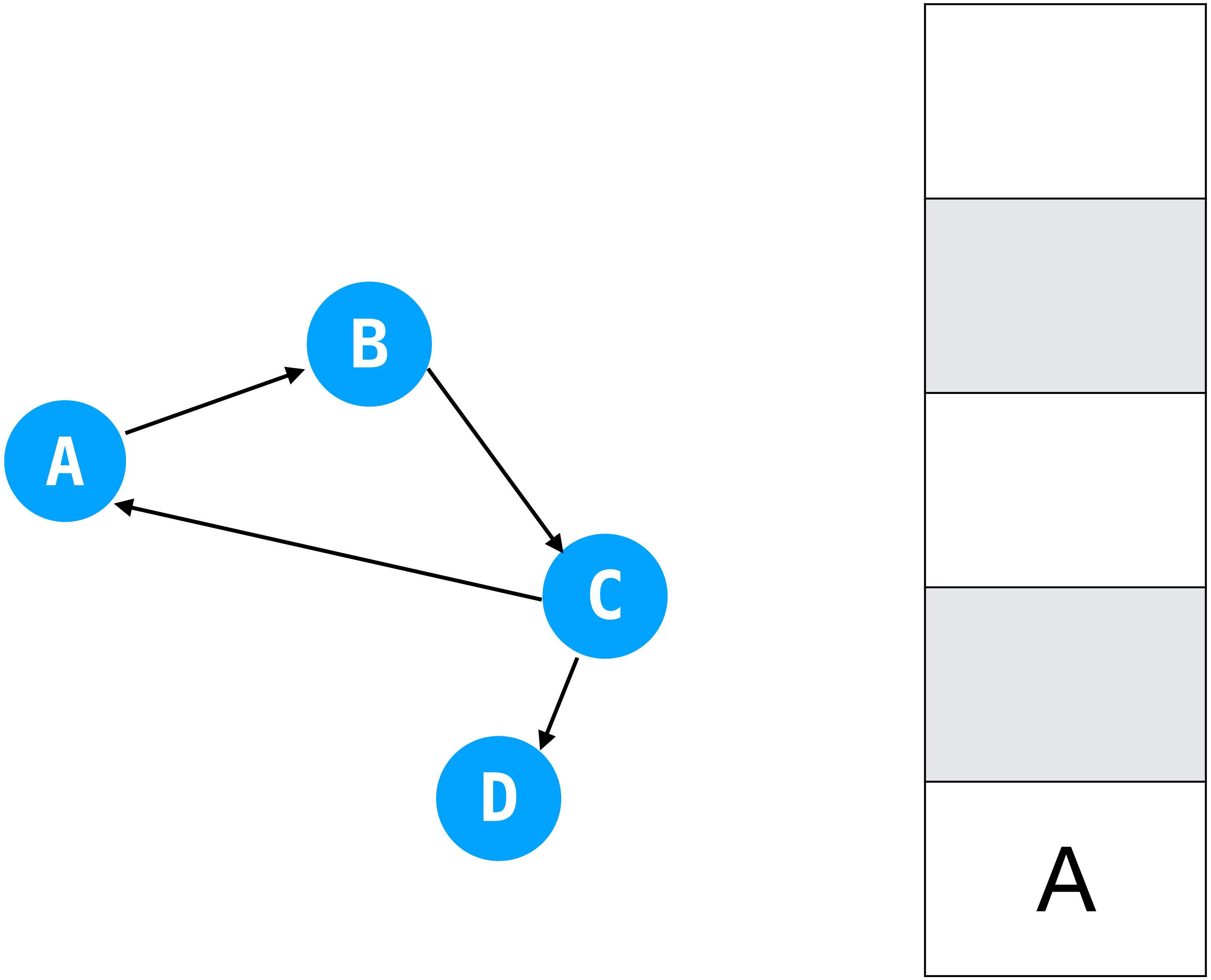
Depth First Search

- Depth-first search is defined just like with trees, with a slight modification.



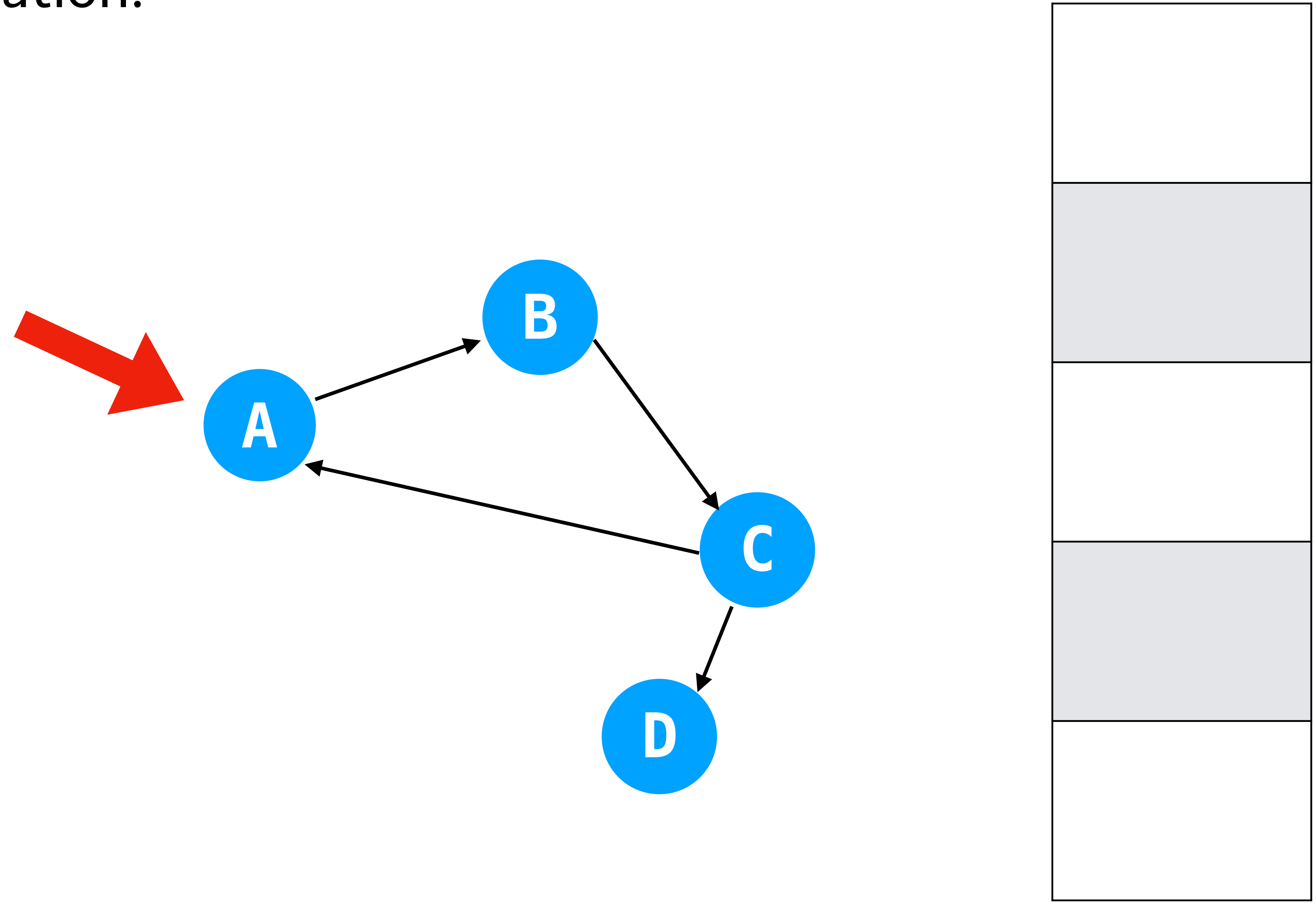
Depth First Search

- Depth-first search is defined just like with trees, with a slight modification.



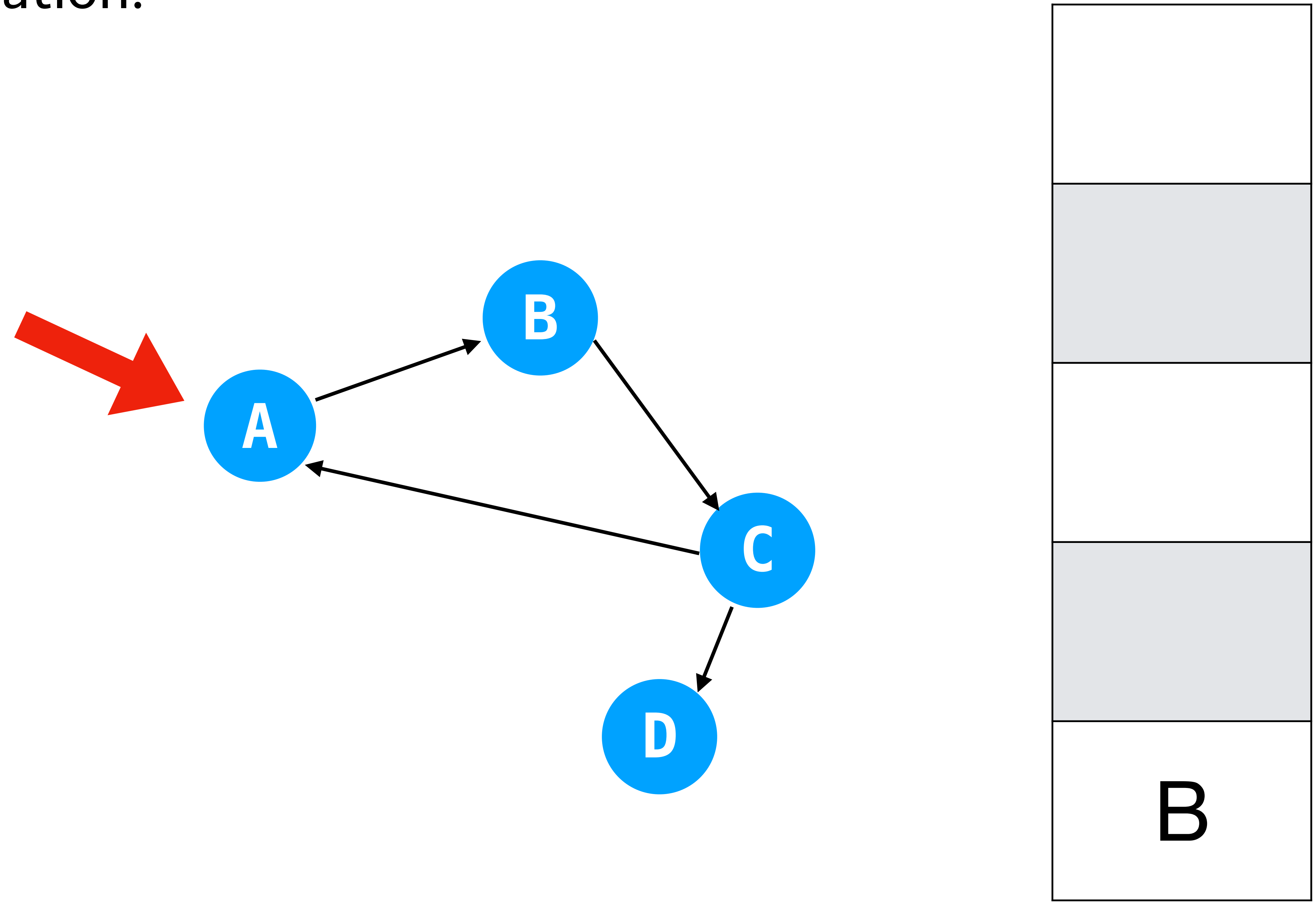
Depth First Search

- Depth-first search is defined just like with trees, with a slight modification.



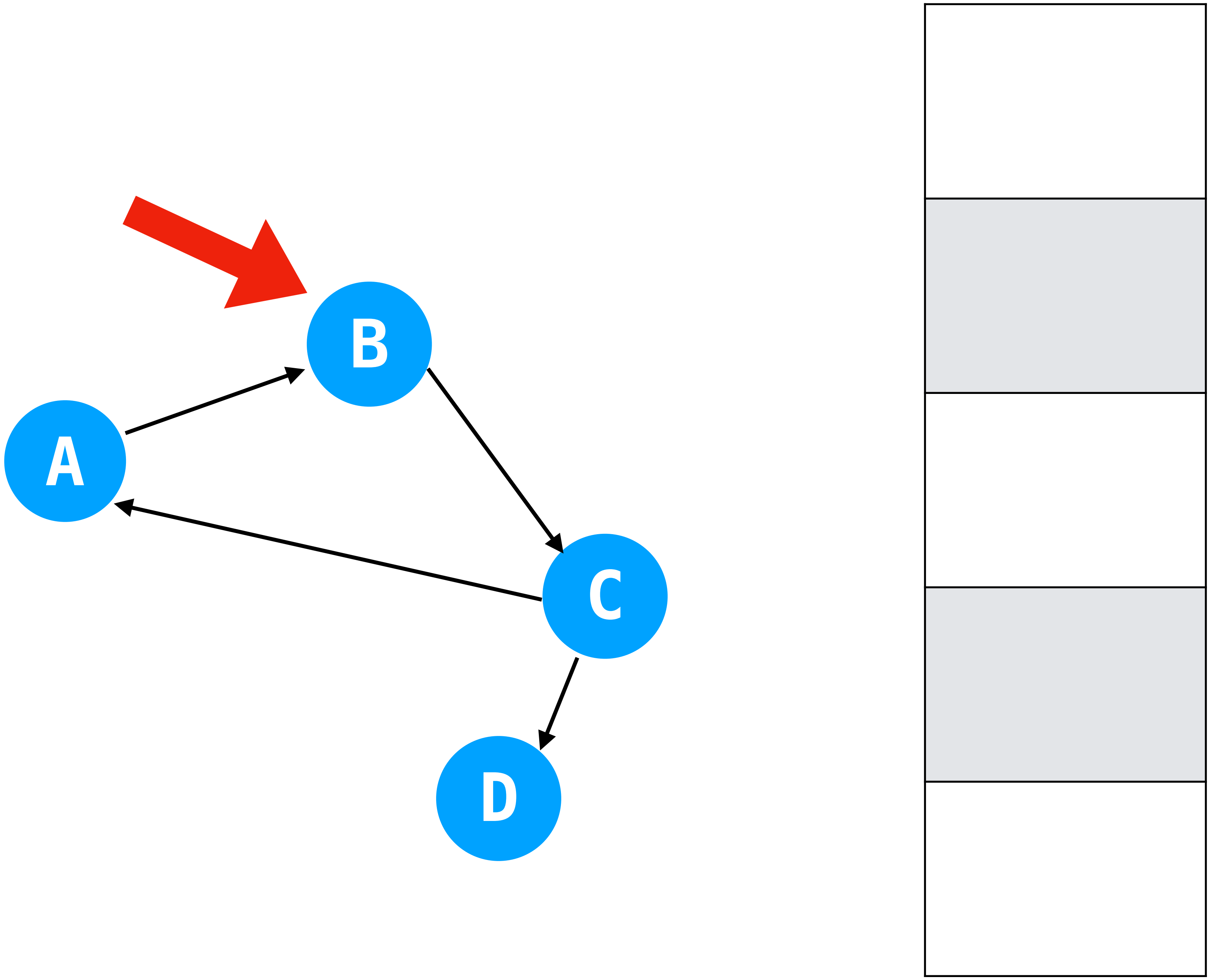
Depth First Search

- Depth-first search is defined just like with trees, with a slight modification.



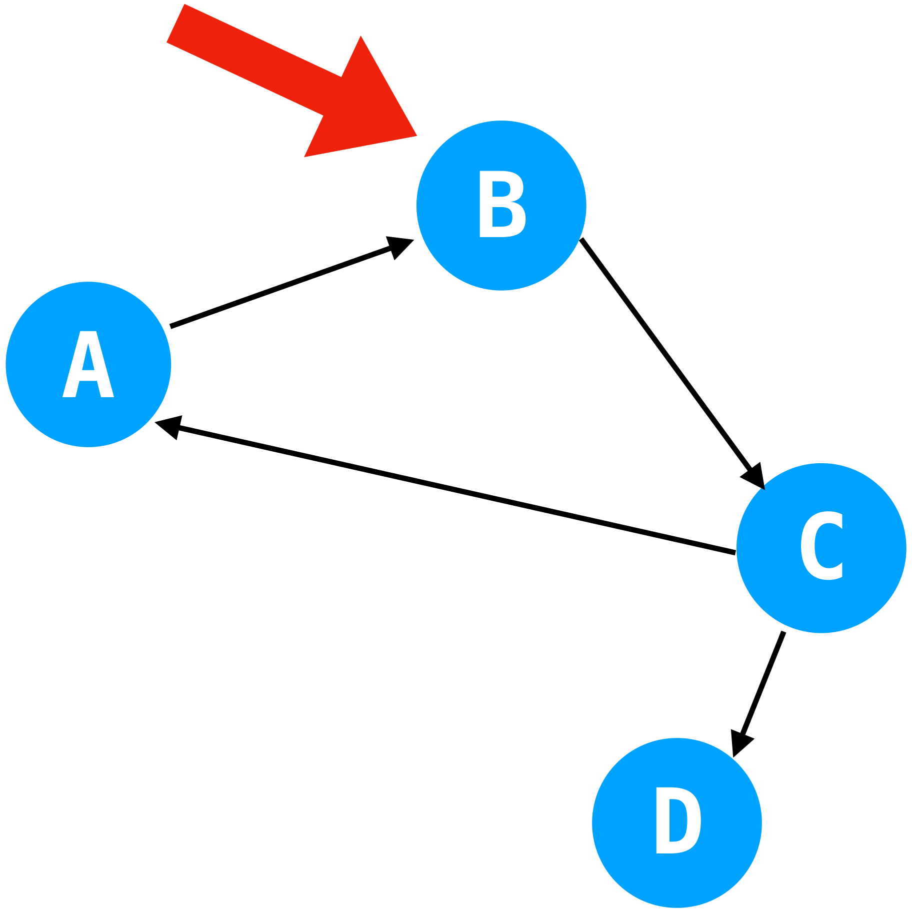
Depth First Search

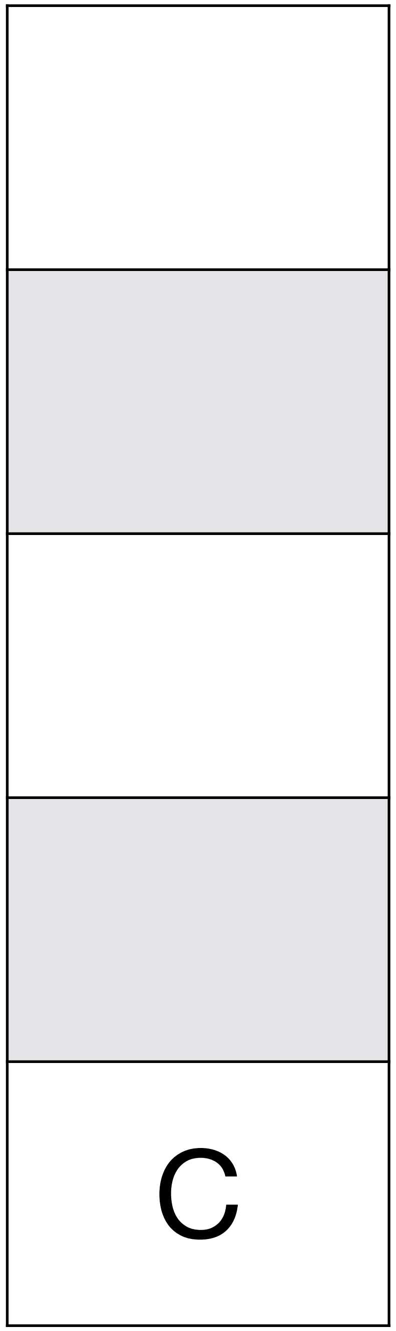
- Depth-first search is defined just like with trees, with a slight modification.



Depth First Search

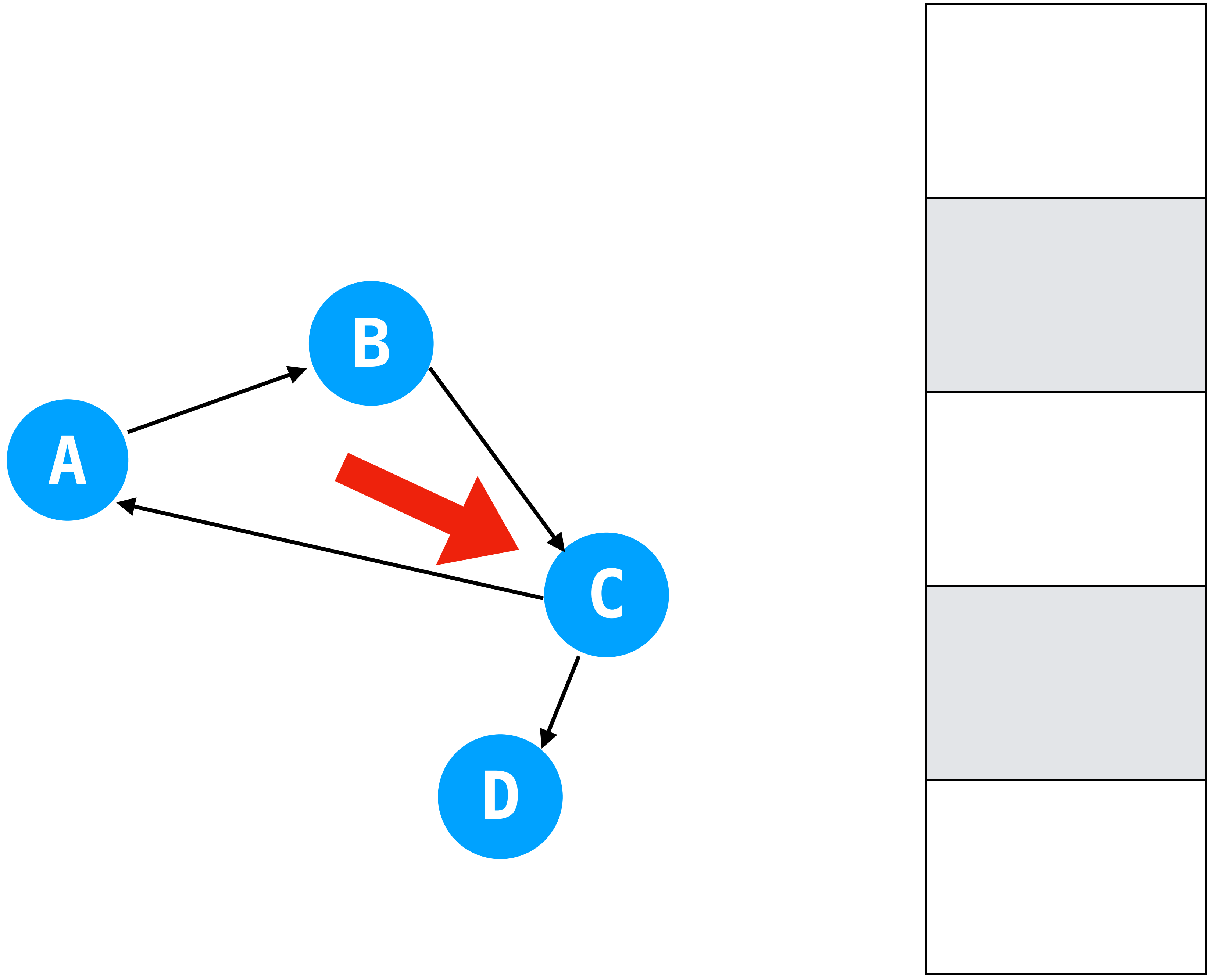
- Depth-first search is defined just like with trees, with a slight modification.





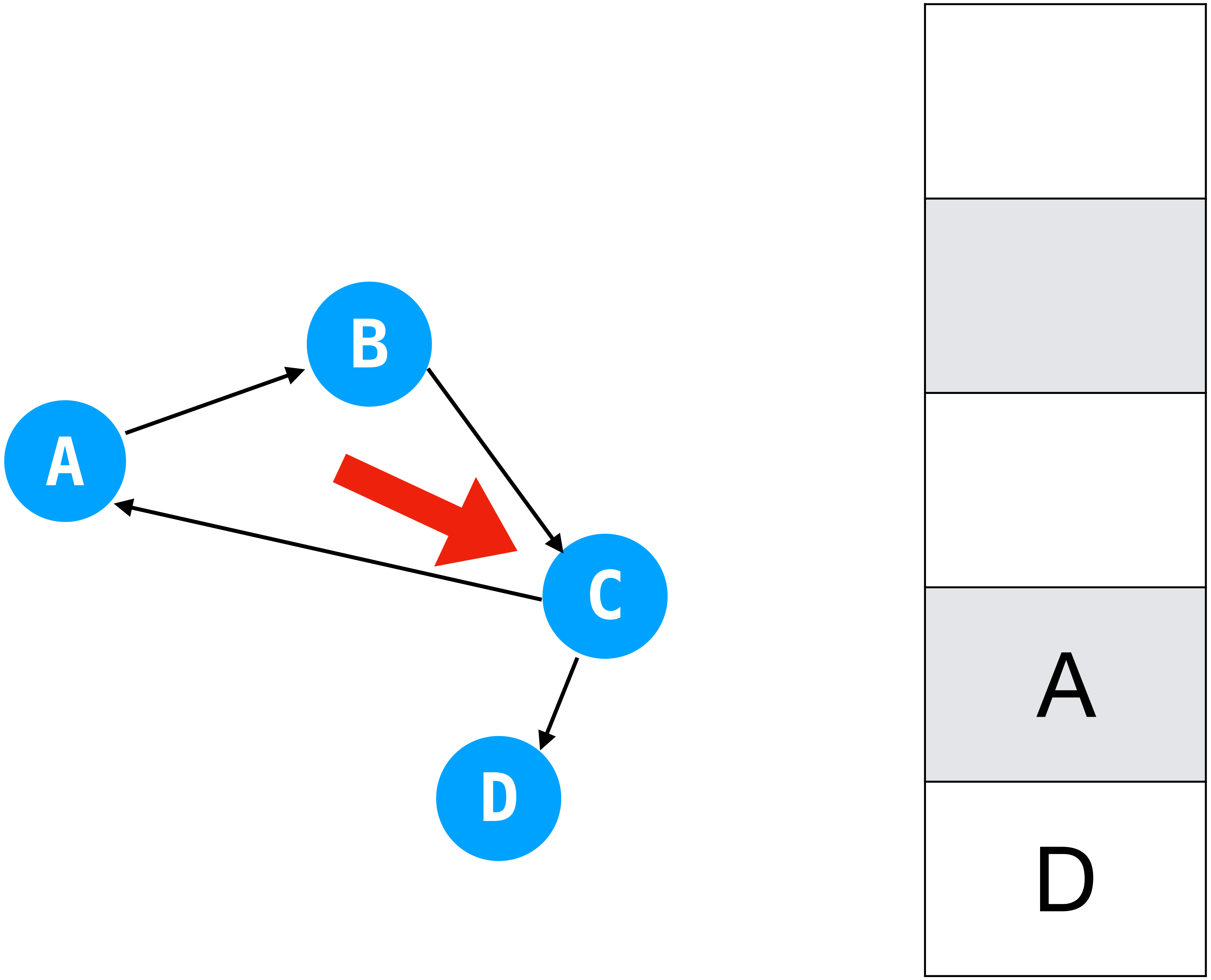
Depth First Search

- Depth-first search is defined just like with trees, with a slight modification.



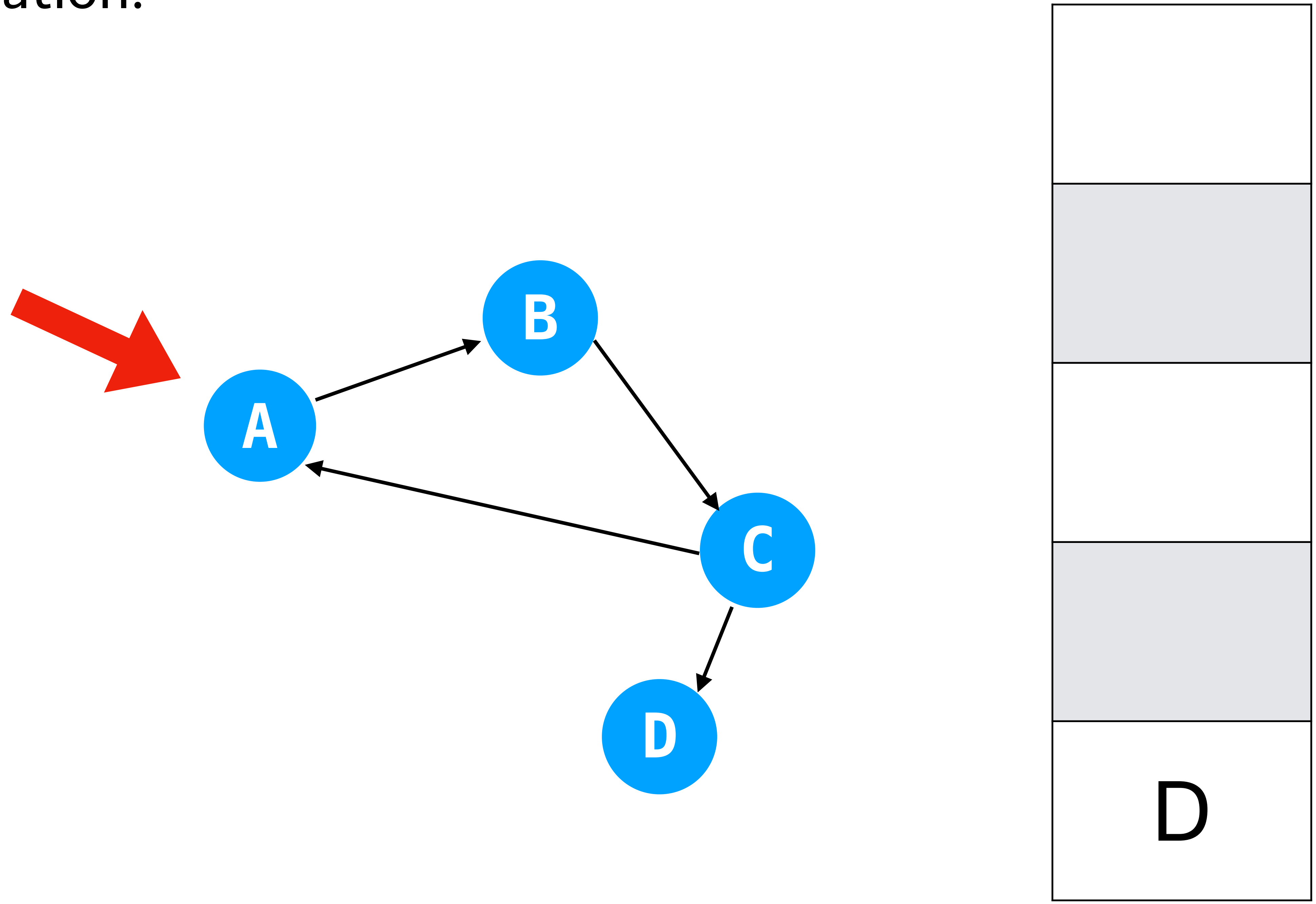
Depth First Search

- Depth-first search is defined just like with trees, with a slight modification.



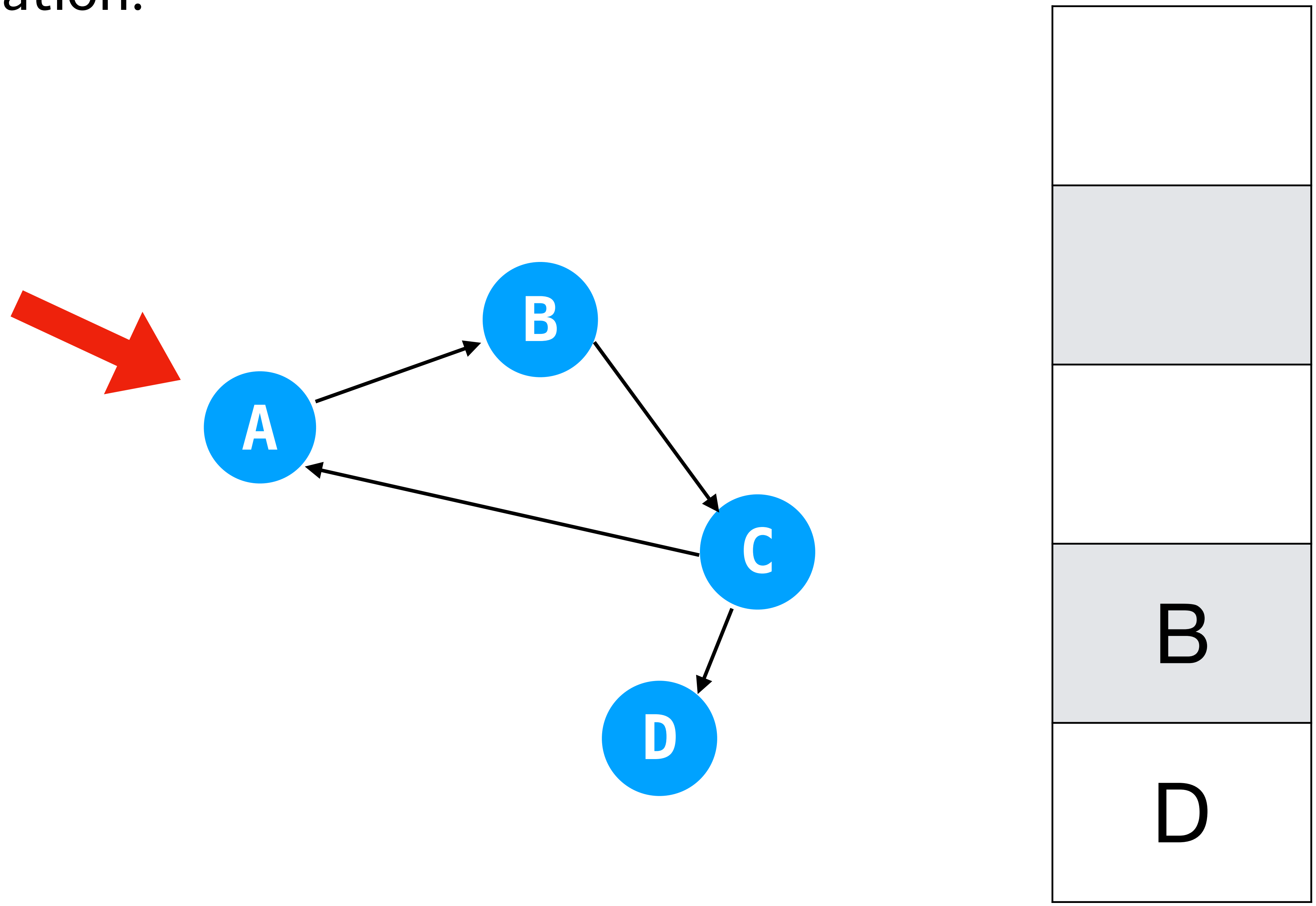
Depth First Search

- Depth-first search is defined just like with trees, with a slight modification.

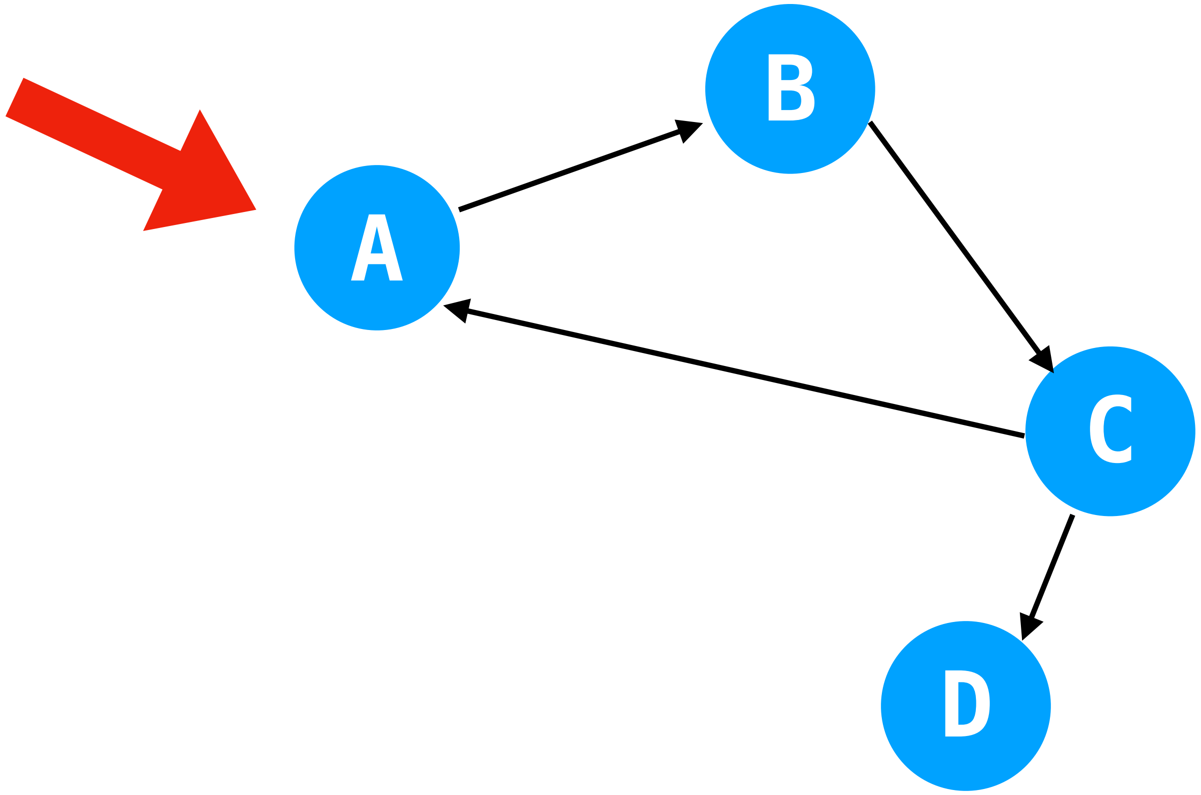


Depth First Search

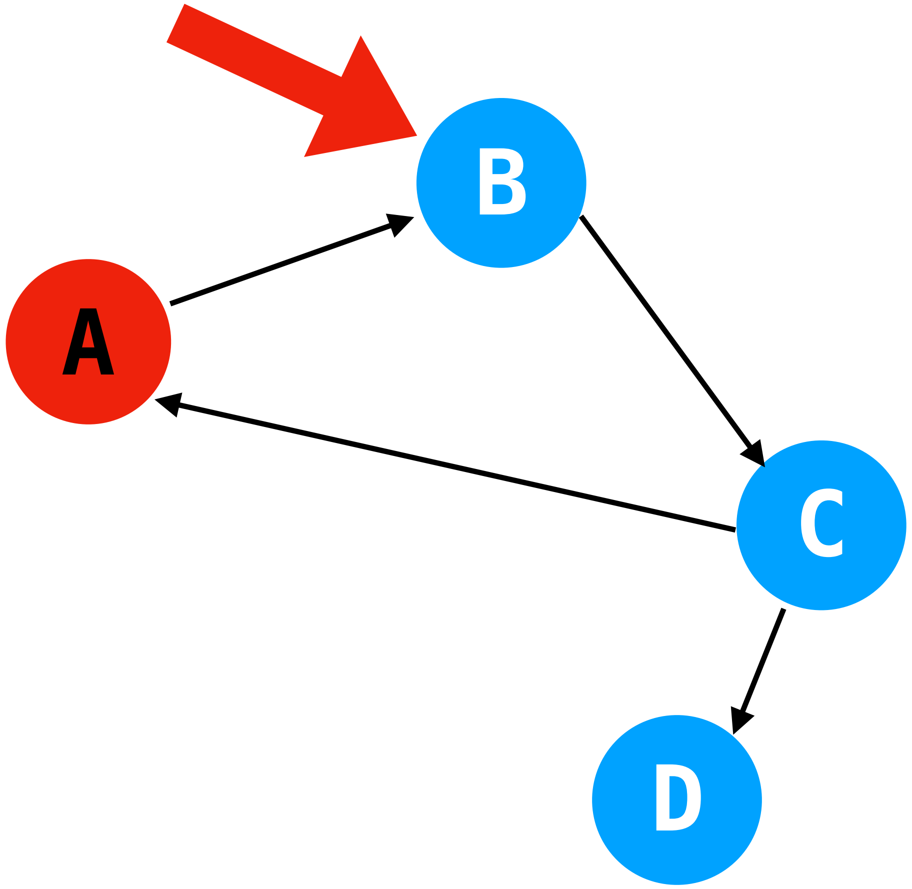
- Depth-first search is defined just like with trees, with a slight modification.



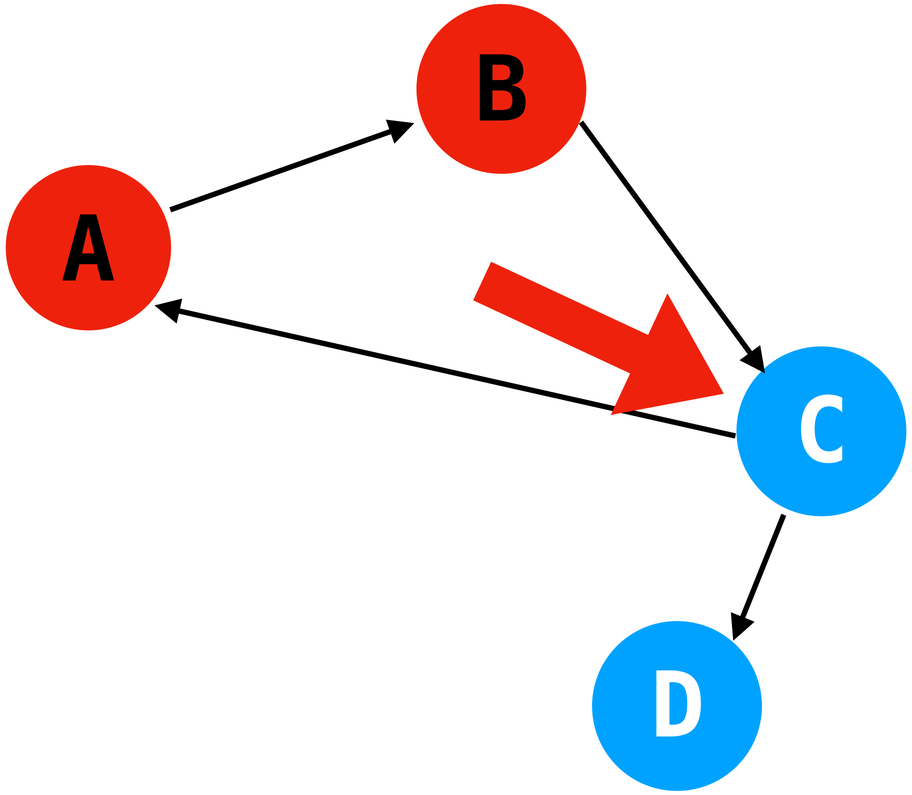
We need to remember visited nodes.



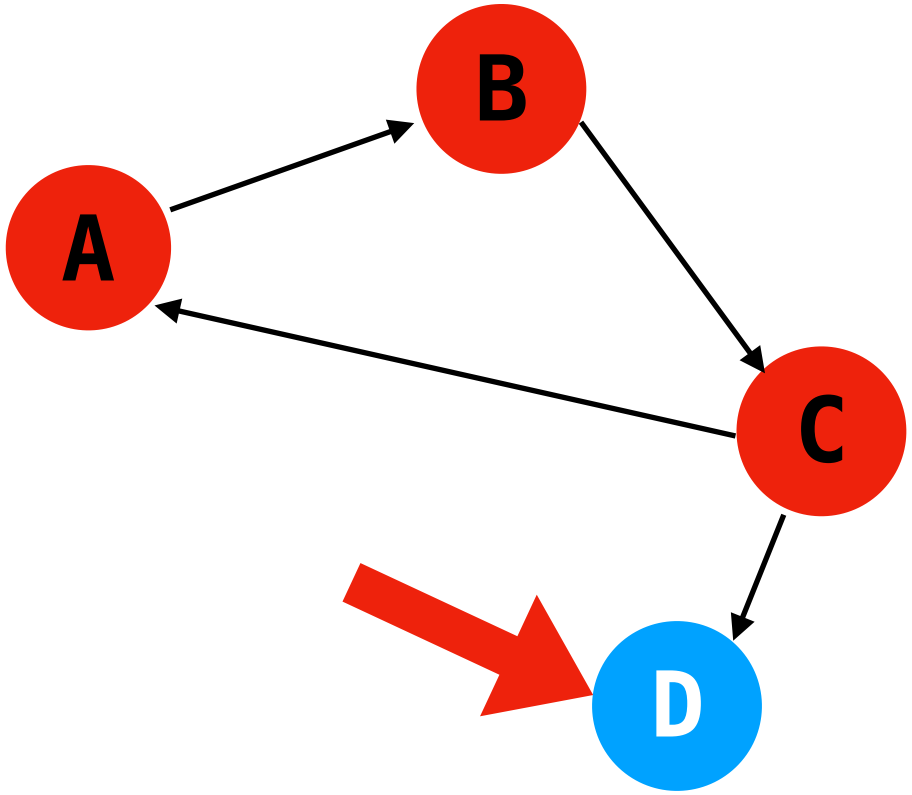
We need to remember visited nodes.



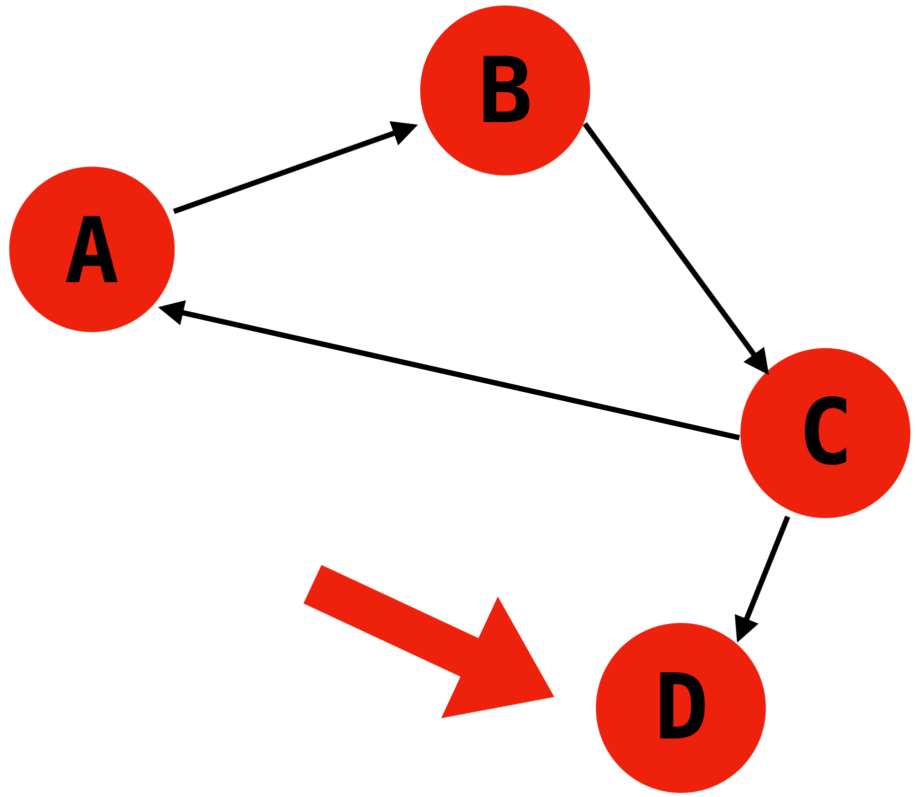
We need to remember visited nodes.



We need to remember visited nodes.



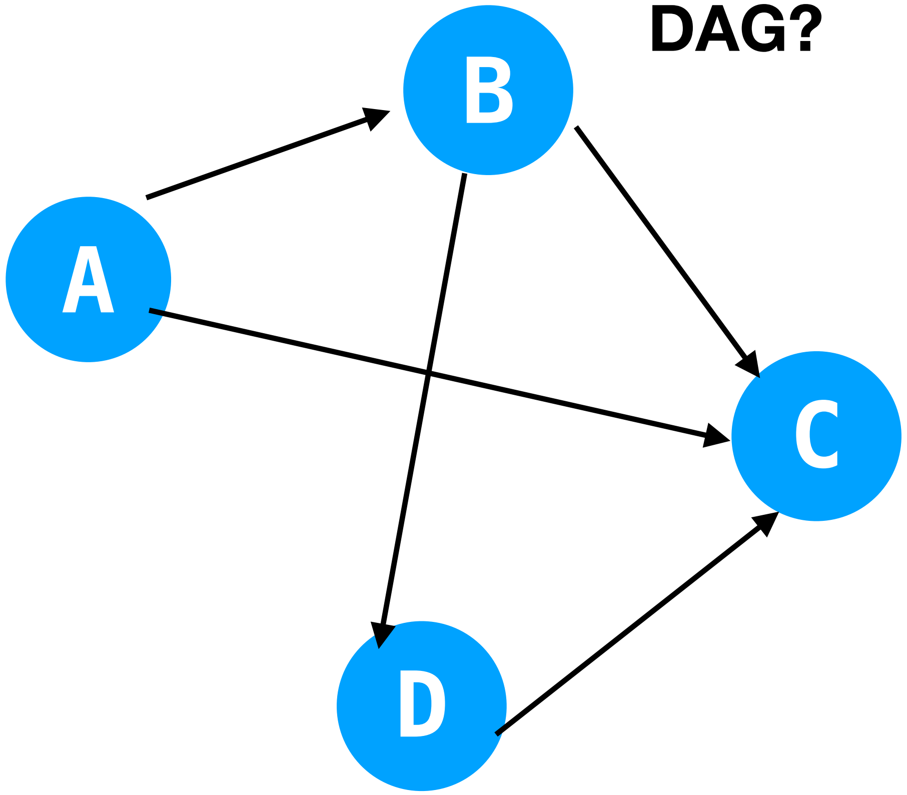
We need to remember visited nodes.



BFS is the same story: don't revisit nodes.

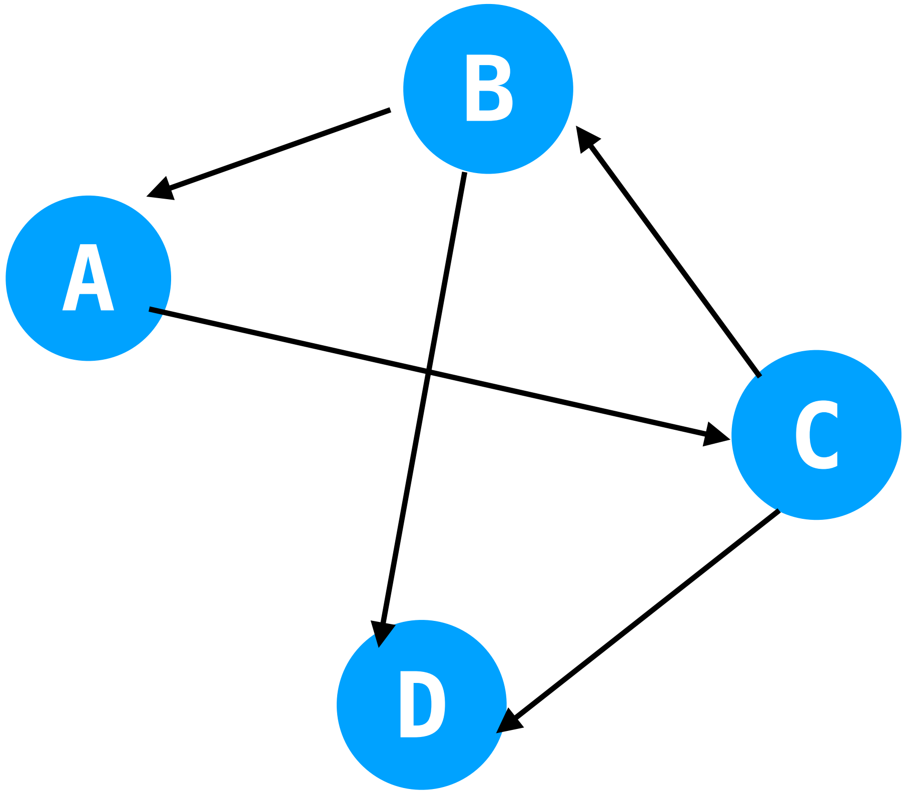
Directed Acyclic Graphs

- Directed Acyclic Graphs (aka DAGs) are graphs that have directed edges and no cycles.



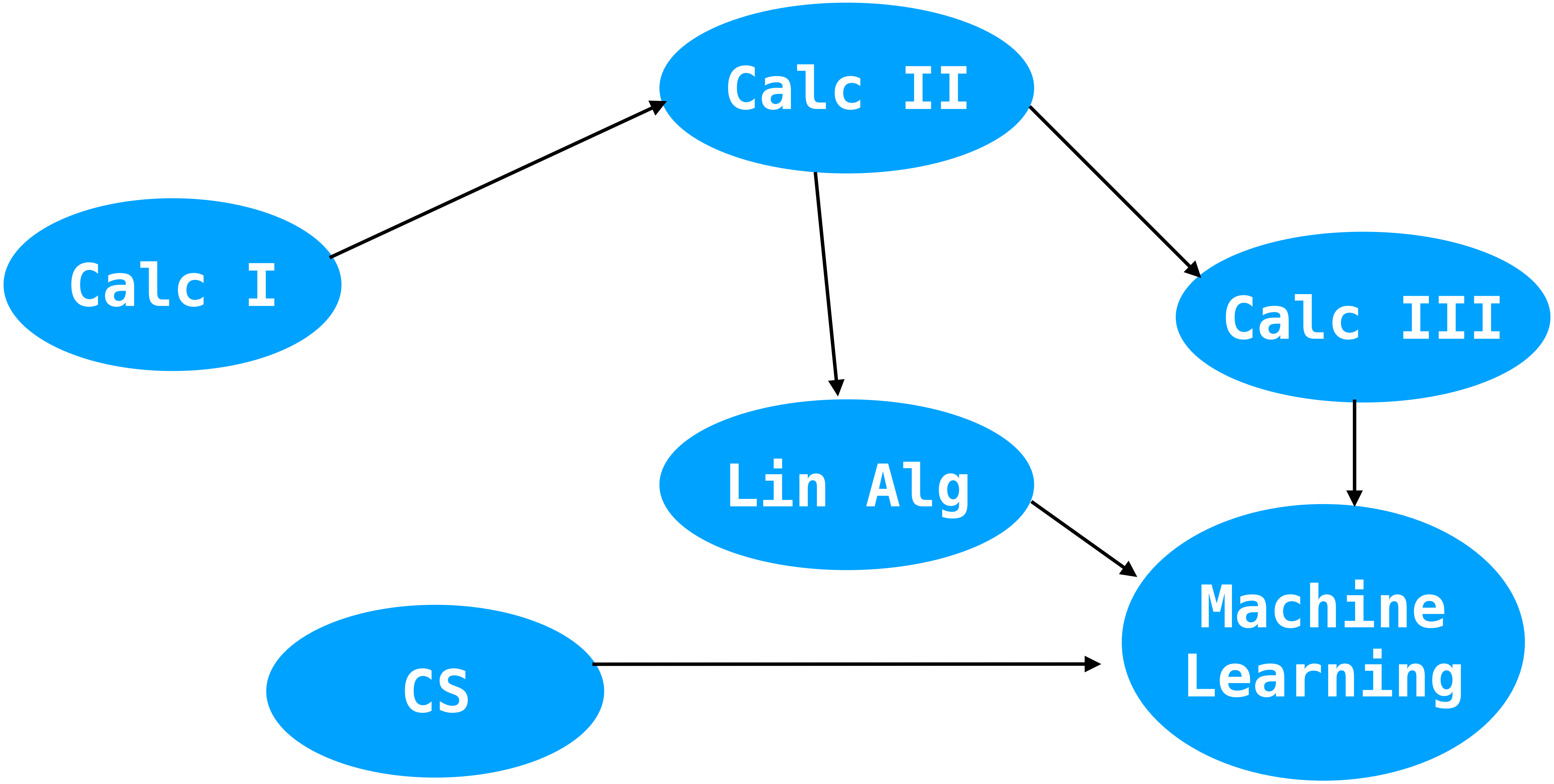
DAG?

DAG?



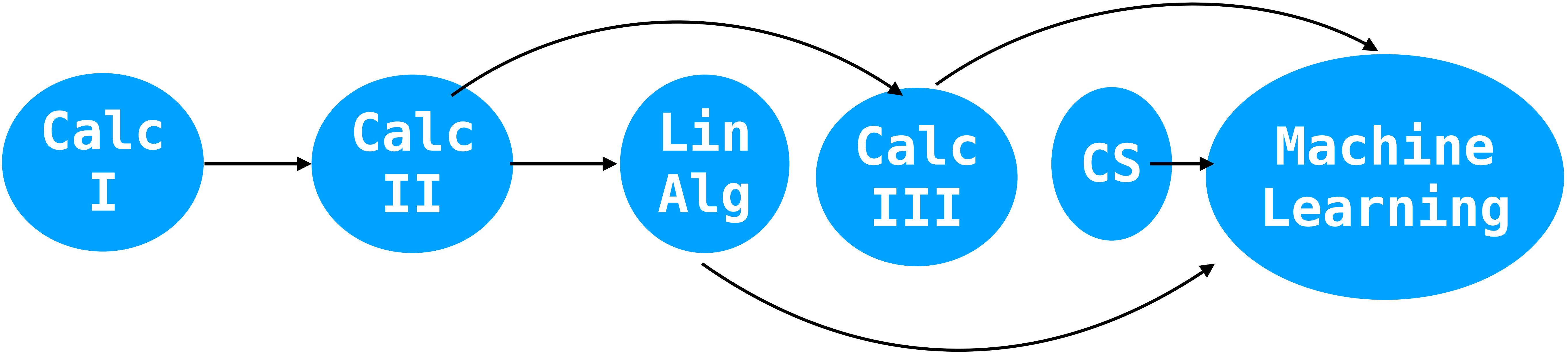
DAGs

- DAGs are great for representing dependency graphs.



Topological Sorting

- The properties of a DAG allow us to organize the nodes into a topological order, where nodes are arranged in order and edges are directed forward only.
- There may be more than one possible topological sort.
- One way of thinking about it is ordering courses so you don't violate any prerequisites.

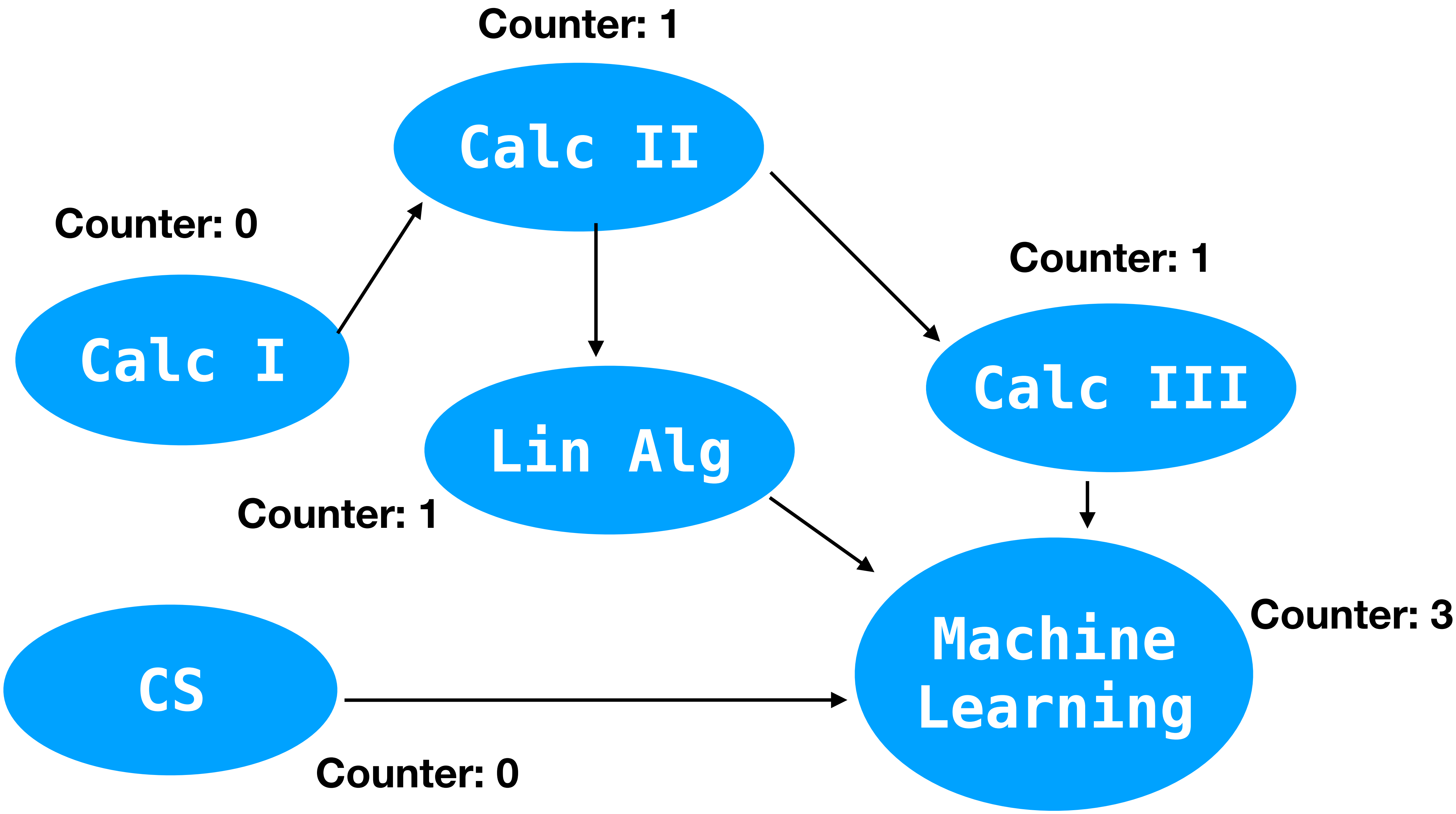


How to Topologically Sort?

1. Each node will have a counter that is initialized to the number of incoming edges.
2. Take all nodes with counter = 0 and add them to your result list.
3. Decrement the counter of neighbors by 1.
4. Repeat step 2.

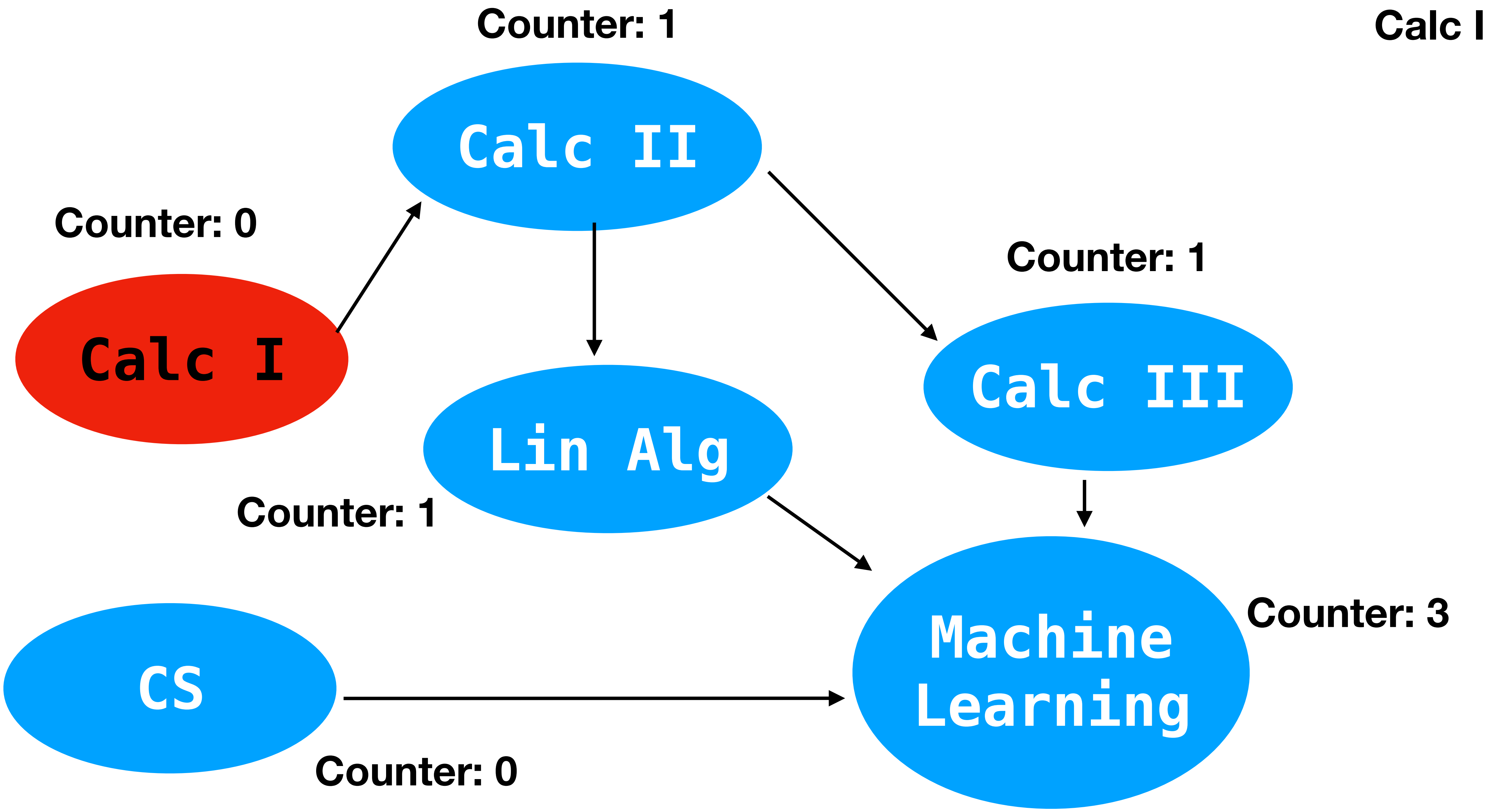
How to Topologically Sort?

- We'll run a modified DFS.



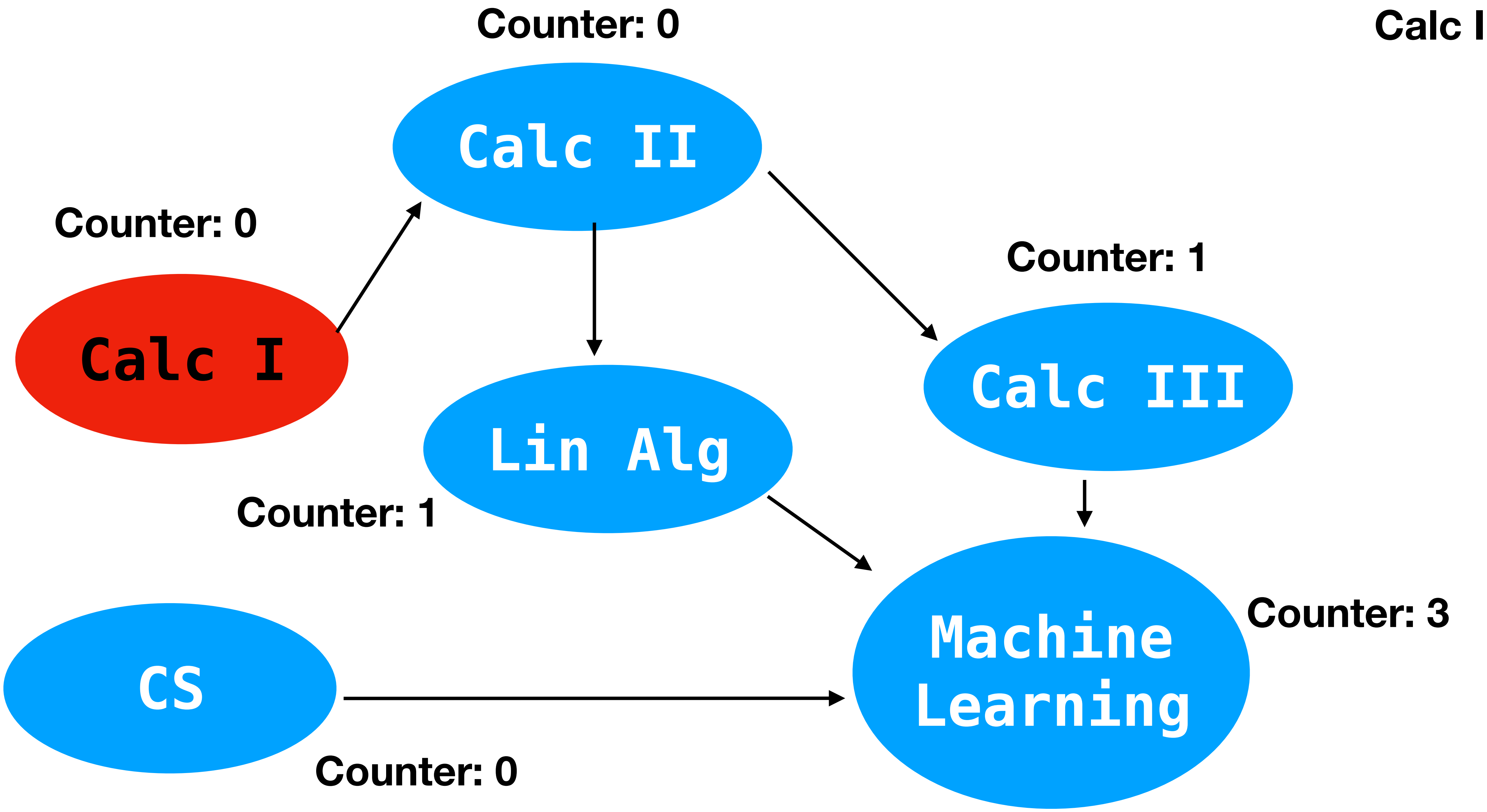
How to Topologically Sort?

- We'll run a modified DFS.



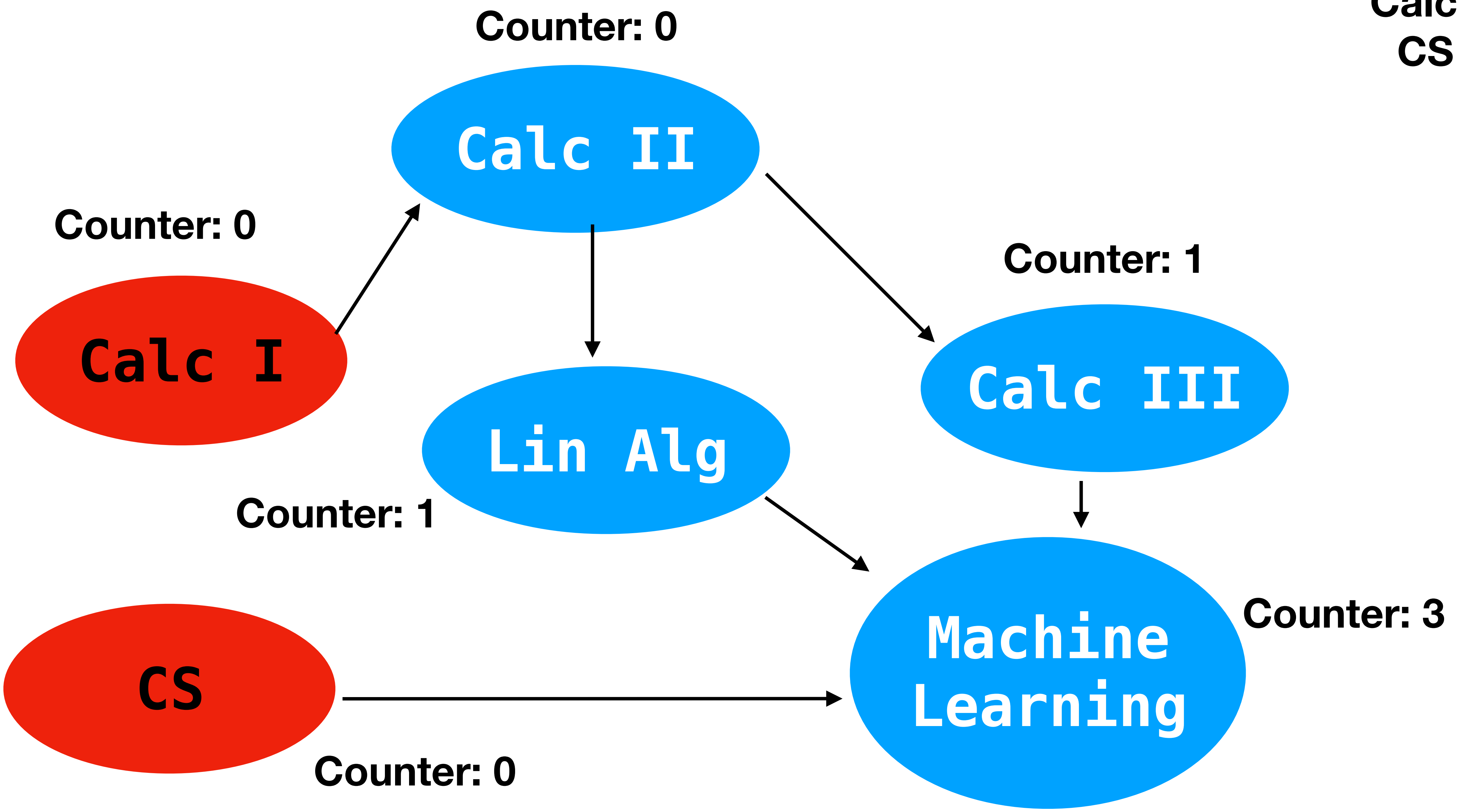
How to Topologically Sort?

- We'll run a modified DFS.



How to Topologically Sort?

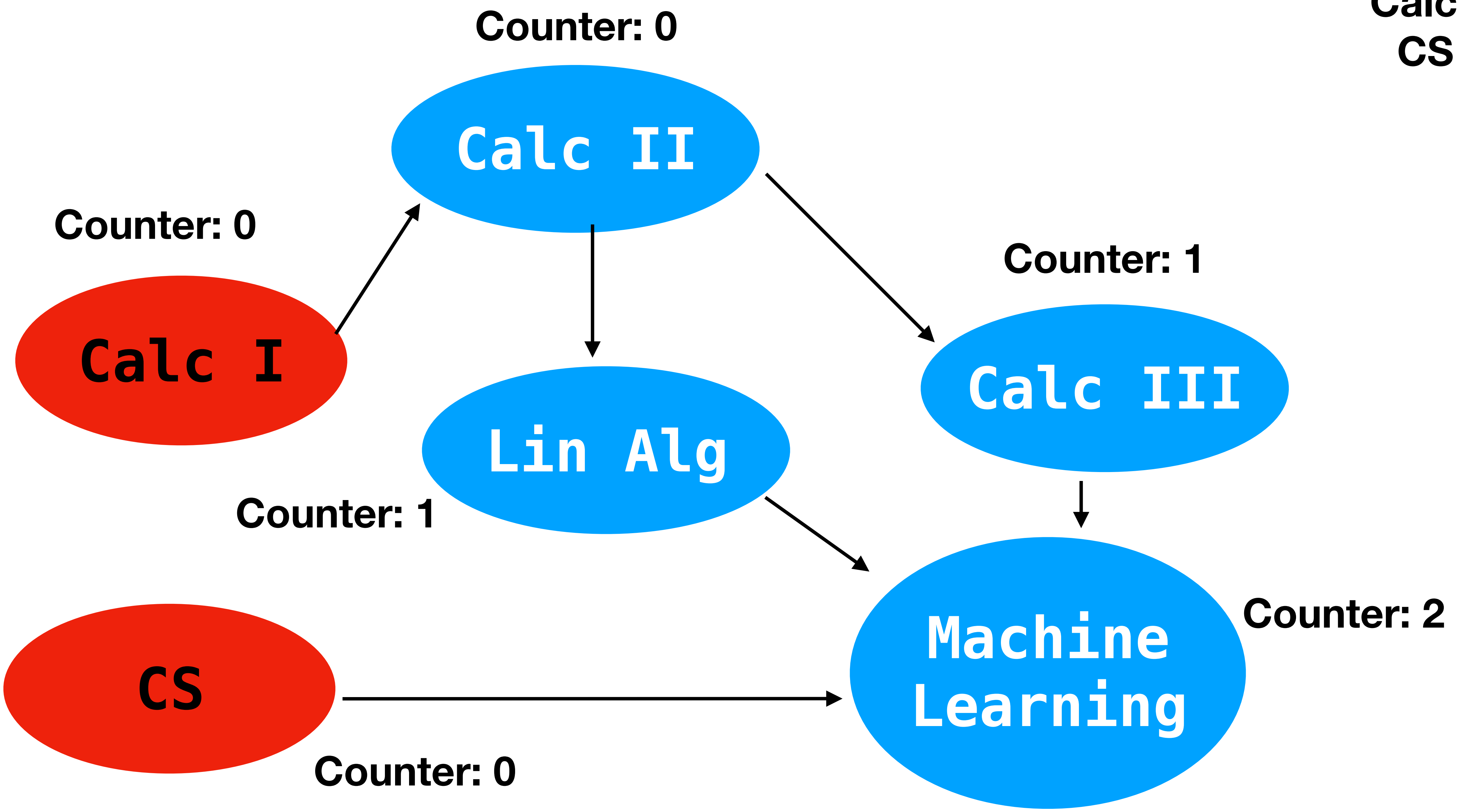
- We'll run a modified DFS.



Calc I
CS

How to Topologically Sort?

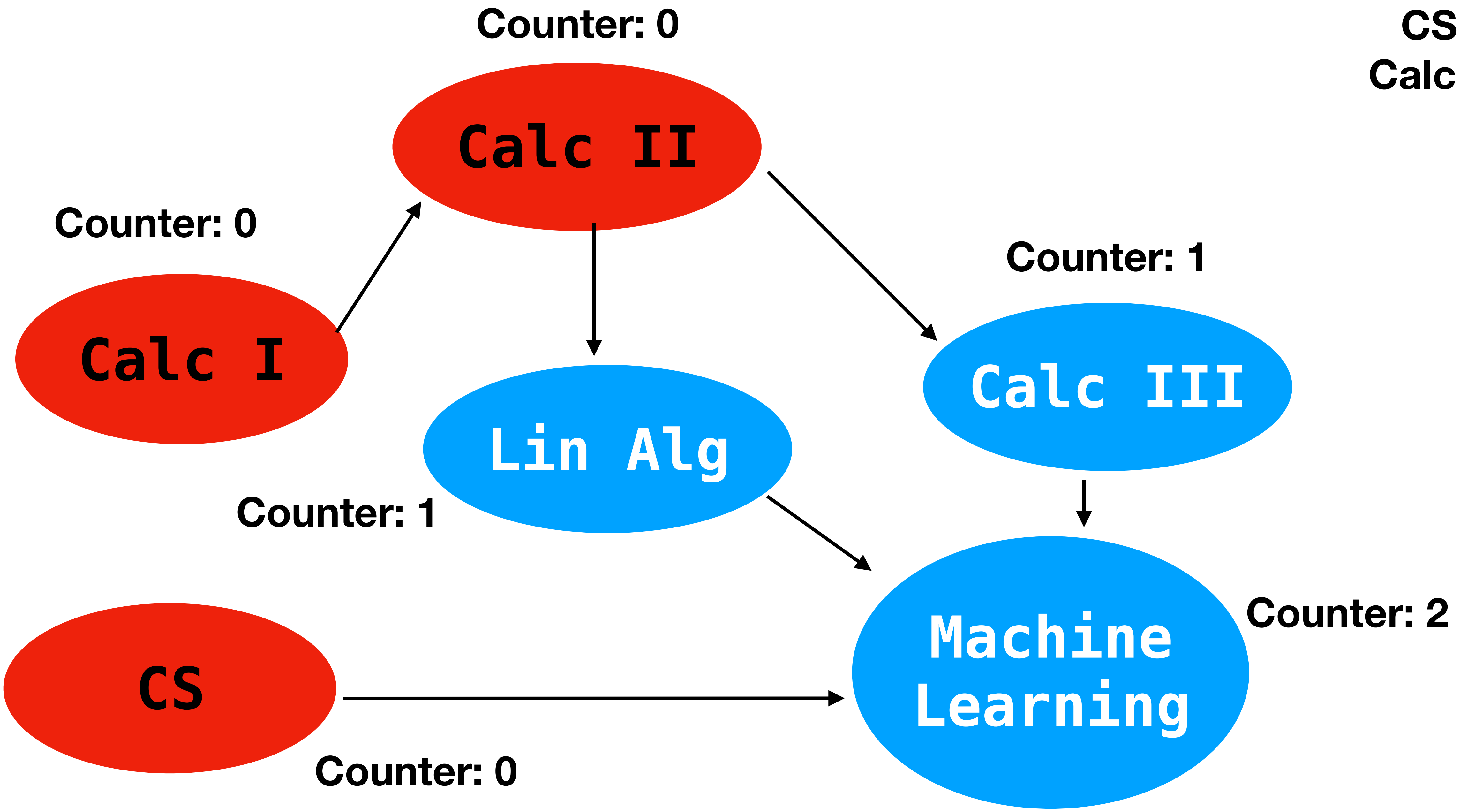
- We'll run a modified DFS.



Calc I
CS

How to Topologically Sort?

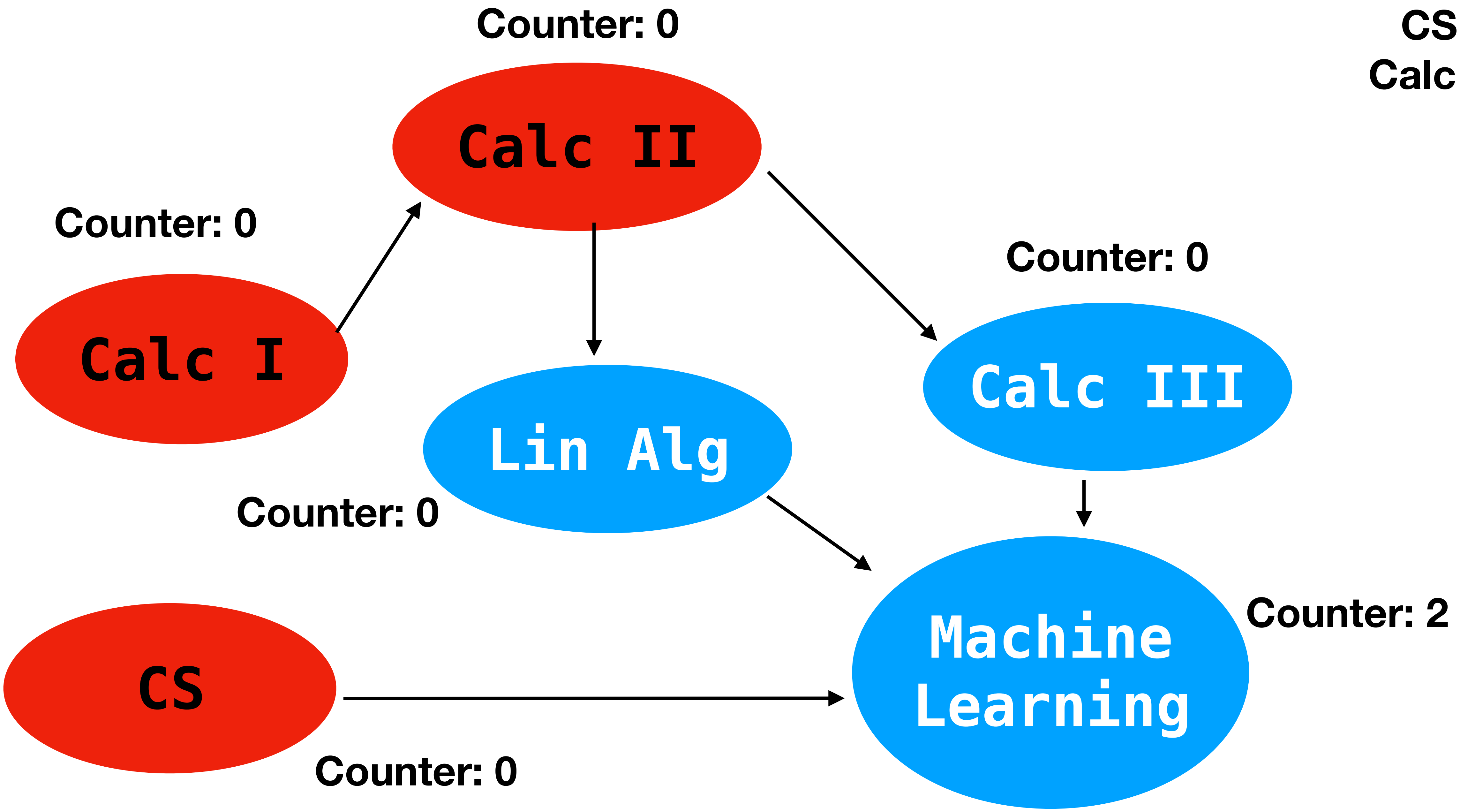
- We'll run a modified DFS.



Calc I
CS
Calc II

How to Topologically Sort?

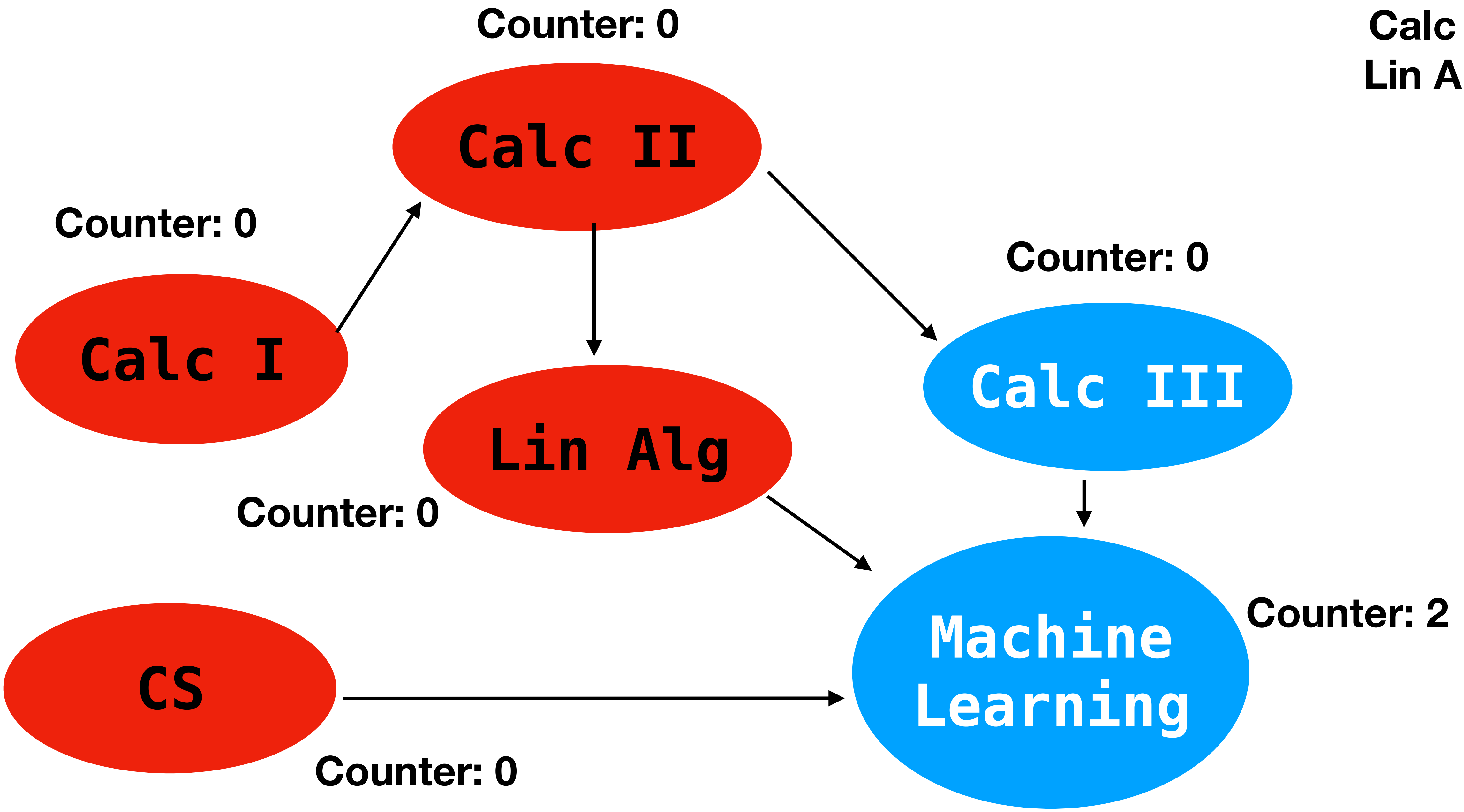
- We'll run a modified DFS.



Calc I
CS
Calc II

How to Topologically Sort?

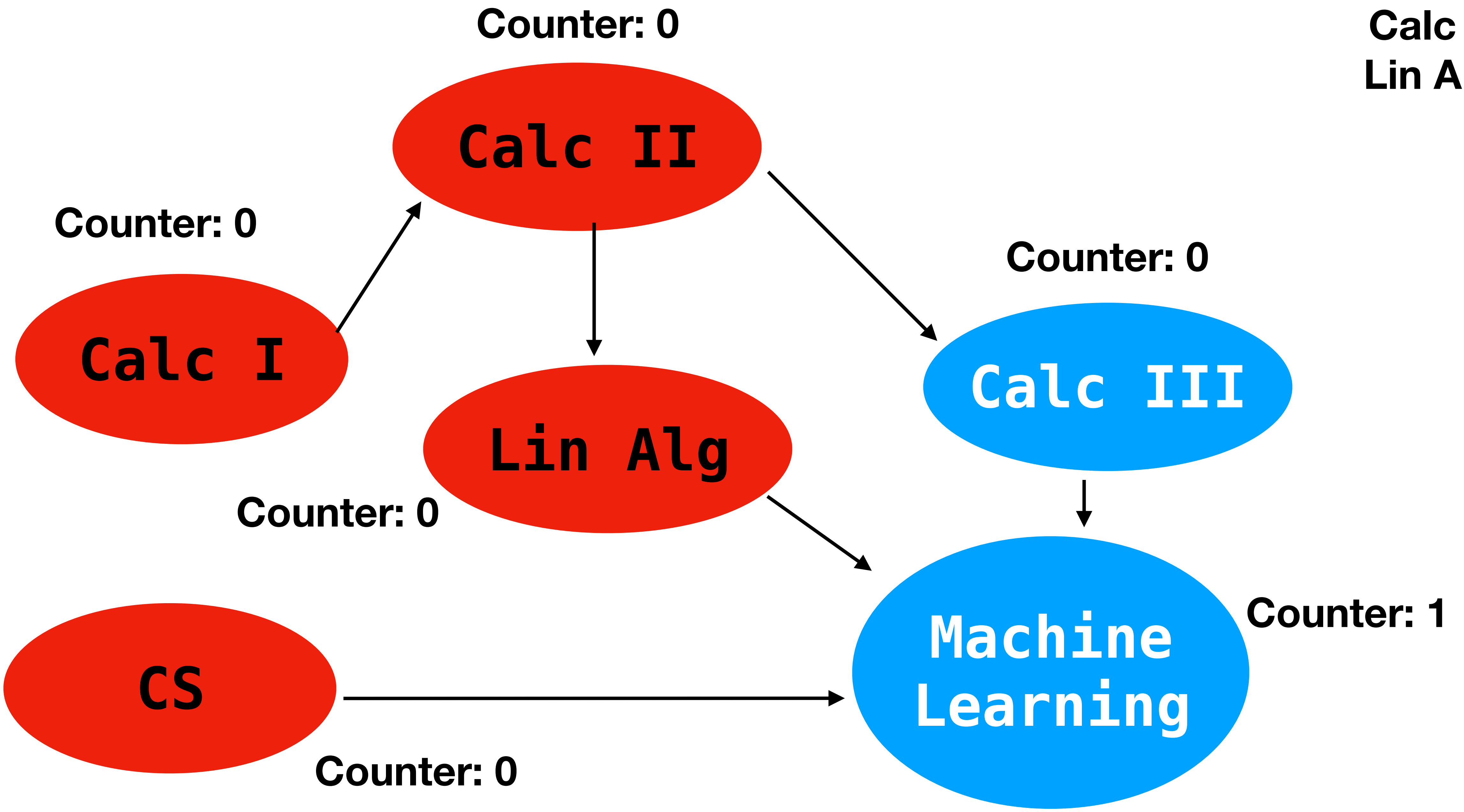
- We'll run a modified DFS.



Calc I
CS
Calc II
Lin Alg

How to Topologically Sort?

- We'll run a modified DFS.

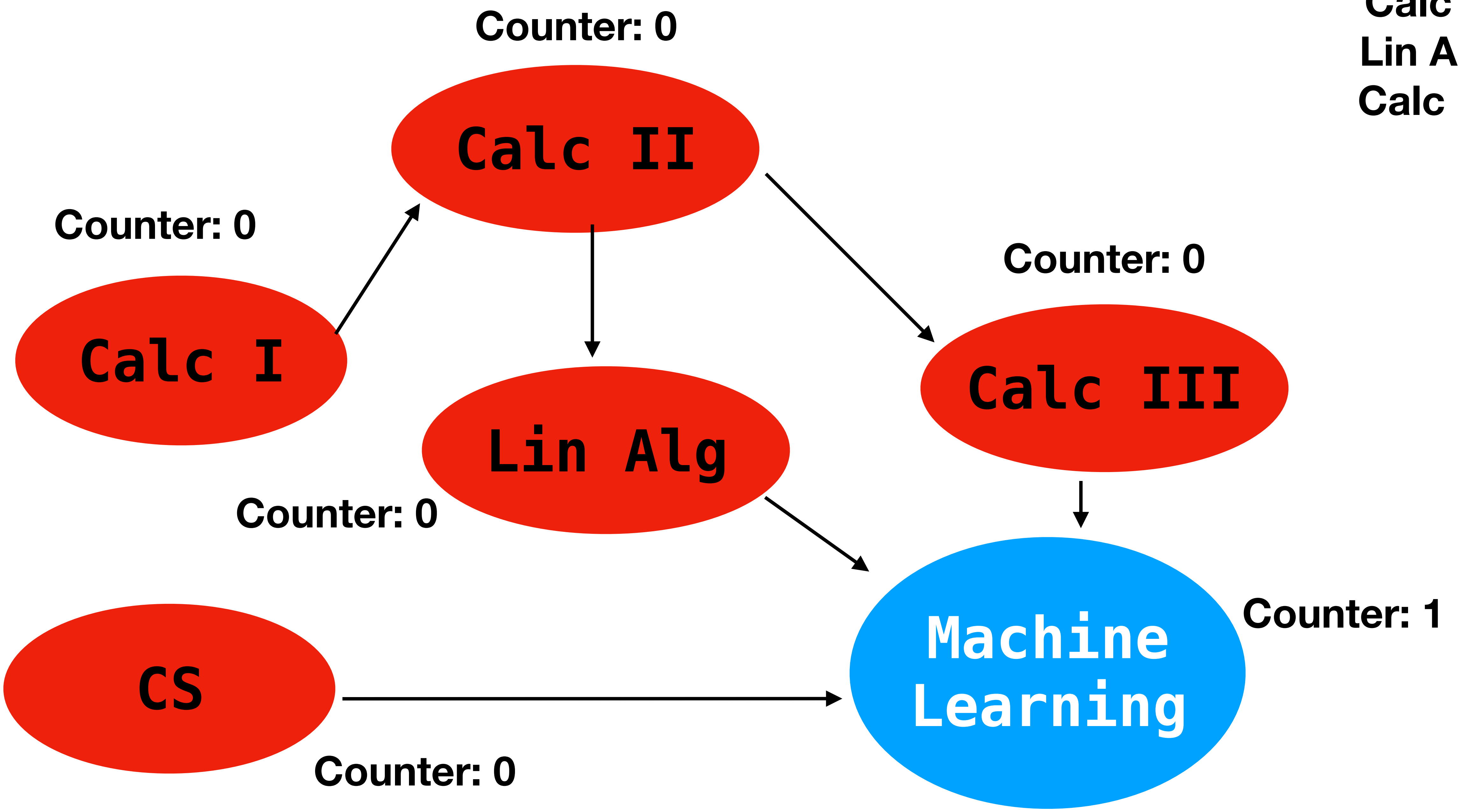


Calc I
CS
Calc II
Lin Alg

How to Topologically Sort?

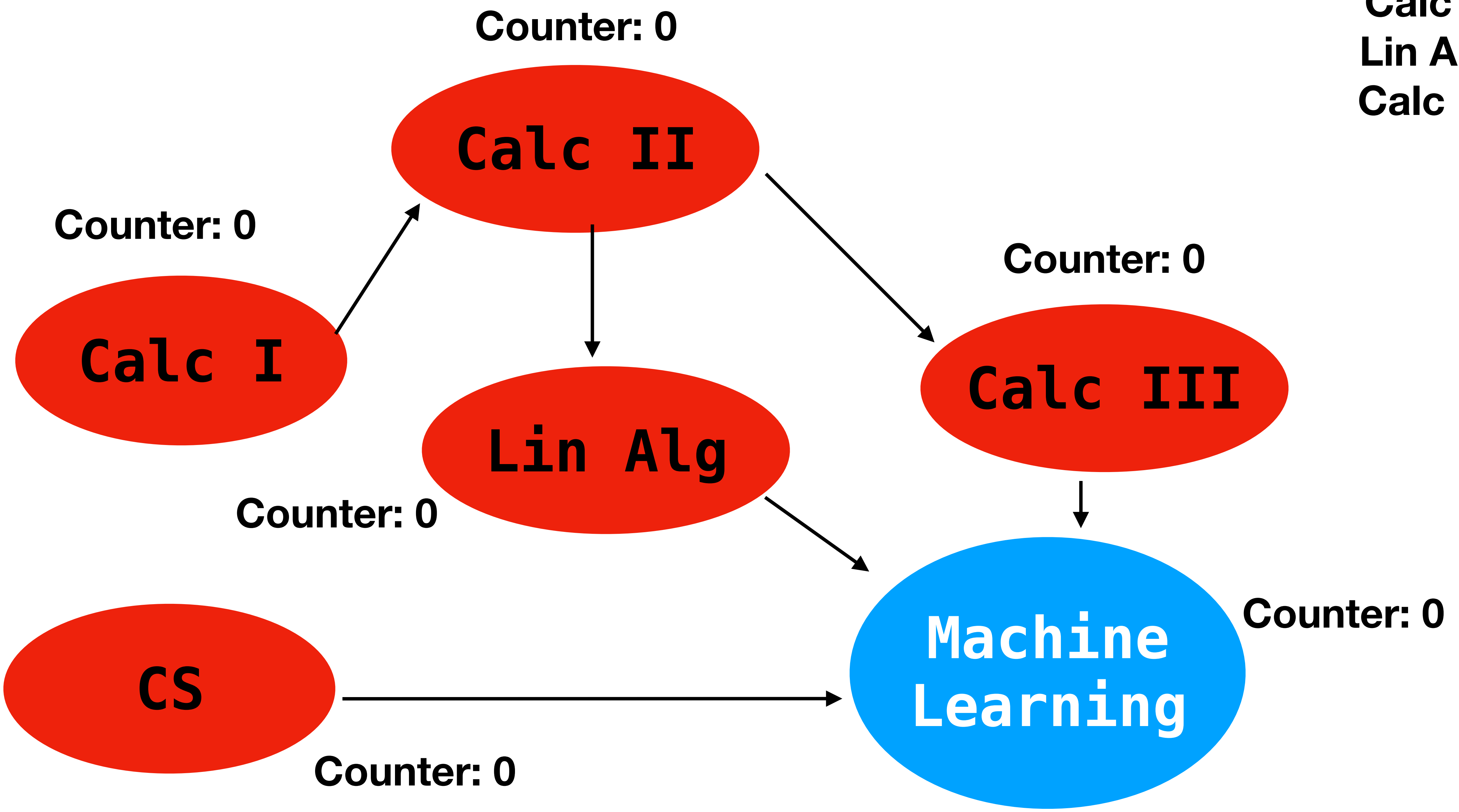
- We'll run a modified DFS.

Calc I
CS
Calc II
Lin Alg
Calc III



How to Topologically Sort?

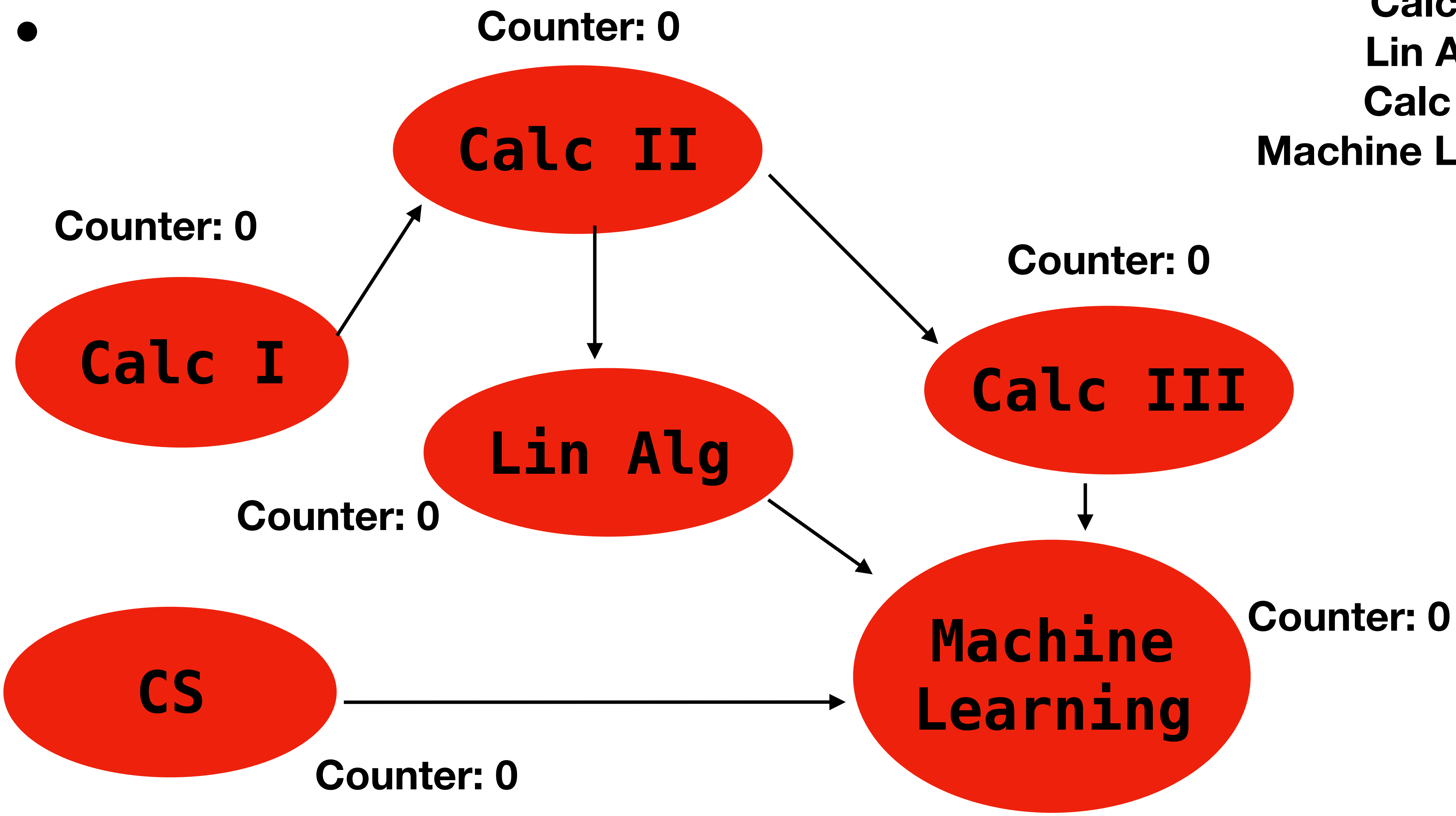
- We'll run a modified DFS.



Calc I
CS
Calc II
Lin Alg
Calc III

How to Topologically Sort?

- We'll run a modified DFS.
-



Calc I
CS
Calc II
Lin Alg
Calc III
Machine Learning