Scheduling

Set of tasks for baking cookies:

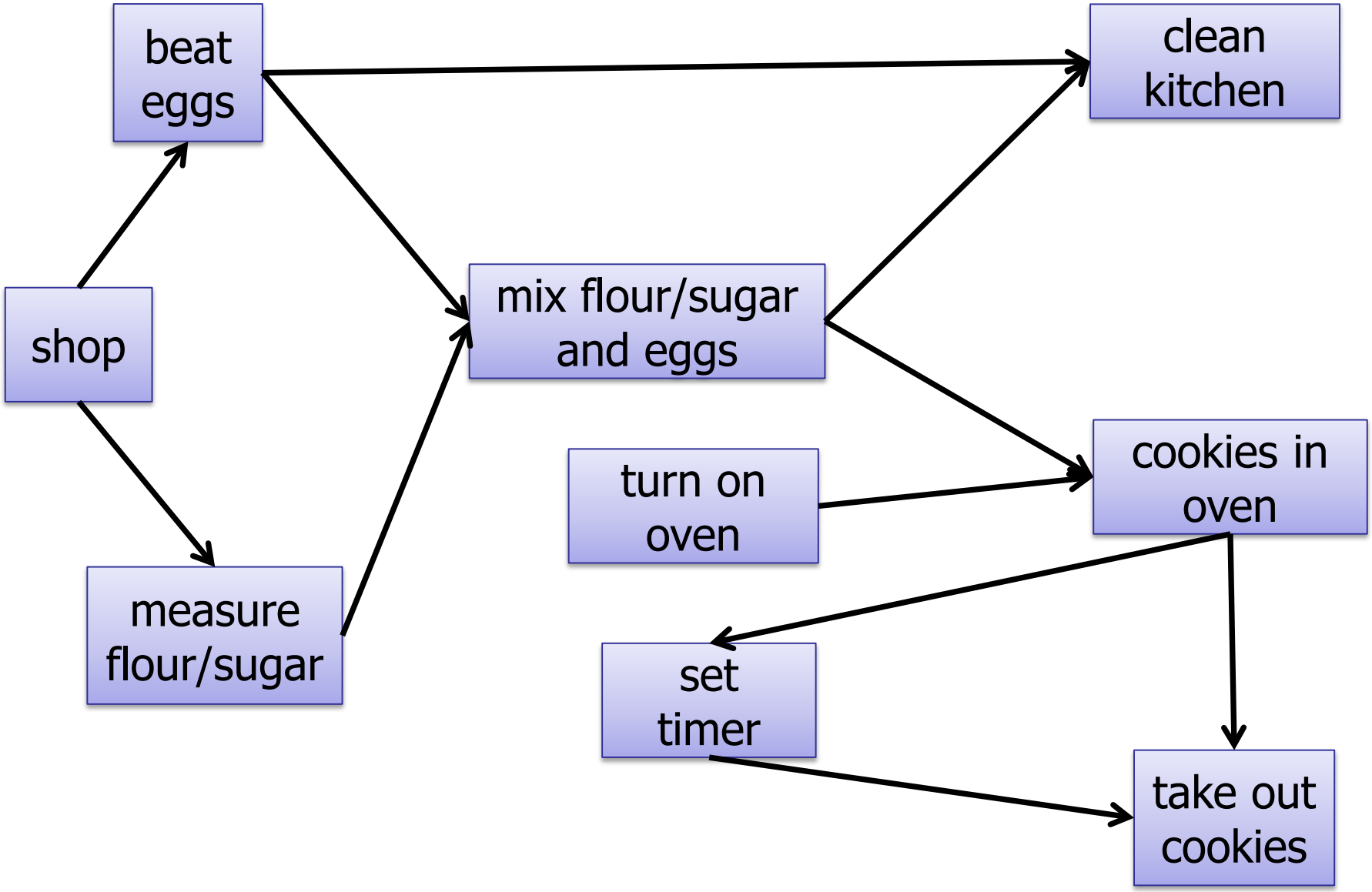
- Shop for groceries
- Put the cookies in the oven
- Clean the kitchen
- Beat the eggs in a bowl
- Measure the flour and sugar in a bowl
- Mix the eggs with the flour and sugar
- Turn on the oven
- Set the timer
- Take out the cookies

Scheduling

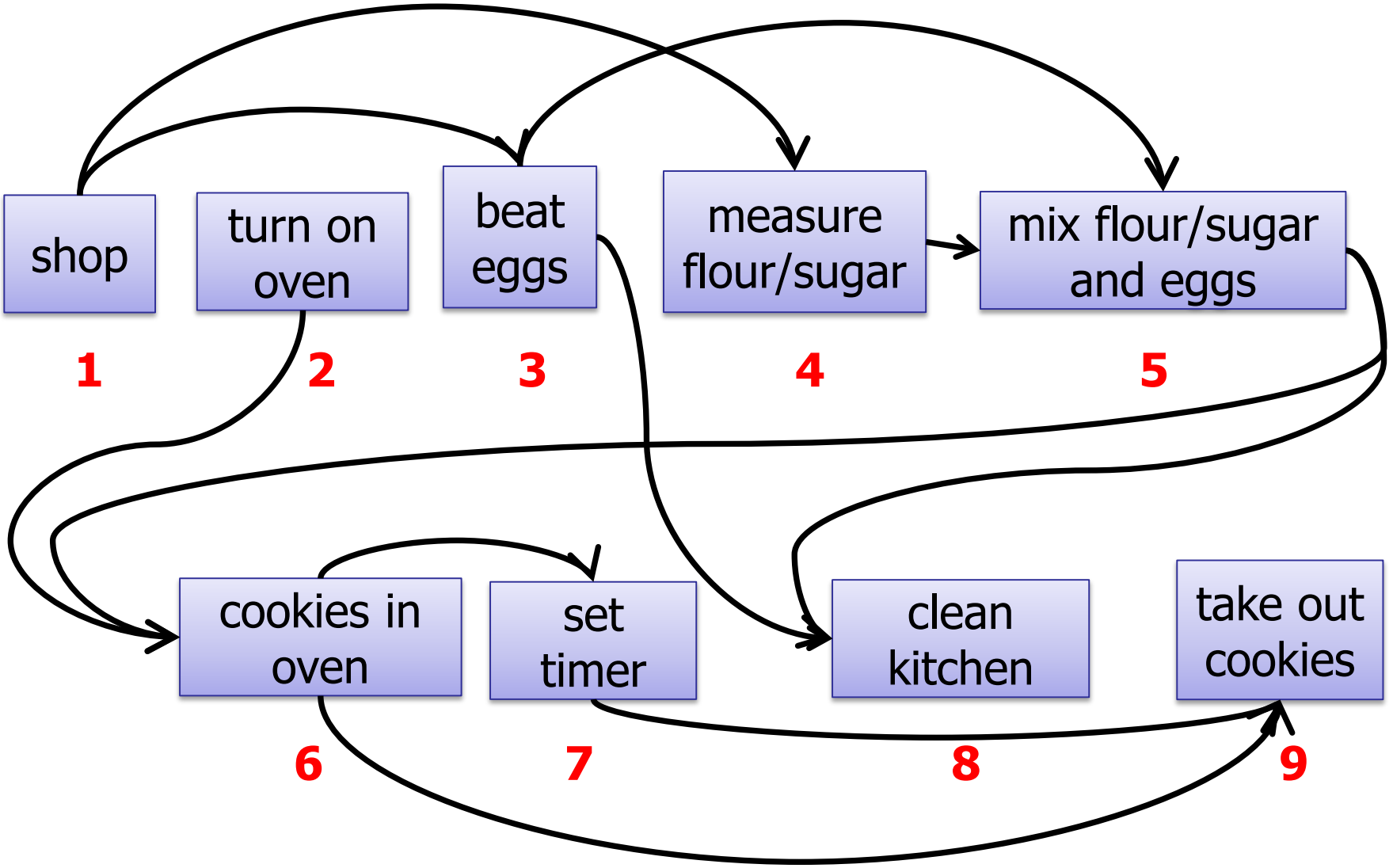
Ordering:

- Shop for groceries before beat the eggs
- Shop for groceries before measure the flour
- Turn on the oven before put the cookies in the oven
- Beat the eggs before mix the eggs with the flour
- Measure the flour before mix the eggs with the flour
- Put the cookies in the oven before set the timer
- Measure the flour before clean the kitchen
- Beat the eggs before clean the kitchen
- Mix the flour and the eggs before clean the kitchen

Scheduling


beat eggs
clean kitchen
shop
mix flour/sugar and eggs
turn on oven
cookies in oven
measure flour/sugar
set timer
take out cookies

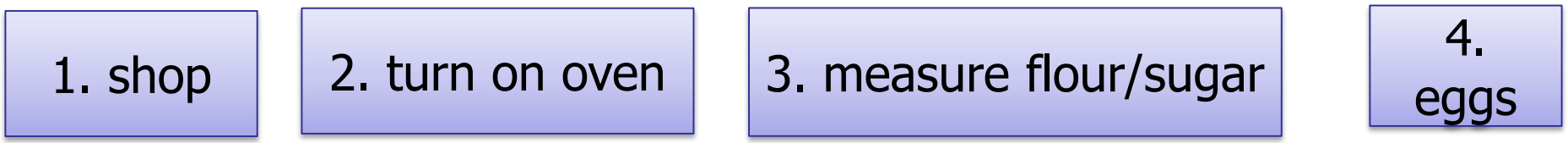
Topological Ordering



Topological Order

Properties:

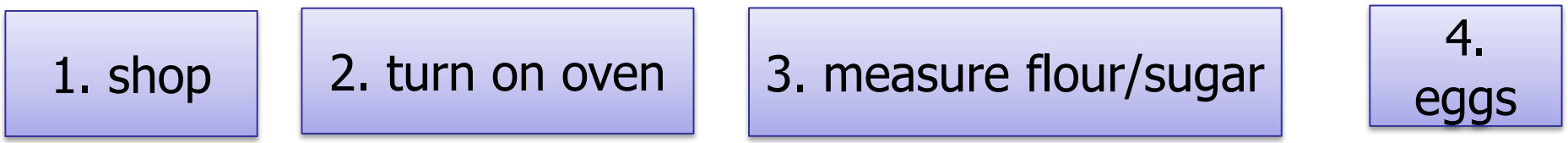
1. Sequential total ordering of all nodes



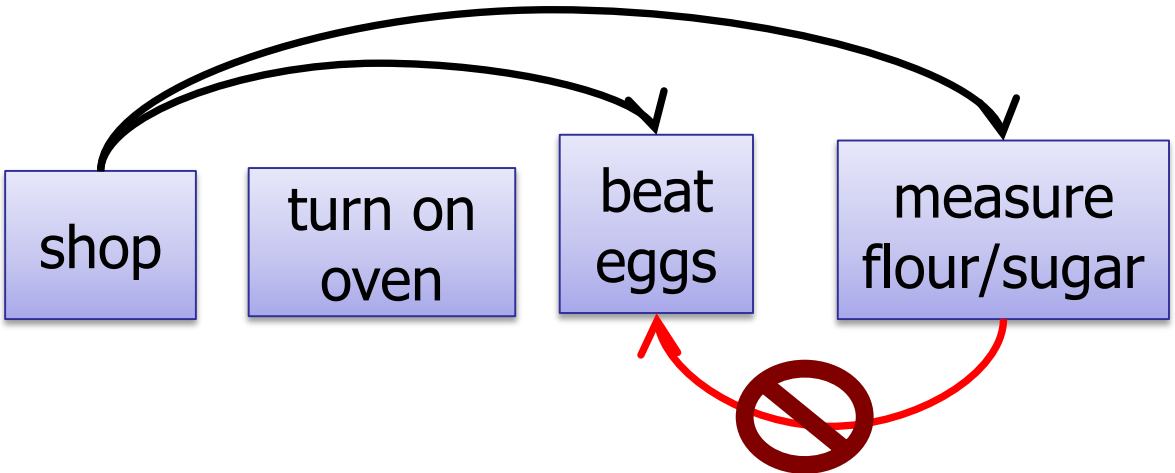
Topological Order

Properties:

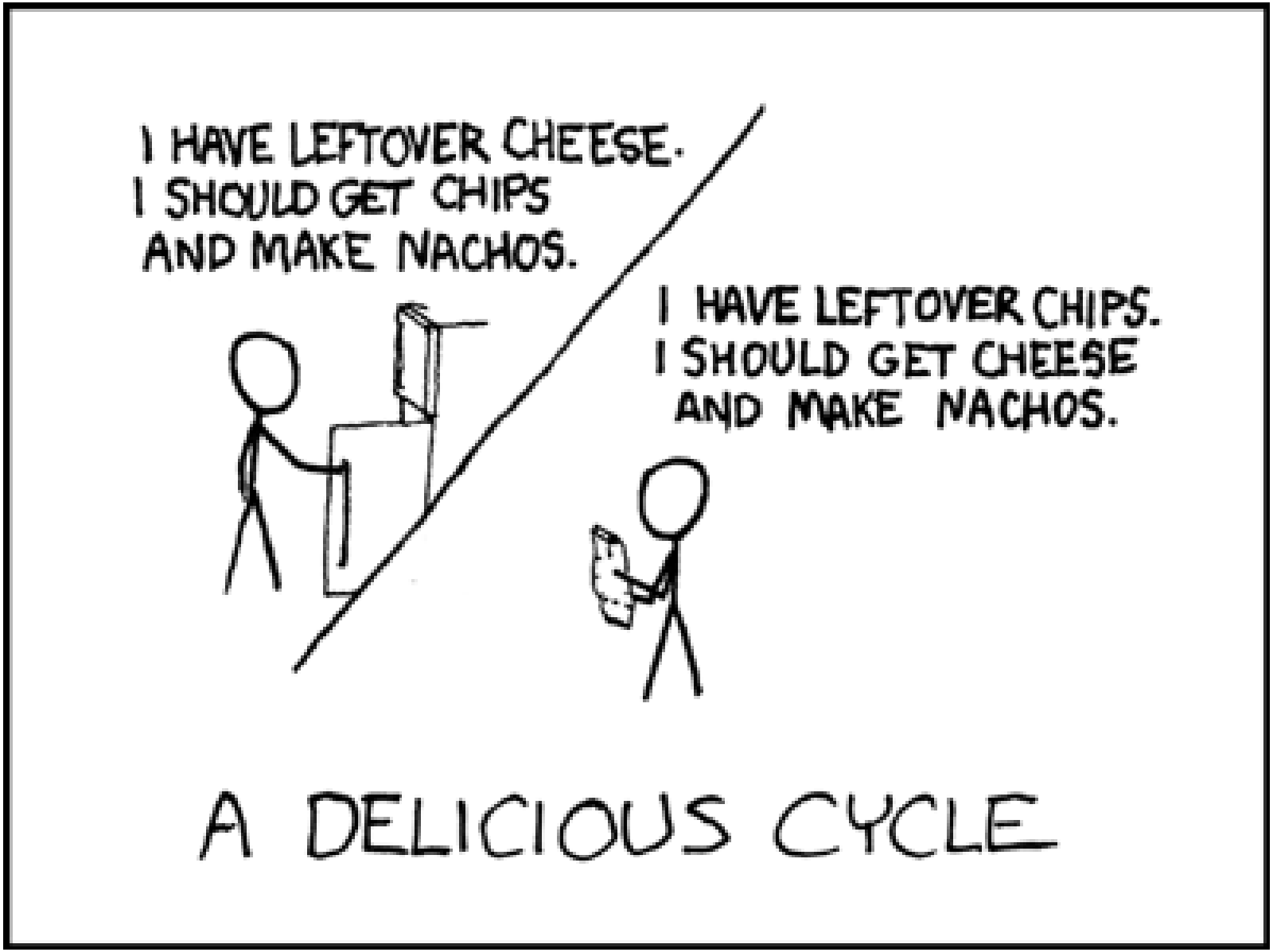
1. Sequential total ordering of all nodes



2. Edges only point forward

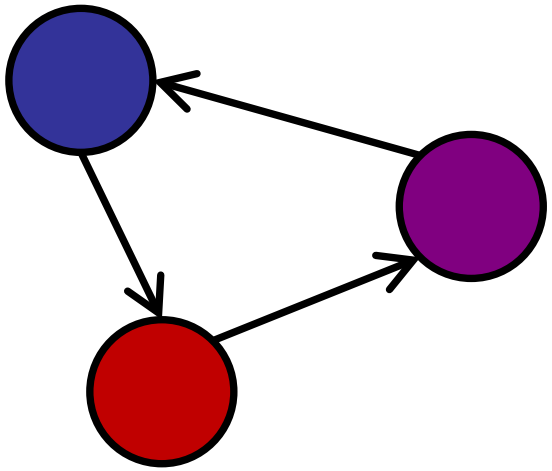


Which one should we do first?

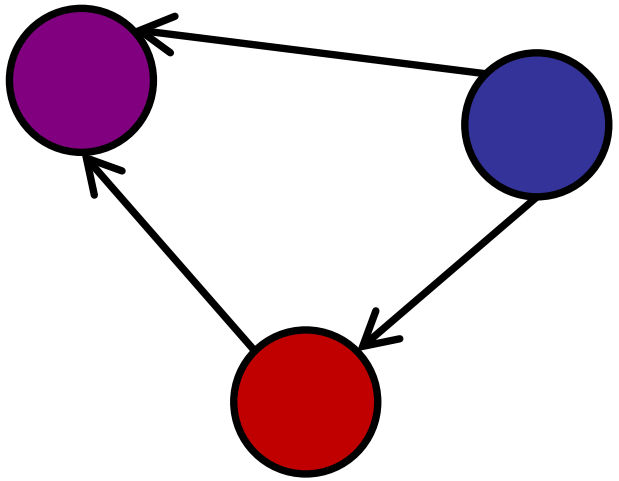


Directed Acyclic Graphs

Cyclic

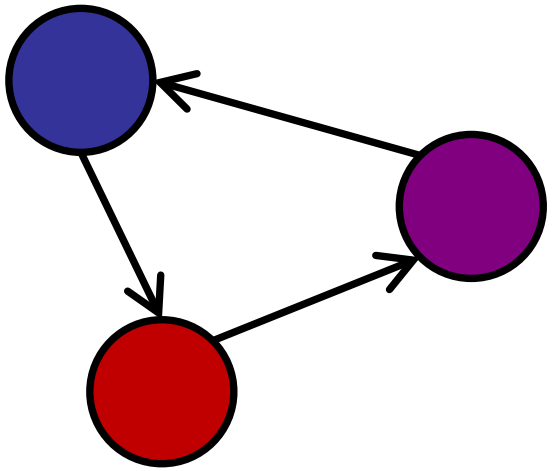


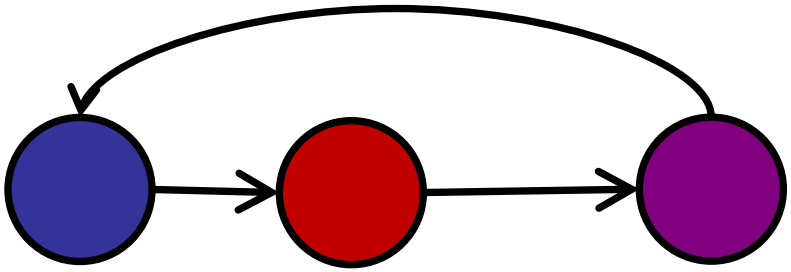
Acyclic



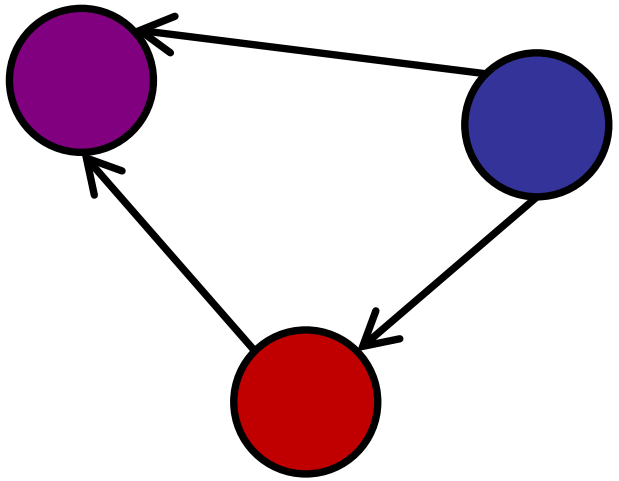
Directed Acyclic Graphs

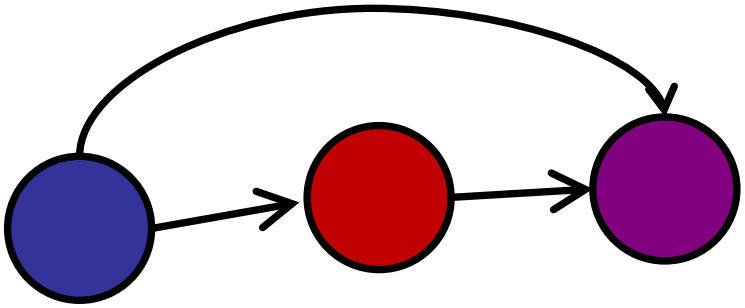
Cyclic





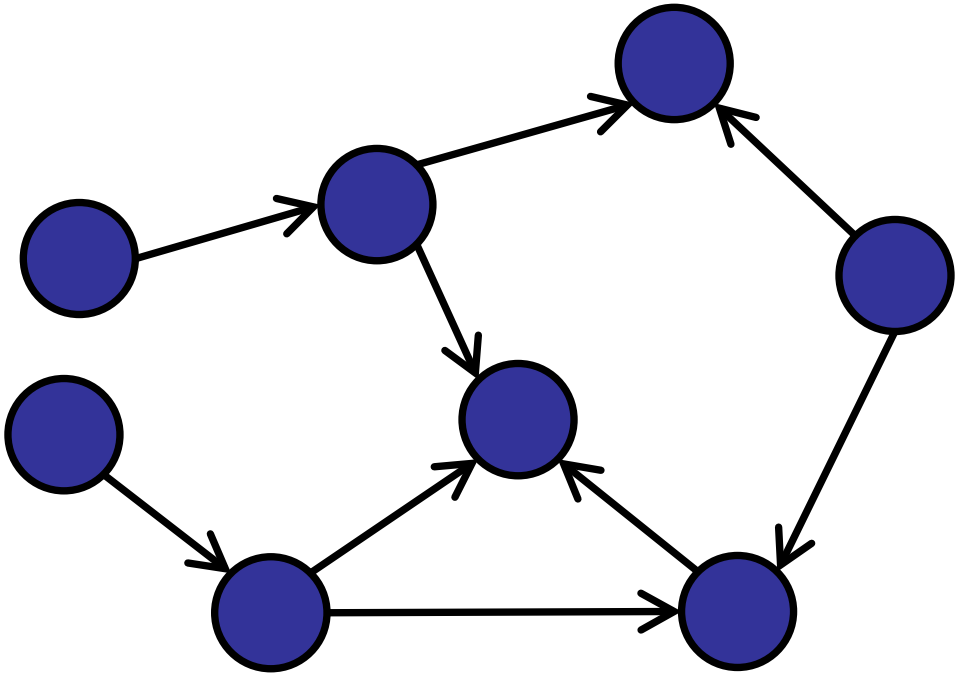
Acyclic

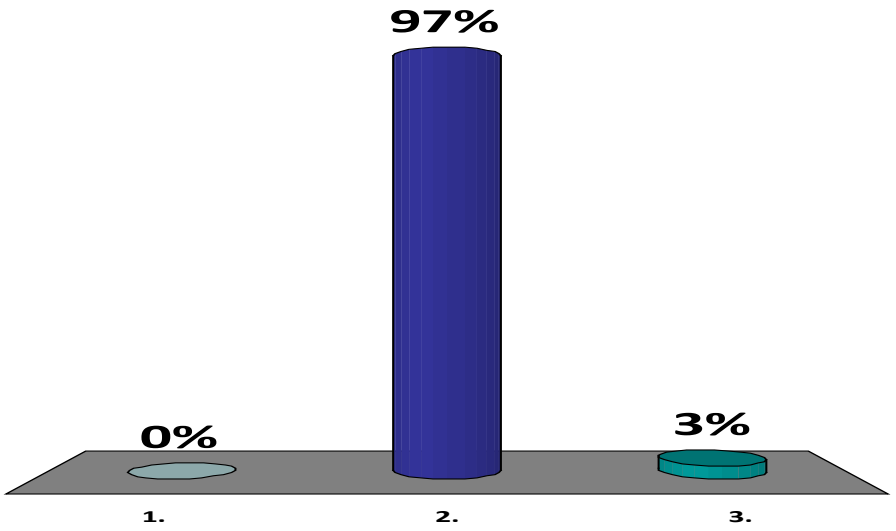




Is this graph:

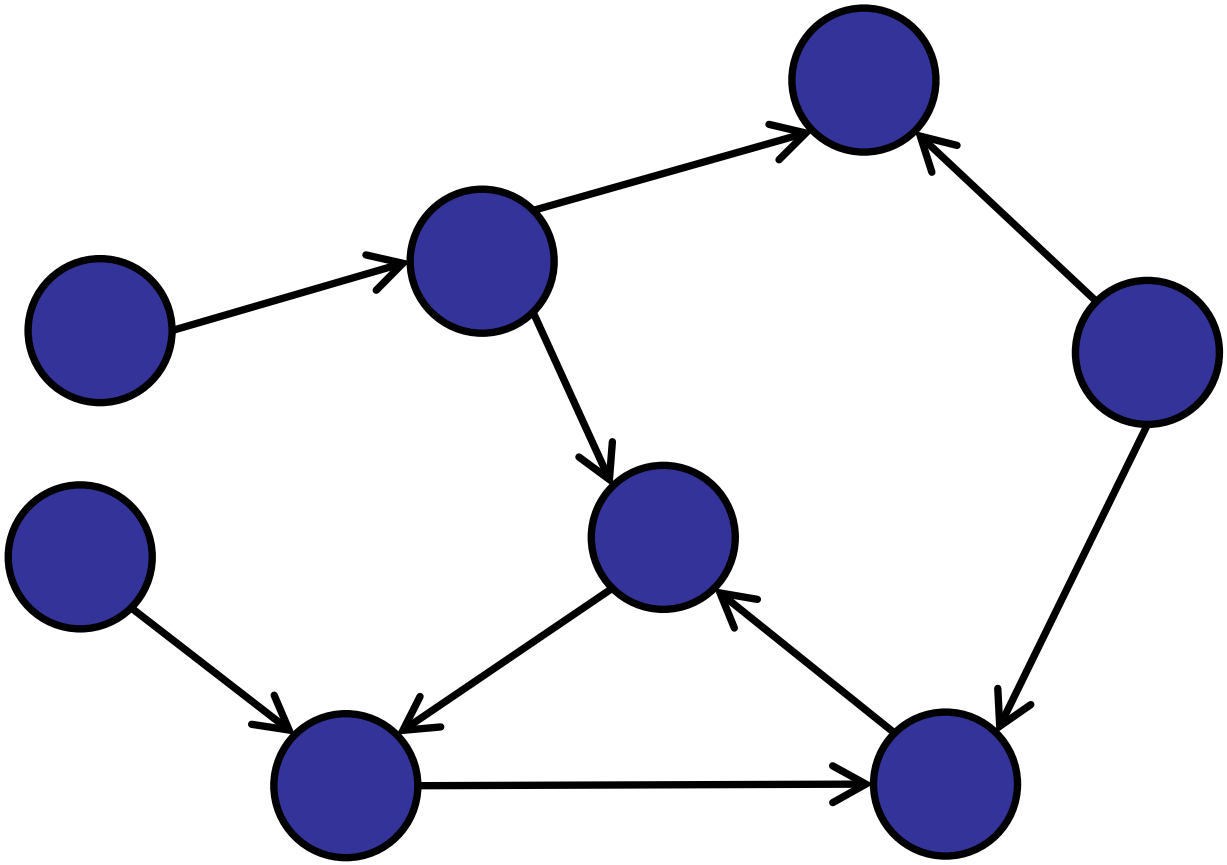
1. Cyclic
2. ✓ Acyclic
3. Transcendental



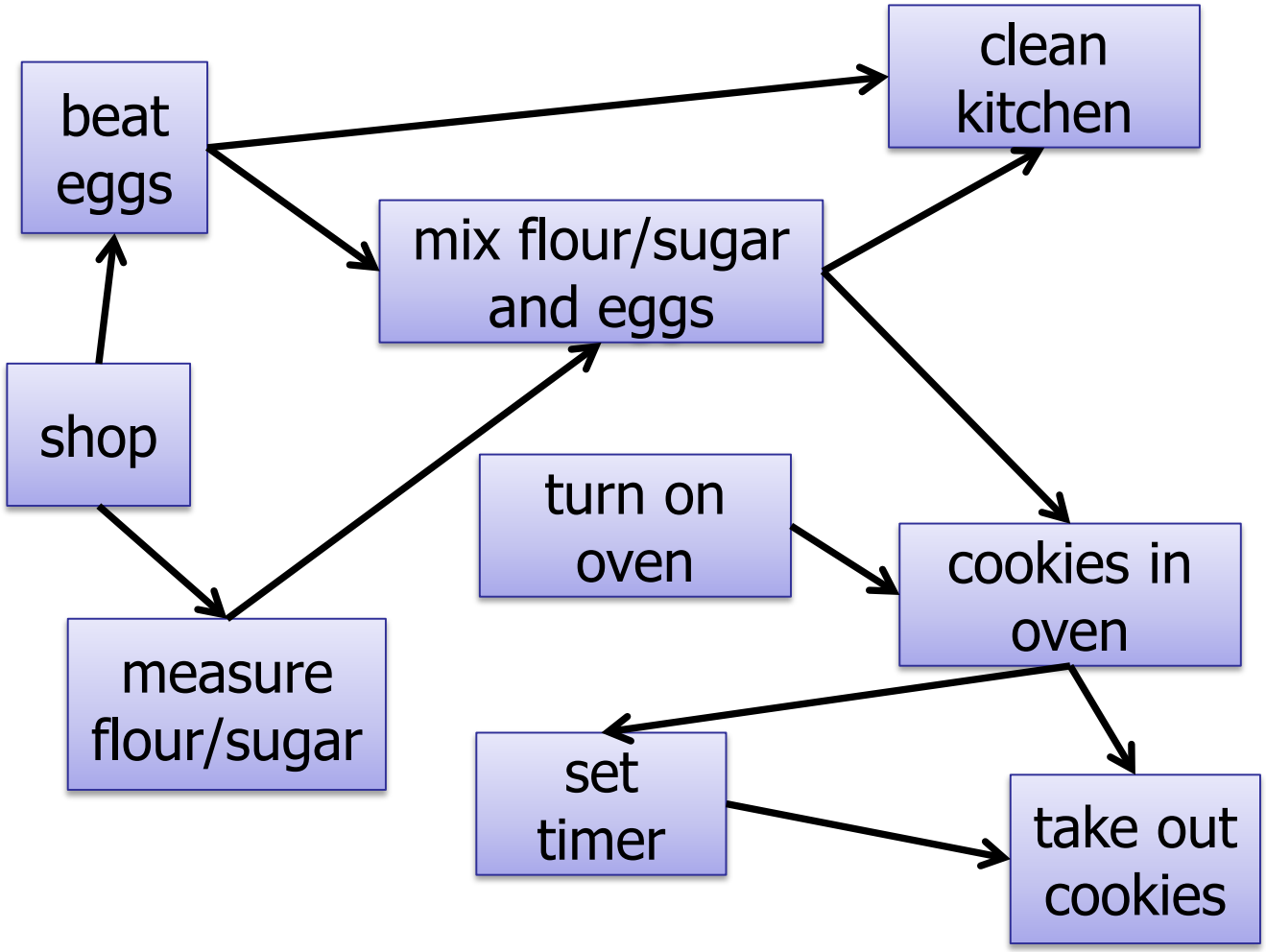


Directed Acyclic Graphs

Cyclic or Acyclic?



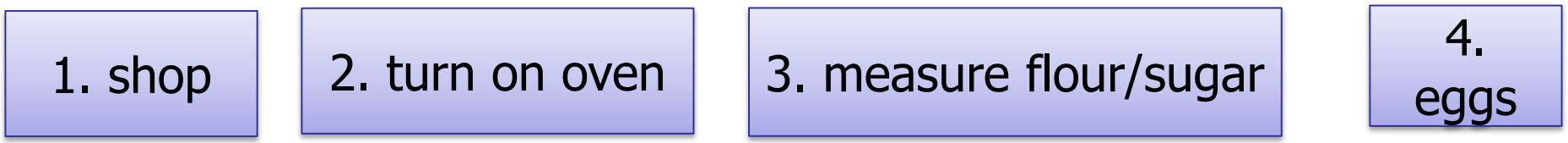
Directed Acyclic Graph (DAG)



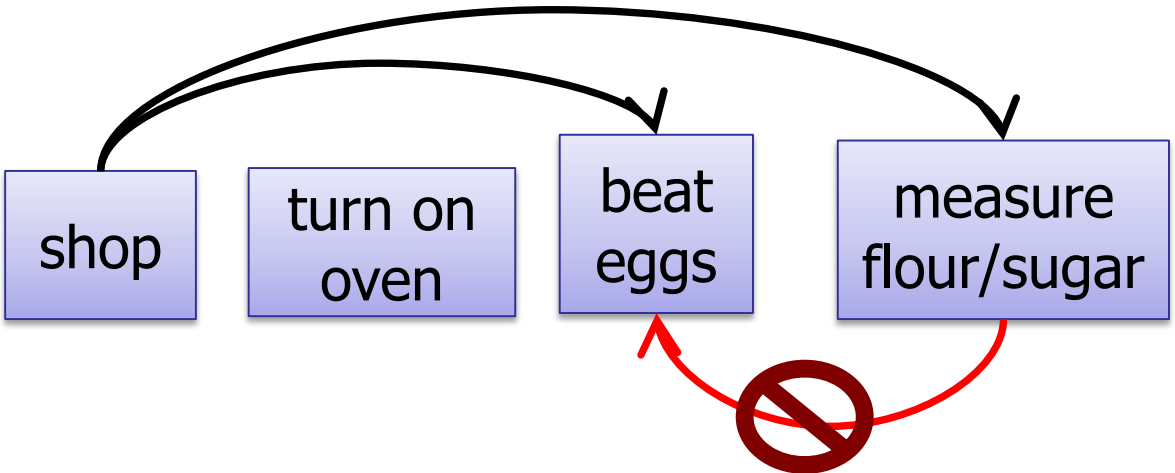
Topological Order

Properties:

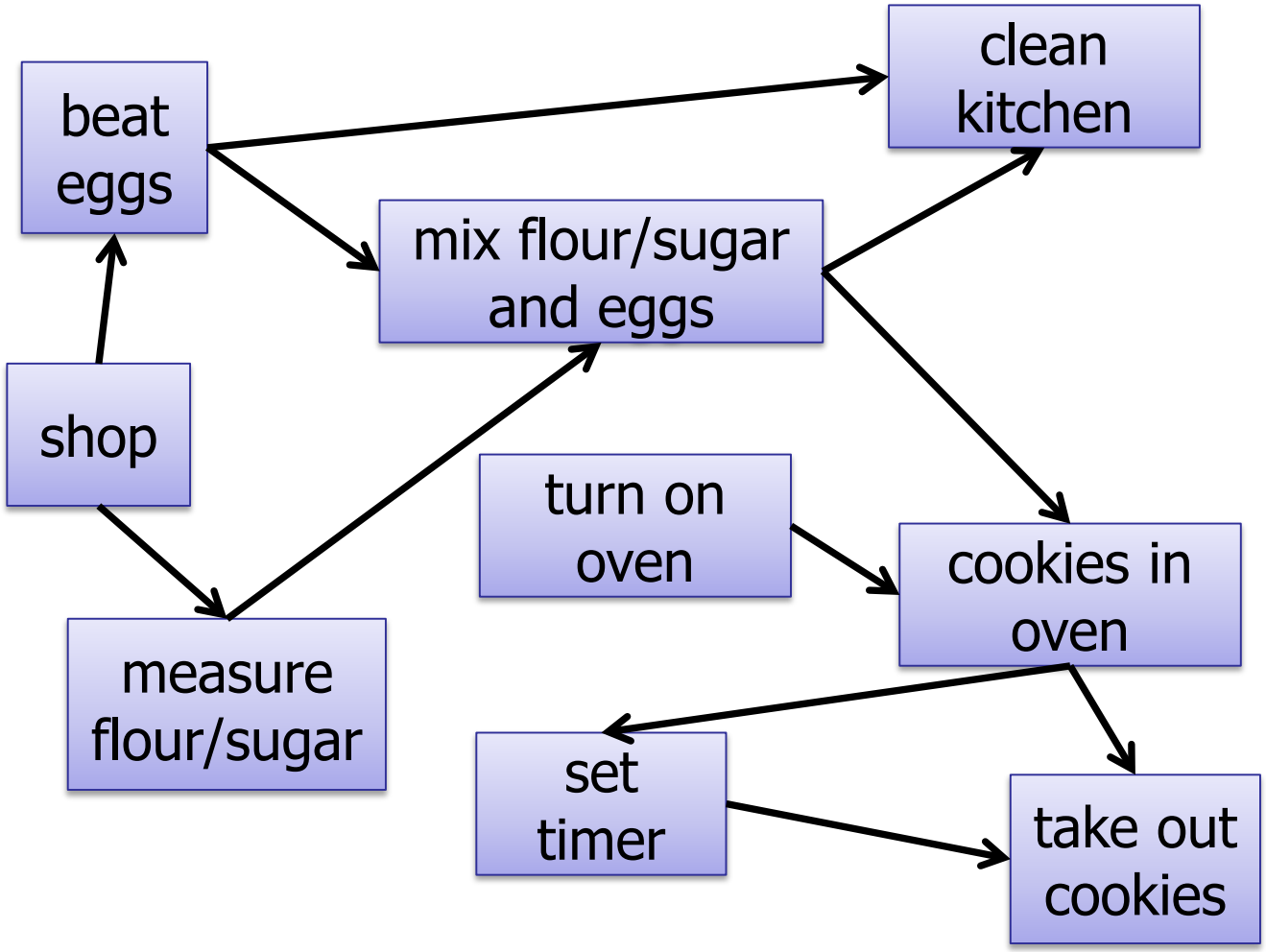
1. Sequential total ordering of all nodes



2. Edges only point forward

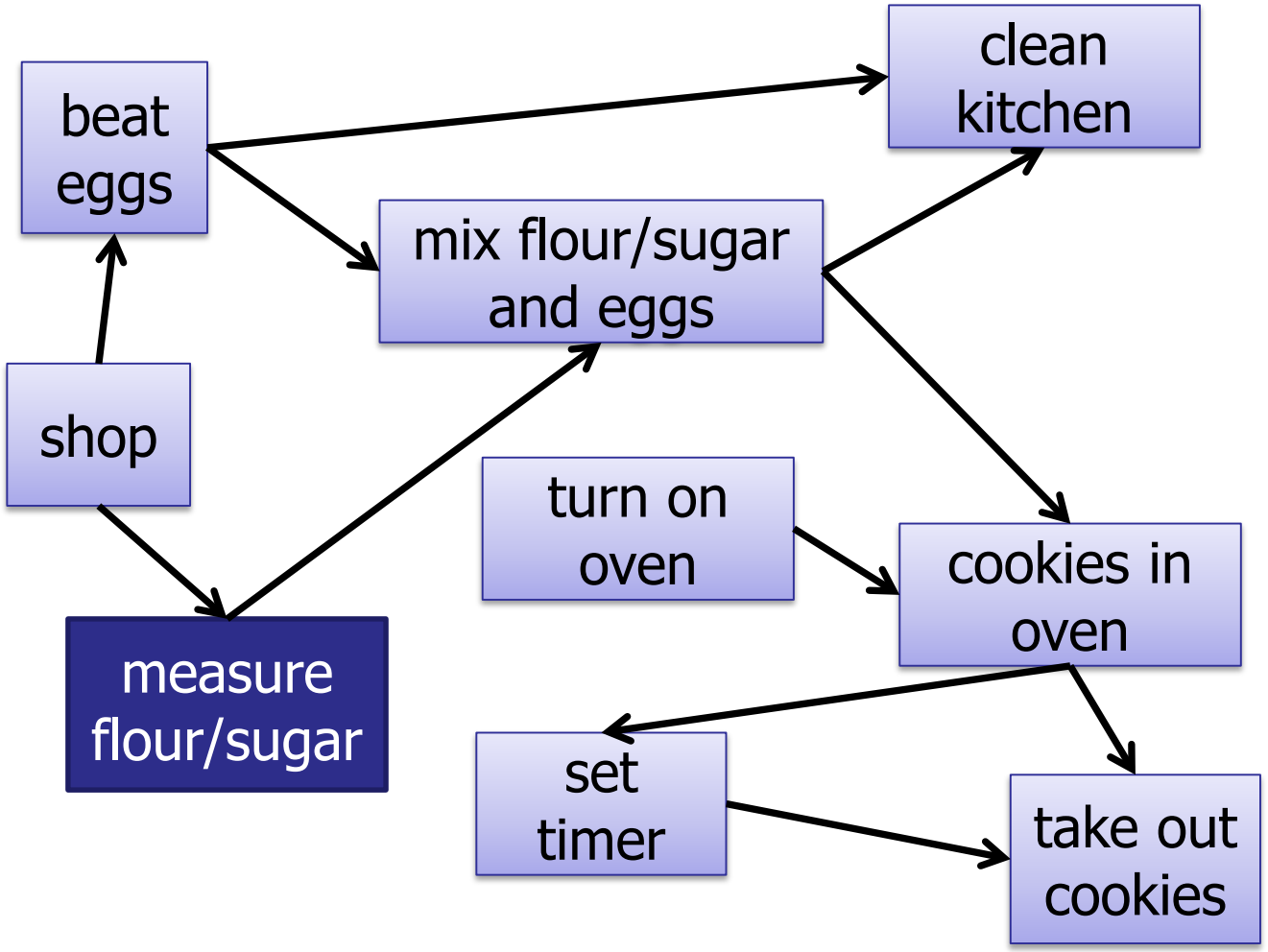


Depth-First Search (First Try)



Depth-First Search

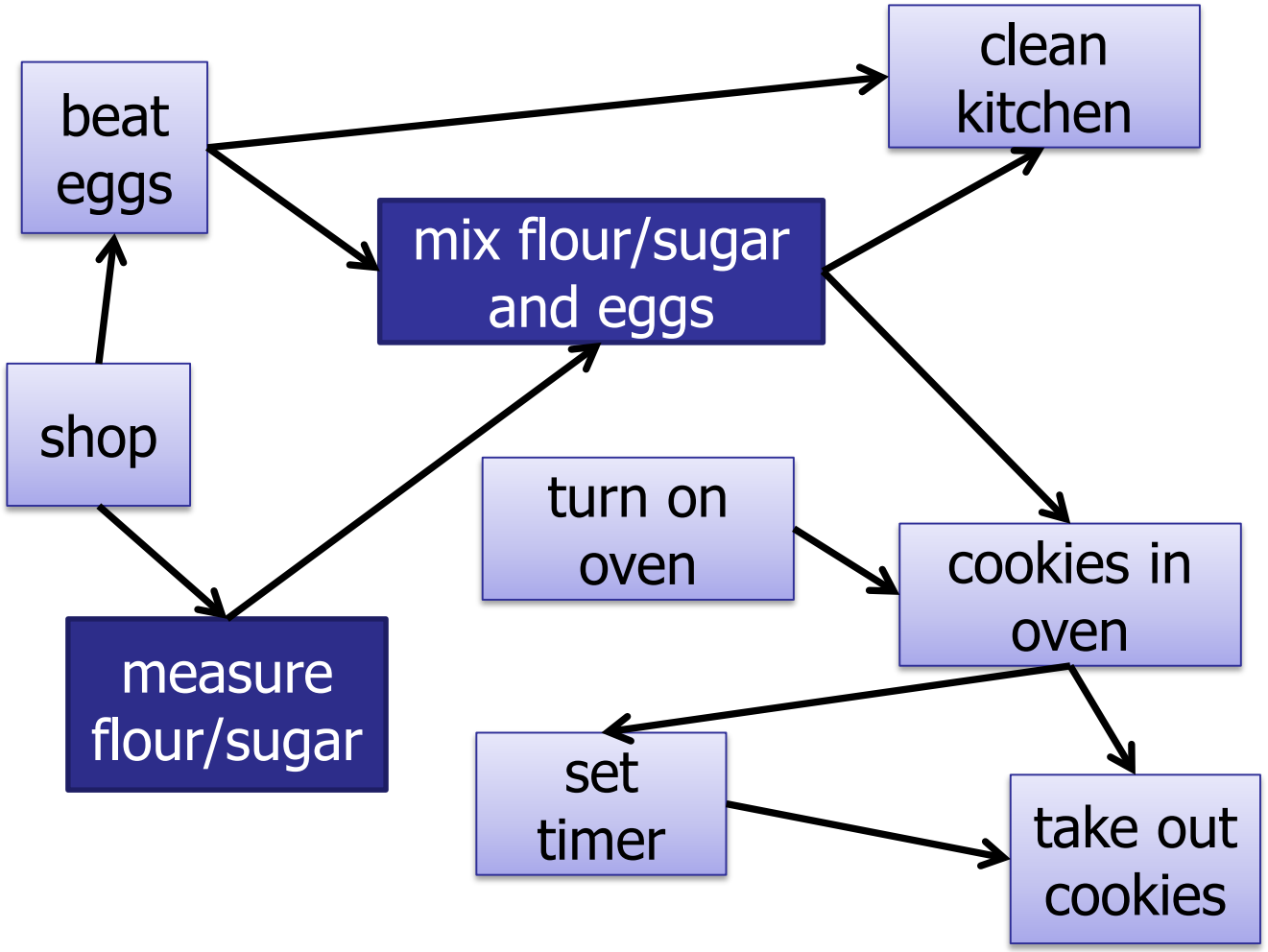
1. measure



Deep Blue: First Visit
Black: Finished Visit

Depth-First Search

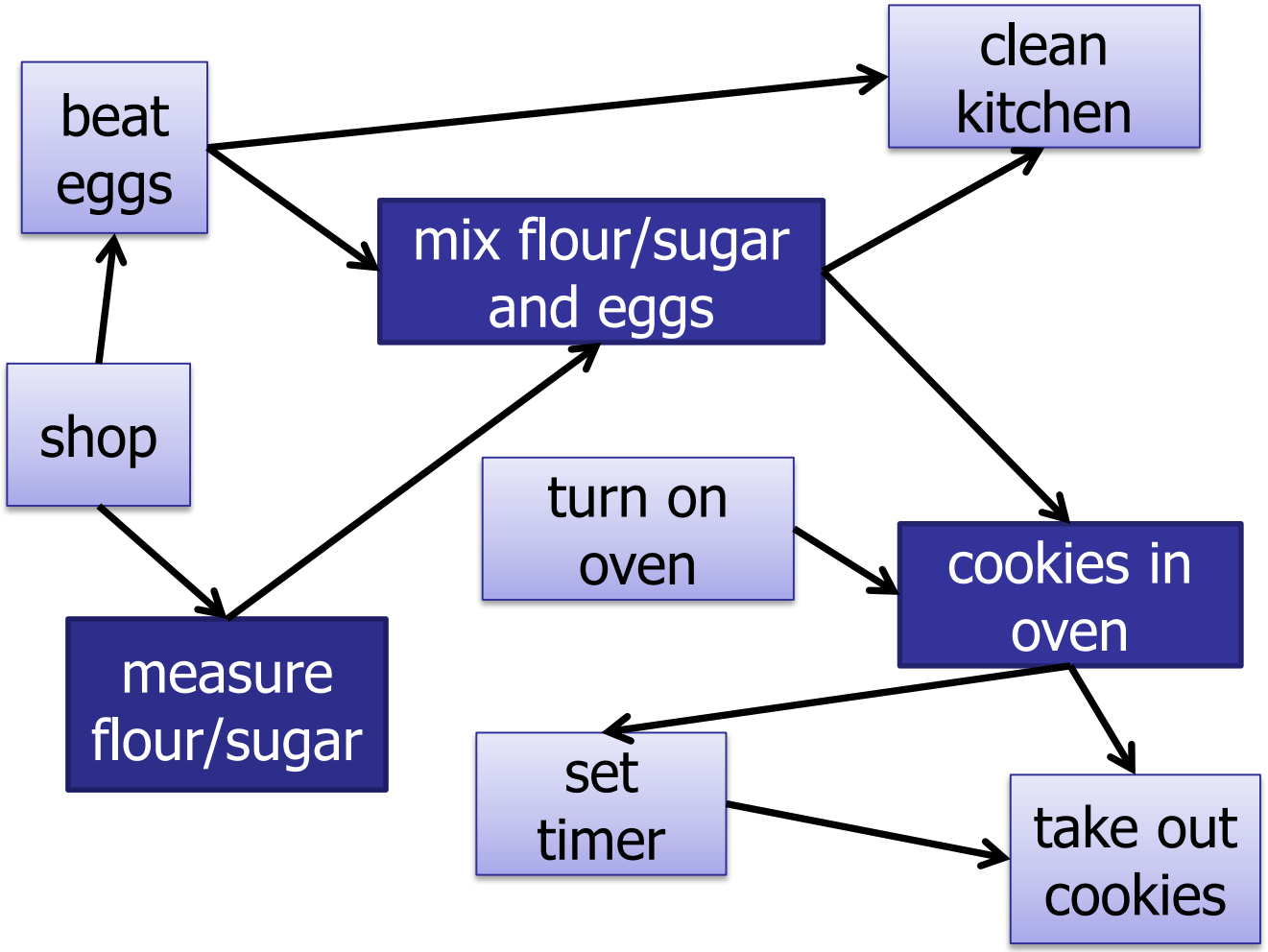
1. measure
2. mix



Deep Blue: First Visit
Black: Finished Visit

Depth-First Search

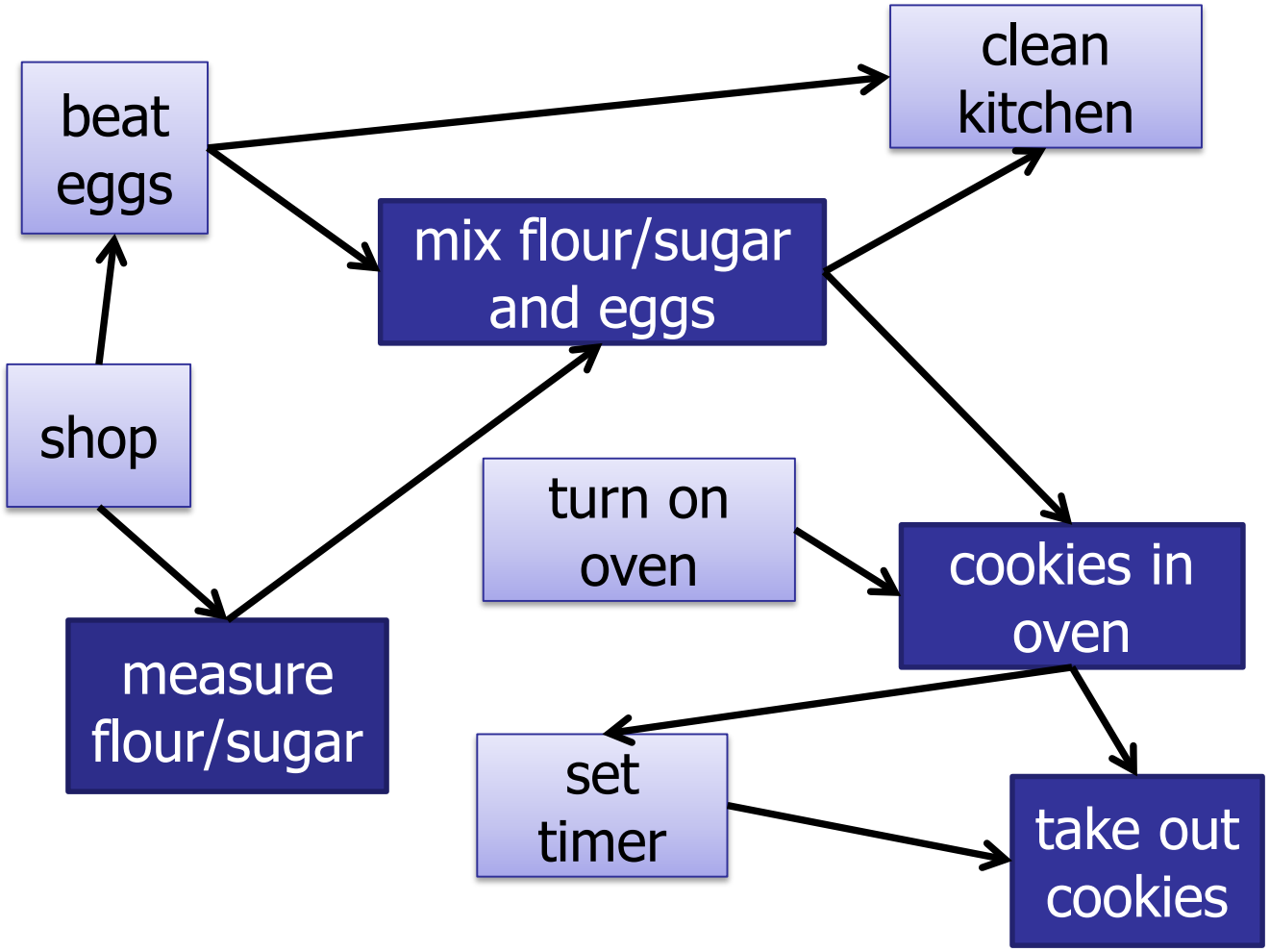
1. measure
2. mix
3. in oven



Deep Blue: First Visit
Black: Finished Visit

Depth-First Search

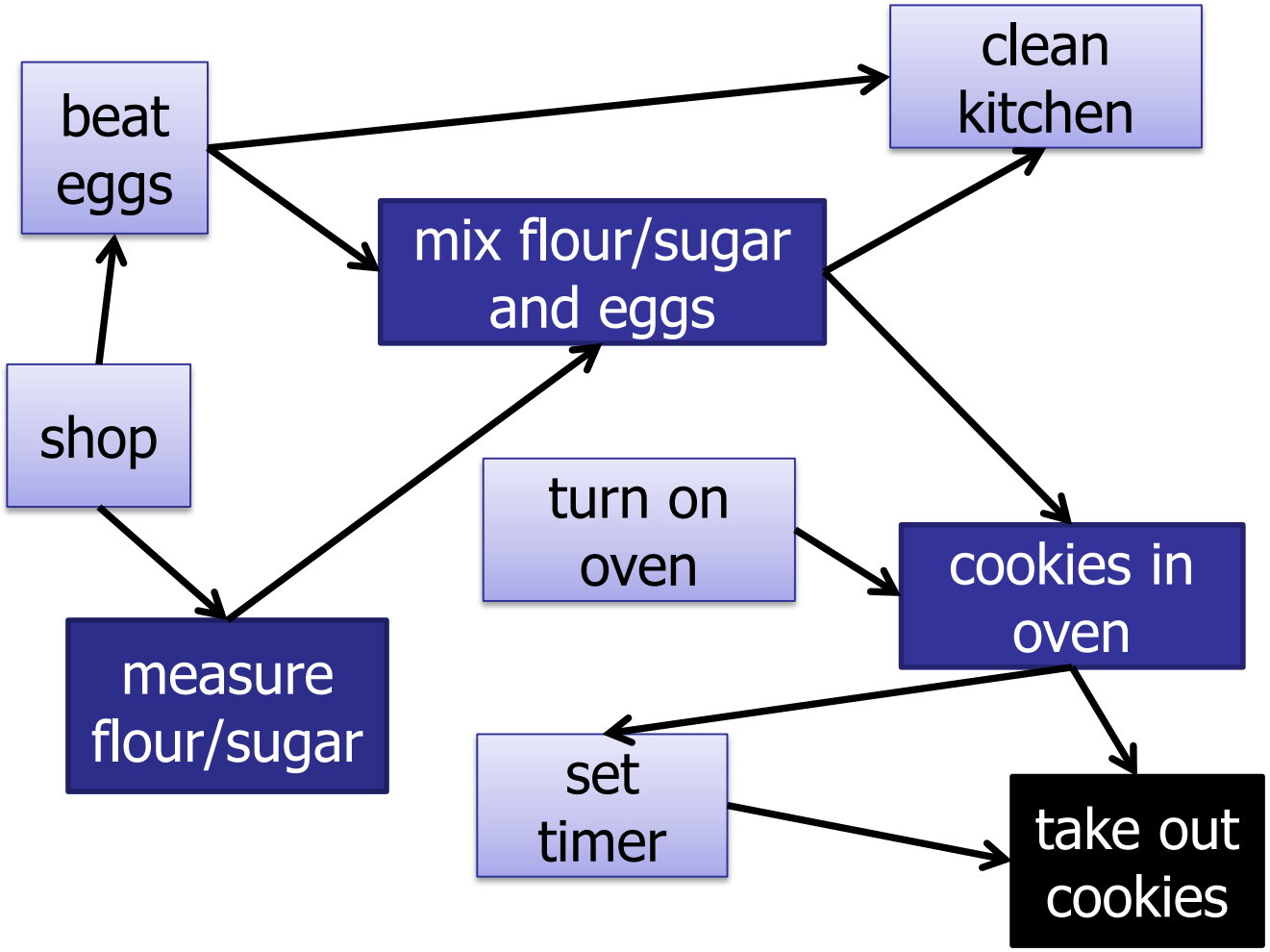
1. measure
2. mix
3. in oven
4. take out



Deep Blue: First Visit
Black: Finished Visit

Depth-First Search

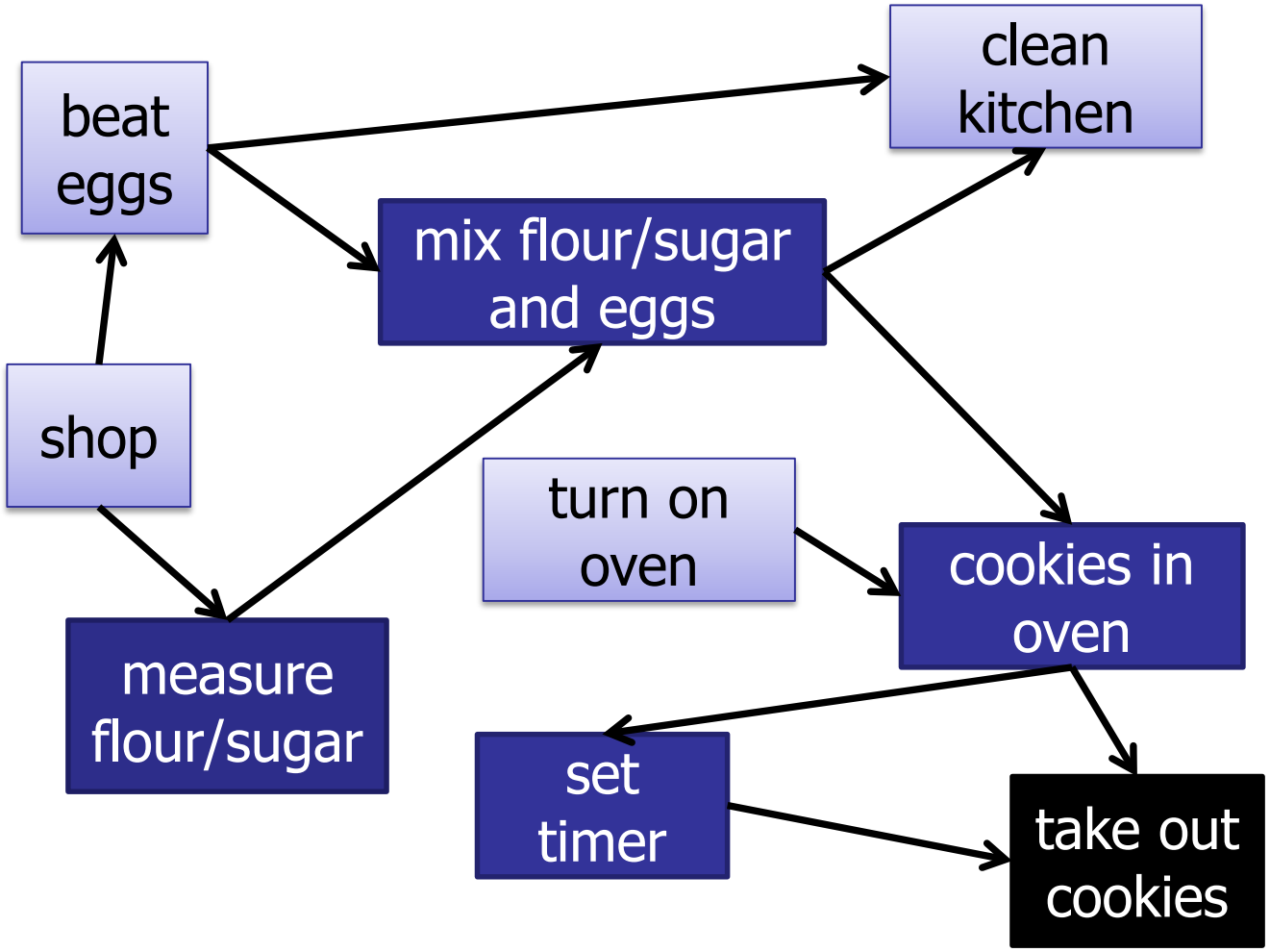
1. measure
2. mix
3. in oven
4. take out



Deep Blue: First Visit
Black: Finished Visit

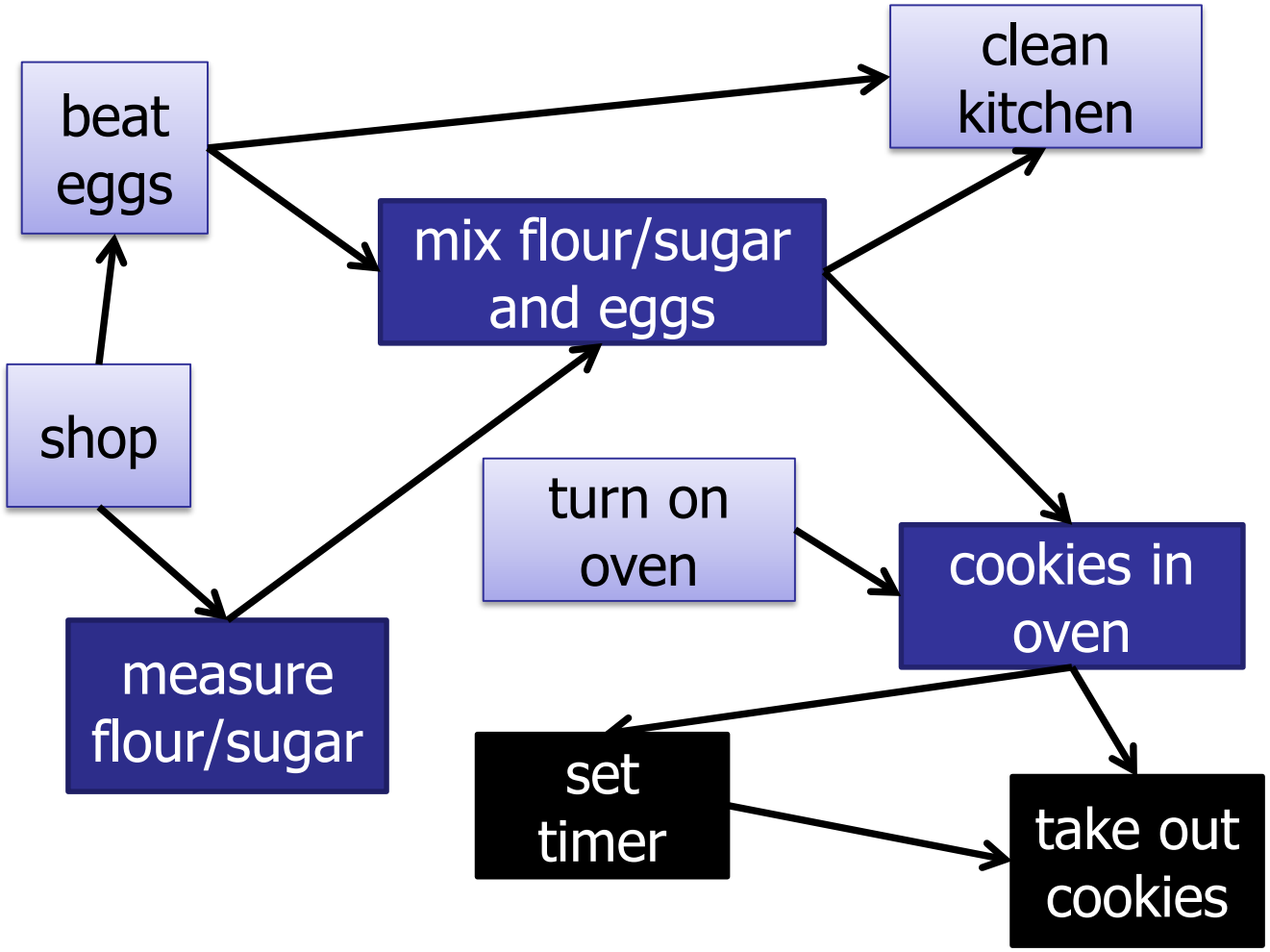
Depth-First Search

1. measure
2. mix
3. in oven
4. take out
5. set timer



Depth-First Search

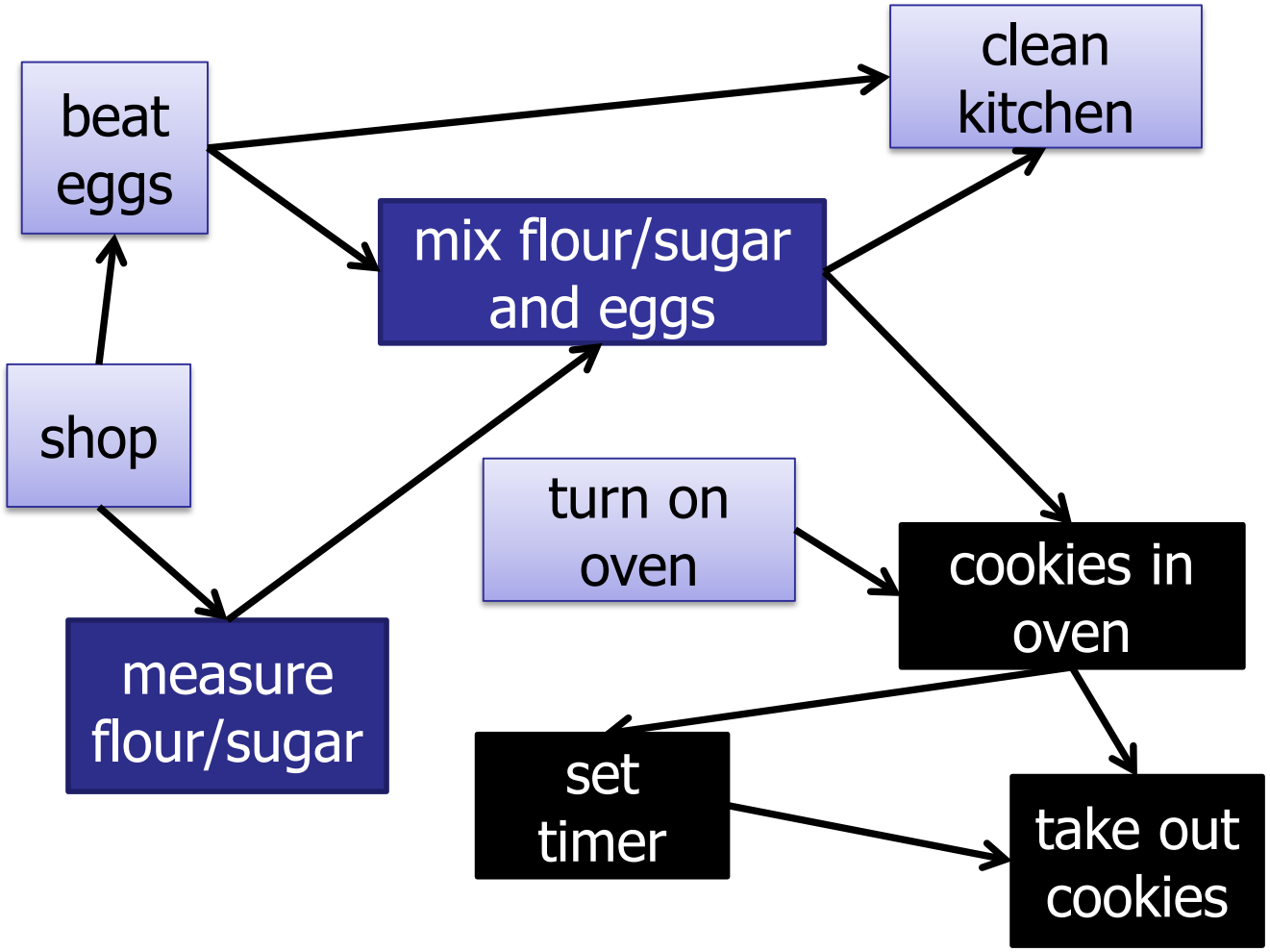
1. measure
2. mix
3. in oven
4. take out
5. set timer



Deep Blue: First Visit
Black: Finished Visit

Depth-First Search

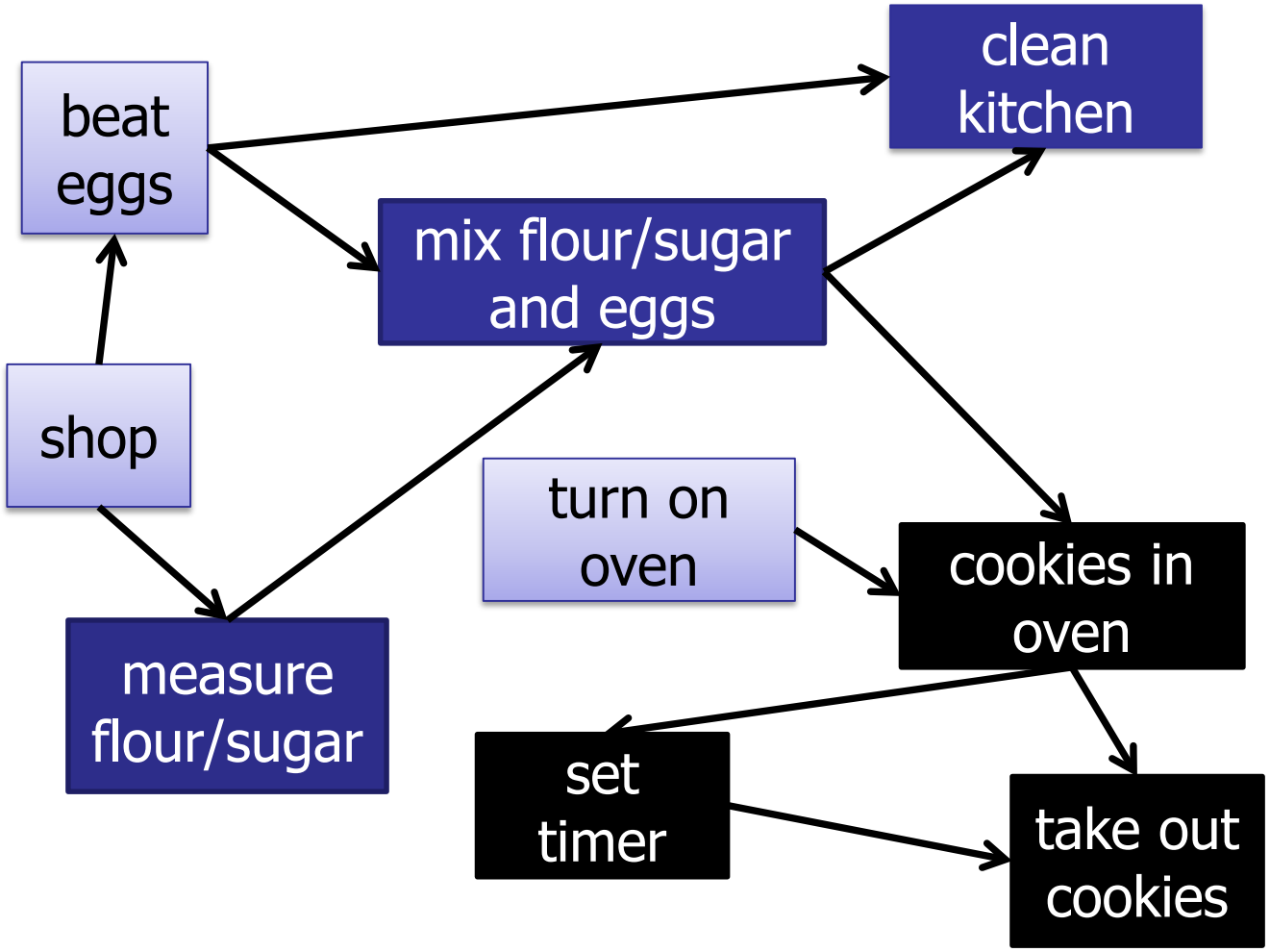
1. measure
2. mix
3. in oven
4. take out
5. set timer



Deep Blue: First Visit
Black: Finished Visit

Depth-First Search

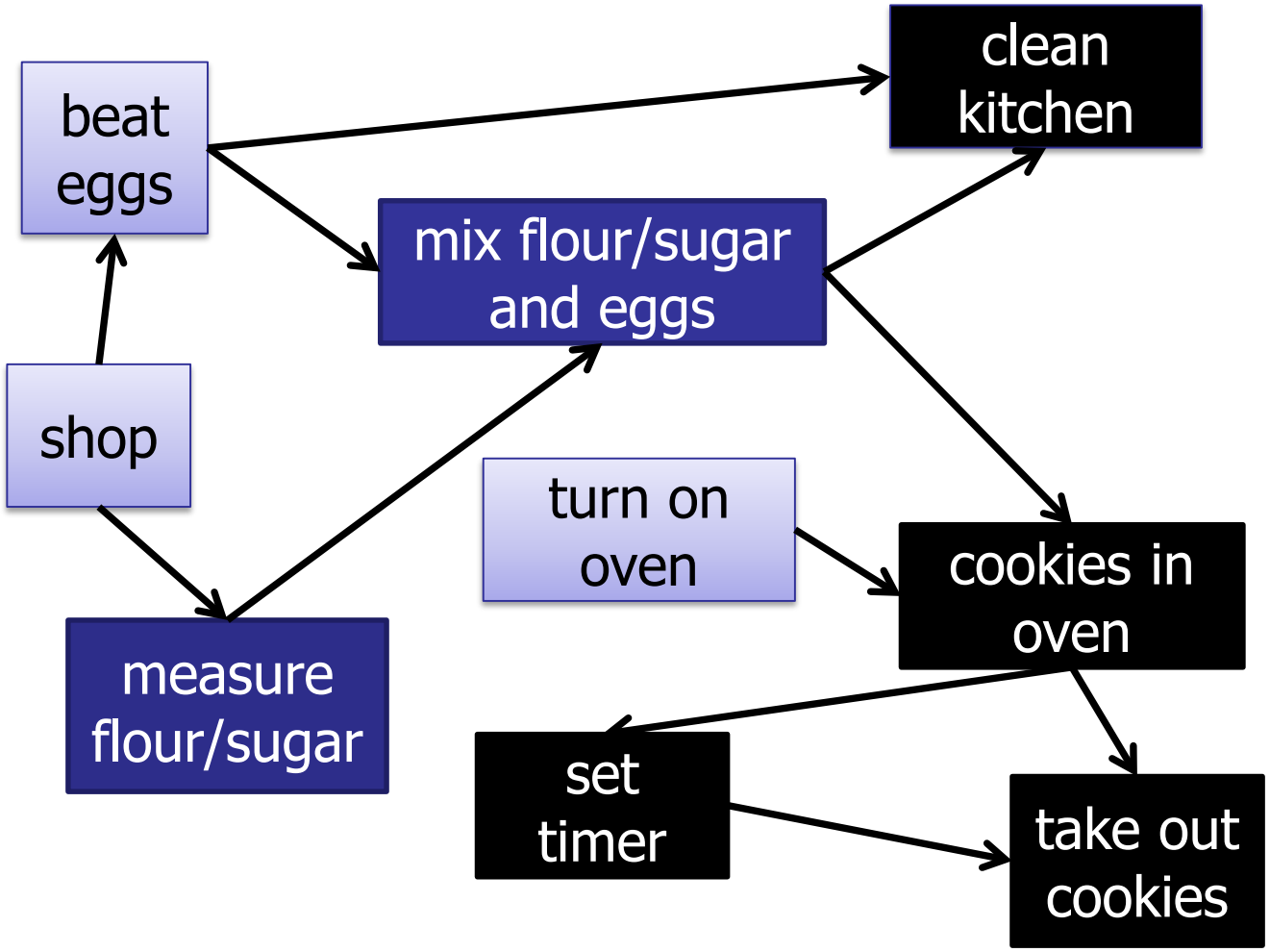
1. measure
2. mix
3. in oven
4. take out
5. set timer
6. clean



Deep Blue: First Visit
Black: Finished Visit

Depth-First Search

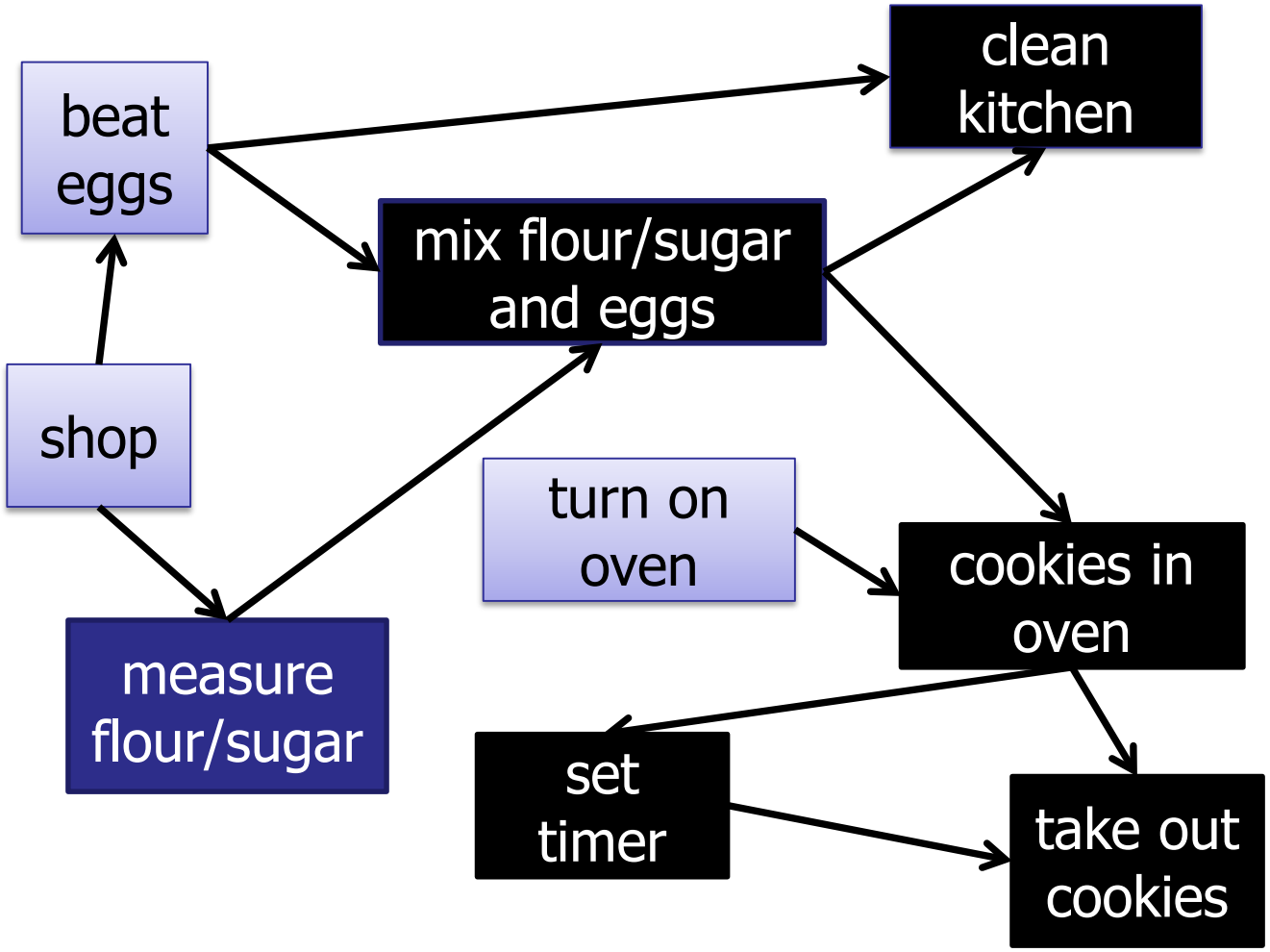
1. measure
2. mix
3. in oven
4. take out
5. set timer
6. clean



Deep Blue: First Visit
Black: Finished Visit

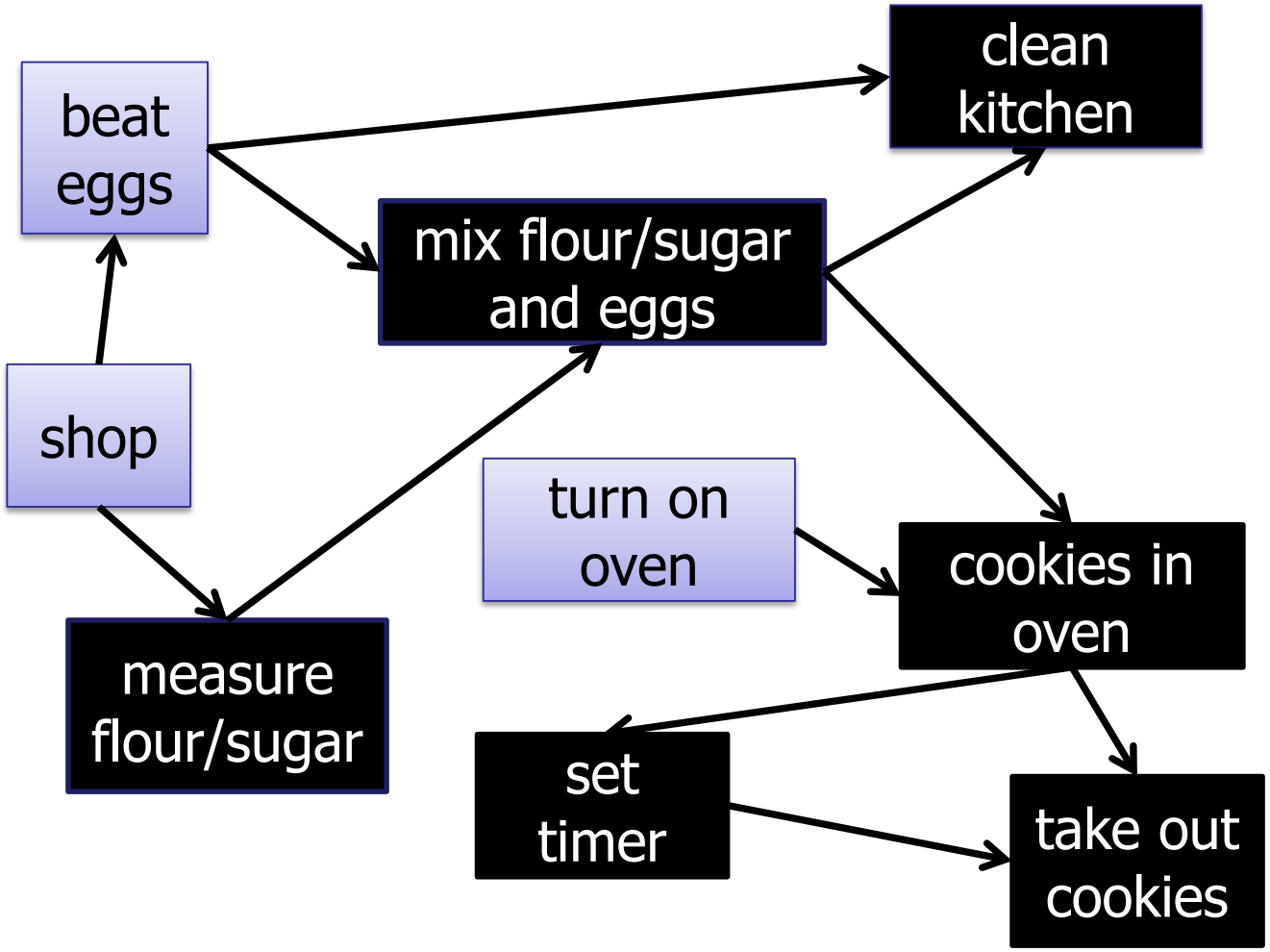
Depth-First Search

1. measure
2. mix
3. in oven
4. take out
5. set timer
6. clean



Depth-First Search

1. measure
2. mix
3. in oven
4. take out
5. set timer
6. clean



Deep Blue: First Visit
Black: Finished Visit

Searching a (Directed) Graph

Pre-Order Depth-First Search:

- Process each node when it is *first* visited.

Searching a (Directed) Graph

Pre-Order Depth-First Search:

- Process each node when it is *first* visited.

Post-Order Depth-First Search:

- Process each node when it is *last* visited.
- Or, when all the neighbors are visited
- Or, when it is finished

Depth-First Search

```
DFS-visit(Node[] nodeList, boolean[] visited, int startId){
  for (every neighbor v of startId) {
     if (!visited[v]){
           visited[v] = true;
           ProcessNode(v);
           DFS-visit(nodeList, visited, v);
     }
  }
}
```

Depth-First Search

```
DFS-visit(Node[] nodeList, boolean[] visited, int startId){
  for (every neighbor v of startId) {
     if (!visited[v]){
           visited[v] = true;
           DFS-visit(nodeList, visited, v);
           ProcessNode(v);
     }
  }
}
```

Searching a (Directed) Graph

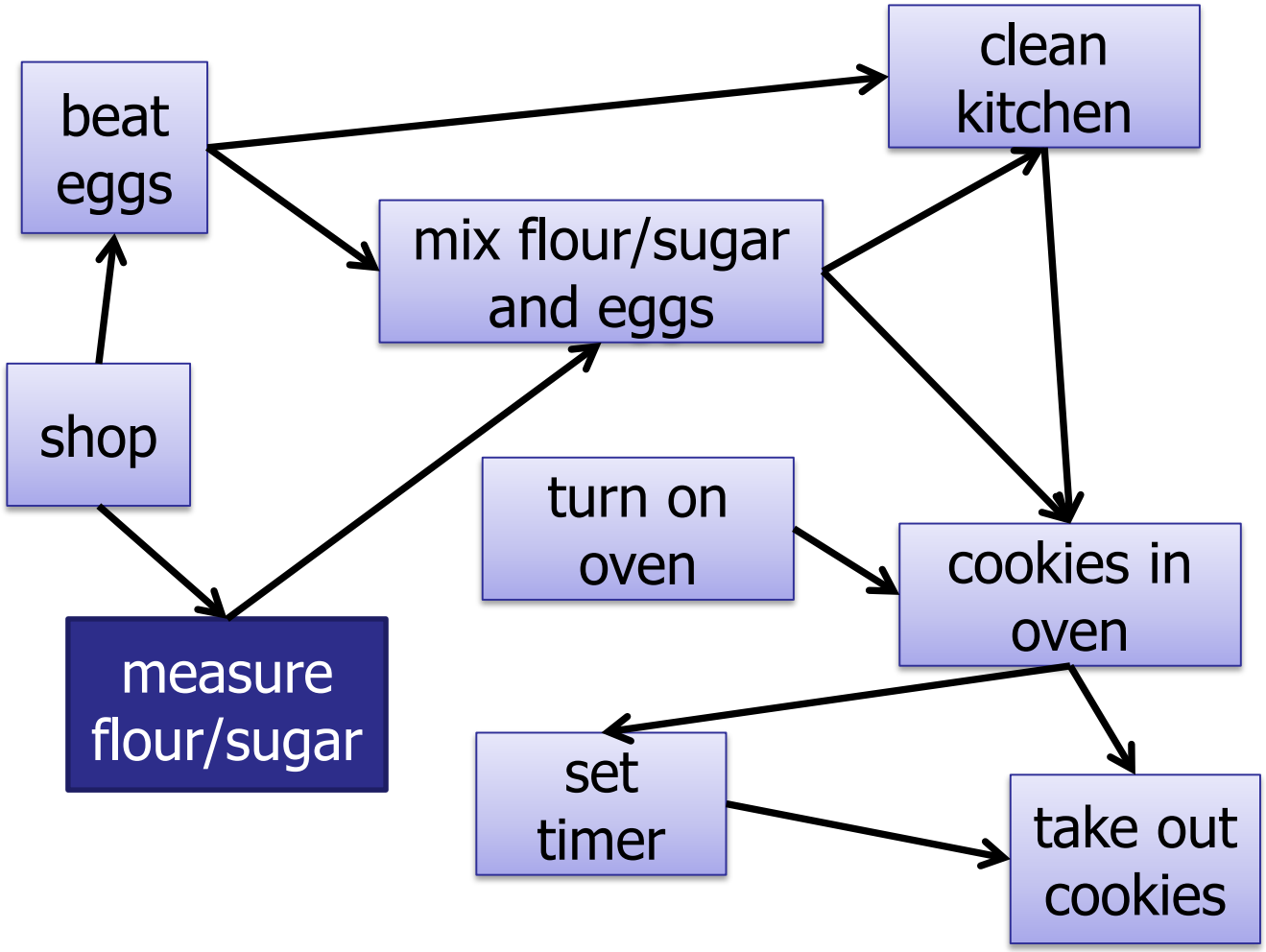
Pre-Order Depth-First Search:

- Process each node when it is *first* visited.

Post-Order Depth-First Search:

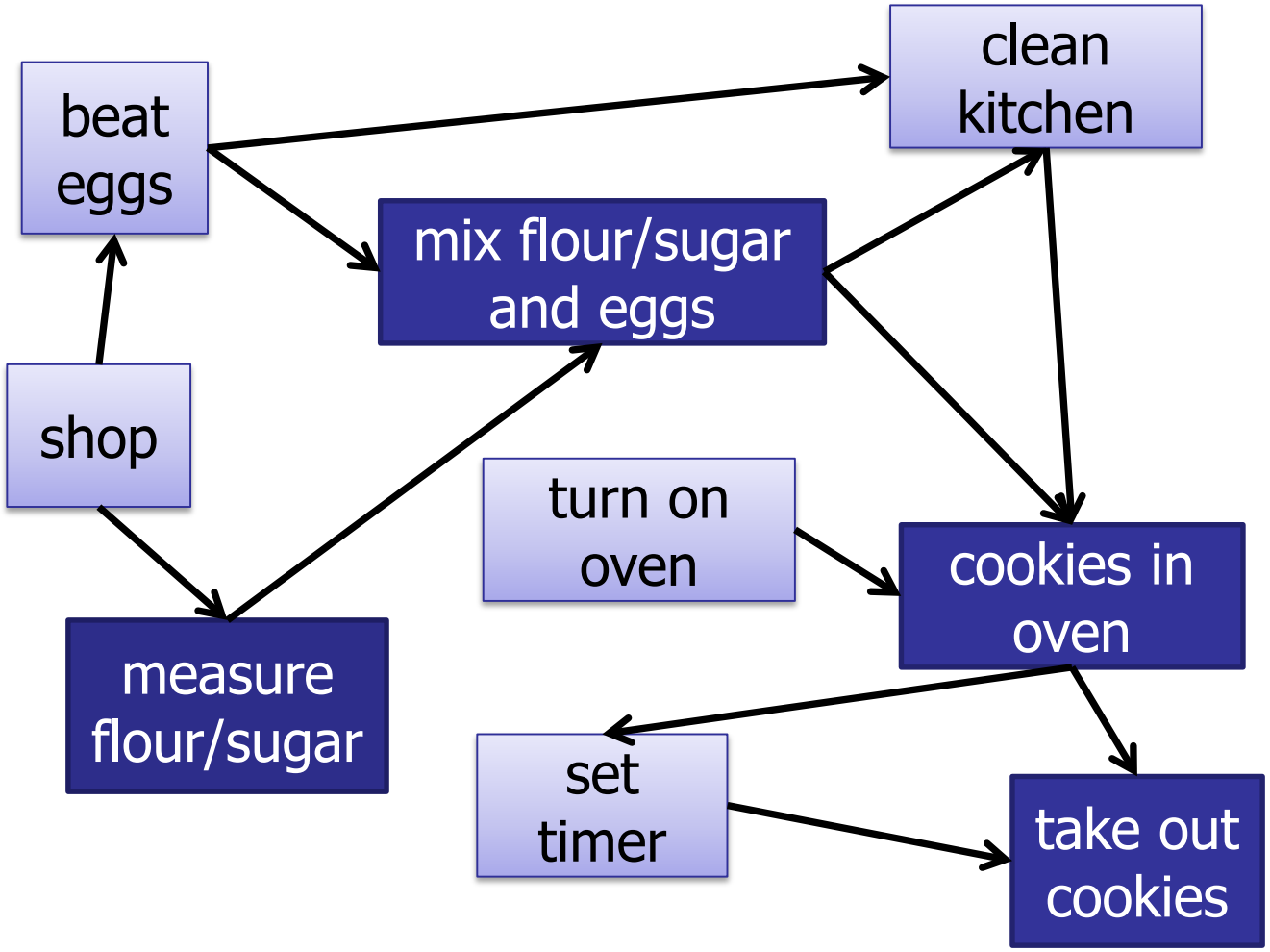
- Process each node when it is *last* visited.
- Or, when all the neighbors are visited
- Or, when it is finished

Post-Order Depth-First Search



Deep Blue: First Visit
Black: Finished Visit

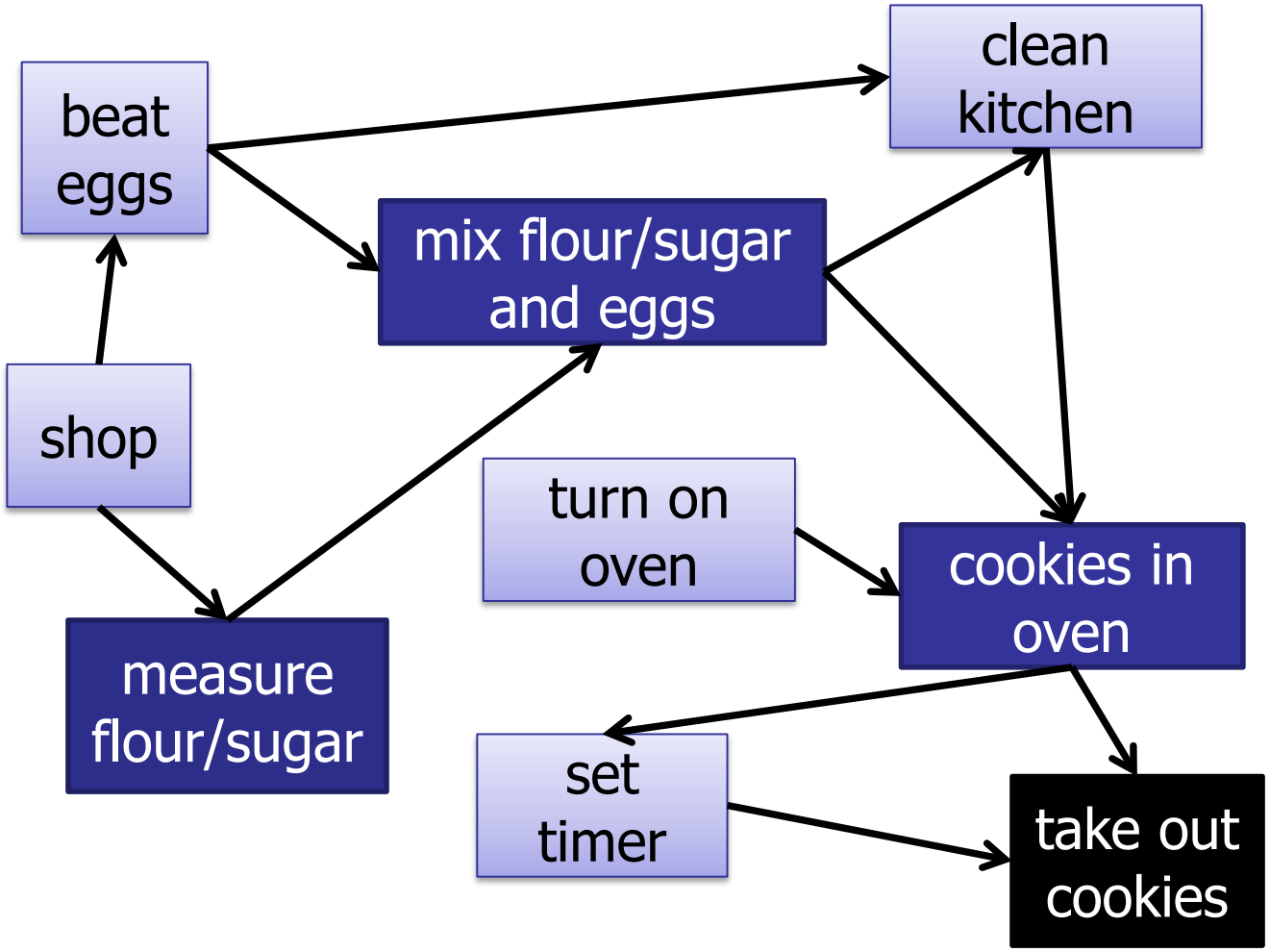
Post-Order Depth-First Search



Deep Blue: First Visit
Black: Finished Visit

Post-Order Depth-First Search

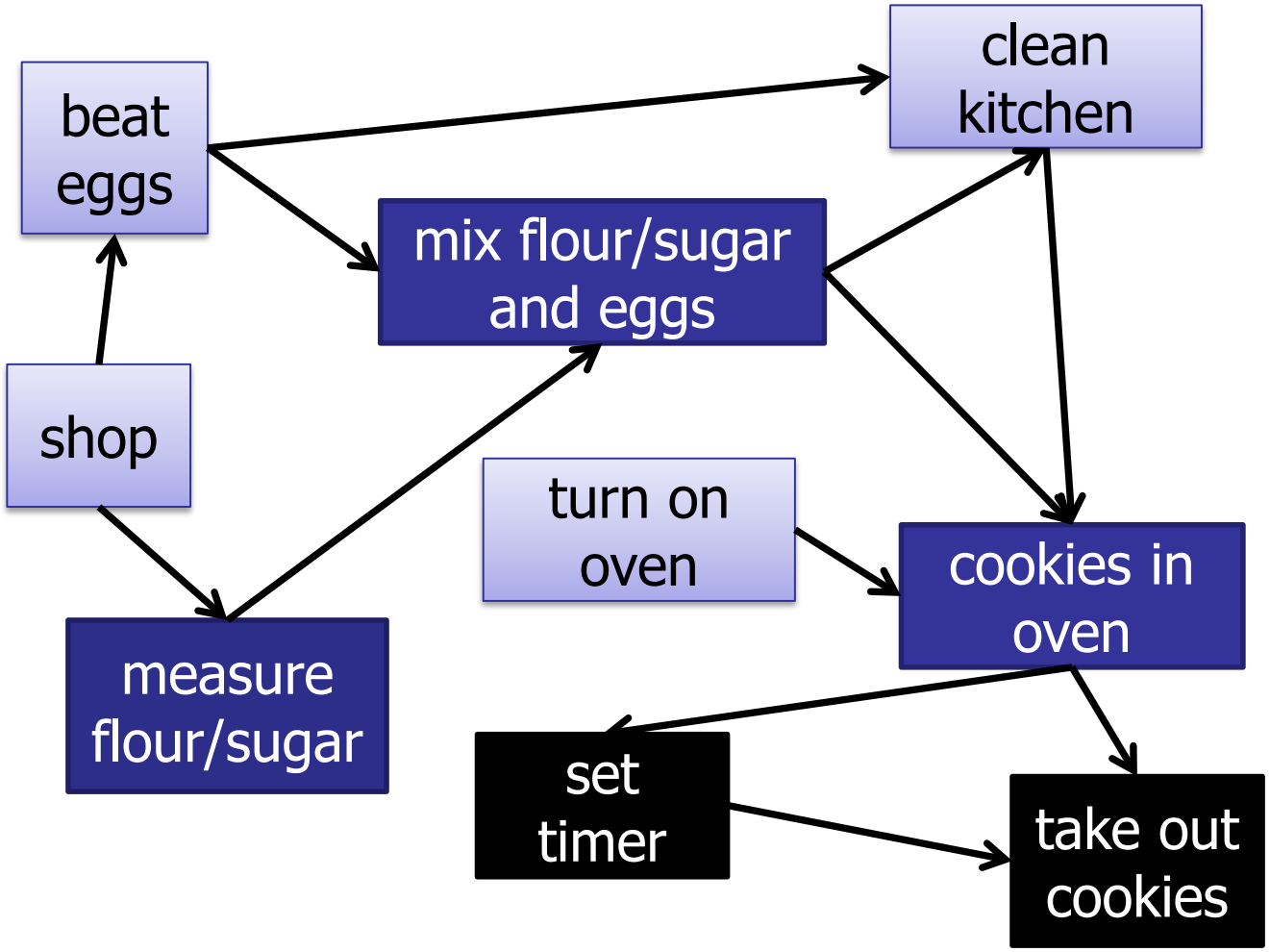
1.
2.
3.
4.
5.
6.
7.
8.
9. take out



Deep Blue: First Visit
Black: Finished Visit

Post-Order Depth-First Search

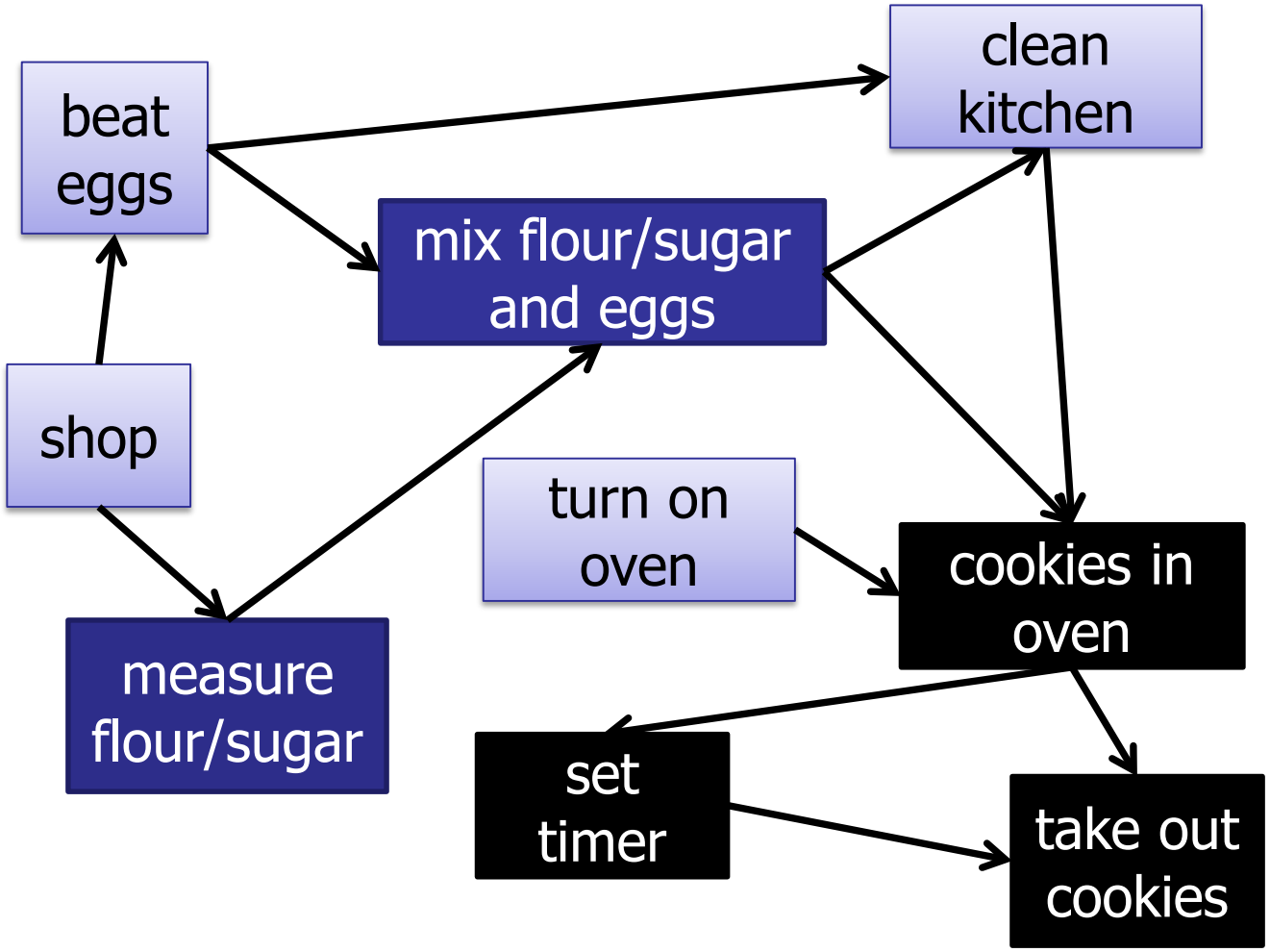
1.
2.
3.
4.
5.
6.
7.
8. set timer
9. take out



Deep Blue: First Visit
Black: Finished Visit

Post-Order Depth-First Search

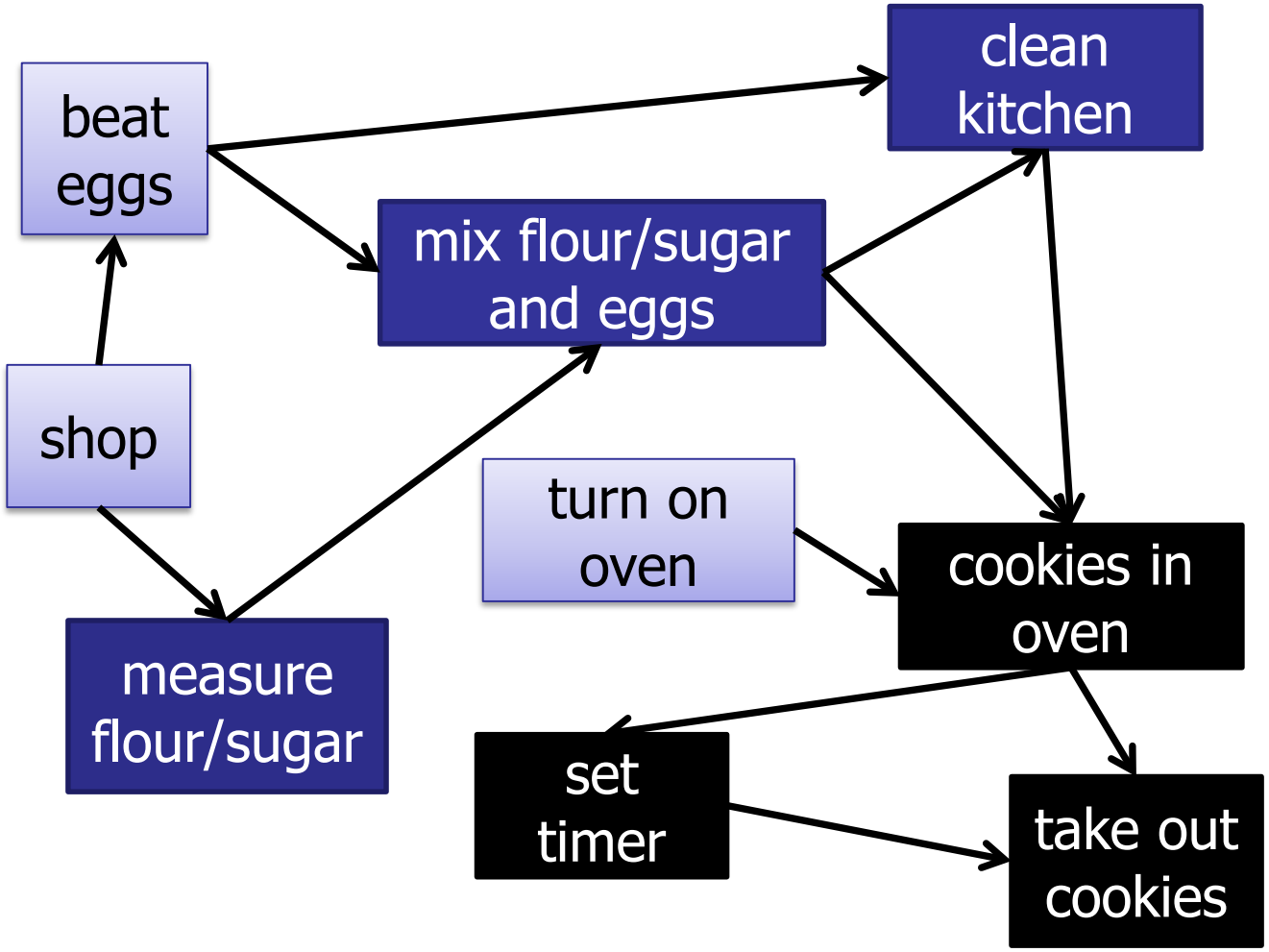
1.
2.
3.
4.
5.
6.
7. in oven
8. set timer
9. take out



Deep Blue: First Visit
Black: Finished Visit

Post-Order Depth-First Search

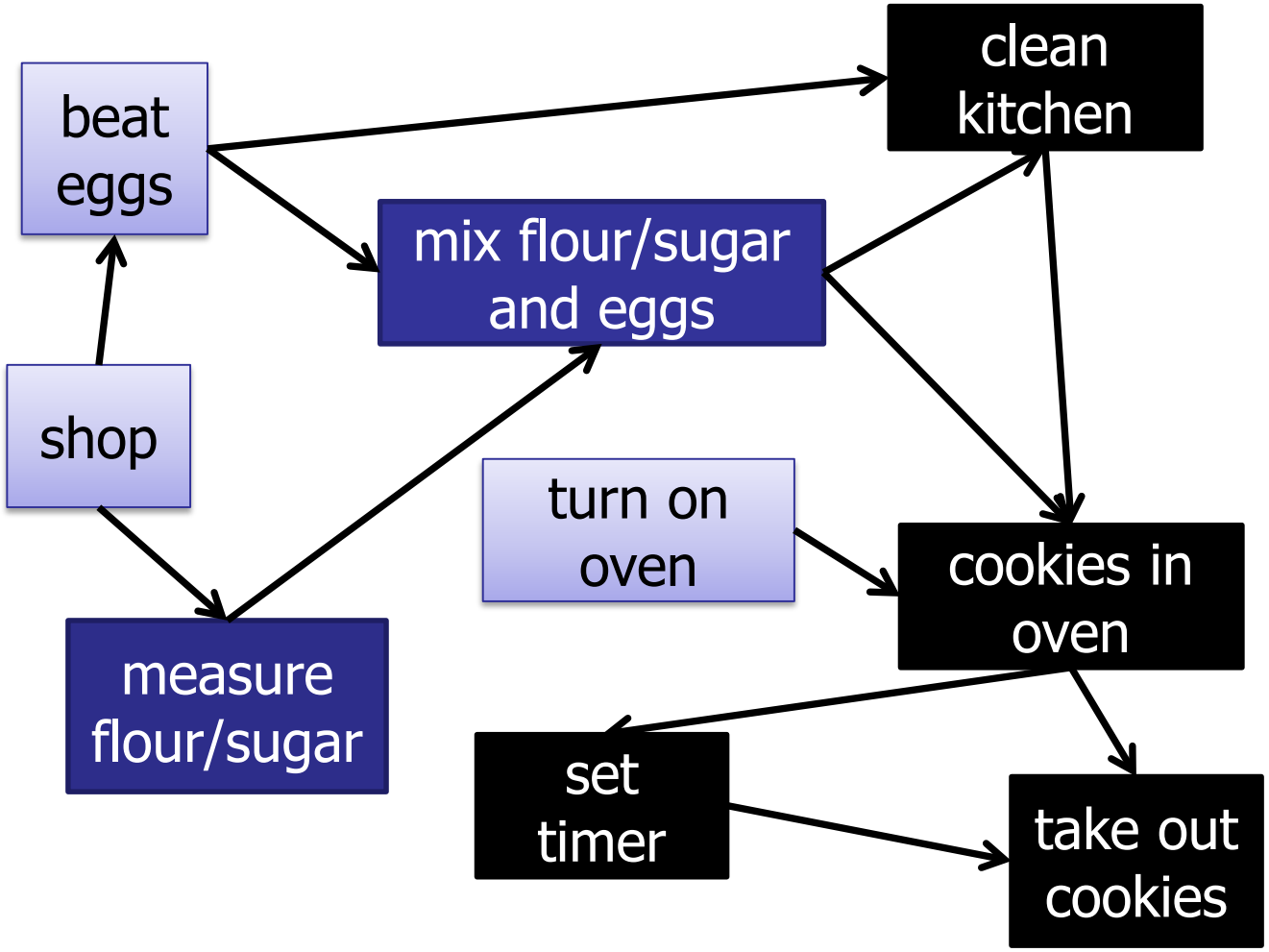
1.
2.
3.
4.
5.
6.
7. in oven
8. set timer
9. take out



Deep Blue: First Visit
Black: Finished Visit

Post-Order Depth-First Search

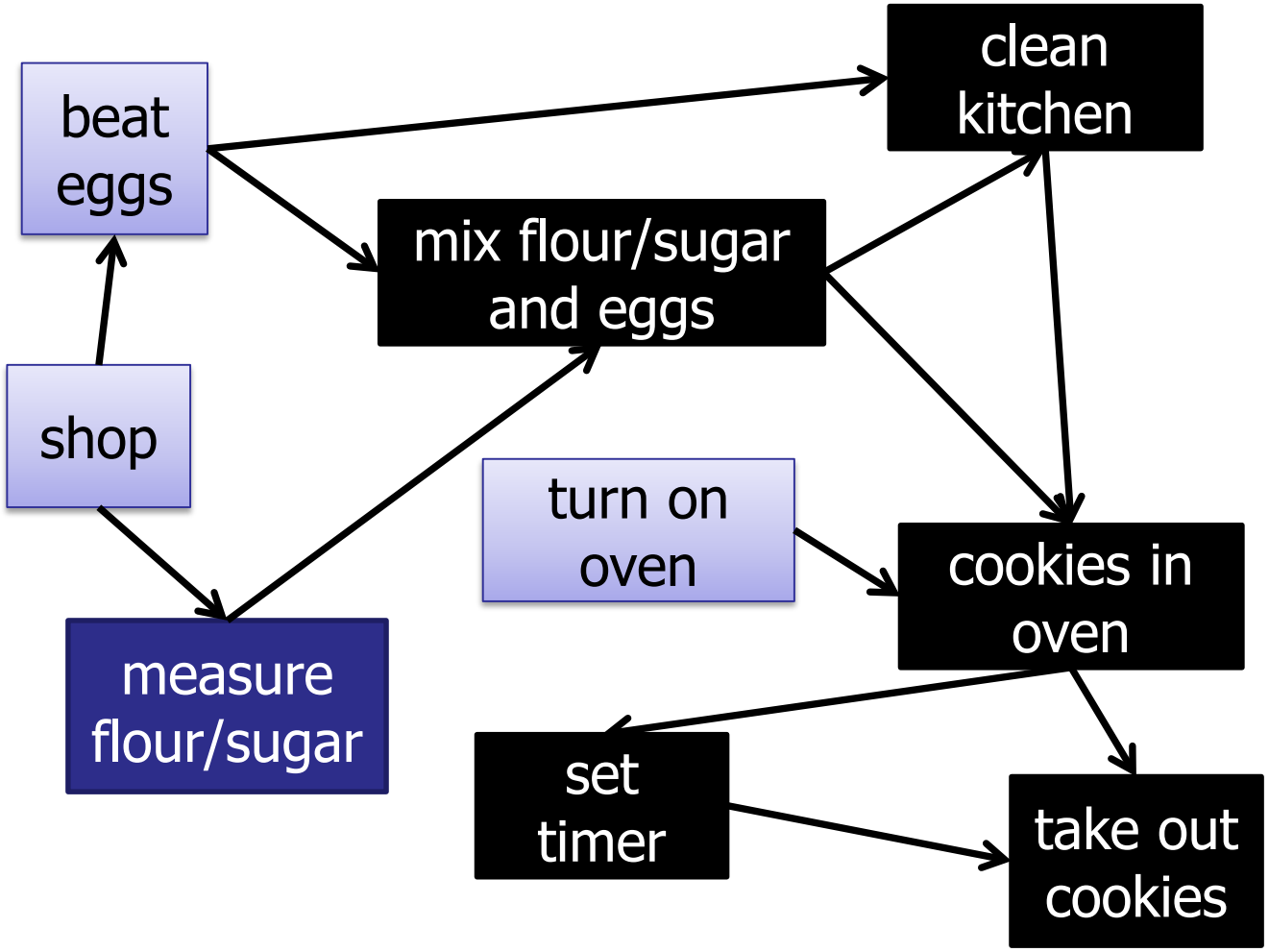
1.
2.
3.
4.
5.
6. clean
7. in oven
8. set timer
9. take out



Deep Blue: First Visit
Black: Finished Visit

Post-Order Depth-First Search

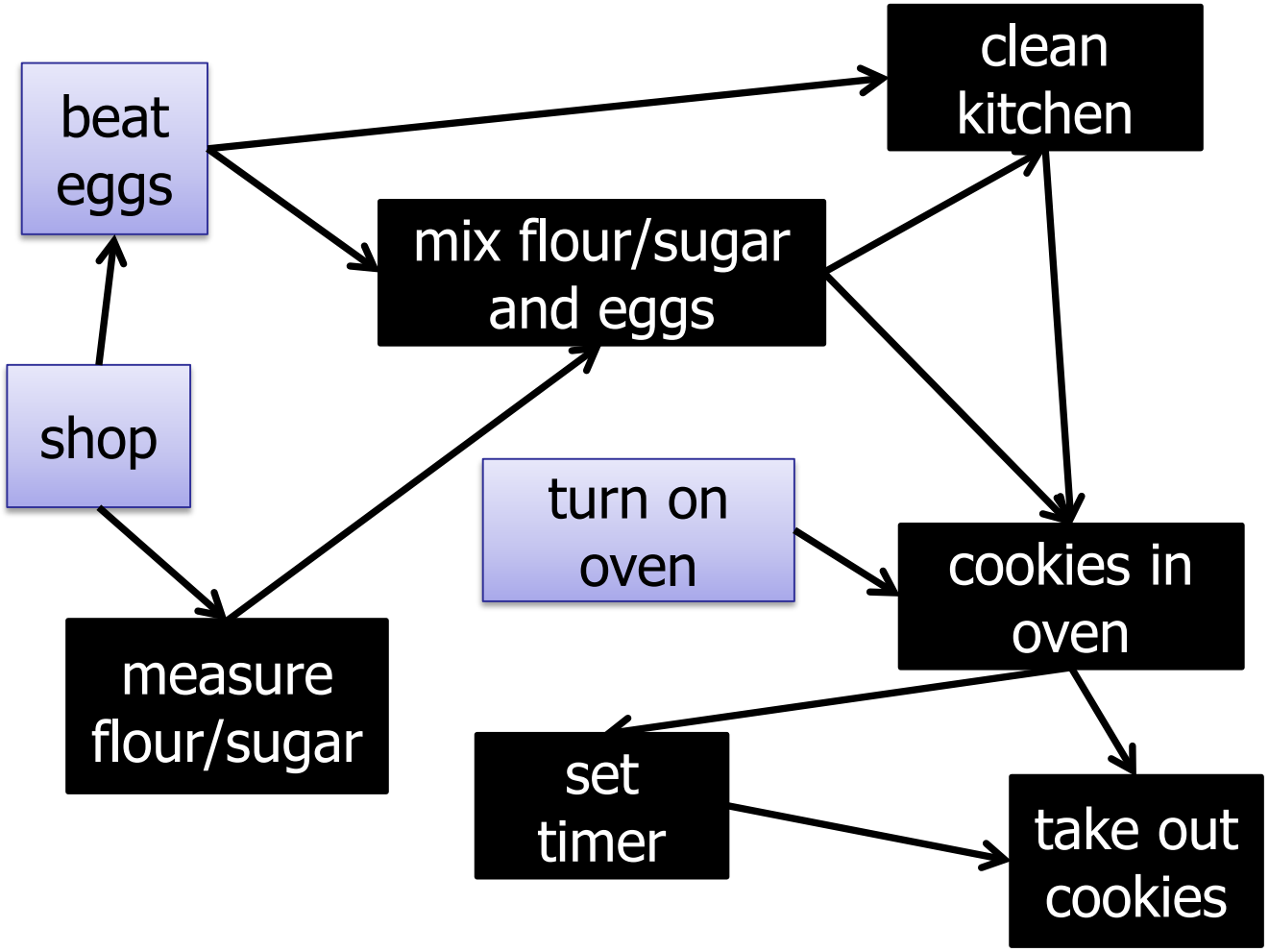
1.
2.
3.
4.
5. mix
6. clean
7. in oven
8. set timer
9. take out



Deep Blue: First Visit
Black: Finished Visit

Post-Order Depth-First Search

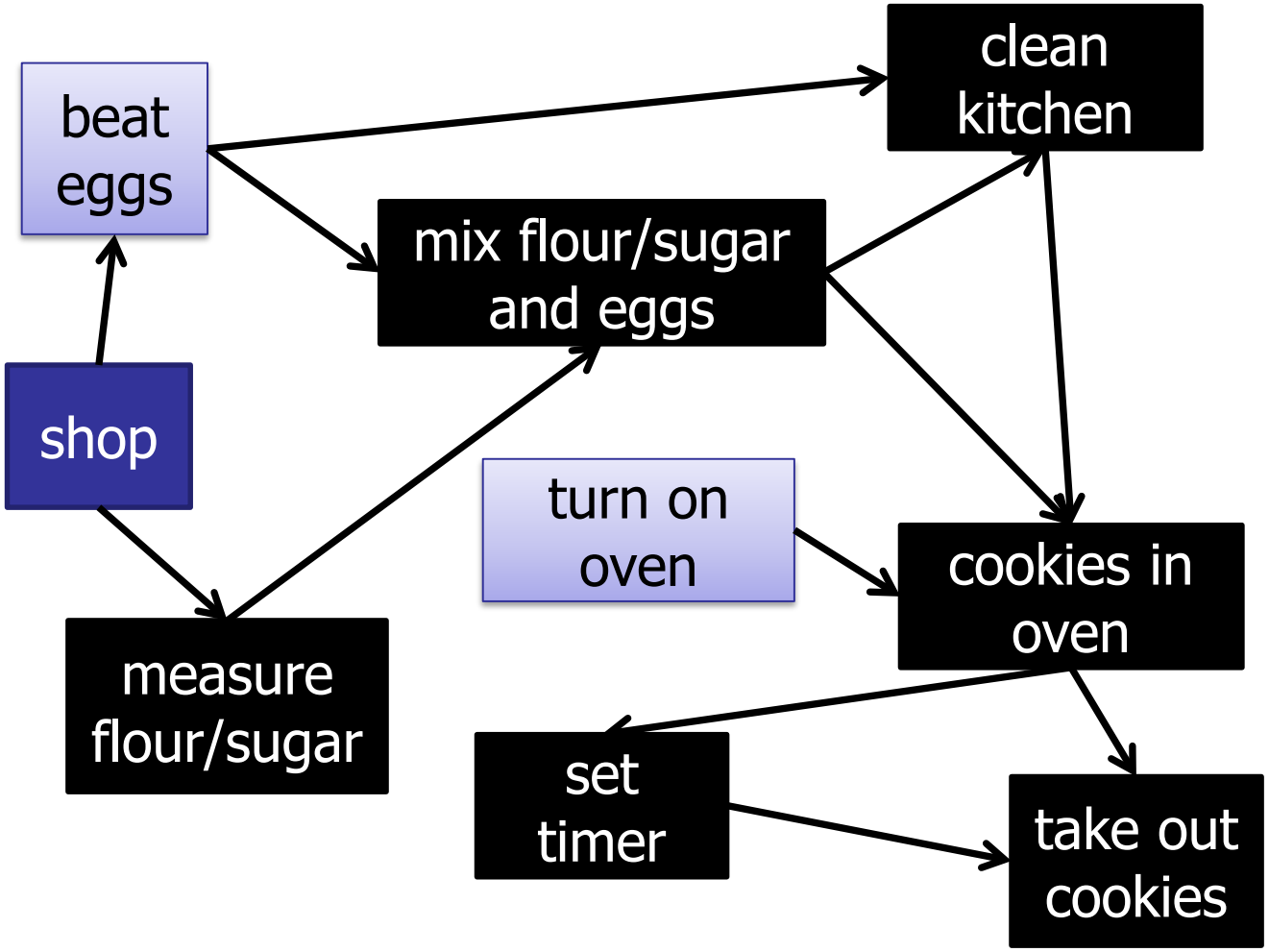
1.
2.
3.
4. measure
5. mix
6. clean
7. in oven
8. set timer
9. take out



Deep Blue: First Visit
Black: Finished Visit

Post-Order Depth-First Search

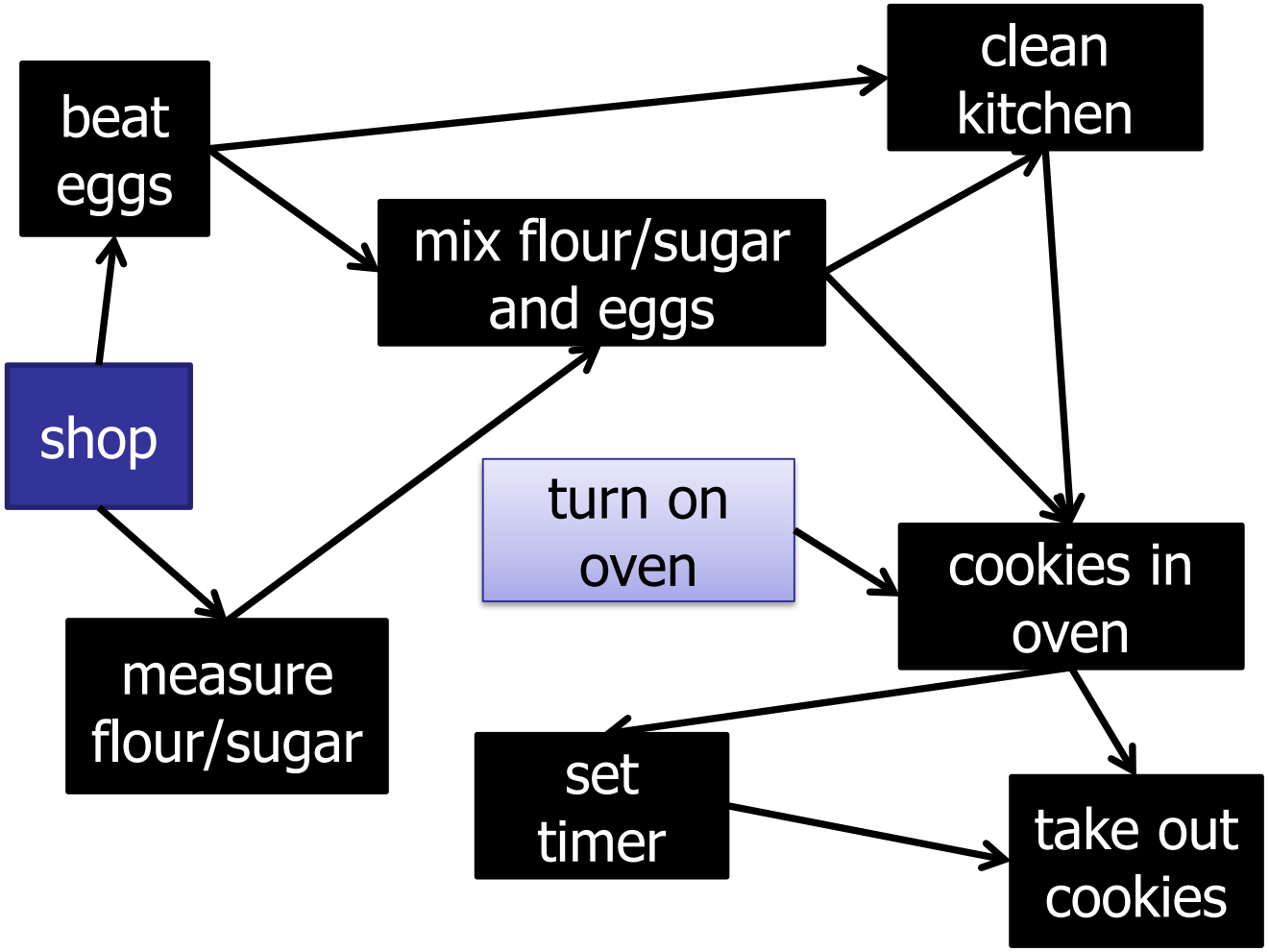
1.
2.
3.
4. measure
5. mix
6. clean
7. in oven
8. set timer
9. take out



Deep Blue: First Visit
Black: Finished Visit

Post-Order Depth-First Search

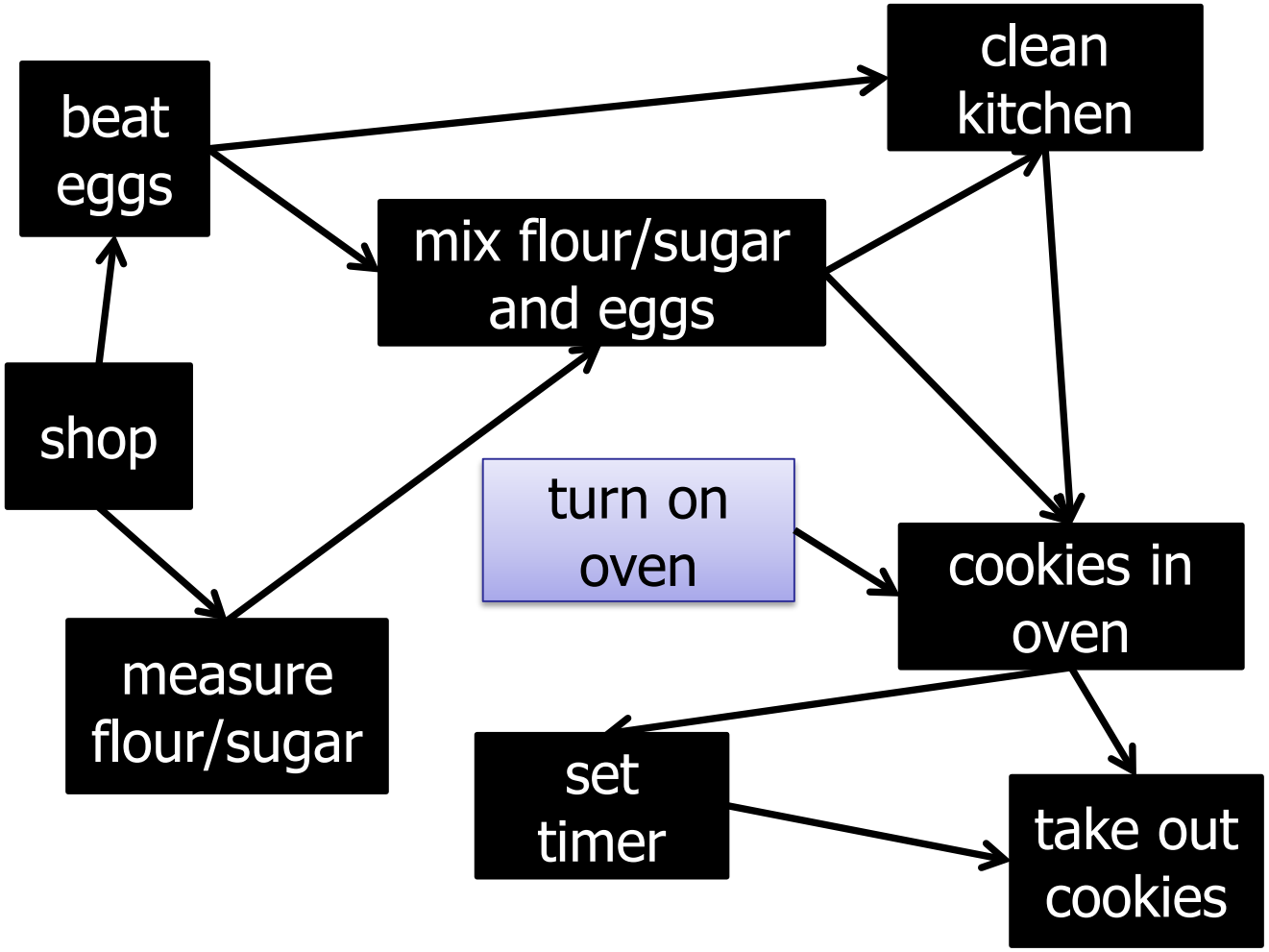
1.
2.
3. beat
4. measure
5. mix
6. clean
7. in oven
8. set timer
9. take out



Deep Blue: First Visit
Black: Finished Visit

Post-Order Depth-First Search

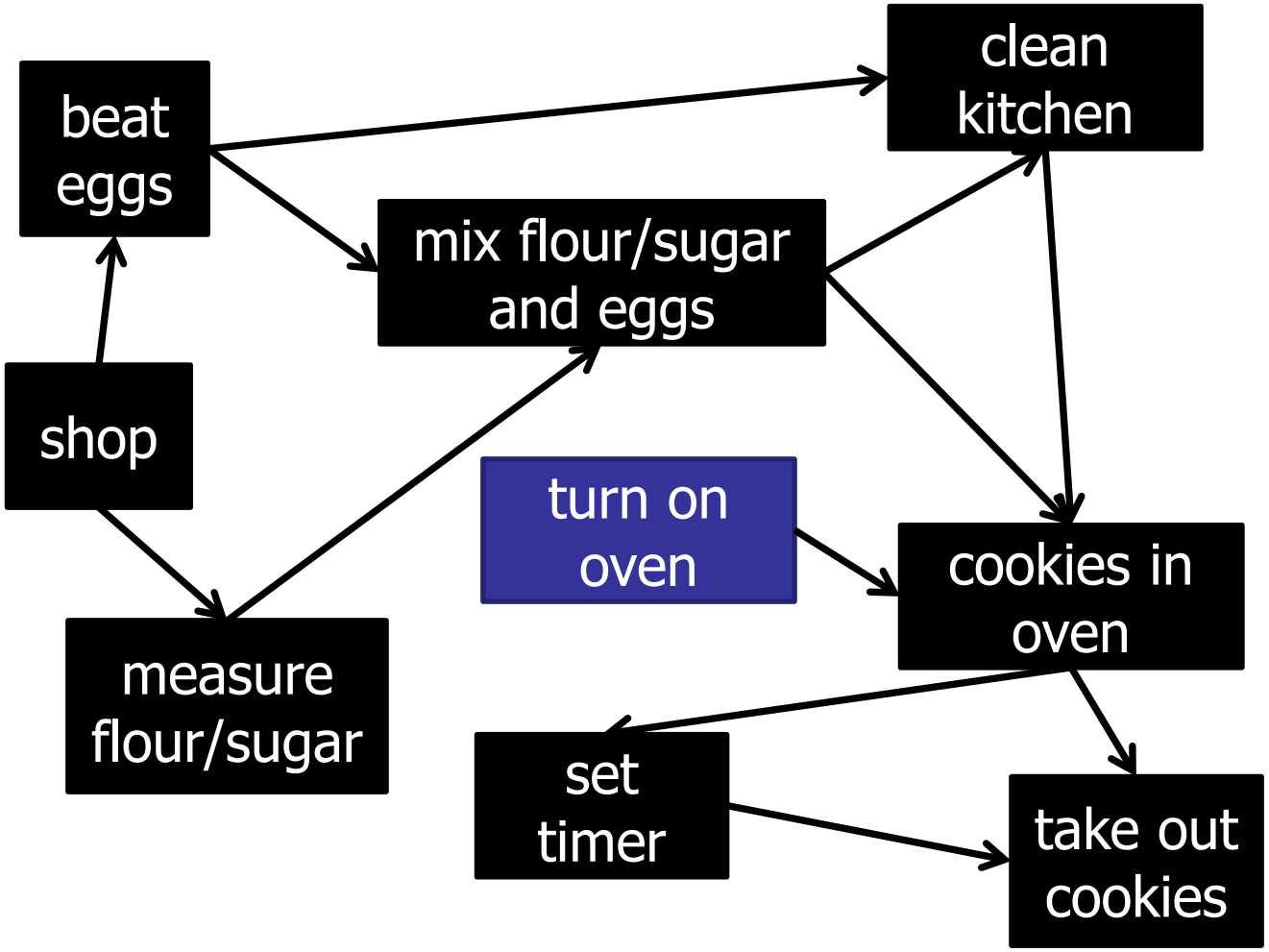
1.
2. shop
3. beat
4. measure
5. mix
6. clean
7. in oven
8. set timer
9. take out



Deep Blue: First Visit
Black: Finished Visit

Post-Order Depth-First Search

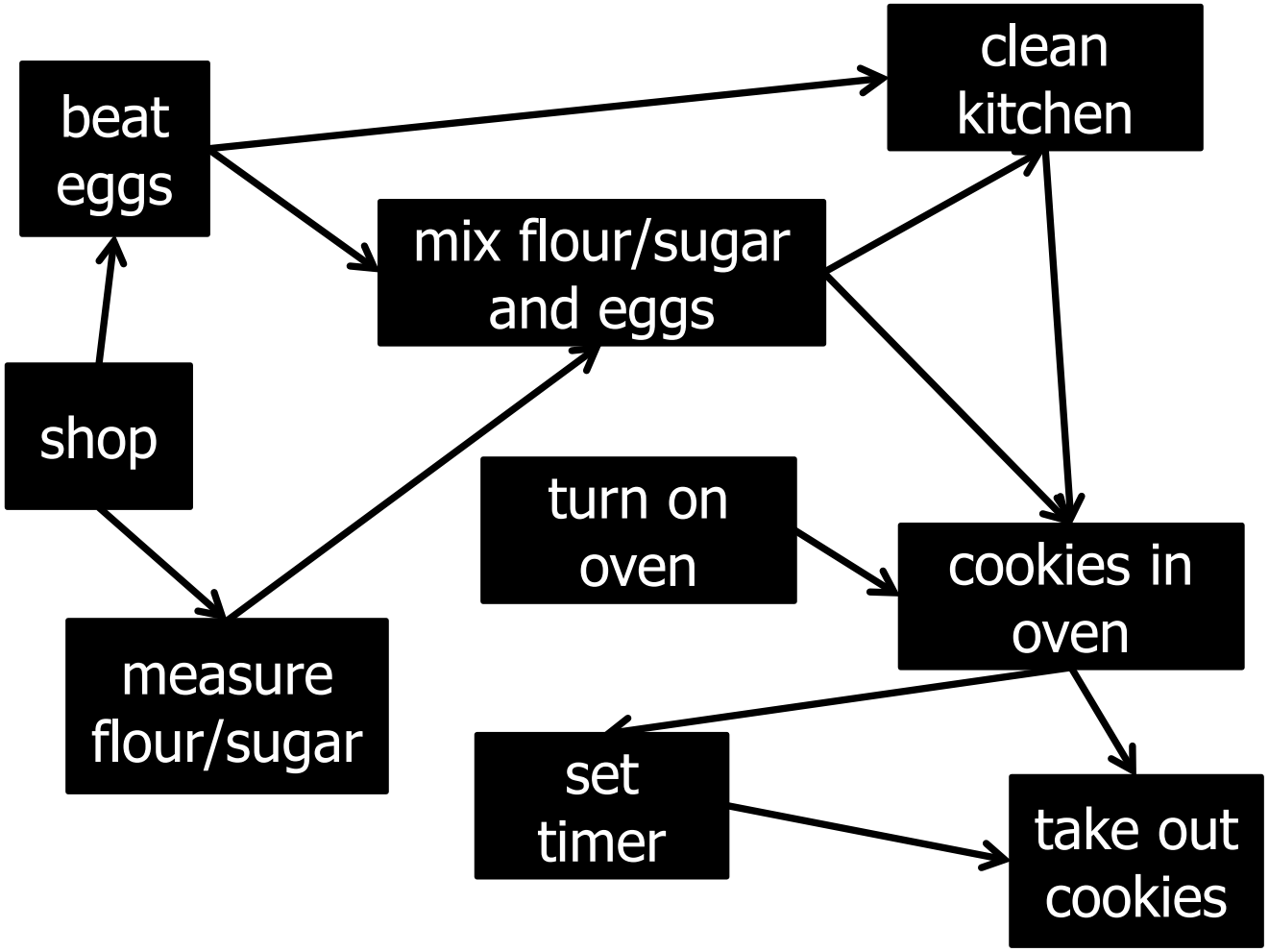
1.
2. shop
3. beat
4. measure
5. mix
6. clean
7. in oven
8. set timer
9. take out



Deep Blue: First Visit
Black: Finished Visit

Post-Order Depth-First Search

1. on oven
2. shop
3. beat
4. measure
5. mix
6. clean
7. in oven
8. set timer
9. take out



Deep Blue: First Visit
Black: Finished Visit

Topological Sort

What is the time complexity of topological sort?

DFS: O(V+E)

Depth-First Search

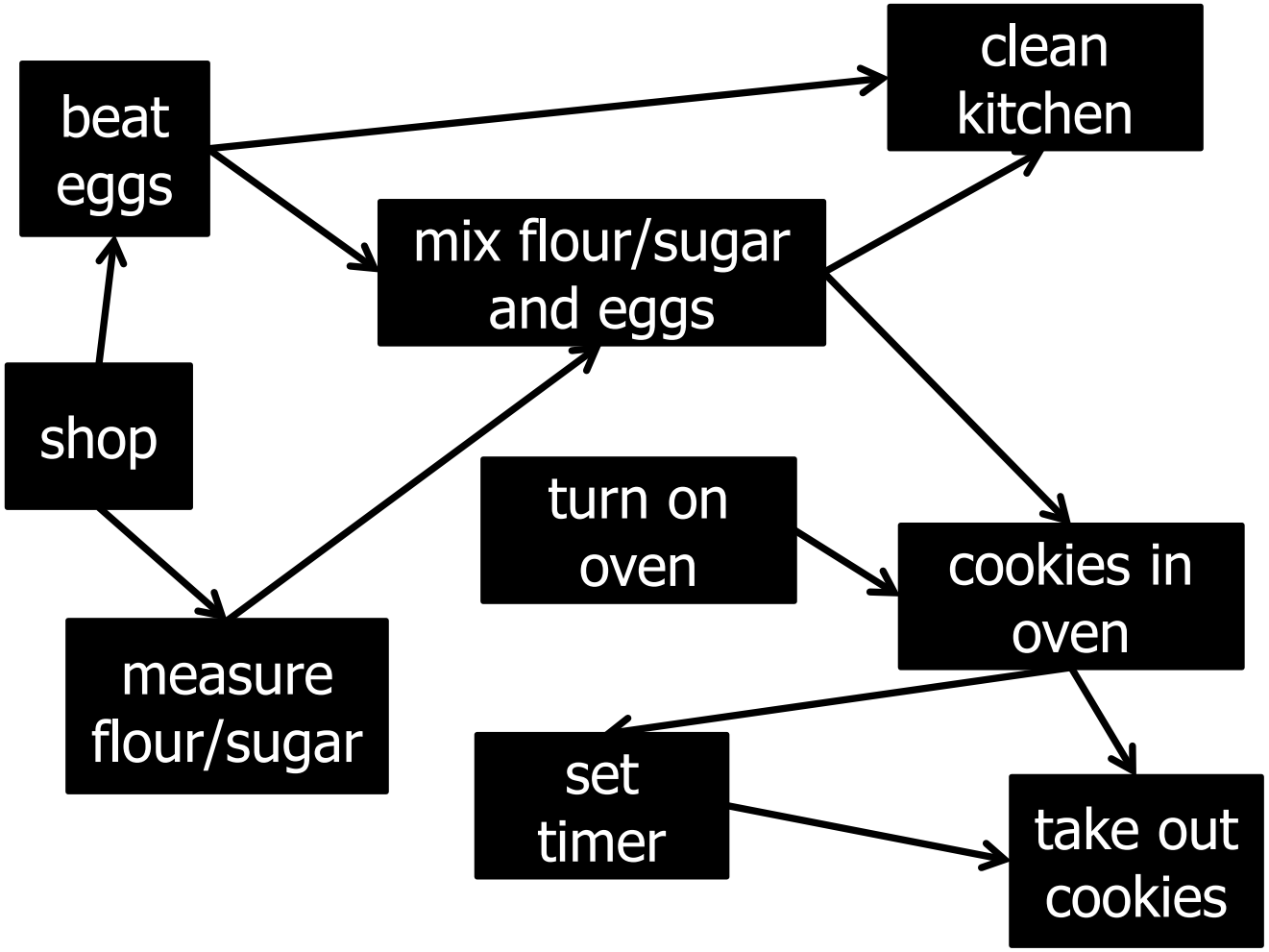
```
DFS-visit(Node[] nodeList, boolean[] visited, int startId){
  for (every neighbor v of startId) {
     if (!visited[v]){
           visited[v] = true;
           DFS-visit(nodeList, visited, v);
           schedule.prepend(v);
     }
  }
}
```

Depth-First Search

```
  for (start = i; start<nodeList.length; start++) {
     if (!visited[start]){
           visited[start] = true;
           DFS-visit(nodeList, visited, start);
           schedule.prepend(v);
     }
  }
}
```

Post-Order Depth-First Search

1. **on oven**
2. **shop**
3. beat
4. measure
5. mix
6. **clean**
7. in oven
8. **set timer**
9. take out



Topological Sort

Input:

- Directed Acyclic Graph (DAG)

Output:

- Total ordering of nodes, where all edges point forwards.

Algorithm:

- Post-order Depth-First Search
- O(V + E) time complexity

Topological Sort

Alternative algorithm:

Input: directed graph G

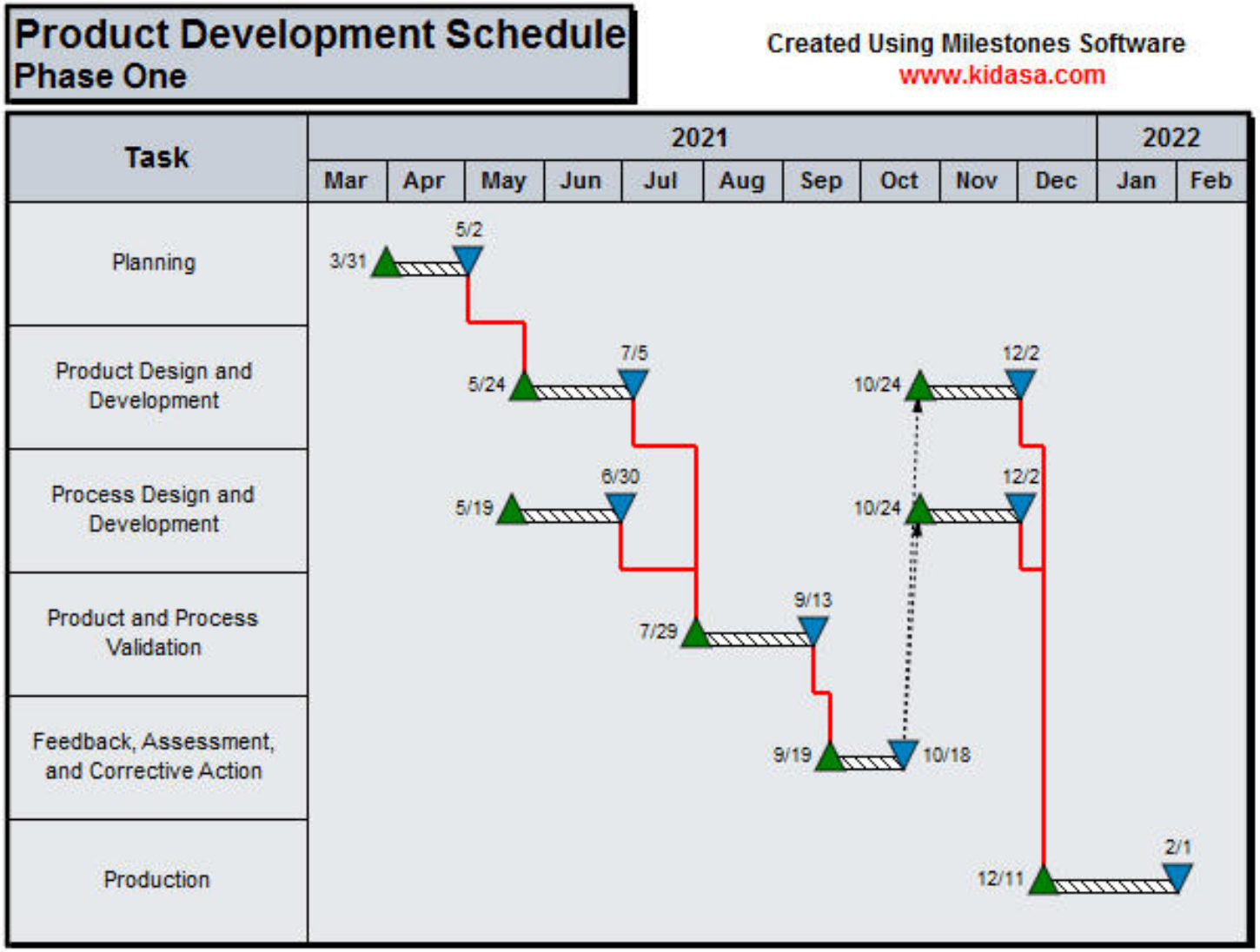
Repeat:

- S = all nodes in G that have *no* incoming edges.
- Add nodes in S to the topo-order
- Remove all edges adjacent to nodes in S
- Remove nodes in S from the graph

Time:

– O(V + E) time complexity

SW Project Dependency



Roadmap

Part I: Directed Graphs

- What is a directed graph?
- Searching directed graphs (DFS / BFS)
- Topological Sort
- Connected Components

Part II: Shortest Paths

- The SSSP Problem
- Bellman-Ford

Roadmap

Part I: Directed Graphs

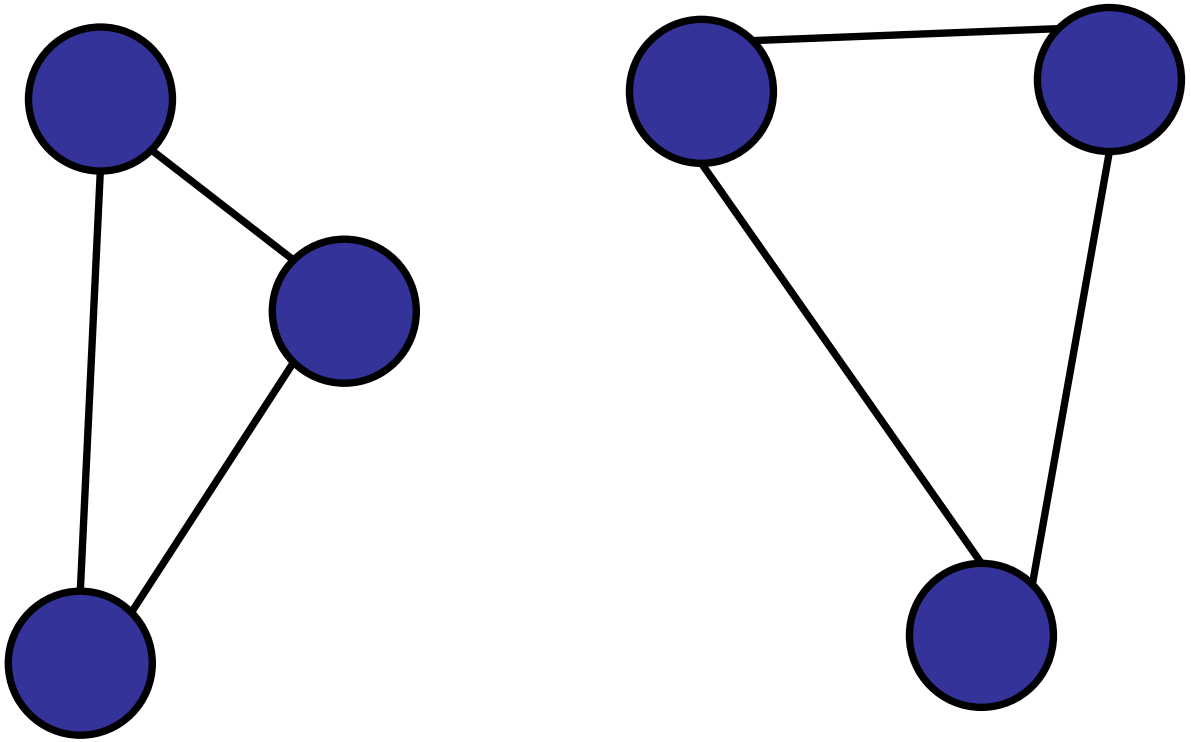
- What is a directed graph?
- Searching directed graphs (DFS / BFS)
- Topological Sort
- Connected Components

Part II: Shortest Paths

- The SSSP Problem
- Bellman-Ford

Connected Components

Undirected graphs

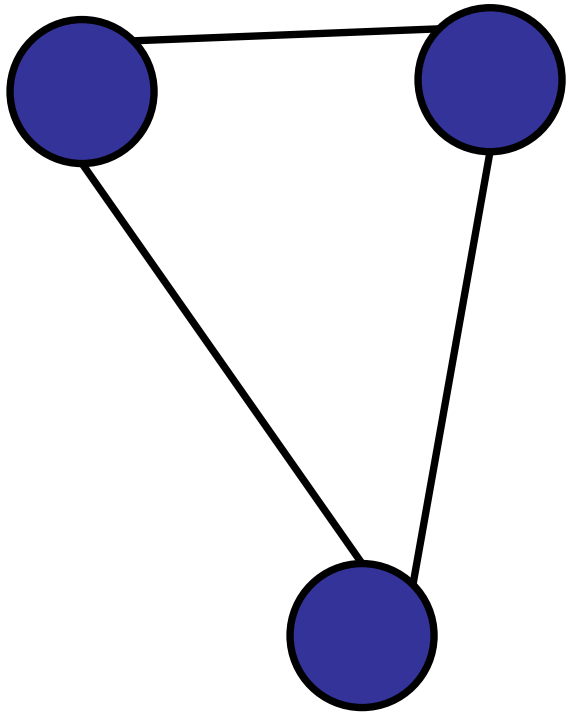


Two connected components

Connected Components

Undirected graphs

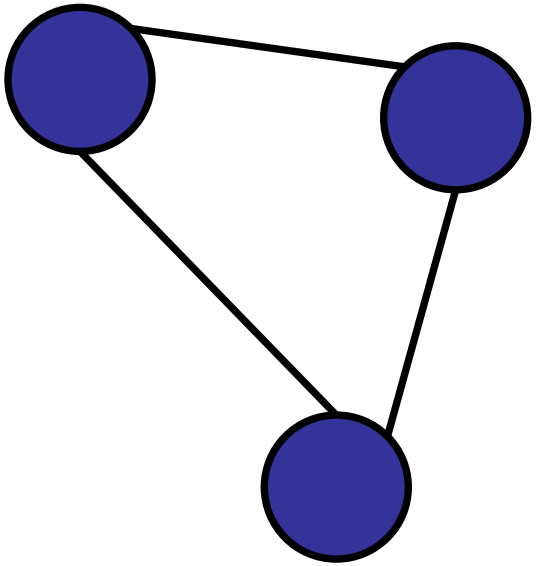
Vertex v and w are in the same connected component if and only if there is a path from v to w.



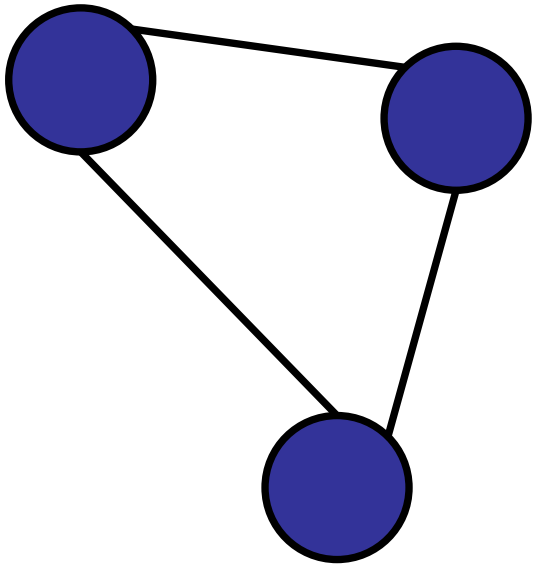
Connected Components

Undirected graphs

Vertex v and w are in the same connected component if and only if there is a path from v to w.

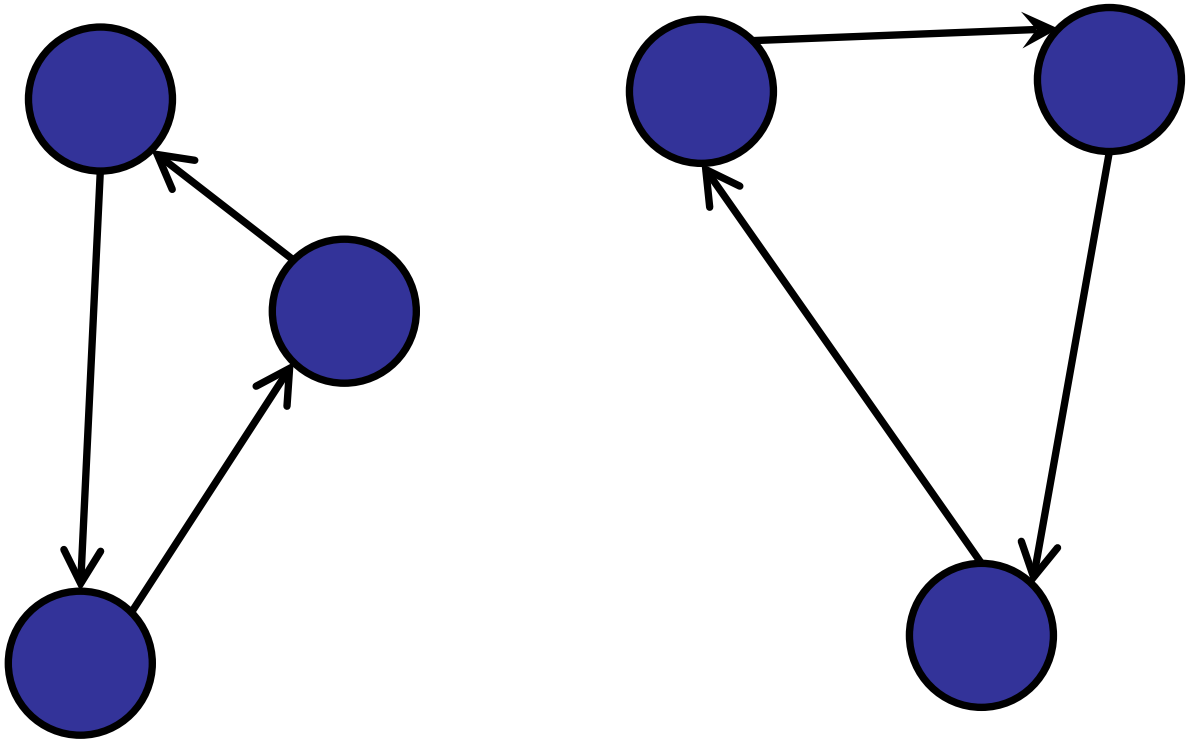


There is a set $\{v_1, v_2, \ldots, v_k\}$ where there is no path from any v_i to v_j if and only if there are k connected components.



Connected Components

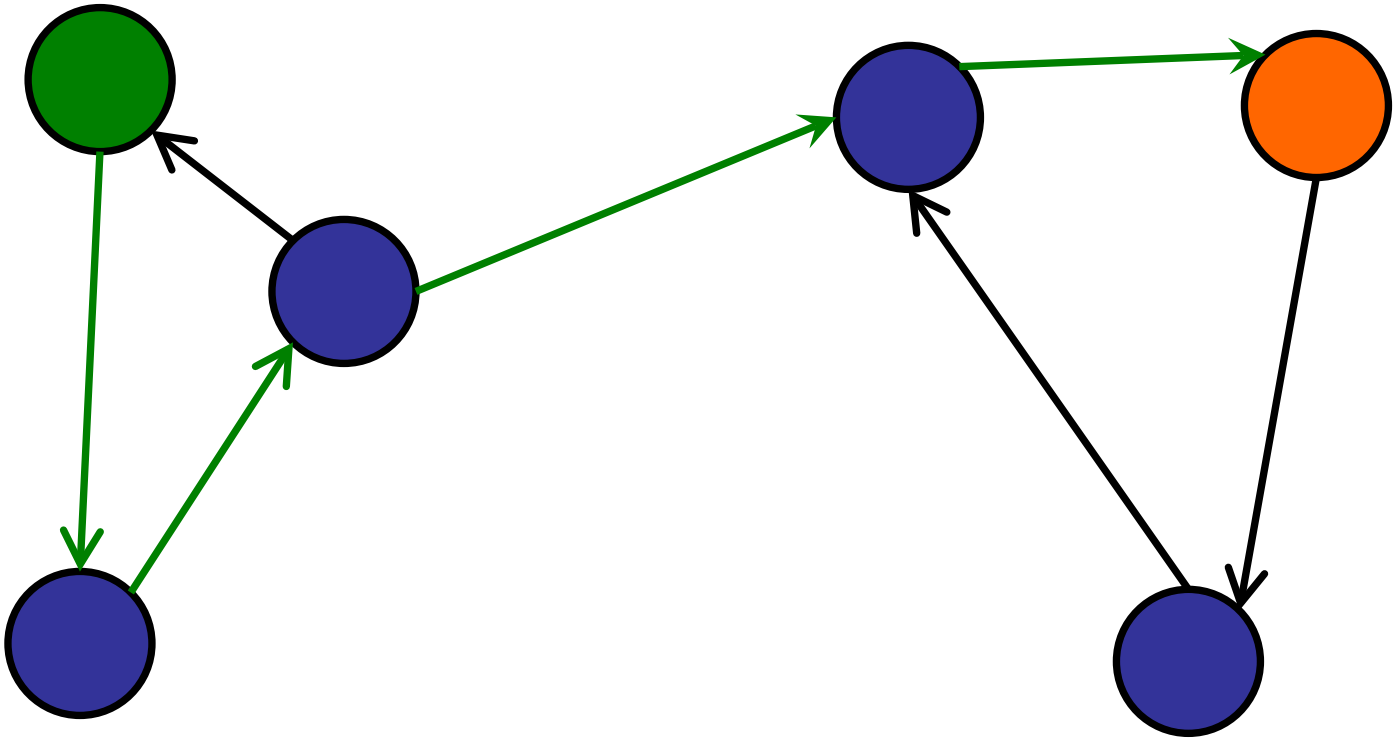
Directed graphs



Two connected components

Connected Components

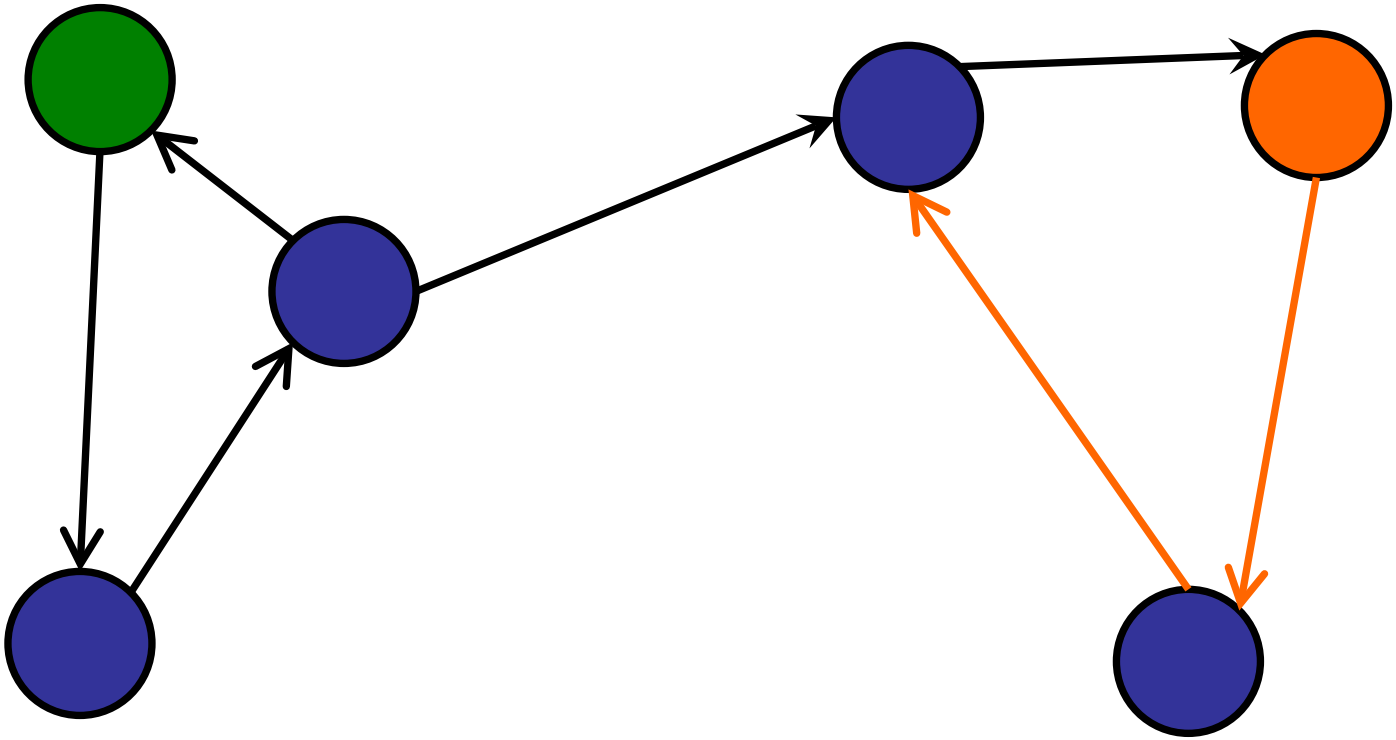
Directed graphs



Two connected components??

Connected Components

Directed graphs



Two connected components??

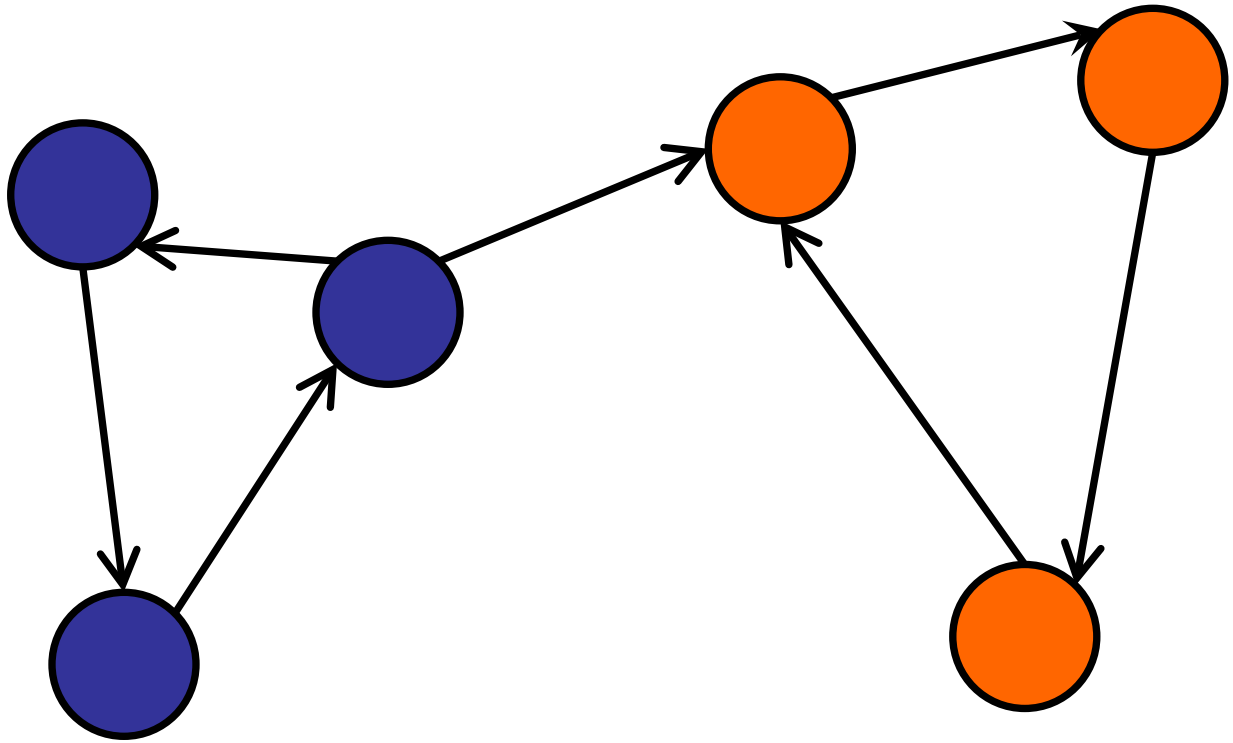
Connected Components

Strongly connected component

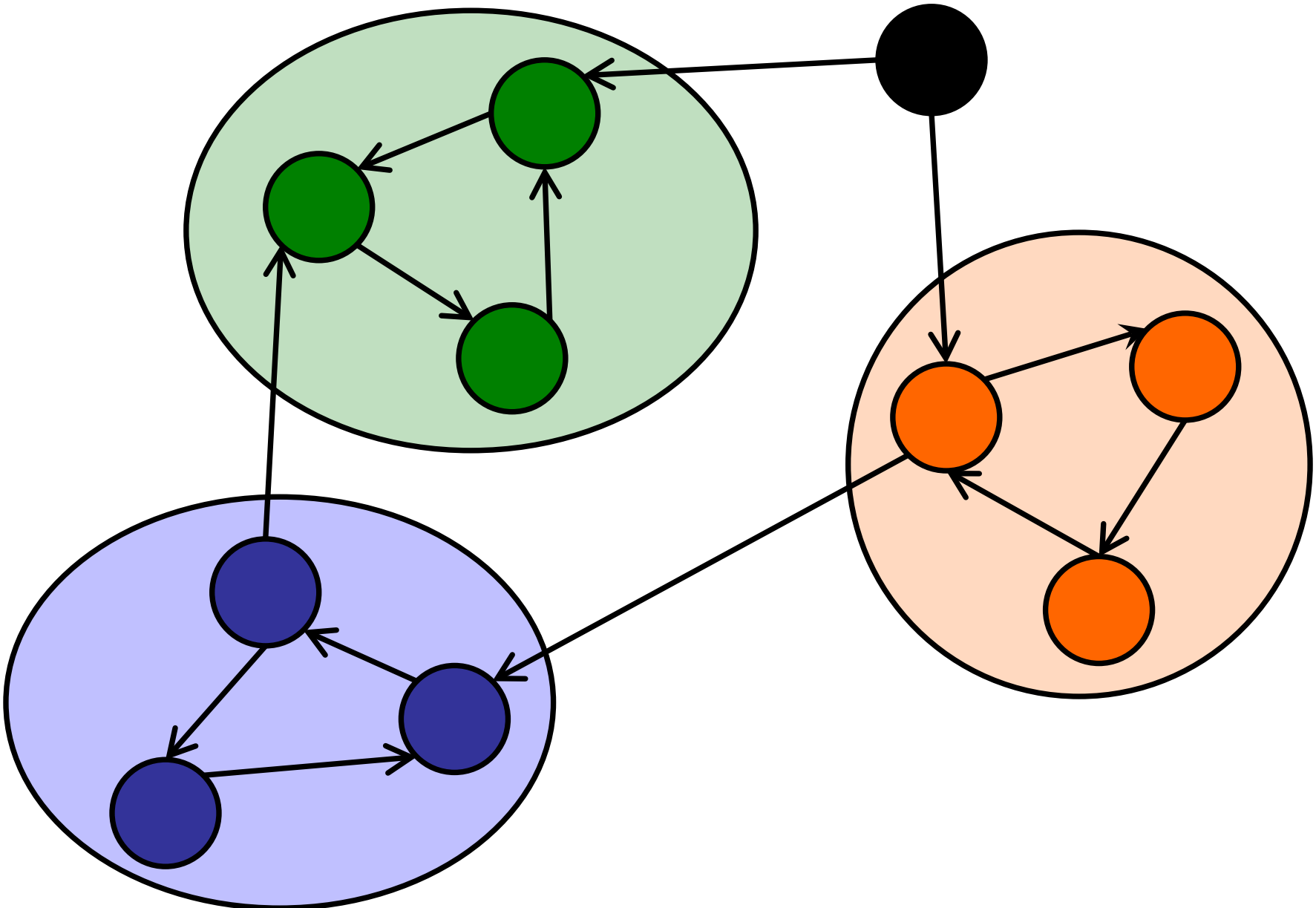
For every vertex v and w:

–There is a path from v to w.

– There is a path from w to v.

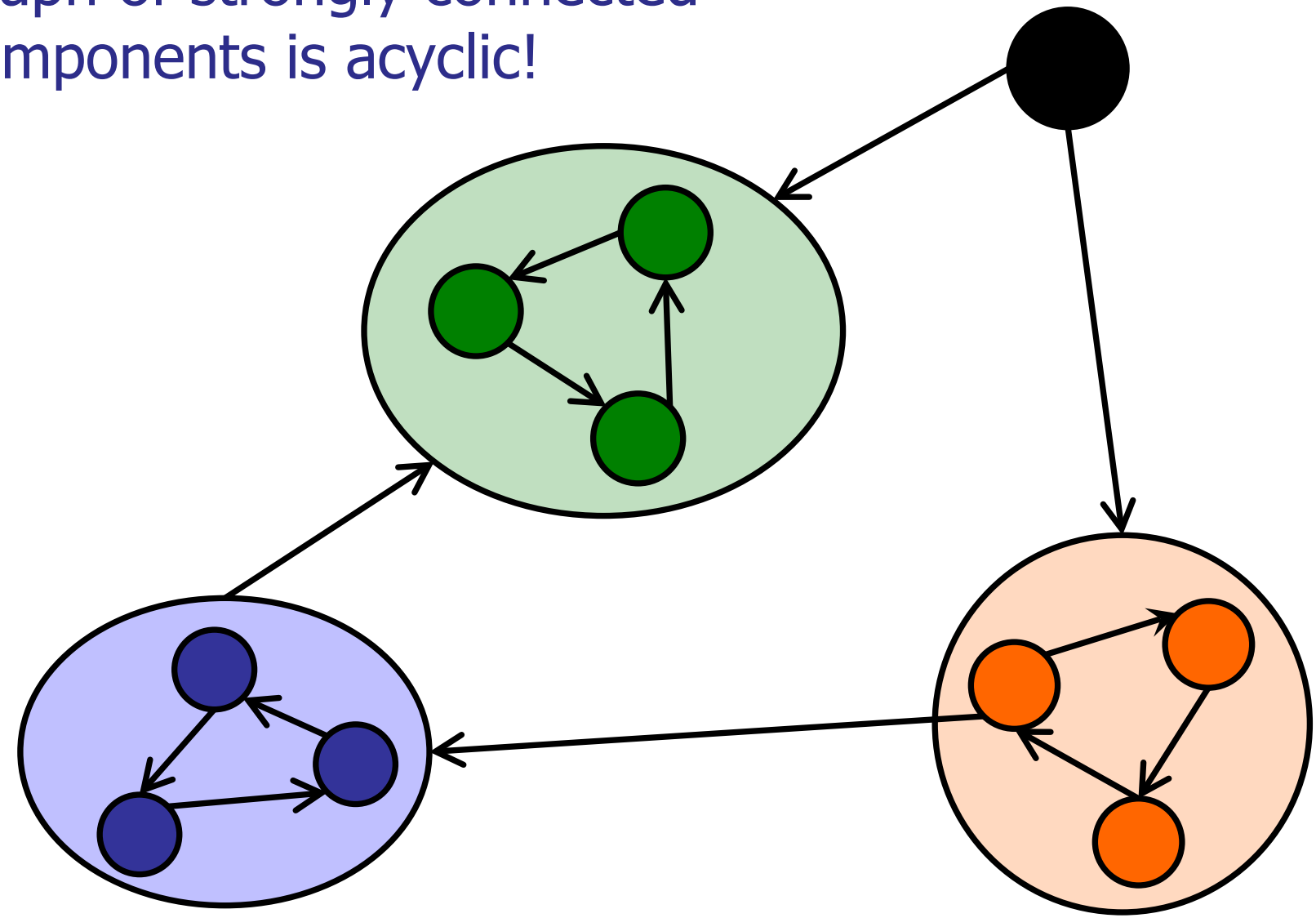


Connected Components



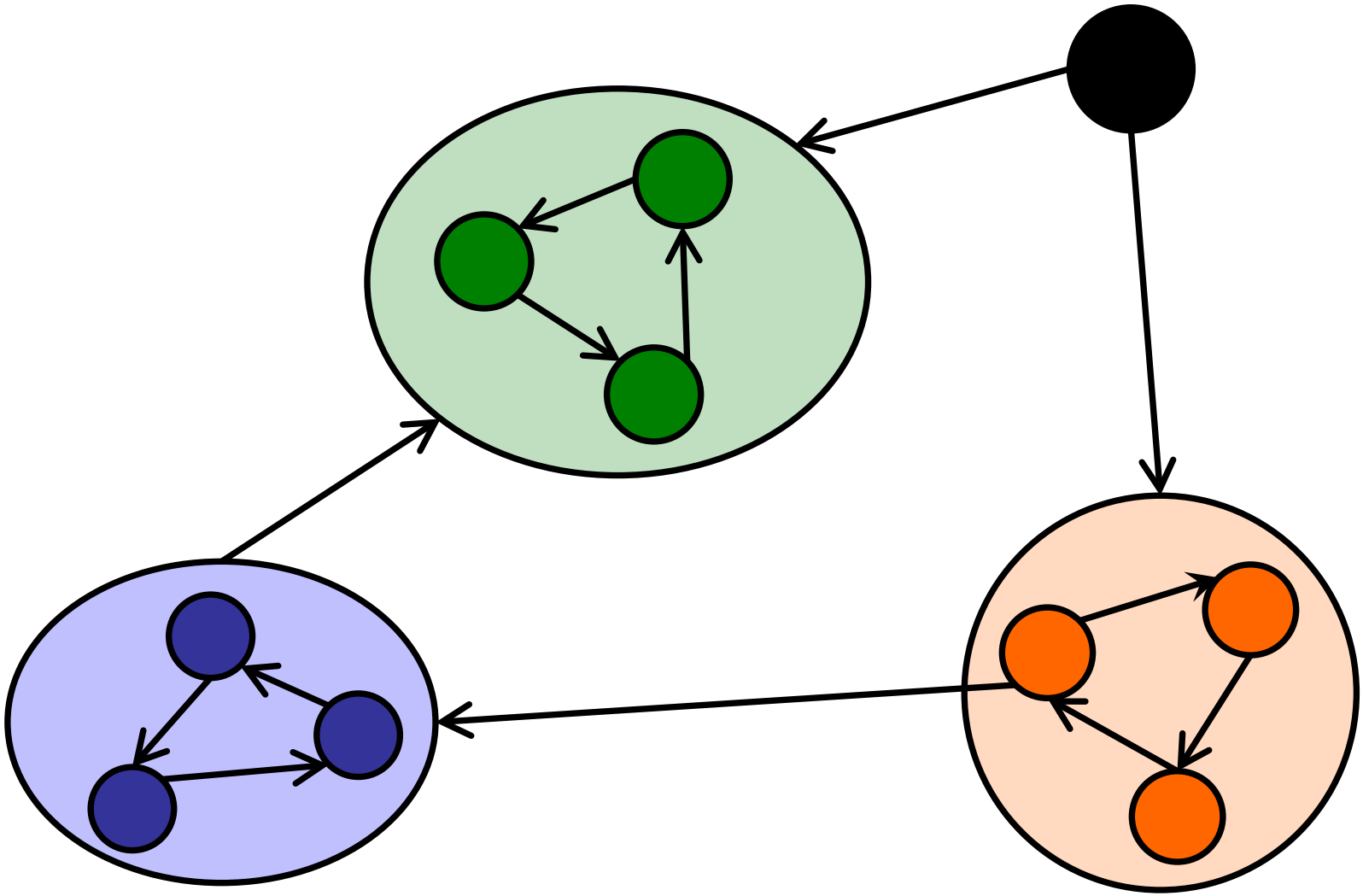
Connected Components

Graph of strongly connected components is acyclic!



Connected Components

Challenge: find all strongly connected components.



Roadmap

Part I: Directed Graphs

- What is a directed graph?
- Searching directed graphs (DFS / BFS)
- Topological Sort
- Connected Components

Part II: Shortest Paths

- The SSSP Problem
- Bellman-Ford