Kosaraju’s Algorithm for Strongly Connected Components

https://cs.pomona.edu/classes/cs140/

Outline

Topics and Learning Objectives

- Review topological orderings
- Discuss strongly connected components
- Cover Kosaraju's Algorithm

Exercise

- Work through Kosaraju's Algorithm

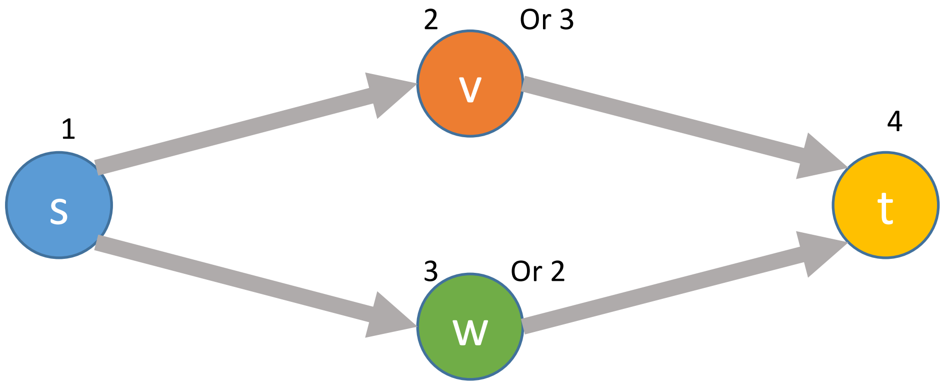
Extra Resources

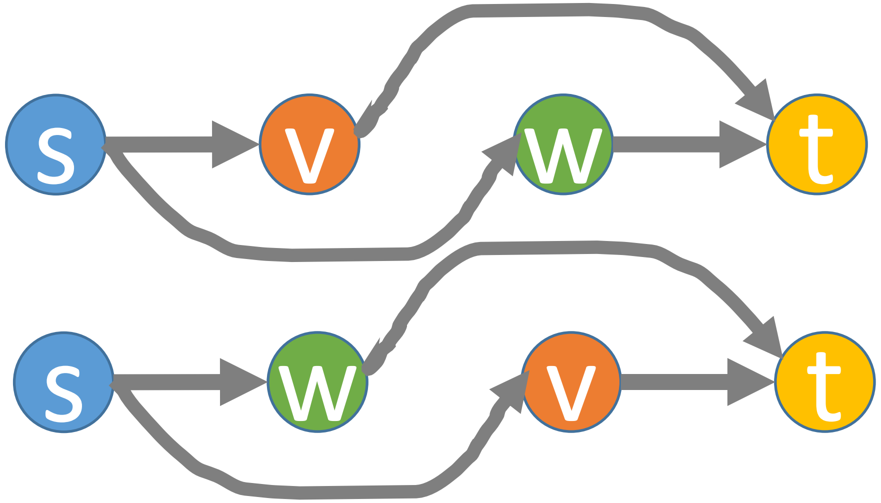
- Introduction to Algorithms, 3rd, chapter 22

Topological Orderings

Definition: a topological ordering of a directed acyclic graph is a labelling f of the graph's vertices such that:

1. The f-values are of the set {1, 2, …, n}
2. For an edge (u, v) of G, f(u) < f(v)





Solve with DFS

```
FUNCTION TopologicalOrdering(G)
    found = {v: FALSE FOR v IN G.vertices}
    fValues = {v: INFINITY FOR v IN G.vertices}
    f = G.vertices.length
    FOR v IN G.vertices
        IF found[v] == FALSE
            DFSTopological(G, v, found, f, fValues)
    RETURN fValues
```

```
FUNCTION DFSTopological(G, v, found, f, fValues)
    found[v] = TRUE
    FOR vOther IN G.edges[v]
        IF found[vOther] == FALSE
            DFSTopological(G, vOther, found, f, fValues)
    fValues[v] = f
    f = f - 1
```

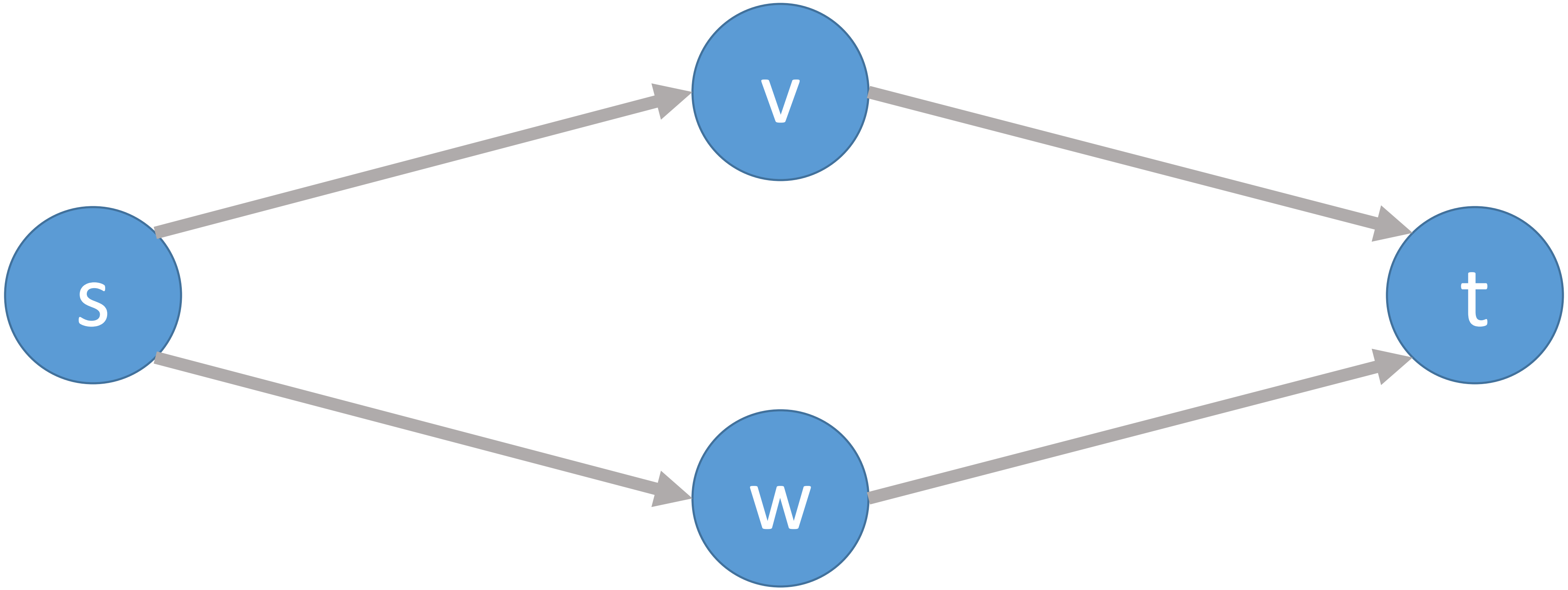
Strongly Connected Components

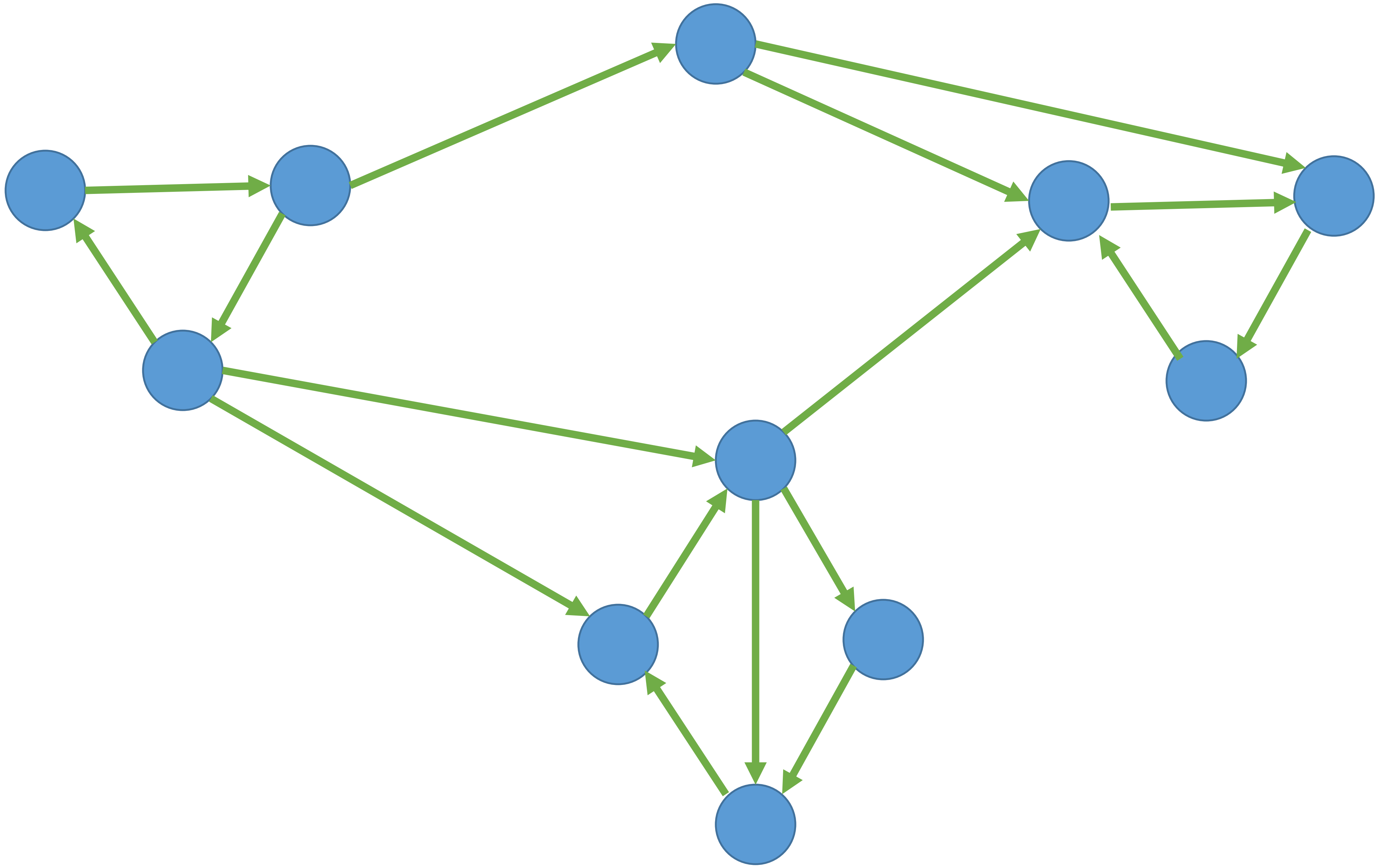
- Topological orderings are useful in their own right, but they also let us efficiently calculate the strongly connected components (SCCs) of a graph

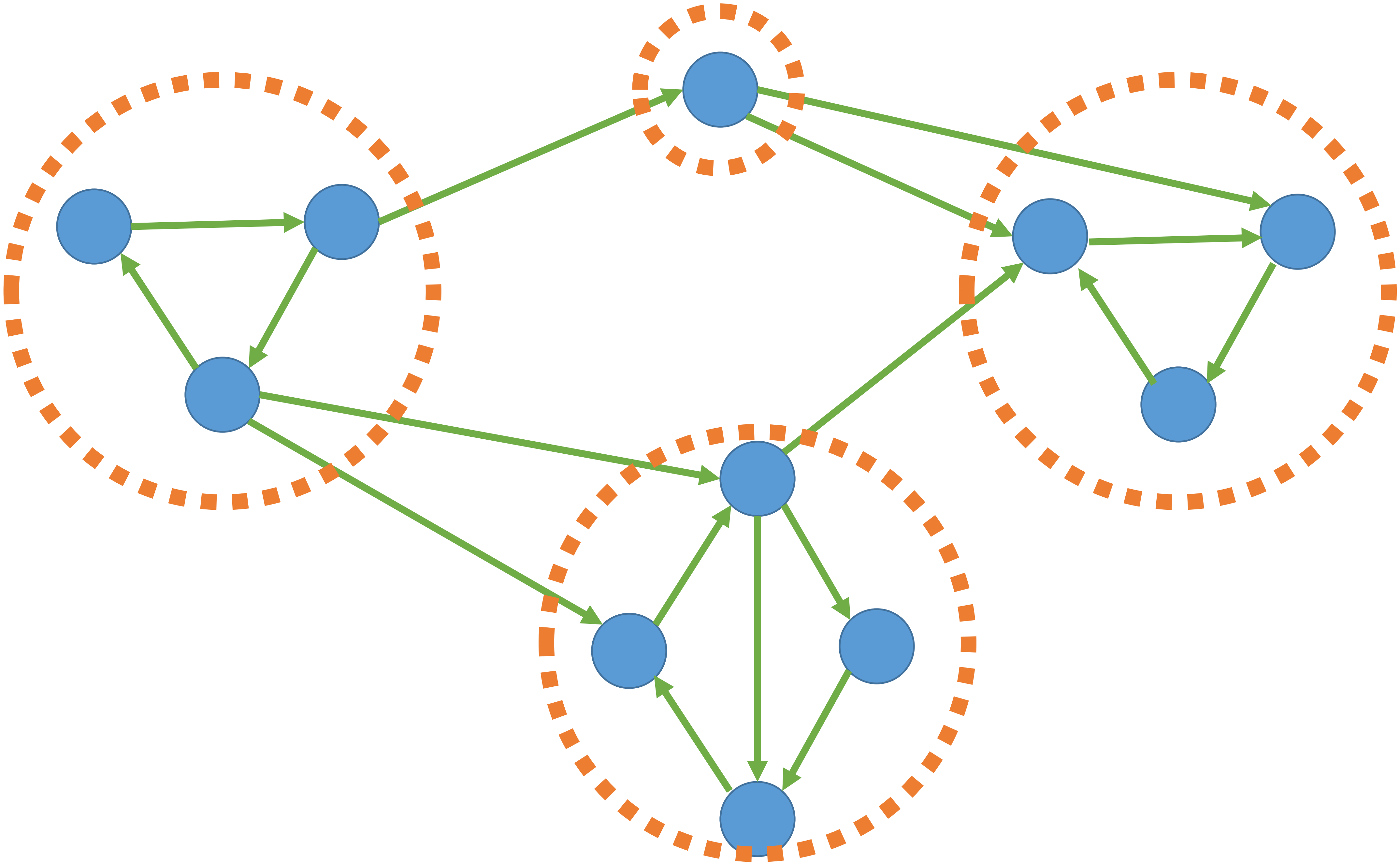
- A component (set of vertices) of a graph is strongly connected if we can find a path from any vertex to any other vertex
- This is a concept for **directed** graphs only
- (just *connected components* for undirected graphs)

Why are SCCs useful?

What are the strongly connected components of this graph?





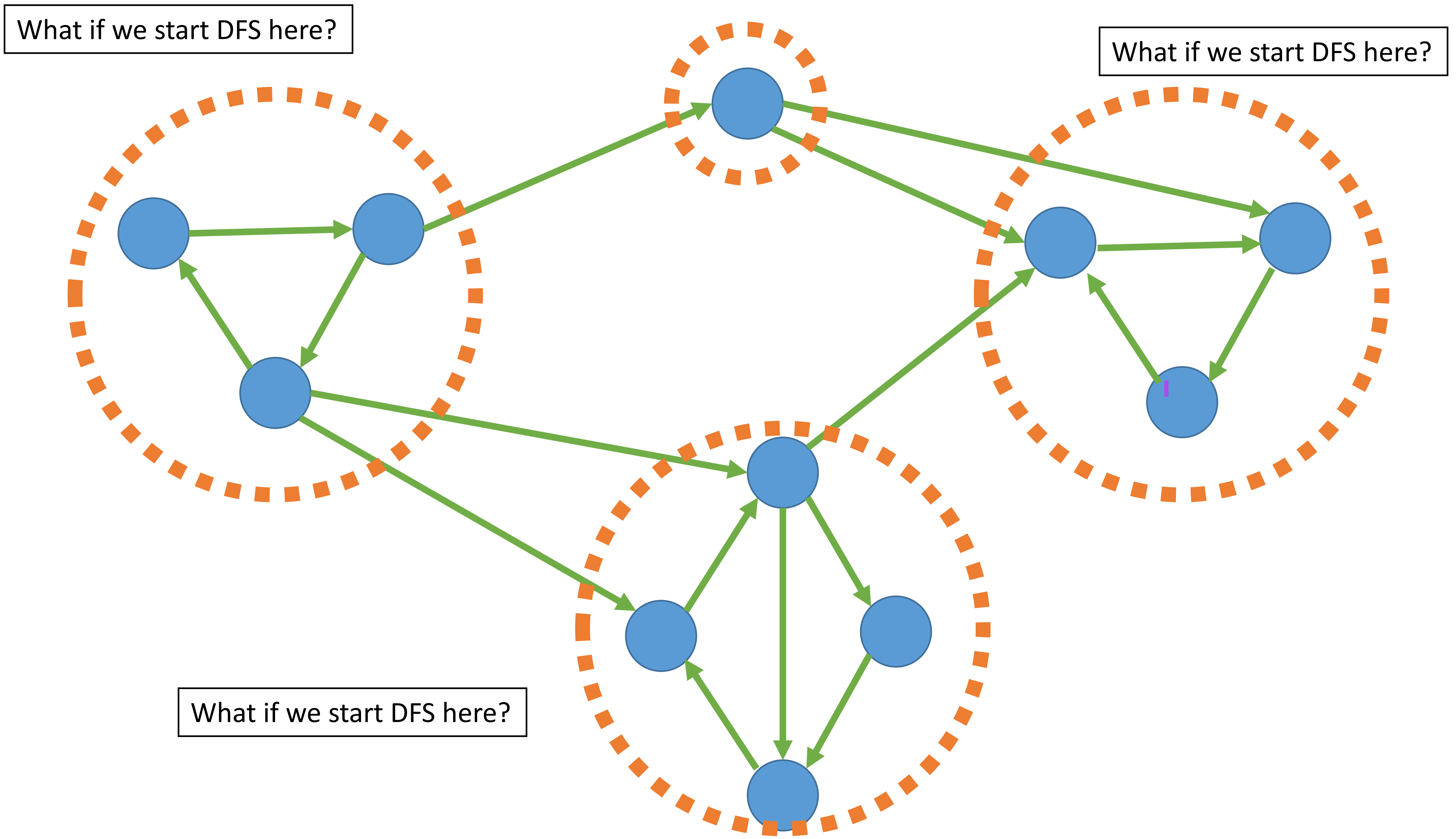


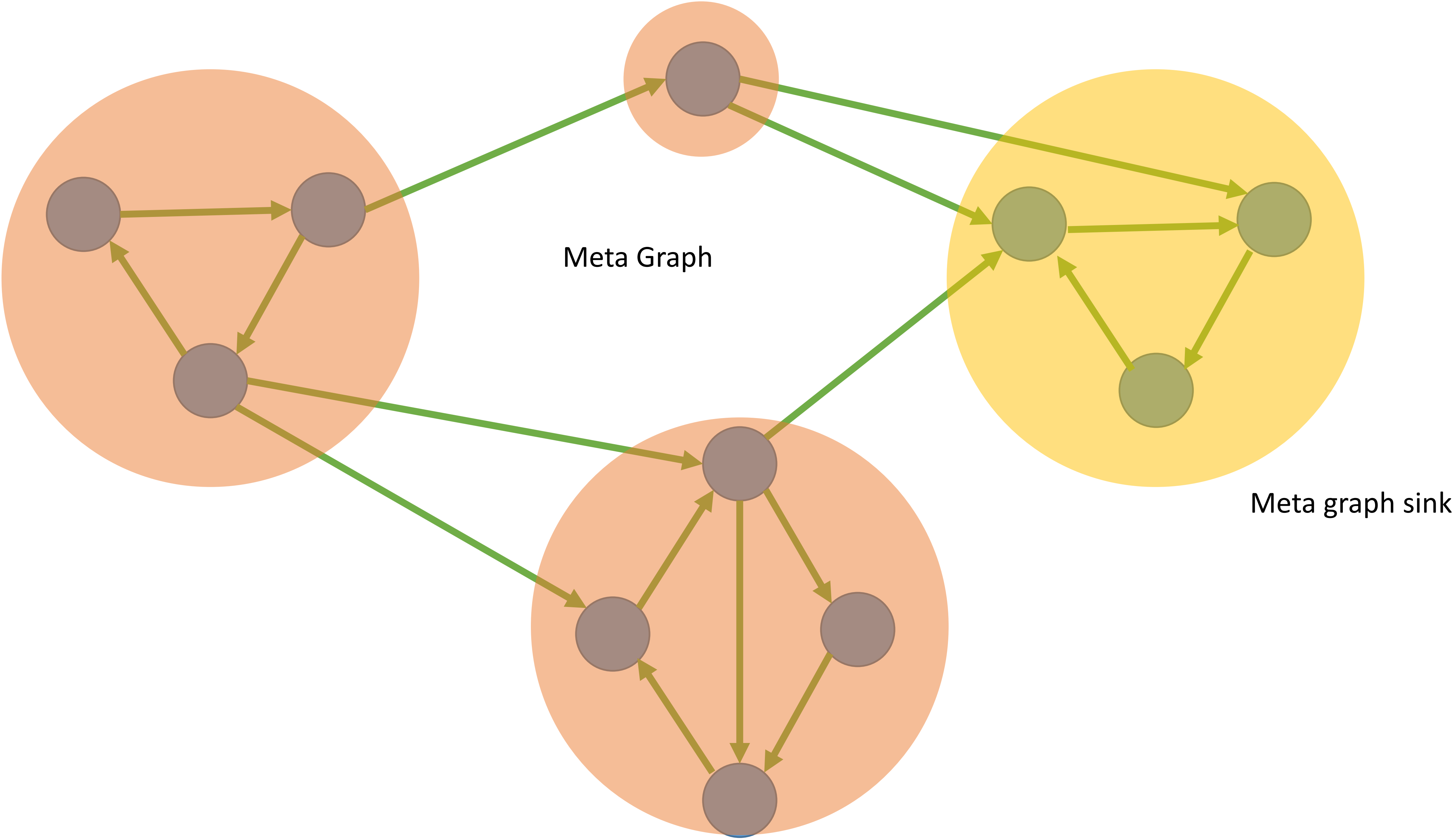
Can we use DFS?

What does a DFS do?

- Finds everything that is findable
- Does not visit any vertex more than once

So, what can we find from each of the different nodes?


What if we start DFS here?
What if we start DFS here?
What if we start DFS here?

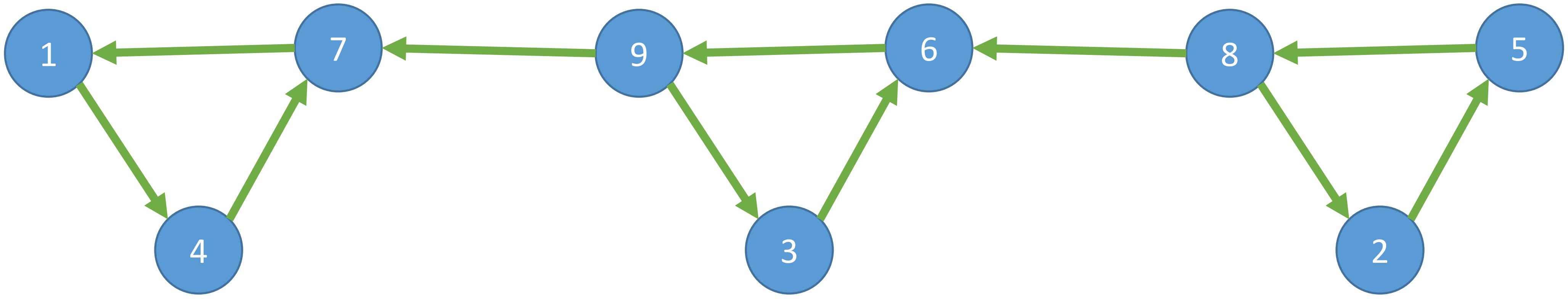

Meta Graph
Meta graph sink

Kosaraju

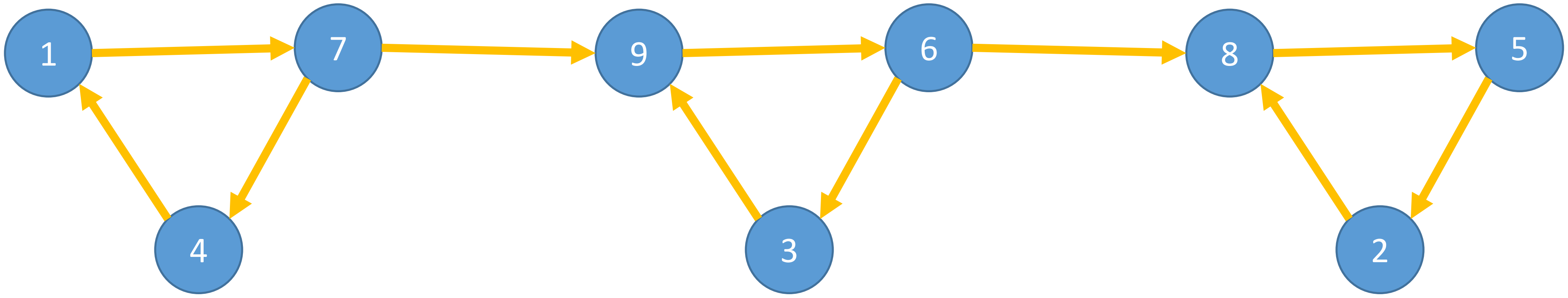
Computes the SCCs in O(m + n) time **(linear!)**

1. Create a reverse version of the G called G_reversed

G



G_reversed



Kosaraju

Computes the SCCs in O(m + n) time **(linear!)**

1. Create a reverse version of the G called G_reversed
2. Run `KosarajuLabels` on G_reversed

   Compute a topological order of the meta graph
3. Create a relabeled version of the G called G_relabeled
4. Run `KosarajuLeaders` on G_relabeled

   Explore vertices in the new order

```
FUNCTION Kosaraju(G)
    G_reversed = reverse_graph(G)
    new_labels = KosarajuLabels(G_reversed)

    G_relabeled = relabel_graph(G, new_labels)
    leaders = KosarajuLeaders(G_relabeled)

    RETURN leaders
```

```
FUNCTION Kosaraju(G)
   G_reversed = reverse_graph(G)
   new_labels = KosarajuLabels(G_reversed)

   G_relabeled = relabel_graph(G, new_labels)
   leaders = KosarajuLeaders(G_relabeled)

   RETURN leaders
```

```
FUNCTION KosarajuLabels(G)
   found = {v: FALSE FOR v IN G.vertices}
   label = 0
   labels = {v: NONE FOR v IN G.vertices}

   FOR v IN G.vertices.reverse_order
      IF found[v] == FALSE
         DFSLabels(G, v, found, label, labels)

   RETURN labels
```

```
FUNCTION DFSLabels(G, v, found, label, labels)
   found[v] = TRUE
   FOR vOther IN G.edges[v]
      IF found[vOther] == FALSE
         DFSLabels(G, vOther, found, label, labels)
   label = label + 1
   labels[v] = label
```

```
FUNCTION Kosaraju(G)
   G_reversed = reverse_graph(G)
   new_labels = KosarajuLabels(G_reversed)

   G_relabeled = relabel_graph(G, new_labels)
   leaders = KosarajuLeaders(G_relabeled)

   RETURN leaders
```

```
FUNCTION KosarajuLeaders(G)
   found = {v: FALSE FOR v IN G.vertices}
   leaders = {v: NONE FOR v IN G.vertices}

   FOR v IN G.vertices.reverse_order
      IF found[v] == FALSE
         leader = v
         DFSLeaders(G, v, found, leader, leaders)

   RETURN leaders
```

```
FUNCTION DFSLeaders(G, v, found, leader, leaders)
   found[v] = TRUE
   leaders[v] = leader
   FOR vOther IN G.edges[v]
      IF found[vOther] == FALSE
         DFSLeaders(G, vOther, found, leader, leaders)
```

```
FUNCTION KosarajuLabels(G)
   found = {v: FALSE FOR v IN G.vertices}
   label = 0
   labels = {v: NONE FOR v IN G.vertices}

   FOR v IN G.vertices.reverse_order
      IF found[v] == FALSE
         DFSLabels(G, v, found, label, labels)

   RETURN labels
```

```
FUNCTION KosarajuLeaders(G)
   found = {v: FALSE FOR v IN G.vertices}
   leaders = {v: NONE FOR v IN G.vertices}

   FOR v IN G.vertices.reverse_order
      IF found[v] == FALSE
         leader = v
         DFSLeaders(G, v, found, leader, leaders)

   RETURN leaders
```

```
FUNCTION DFSLabels(G, v, found, label, labels)
   found[v] = TRUE
   FOR vOther IN G.edges[v]
      IF found[vOther] == FALSE
         DFSLabels(G, vOther, found, label, labels)
   label = label + 1
   labels[v] = label
```

```
FUNCTION DFSLeaders(G, v, found, leader, leaders)
   found[v] = TRUE
   leaders[v] = leader
   FOR vOther IN G.edges[v]
      IF found[vOther] == FALSE
         DFSLeaders(G, vOther, found, leader, leaders)
```

These are typically implemented in a single function

```
FUNCTION KosarajuLabels(G)
    found = {v: FALSE FOR v IN G.vertices}
    label = 0
    labels = {v: NONE FOR v IN G.vertices}

    FOR v IN G.vertices.reverse_order
        IF found[v] == FALSE
            DFSLabels(G, v, found, label, labels)

    RETURN labels
```

```
FUNCTION KosarajuLeaders(G)
    found = {v: FALSE FOR v IN G.vertices}
    leaders = {v: NONE FOR v IN G.vertices}

    FOR v IN G.vertices.reverse_order
        IF found[v] == FALSE
            leader = v
            DFSLeaders(G, v, found, leader, leaders)

    RETURN leaders
```

```
FUNCTION DFSLabels(G, v, found, label, labels)
    found[v] = TRUE
    FOR vOther IN G.edges[v]
        IF found[vOther] == FALSE
            DFSLabels(G, vOther, found, label, labels)
    label = label + 1
    labels[v] = label
```

```
FUNCTION DFSLeaders(G, v, found, leader, leaders)
    found[v] = TRUE
    leaders[v] = leader
    FOR vOther IN G.edges[v]
        IF found[vOther] == FALSE
            DFSLeaders(G, vOther, found, leader, leaders)
```

These are typically implemented in a single function

```
FUNCTION KosarajuLabels(G)
   found = {v: FALSE FOR v IN G.vertices}
   label = 0
   labels = {v: NONE FOR v IN G.vertices}

   FOR v IN G.vertices.reverse_order
      IF found[v] == FALSE
         DFSLabels(G, v, found, label, labels)

   RETURN labels
```

```
FUNCTION KosarajuLeaders(G)
   found = {v: FALSE FOR v IN G.vertices}
   leaders = {v: NONE FOR v IN G.vertices}

   FOR v IN G.vertices.reverse_order
      IF found[v] == FALSE
         leader = v
         DFSLeaders(G, v, found, leader, leaders)

   RETURN leaders
```

```
FUNCTION DFSLabels(G, v, found, label, labels)
   found[v] = TRUE
   FOR vOther IN G.edges[v]
      IF found[vOther] == FALSE
         DFSLabels(G, vOther, found, label, labels)
   label = label + 1
   labels[v] = label
```

```
FUNCTION DFSLeaders(G, v, found, leader, leaders)
   found[v] = TRUE
   leaders[v] = leader
   FOR vOther IN G.edges[v]
      IF found[vOther] == FALSE
         DFSLeaders(G, vOther, found, leader, leaders)
```

These are typically implemented in a single function

```
FUNCTION KosarajuLoop(G)
   found = {v: FALSE FOR v IN G.vertices}
   label = 0
   labels = {v: NONE FOR v IN G.vertices}
   leaders = {v: NONE FOR v IN G.vertices}

   FOR v IN G.vertices.reverse_order
      IF found[v] == FALSE
         leader = v
         KosarajuDFS(G, v, found, label, labels, leader, leaders)

   RETURN labels, leaders

FUNCTION KosarajuDFS(G, v, found, label, labels, leader, leaders)
   found[v] = TRUE
   leaders[v] = leader
   FOR vOther IN G.edges[v]
      IF found[vOther] == FALSE
         KosarajuDFS(G, v, found, label, labels, leader, leaders)
   label = label + 1
   labels[v] = label
```

```
FUNCTION Kosaraju(G)
    G_reversed = reverse_graph(G)
    new_labels = KosarajuLabels(G_reversed)

    G_relabeled = relabel_graph(G, new_labels)
    leaders = KosarajuLeaders(G_relabeled)

    RETURN leaders
```

```
FUNCTION Kosaraju(G)
    G_reversed = reverse_graph(G)
    new_labels, _ = KosarajuLoop(G_reversed)

    G_relabeled = relabel_graph(G, new_labels)
    _, leaders = KosarajuLoop(G_relabeled)

    RETURN leaders
```

Kosaraju

Computes the SCCs in O(m + n) time **(linear!)**

1. Create a reverse version of the G called G_reversed

2. Run KosarajuLoop on G_reversed

   Compute a topological order of the meta graph

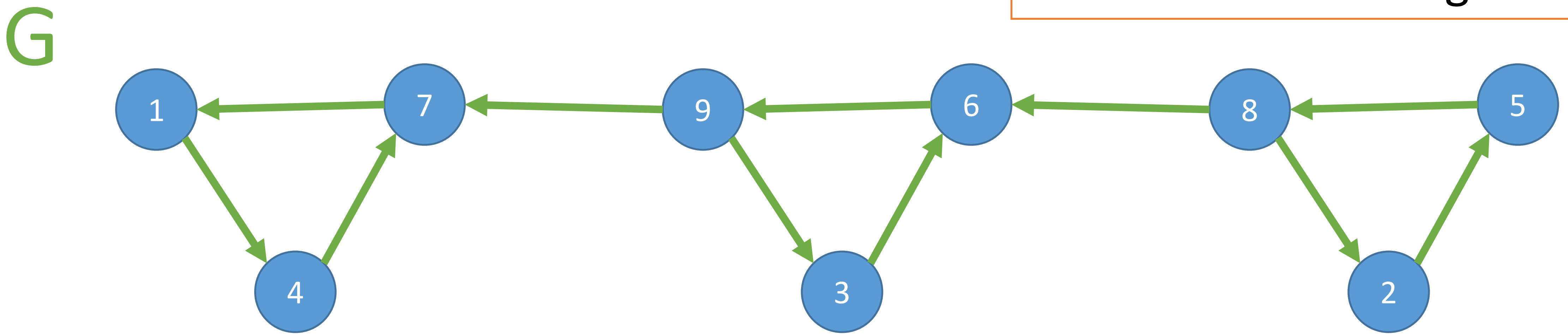
3. Create a relabeled version of the G called G_relabeled

4. Run KosarajuLoop on G_relabeled

   Explore vertices in the new order

```
FUNCTION Kosaraju(G)
    G_reversed = reverse_graph(G)
    new_labels, _ = KosarajuLoop(G_reversed)

    G_relabeled = relabel_graph(G, new_labels)
    _, leaders = KosarajuLoop(G_relabeled)

    RETURN leaders
```

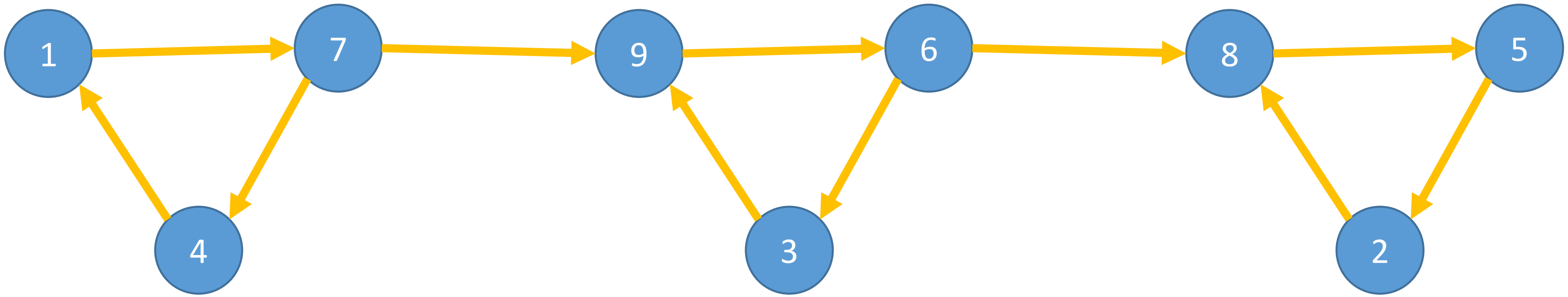
Where do we want to start DFS if we are looking for SCCs?



```
FUNCTION Kosaraju(G)
    G_reversed = reverse_graph(G)
    new_labels, _ = KosarajuLoop(G_reversed)

    G_relabeled = relabel_graph(G, new_labels)
    _, leaders = KosarajuLoop(G_relabeled)

    RETURN leaders
```

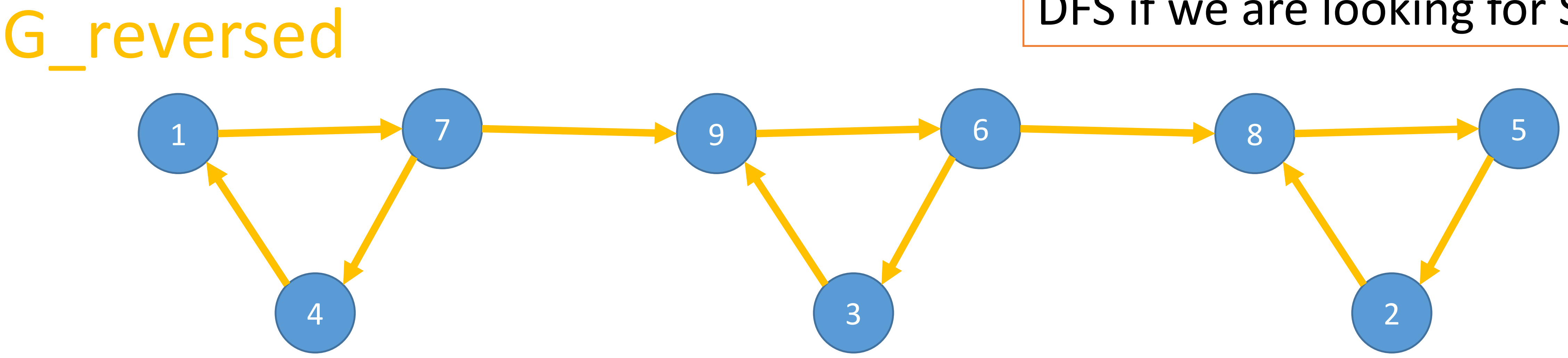
Where do we want to start DFS if we are looking for SCCs?

G_reversed



```
FUNCTION Kosaraju(G)
    G_reversed = reverse_graph(G)
    new_labels, _ = KosarajuLoop(G_reversed)

    G_relabeled = relabel_graph(G, new_labels)
    _, leaders = KosarajuLoop(G_relabeled)

    RETURN leaders
```

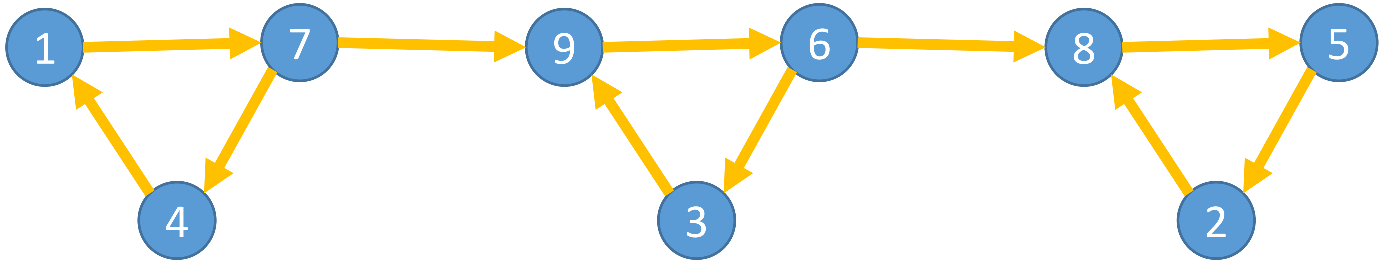
Where do we want to start DFS if we are looking for SCCs?

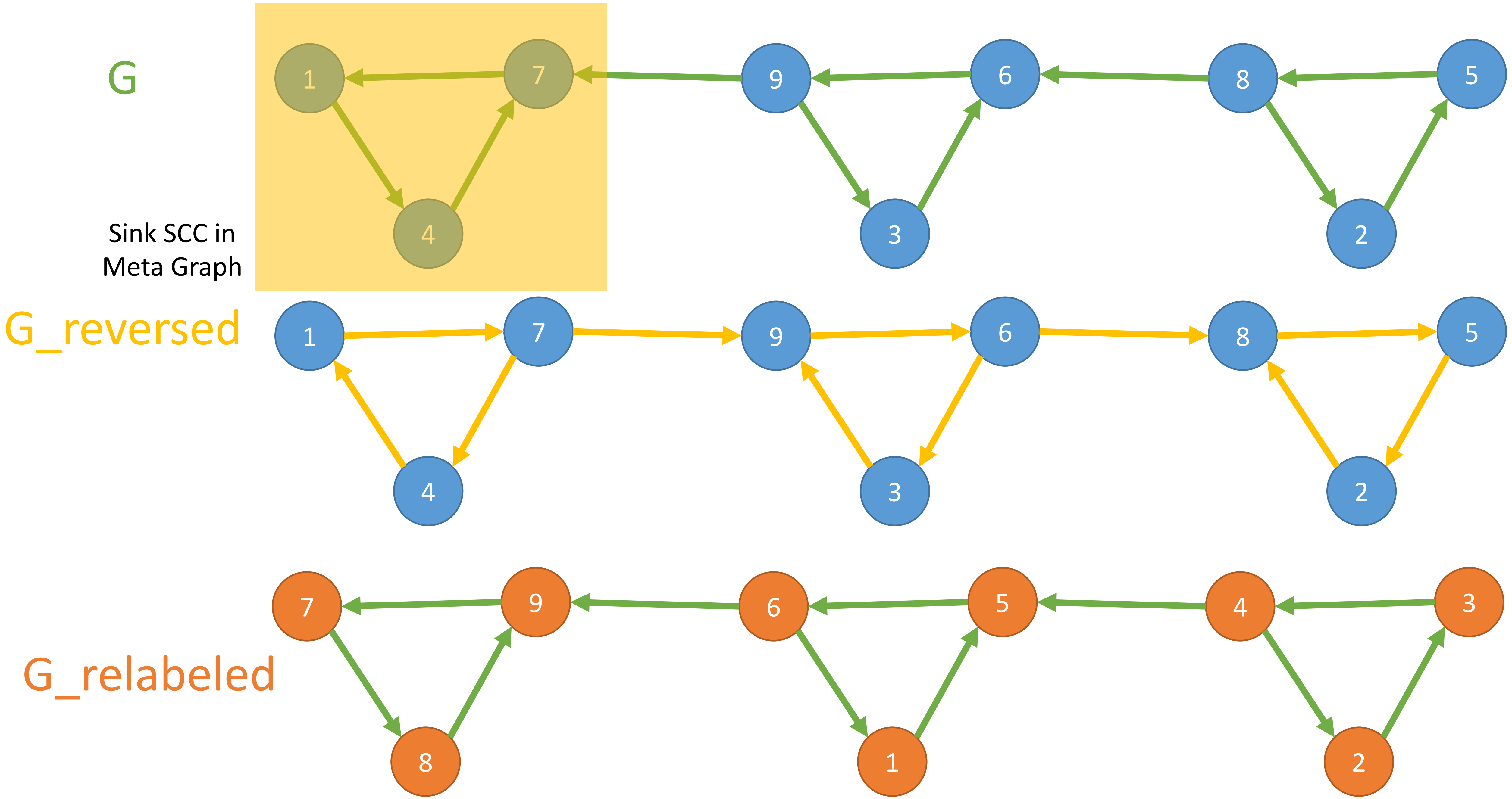


```
FUNCTION KosarajuLoop(G)
   found = {v: FALSE FOR v IN G.vertices}
   label = 0
   labels = {v: NONE FOR v IN G.vertices}
   leaders = {v: NONE FOR v IN G.vertices}

   FOR v IN G.vertices.reverse_order
      IF found[v] == FALSE
         leader = v
         KosarajuDFS(…)

   RETURN labels, leaders
```

```
FUNCTION KosarajuDFS(…)
   found[v] = TRUE
   leaders[v] = leader
   FOR vOther IN G.edges[v]
      IF found[vOther] == FALSE
         KosarajuDFS(…)
   label = label + 1
   labels[v] = label
```

Ignore leaders the first pass
Ignore labels the second pass

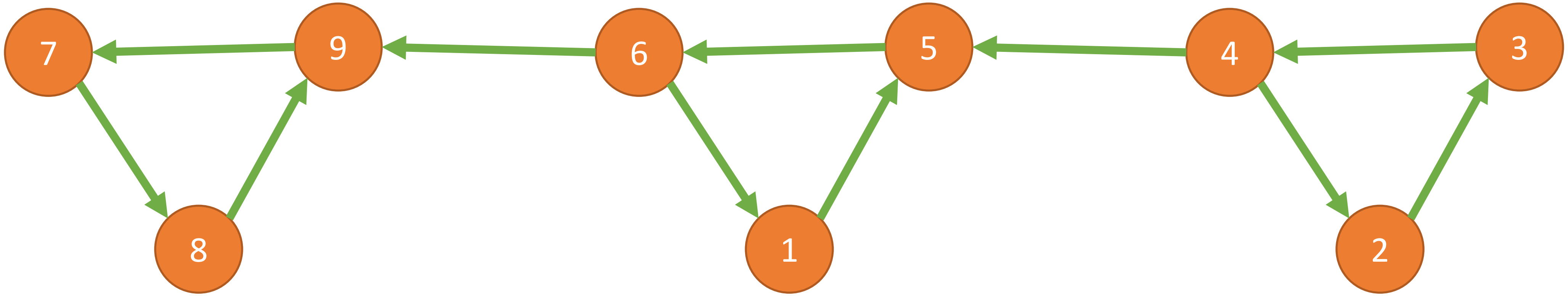




G
1
7
4
9
6
3
8
5
2
Sink SCC in
Meta Graph
G_reversed
1
7
9
6
8
5
4
3
2
G_relabeled
7
9
6
5
4
3
8
1
2

```
FUNCTION Kosaraju(G)
    G_reversed = reverse_graph(G)
    new_labels, _ = KosarajuLoop(G_reversed)

    G_relabeled = relabel_graph(G, new_labels)
    _, leaders = KosarajuLoop(G_relabeled)

    RETURN leaders
```

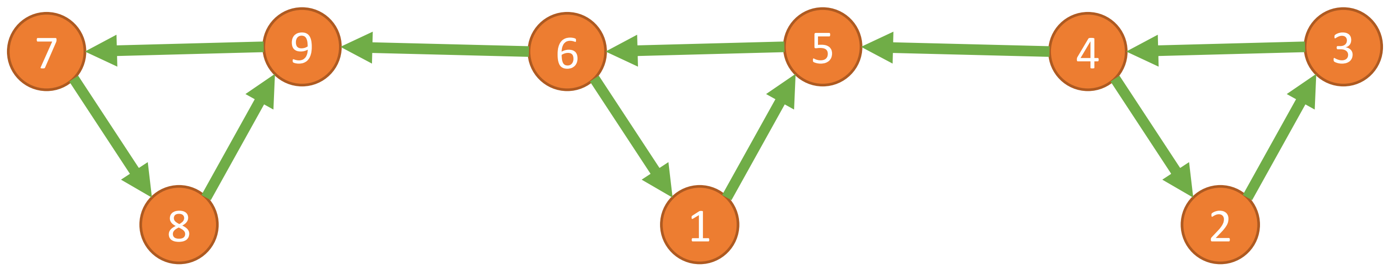
G_relabeled

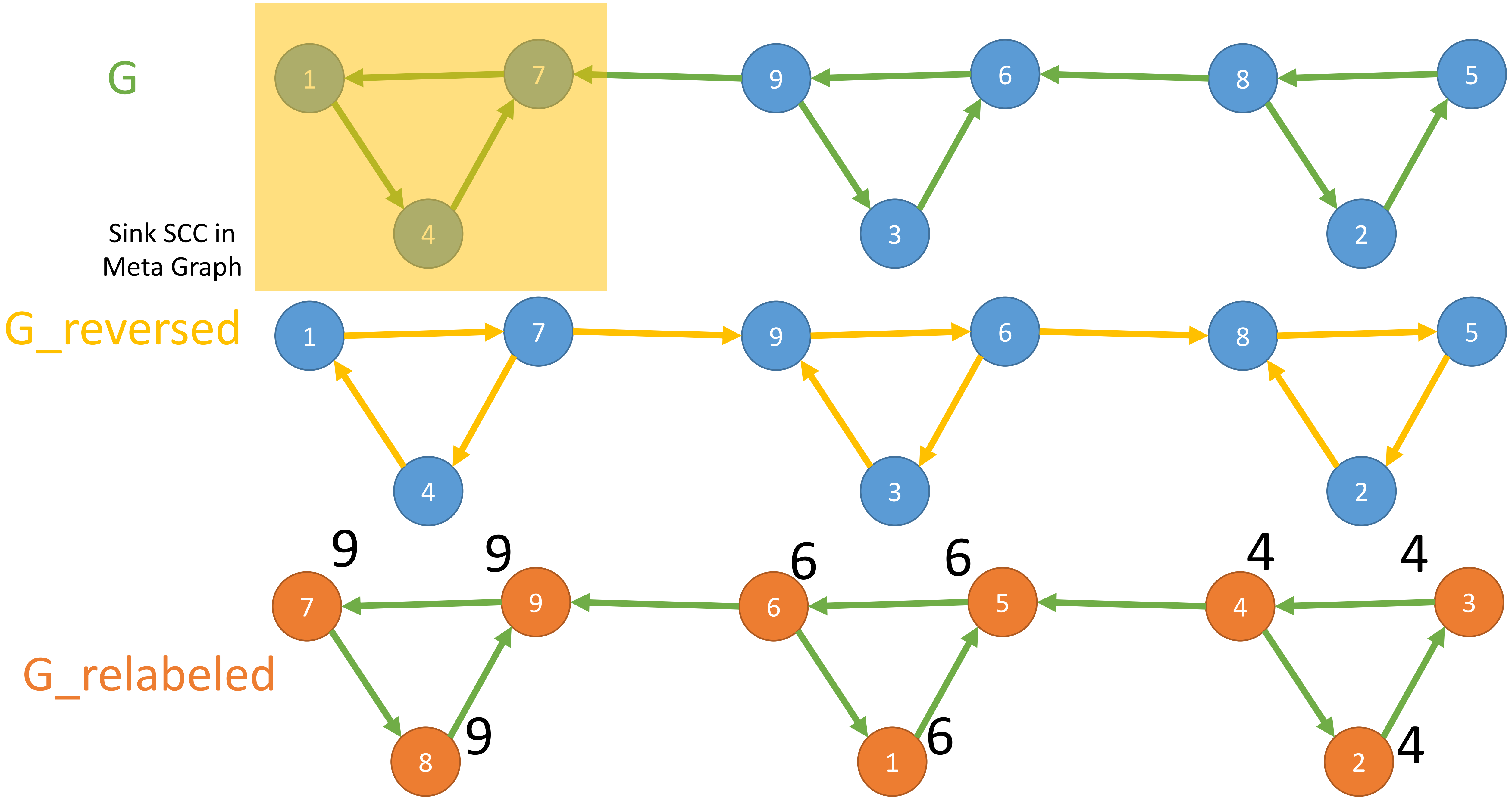


```
FUNCTION KosarajuLoop(G)
    found = {v: FALSE FOR v IN G.vertices}
    label = 0
    labels = {v: NONE FOR v IN G.vertices}
    leaders = {v: NONE FOR v IN G.vertices}

    FOR v IN G.vertices.reverse_order
        IF found[v] == FALSE
            leader = v
            KosarajuDFS(…)

    RETURN labels, leaders
```

```
FUNCTION KosarajuDFS(…)
    found[v] = TRUE
    leaders[v] = leader
    FOR vOther IN G.edges[v]
        IF found[vOther] == FALSE
            KosarajuDFS(…)
    label = label + 1
    labels[v] = label
```

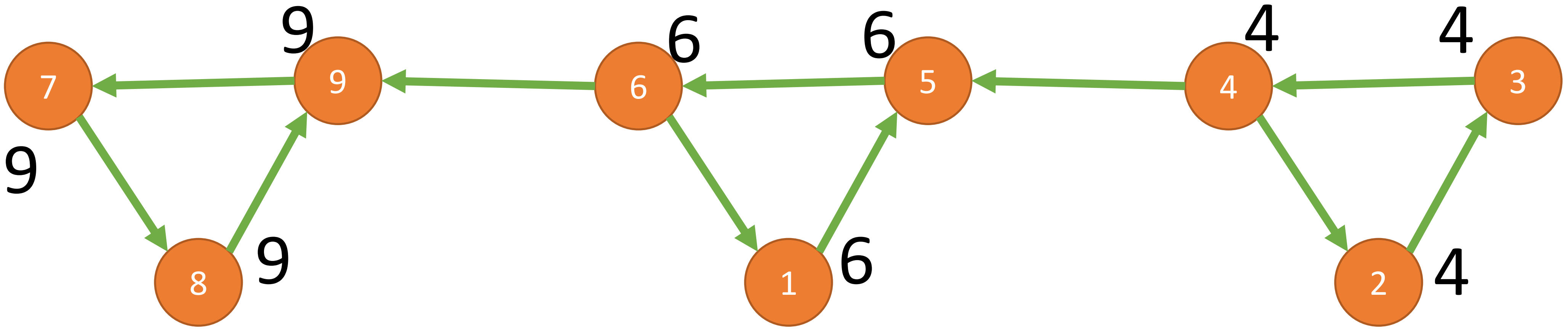
Ignore leaders the first pass
Ignore labels the second pass




G
Sink SCC in
Meta Graph
G_reversed
G_relabeled

```
FUNCTION Kosaraju(G)
    G_reversed = reverse_graph(G)
    new_labels, _ = KosarajuLoop(G_reversed)

    G_relabeled = relabel_graph(G, new_labels)
    _, leaders = KosarajuLoop(G_relabeled)

    RETURN leaders
```

G_relabeled

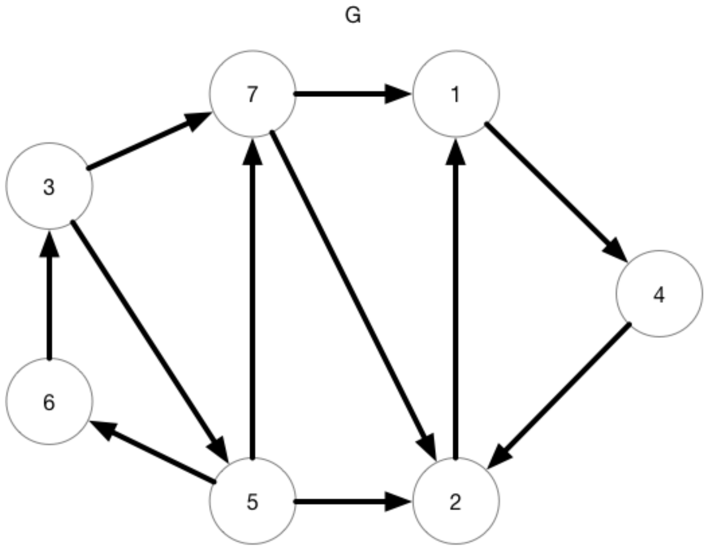


Exercise

```
FUNCTION KosarajuLoop(G)
   found = {v: FALSE FOR v IN G.vertices}
   label = 0
   labels = {v: NONE FOR v IN G.vertices}
   leaders = {v: NONE FOR v IN G.vertices}

   FOR v IN G.vertices.reverse_order
      IF found[v] == FALSE
         leader = v
         KosarajuDFS(G, v, found, label, labels, leader, leaders)

   RETURN labels, leaders
```

```
FUNCTION Kosaraju(G)
   G_reversed = reverse_graph(G)
   new_labels, _ = KosarajuLoop(G_reversed)

   G_relabeled = relabel_graph(G, new_labels)
   _, leaders = KosarajuLoop(G_relabeled)

   RETURN leaders
```

```
FUNCTION KosarajuDFS(G, v, found, label, labels, leader, leaders)
   found[v] = TRUE
   leaders[v] = leader
   FOR vOther IN G.edges[v]
      IF found[vOther] == FALSE
         KosarajuDFS(G, v, found, label, labels, leader, leaders)
   label = label + 1
   labels[v] = label
```

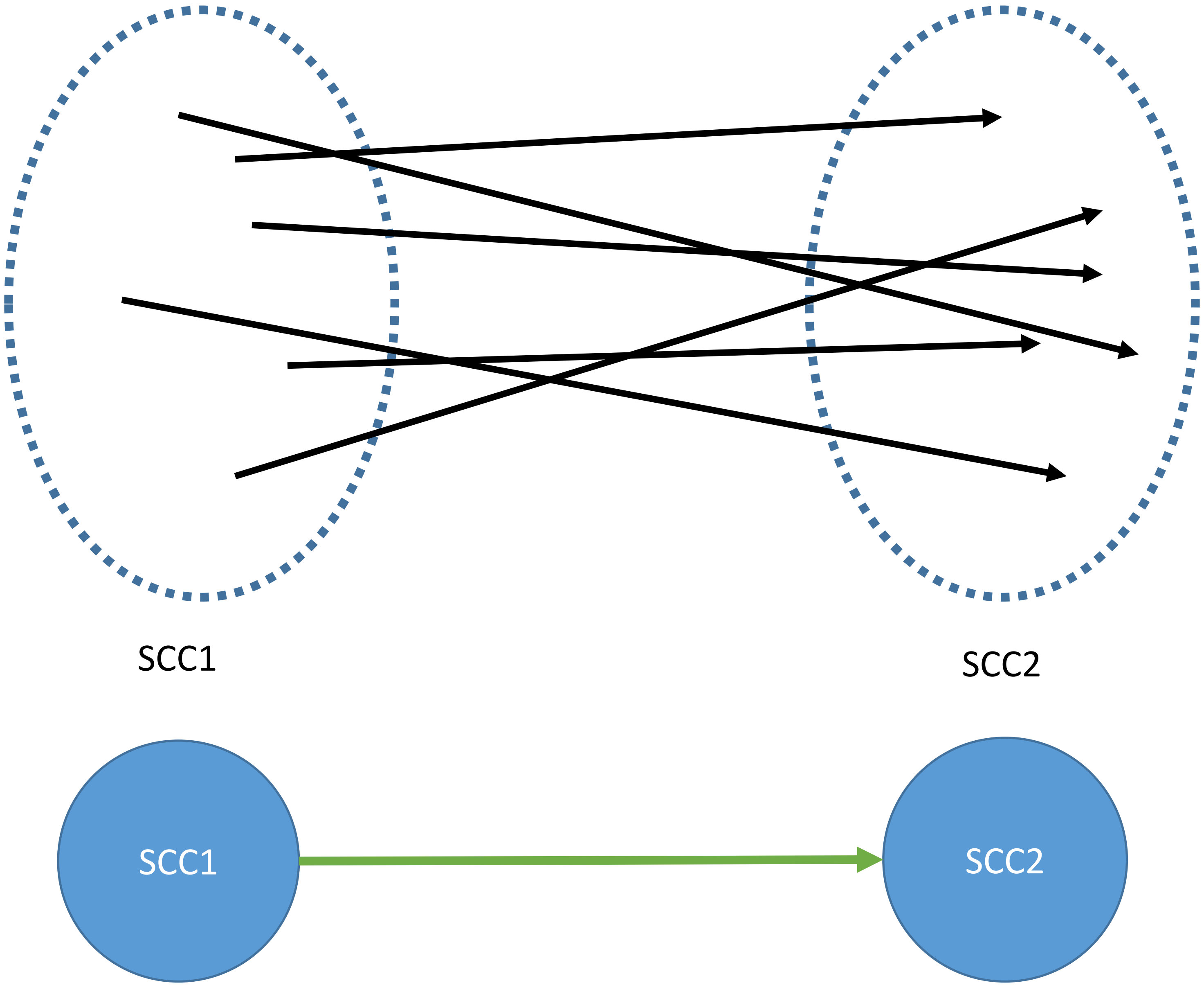


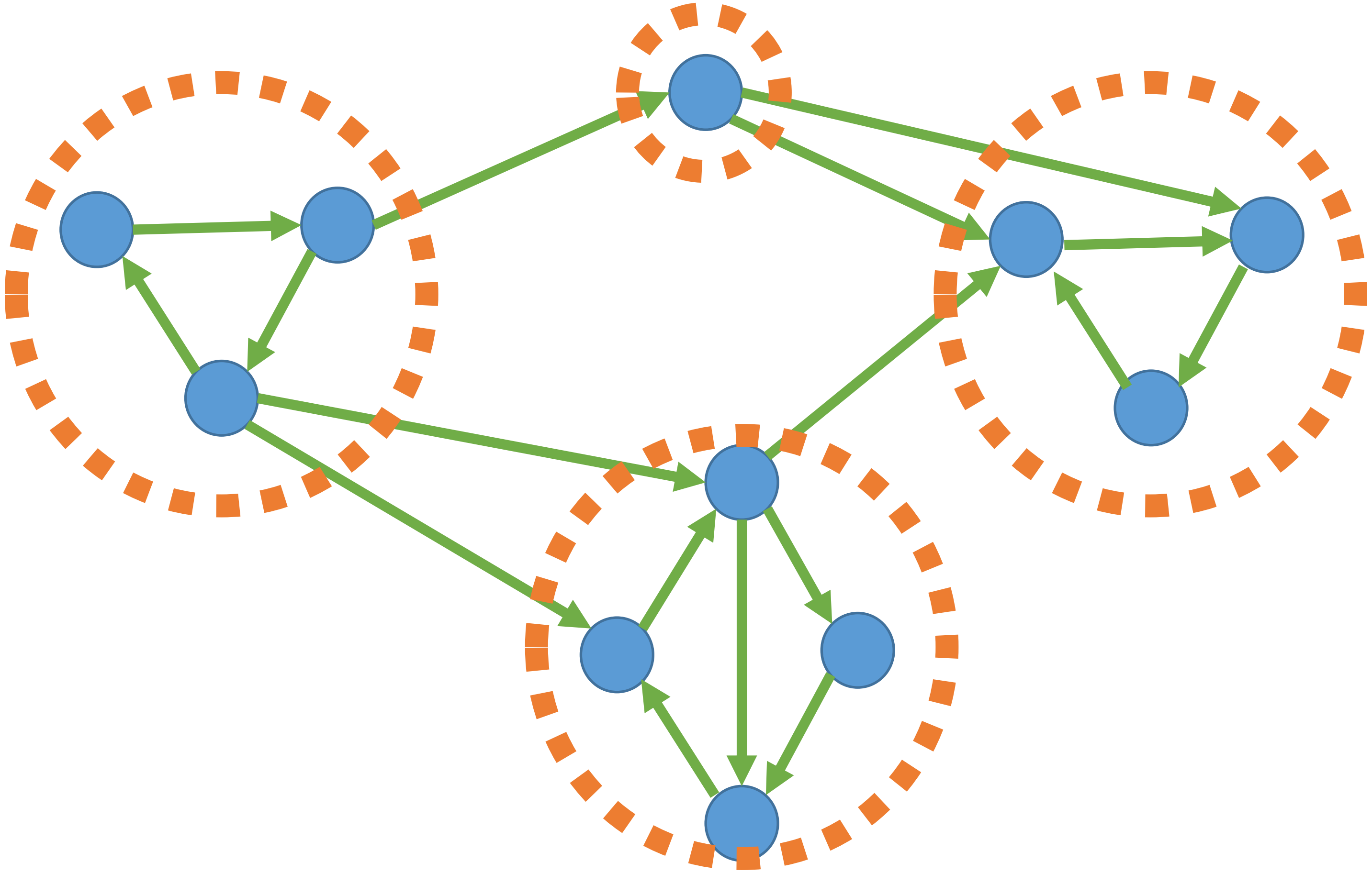
Why does this work?

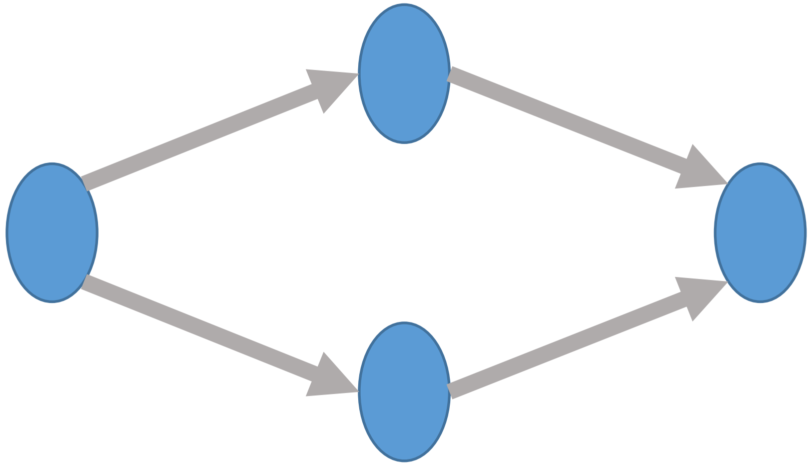
- Does this work for all graphs, or just this example?

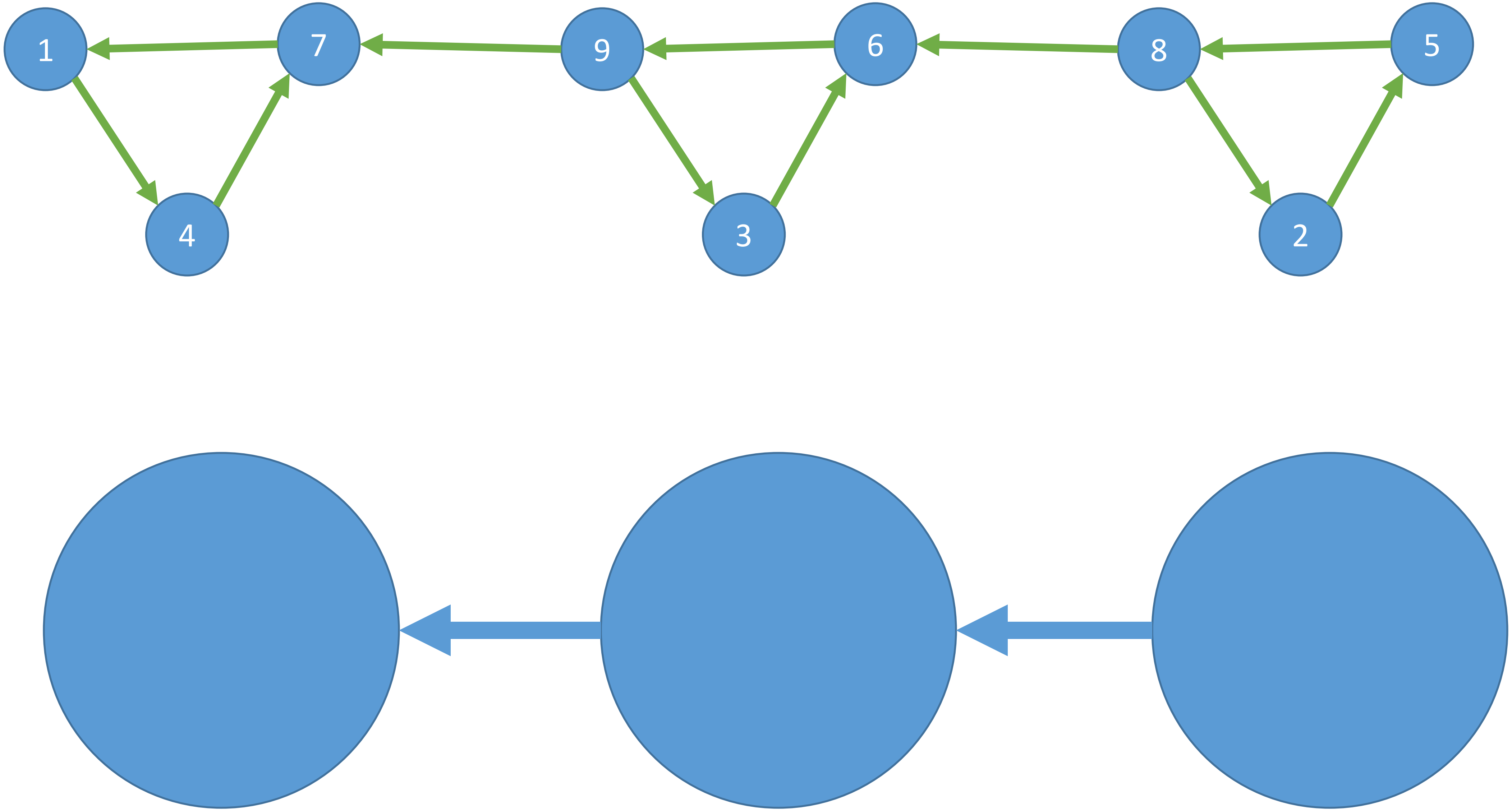
- The SCCs of G create an acyclic “meta-graph”

- For the “meta-graph”
  - Vertices correspond to the SCCs
  - Edges correspond to paths among the SCCs


SCC1
SCC2
SCC1
SCC2

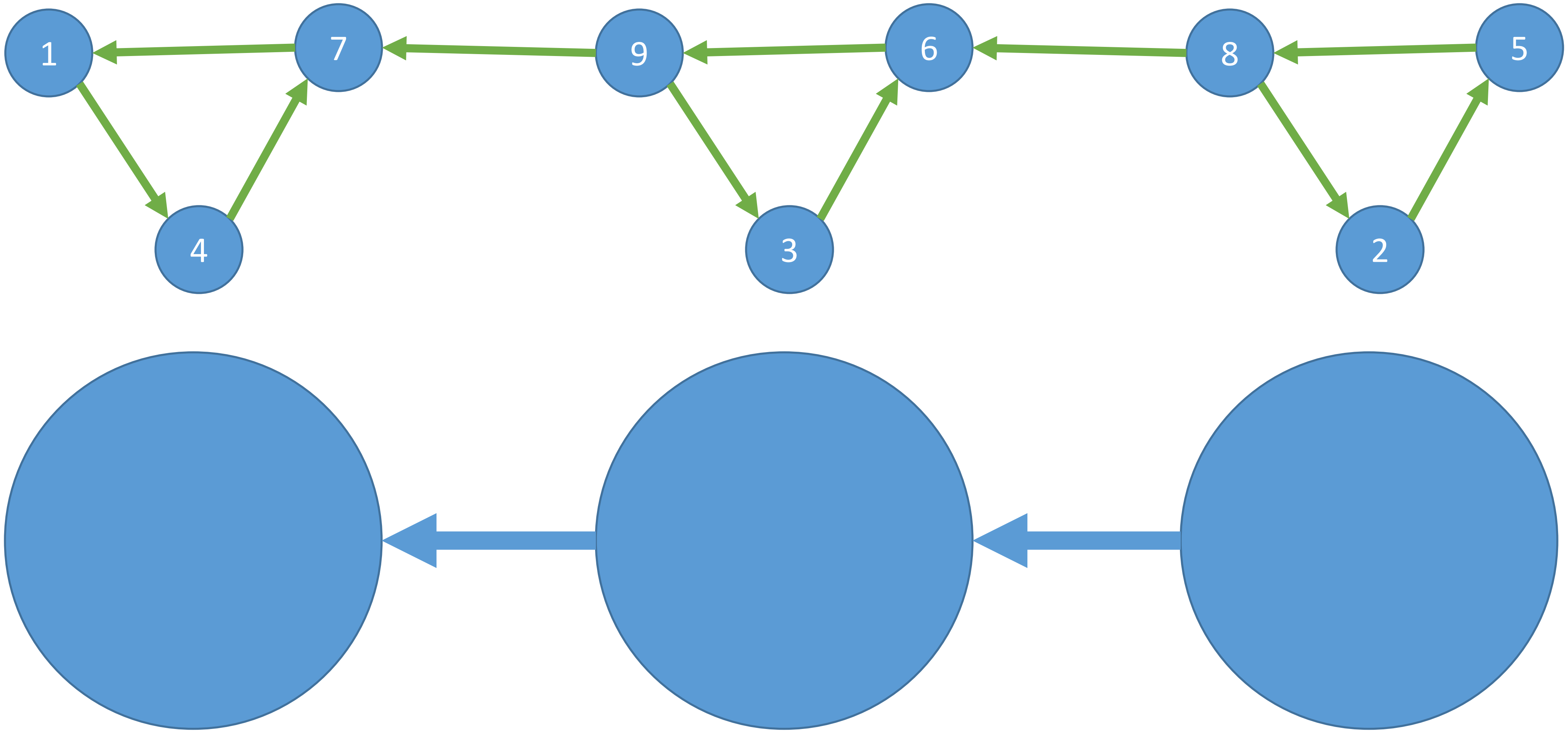




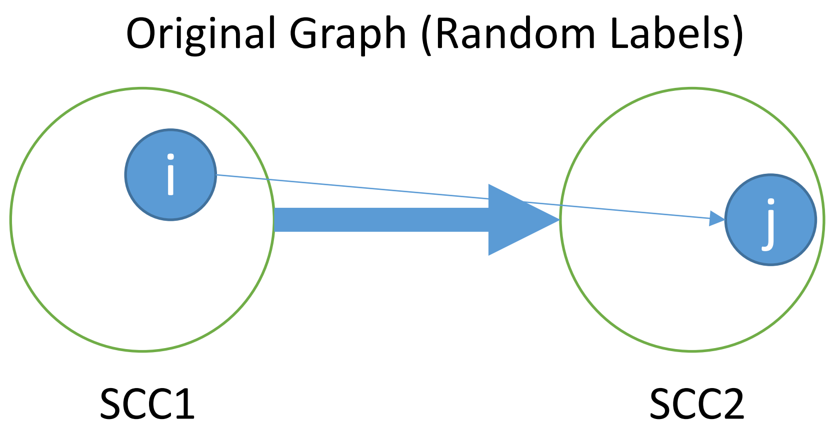


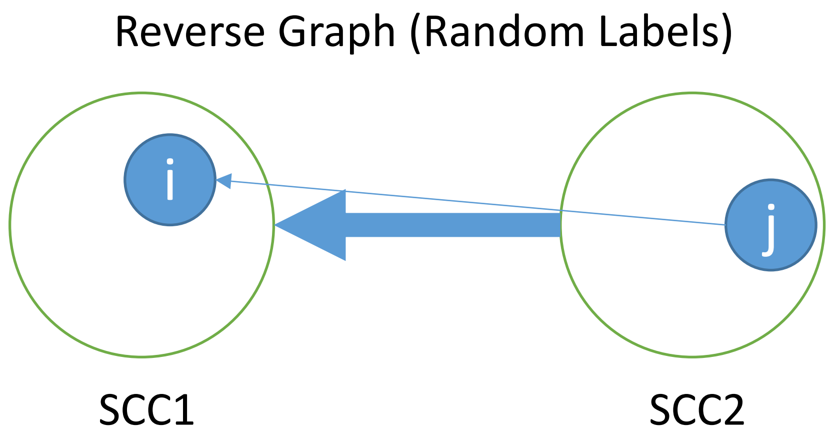
1
7
9
6
8
5
4
3
2

How do we know that the SCC based meta-graph is acyclic?



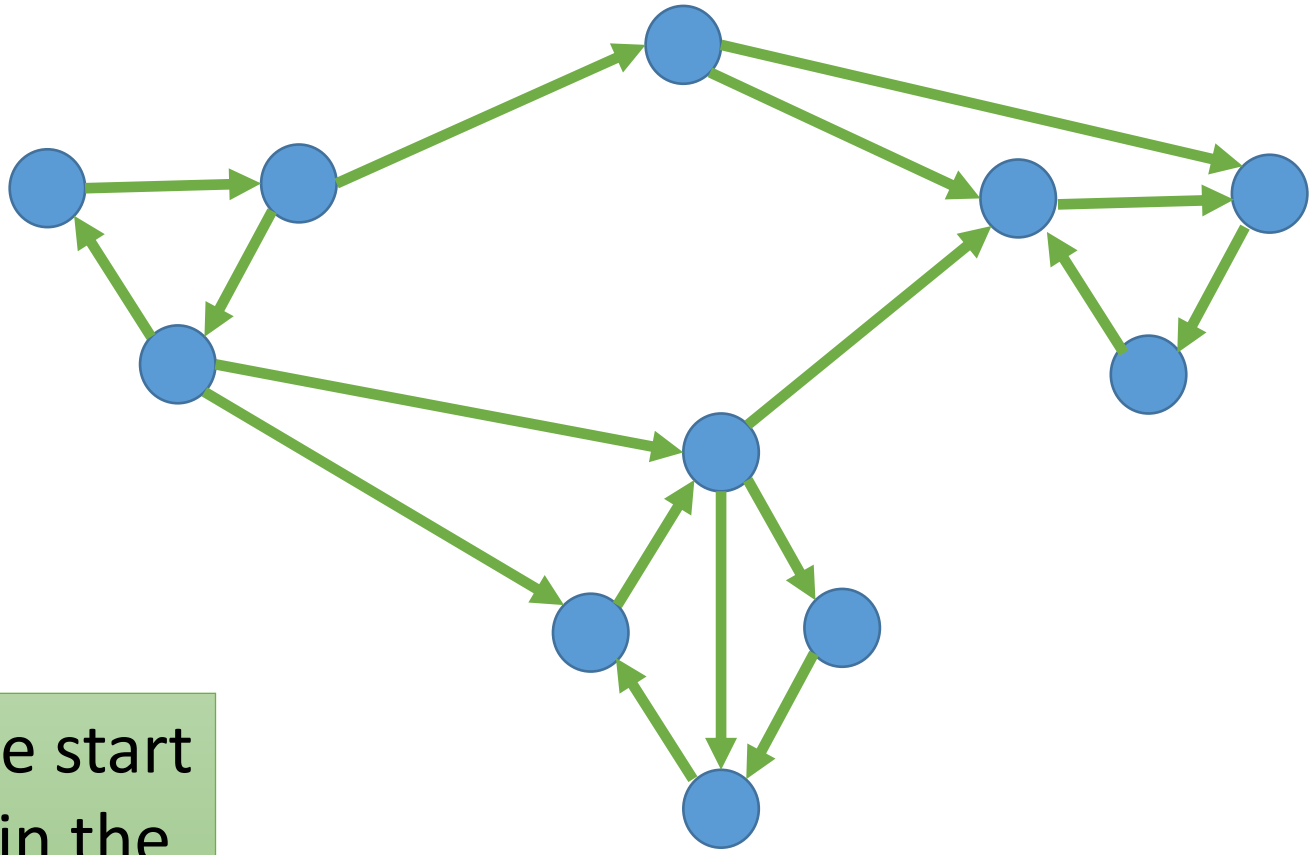
Key Lemma



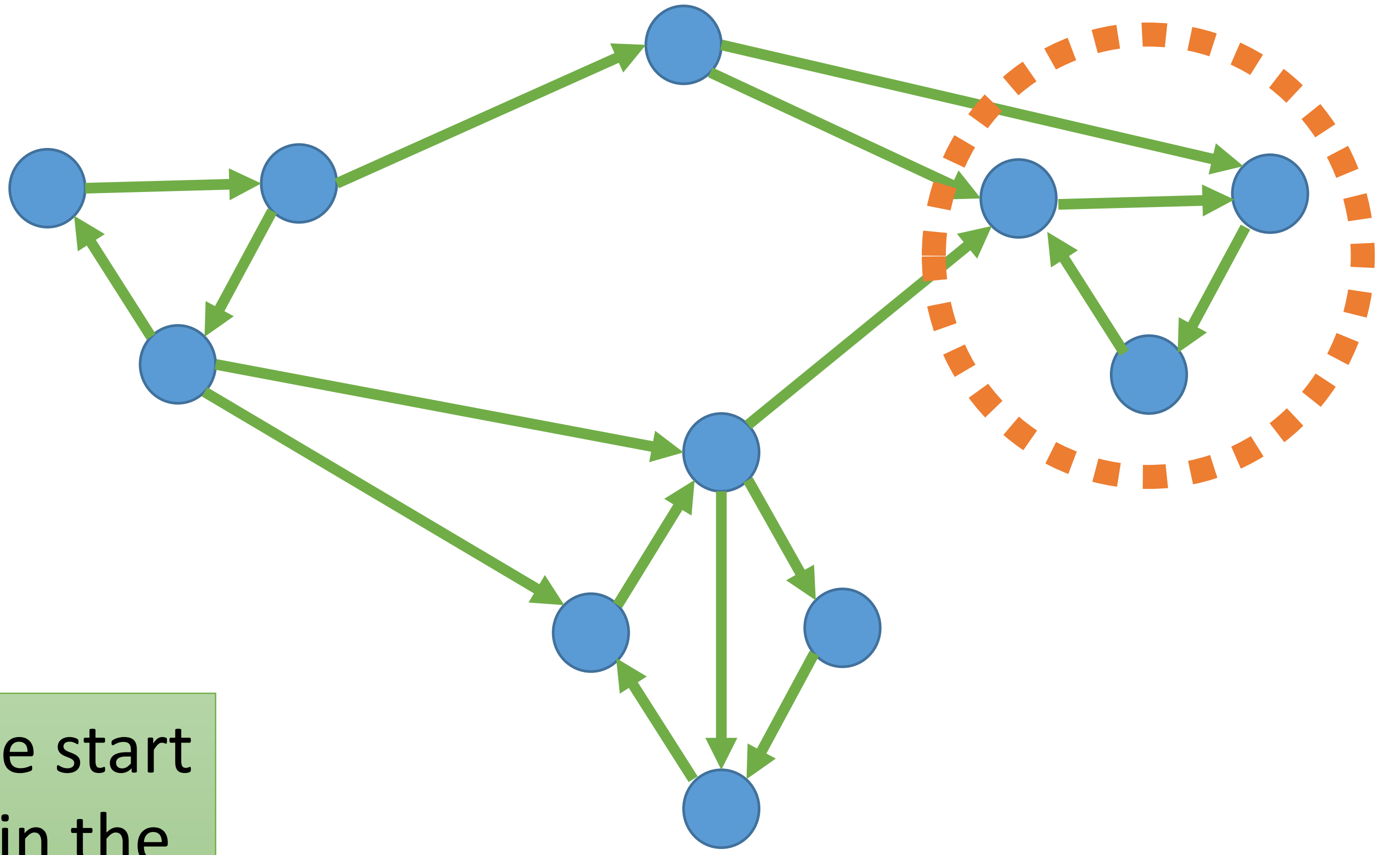


- Consider the two adjacent SCCs in the meta-graph above
- Now consider the re-labeling found from the reverse graph
- Let f(v) = the re-labeling resulting from `KosarajuLoop(G_reversed)`
- Then <u>`max[f(.) in SCC1] < max[f(.) in SCC2]`</u>
- Corollary: the maximum f-value must lie in a “sink SCC” of the original graph

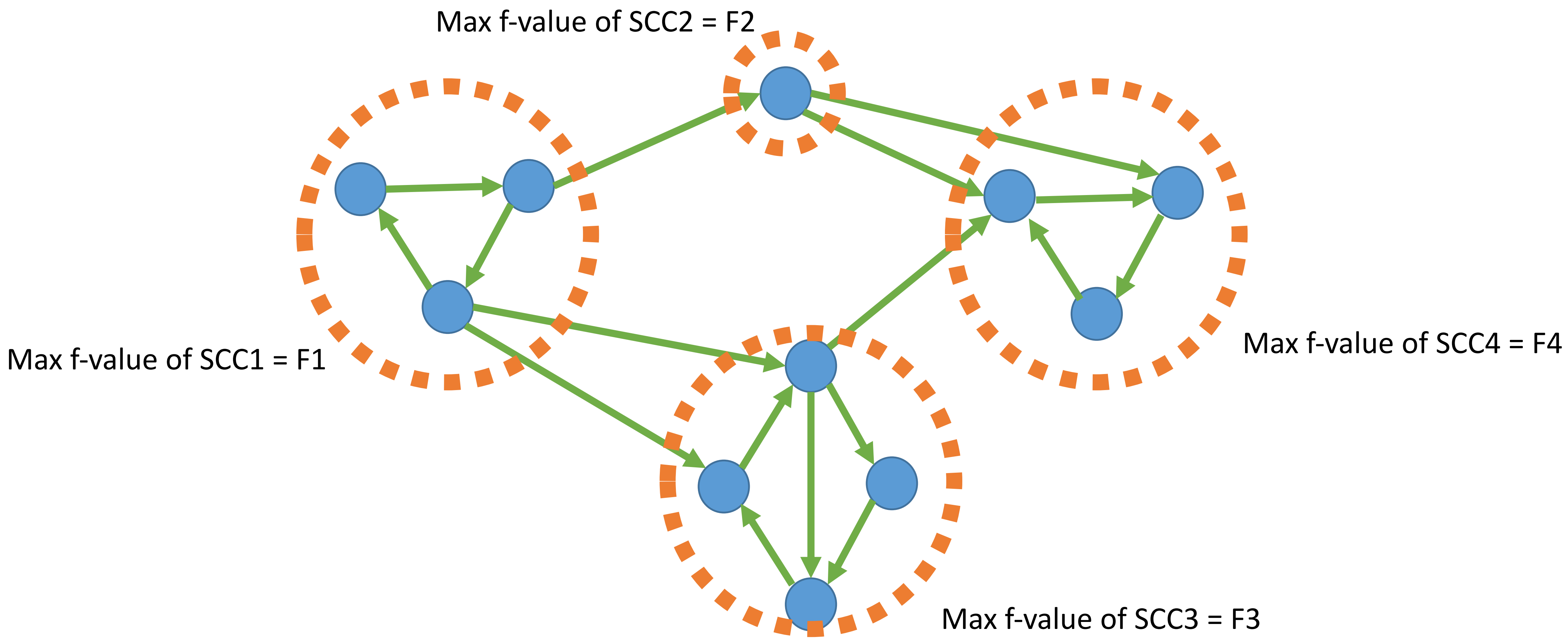
```
FUNCTION KosarajuDFS(…)
    found[v] = TRUE
    leaders[v] = leader
    FOR vOther IN G.edges[v]
        IF found[vOther] == FALSE
            KosarajuDFS(…)
    label = label + 1
    labels[v] = label
```



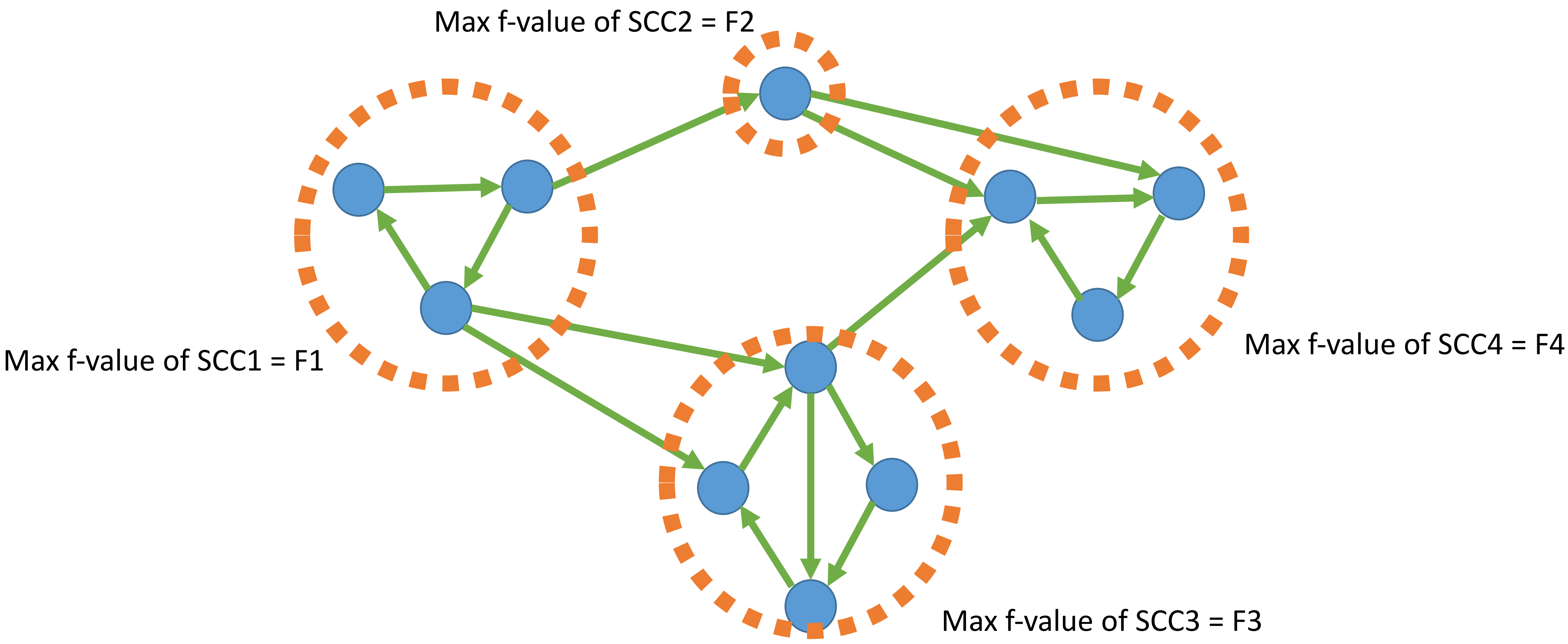
Where should we start labeling leaders in the second pass?



Where should we start labeling leaders in the second pass?



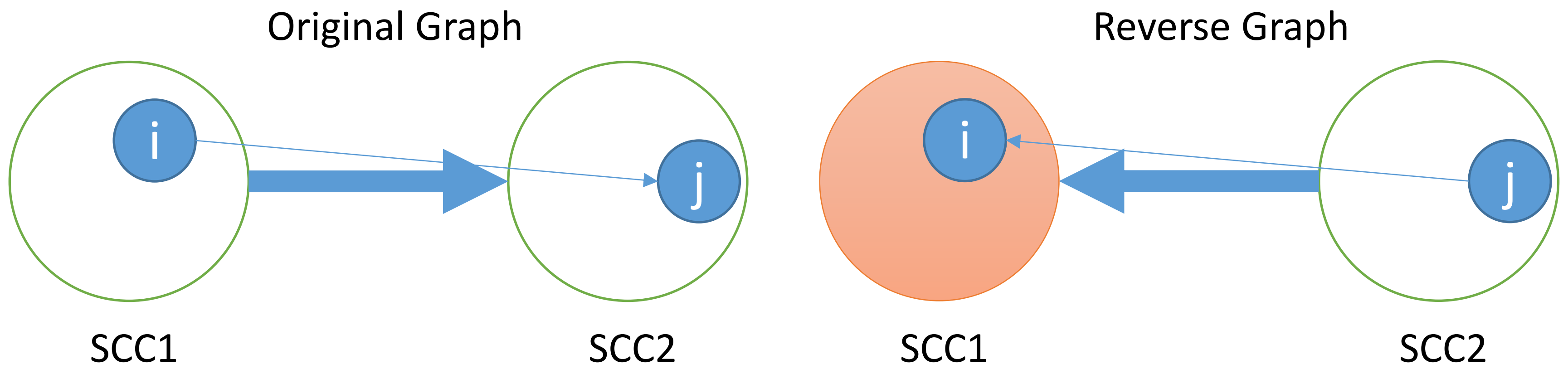
Then F1 < {F2, F3} < F4



Then F1 < {F2, F3} < F4

What would happen if SCC4 had a link back to SCC3?

Proof of Lemma

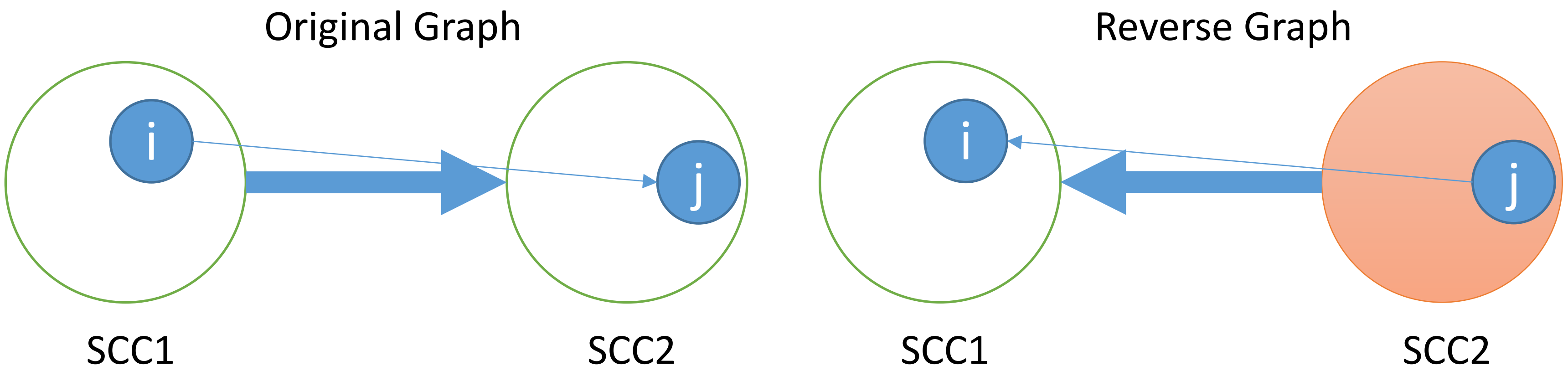


Case 1: consider the case when the first vertex that we explore is in SCC1

- Then all SCC1 is explored before SCC2
- Therefore, all f-values in SCC1 are less than all f-values in SCC2
- So, in the original graph we will start in SCC2 (the sink)

```
FUNCTION KosarajuDFS(…)
    found[v] = TRUE
    leaders[v] = leader
    FOR vOther IN G.edges[v]
        IF found[vOther] == FALSE
            KosarajuDFS(…)
    label = label + 1
    labels[v] = label
```

Proof of Lemma



Case 2: consider the case when the first vertex that we explore is in SCC2

- All other vertices in SSC2 are explored before vertex j
- All vertices in SSC1 are explored before vertex j
- Therefore, all f-values in SSC1 and SSC2 are less than the f-value of vertex j
- So, in the original graph we will start at vertex j in SSC2 (the sink)

```
FUNCTION KosarajuDFS(…)
   found[v] = TRUE
   leaders[v] = leader
   FOR vOther IN G.edges[v]
      IF found[vOther] == FALSE
         KosarajuDFS(…)
   label = label + 1
   labels[v] = label
```

What does this mean?

- We'll start the <u>second</u> `KosarajuLoop` at an "SCC sink"
- That sink will then be removed (by marking all vertices in the SCC as explored) and we'll next move to the newly created sink
- And so on

Kosaraju's Algorithm Summary

Computes the SCCs in O(m + n) time **(linear!)**

1. Create a reverse version of the G called G_reversed

2. Run KosarajuLoop on G_reversed
   - Create a topological ordering on the meta graph

3. Create a relabeled version of the G called G_relabeled

4. Run KosarajuLoop on G_relabeled
   - Find all nodes with the same "leader"