Applications of Depth First Search

Thomas Schwarz, SJ

Topological Sort

- Recall topological sort
  - We are given a directed graph
  - Want to order all vertices such that no edge goes from a higher-numbered vertex to a lower-numbered vertex
  - If this is impossible, then we have a cycle
  - So, our algorithm also detects whether there is a cycle in a directed graph
  - We use DFS for an even better algorithm

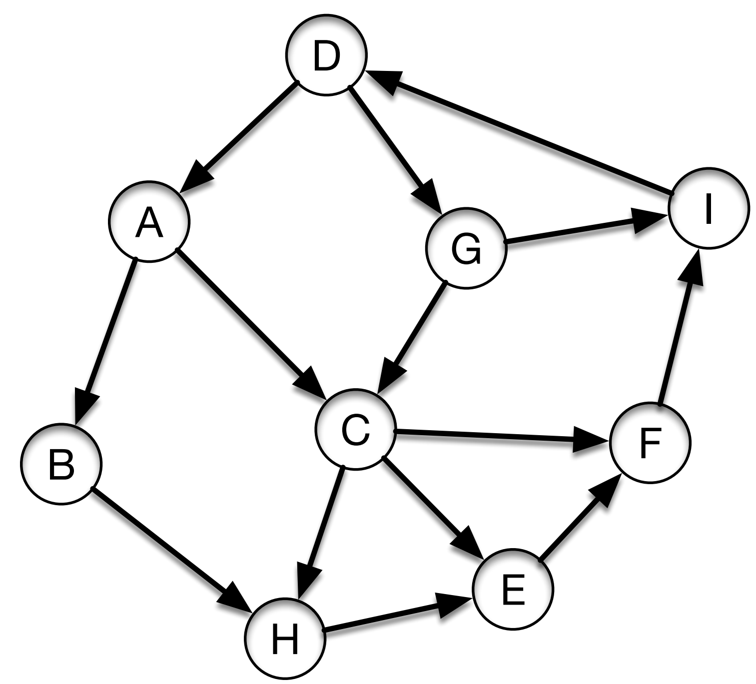
Topological Sort

- Run DFS on all nodes
  - Order nodes according to finish time in descending order

Topological Sort

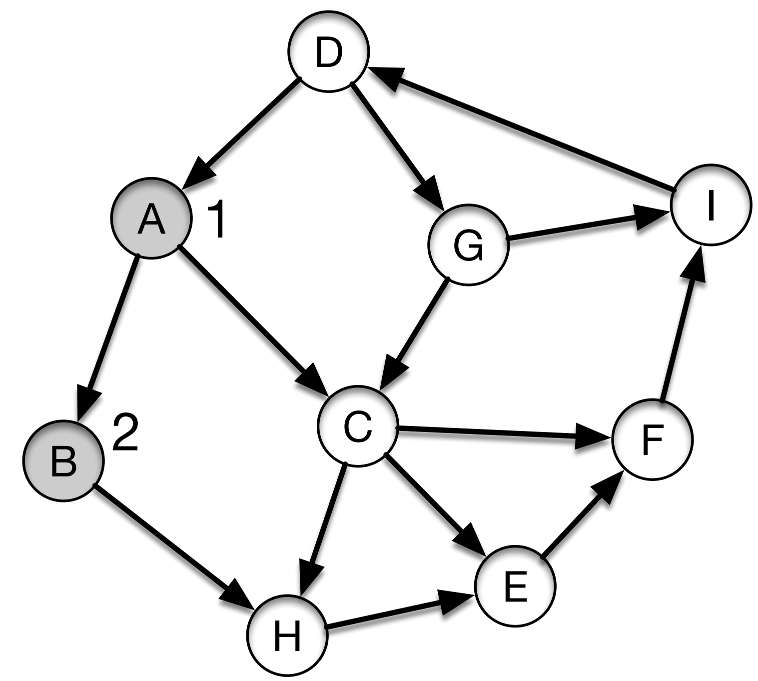
- Start in A
  - Order adjacency lists alphabetically

```
A: B,C
B: H
C: E,F,H
D: A,G
E: F
F: I
G: C
H:
I: D
```



Topological Sort

- Visit B

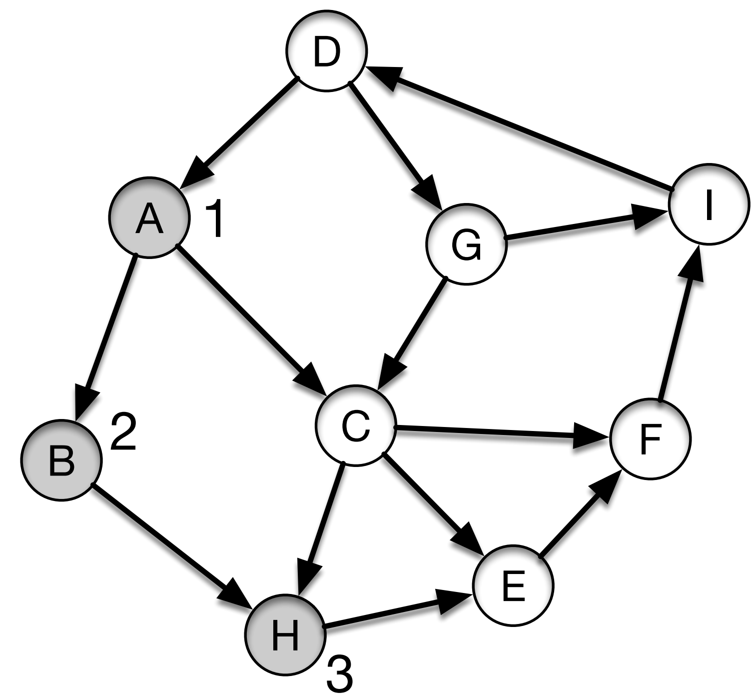



```
A: B,C
B: H
C: E,F,H
D: A,G
E: F
F: I
G: C
H:
I: D
```

```
visit(B)
visit(A)
```

Topological Sort

- Visit H

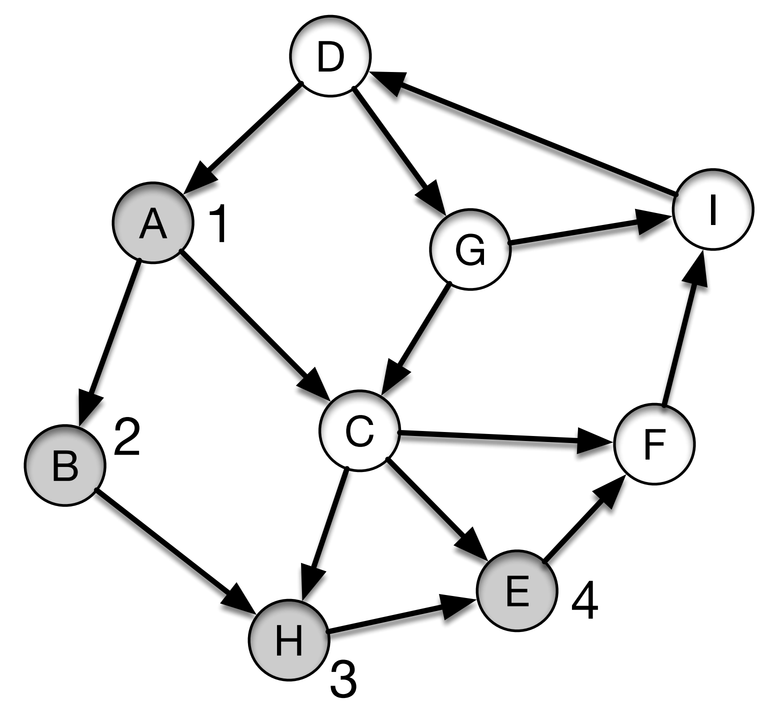



```
A: B,C
B: H
C: E,F,H
D: A,G
E: F
F: I
G: C
H:
I: D
```

```
visit(H)
visit(B)
visit(A)
```

Topological Sort

- Visit E

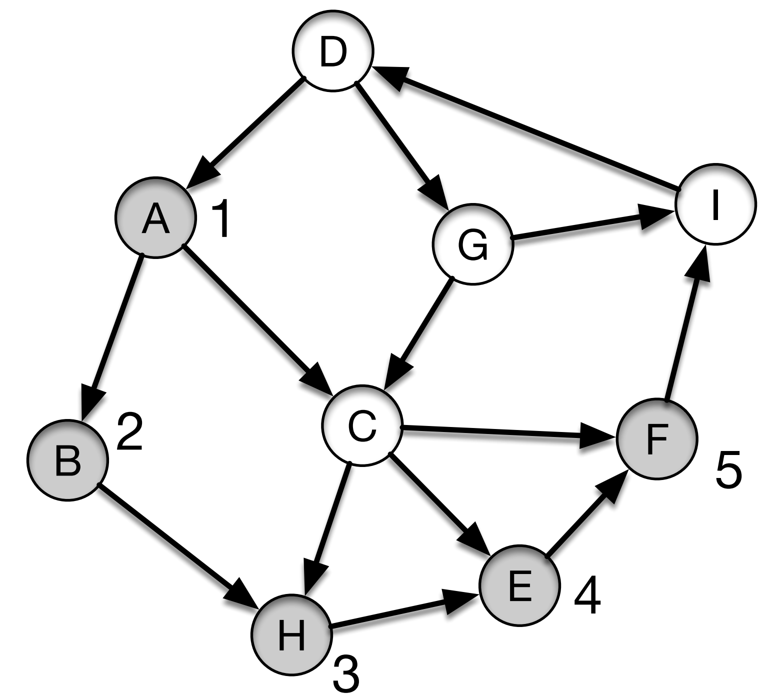



```
A: B,C
B: H
C: E,F,H
D: A,G
E: F
F: I
G: C
H:
I: D
```

```
visit(E)
visit(H)
visit(B)
visit(A)
```

Topological Sort

- Visit F

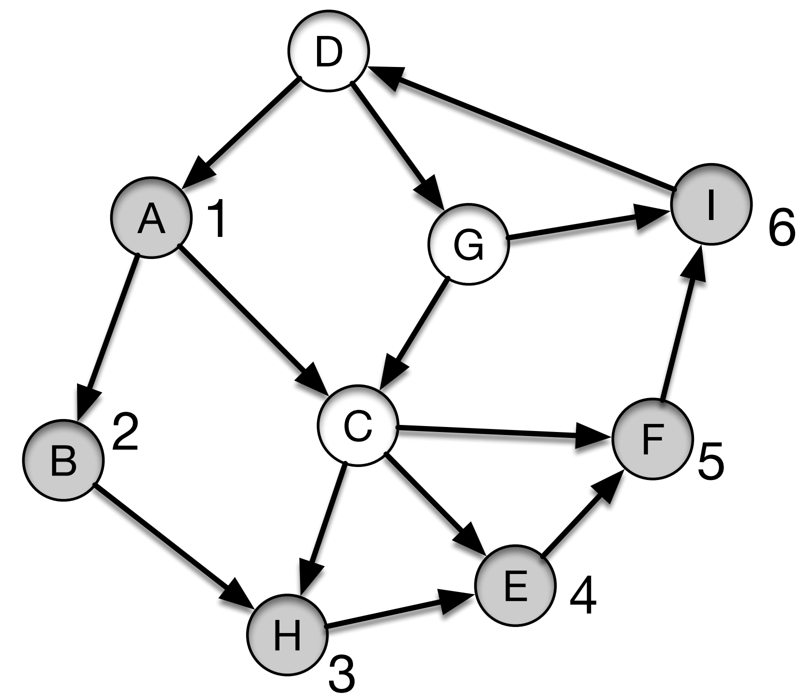



```
A: B,C
B: H
C: E,F,H
D: A,G
E: F
F: I
G: C
H:
I: D
```

```
visit(F)
visit(E)
visit(H)
visit(B)
visit(A)
```

Topological Sort

- Visit I

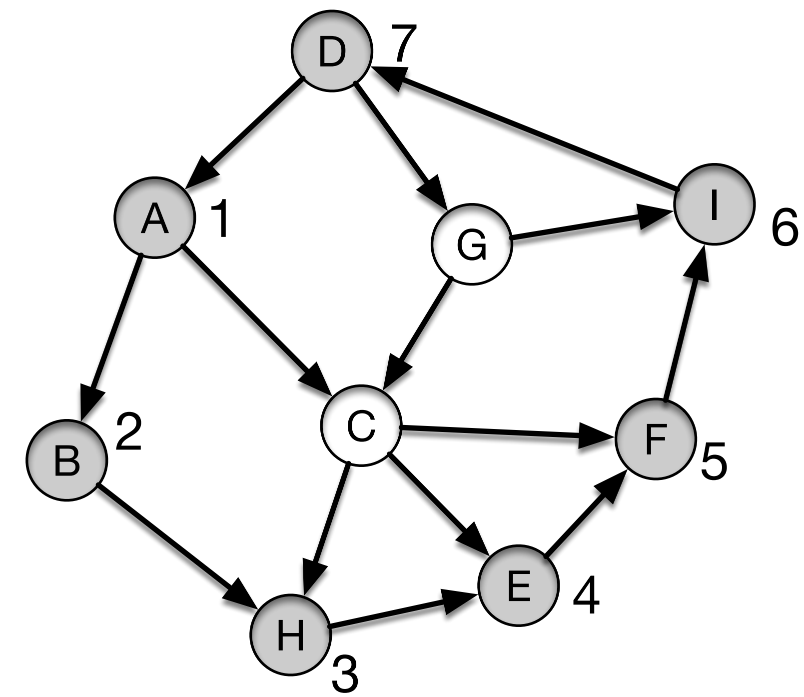



```
A: B,C
B: H
C: E,F,H
D: A,G
E: F
F: I
G: C
H:
I: D
```

```
visit(I)
visit(F)
visit(E)
visit(H)
visit(B)
visit(A)
```

Topological Sort

- Visit D

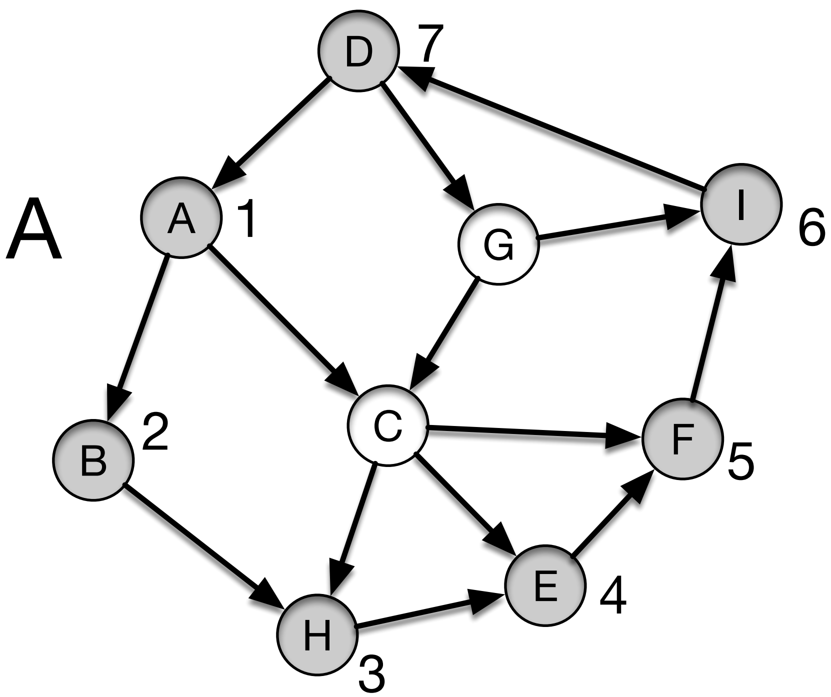



```
A: B,C
B: H
C: E,F,H
D: A,G
E: F
F: I
G: C
H:
I: D
```

```
visit(D)
visit(I)
visit(F)
visit(E)
visit(H)
visit(B)
visit(A)
```

Topological Sort

- At this point:
  - The adjacency list of D starts with A
  - A is gray
  - This edge becomes a back edge!
  - And shows that there is a cycle

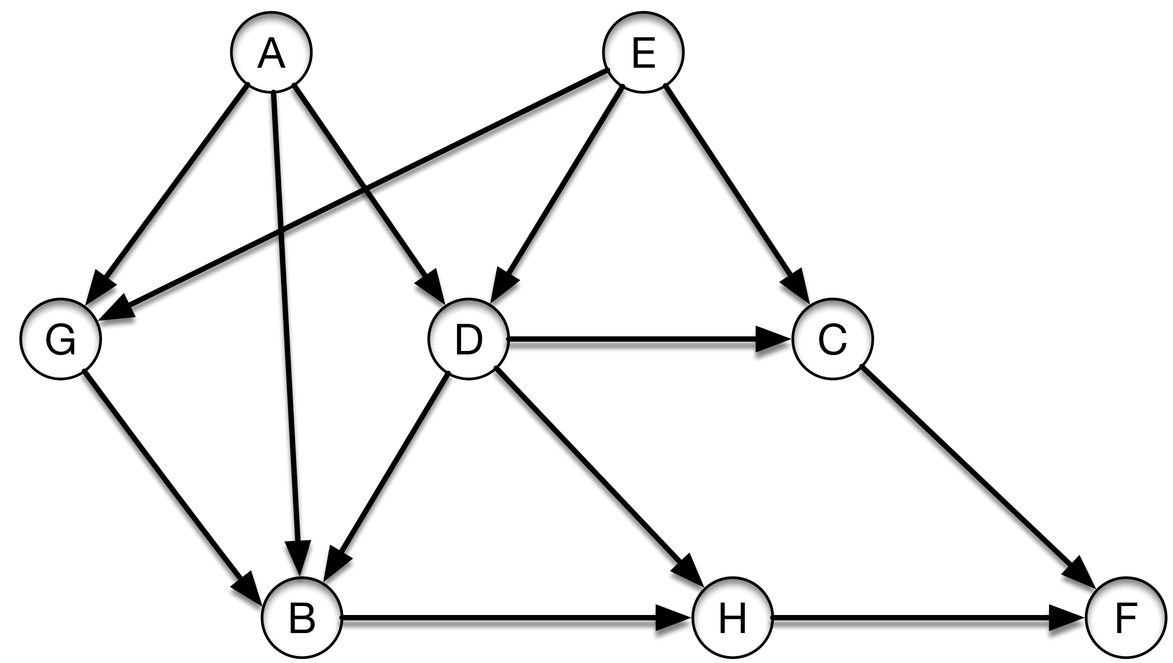



```
A: B,C
B: H
C: E,F,H
D: A,G
E: F
F: I
G: C
H:
I: D
```

```
visit(D)
visit(I)
visit(F)
visit(E)
visit(H)
visit(B)
visit(A)
```

Topological Sort

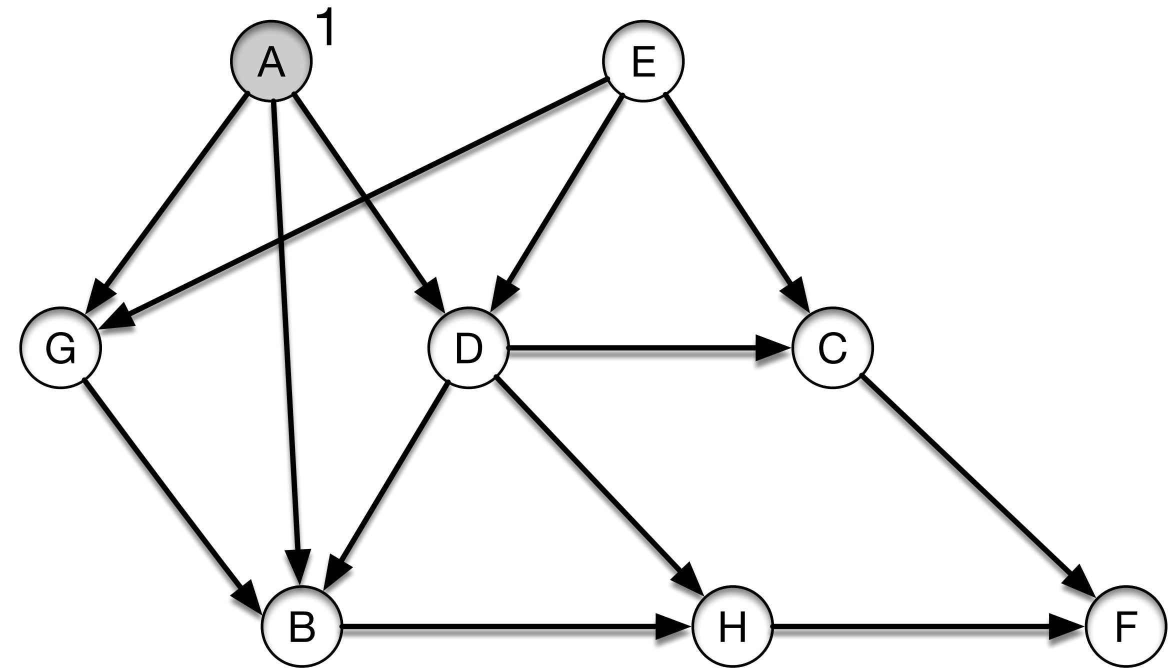
- A different example
  - Start in A




```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```

Topological Sort

- A different example
  - Start in A

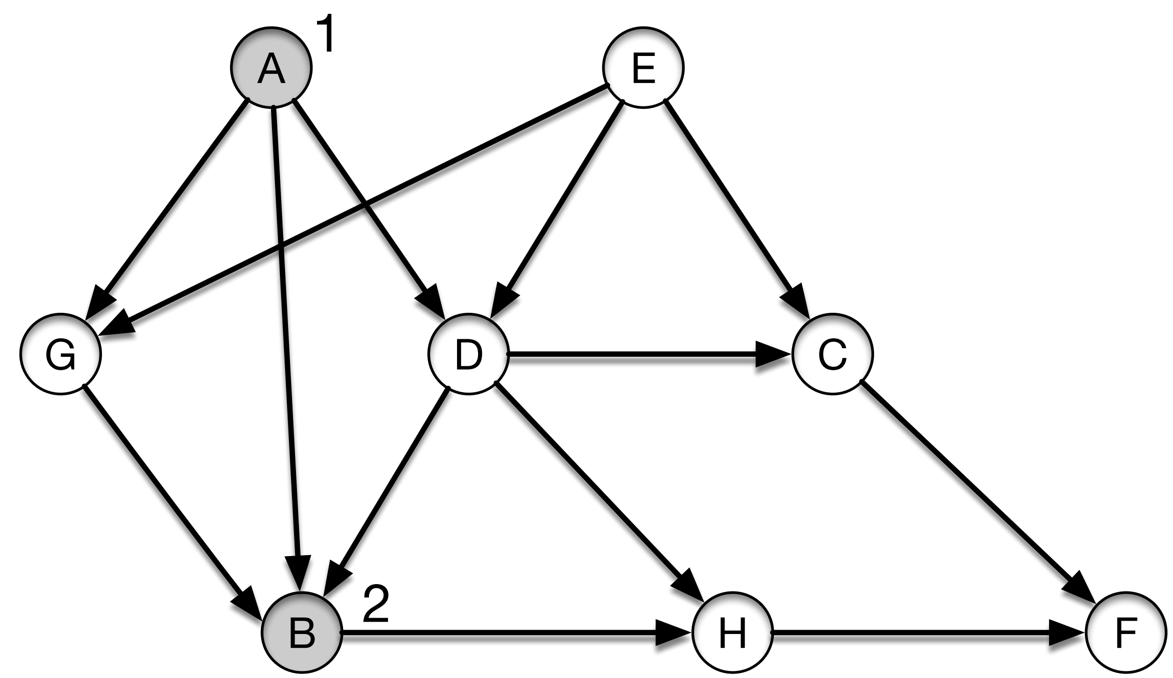



```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```

`visit(A)`

Topological Sort

- Visit B

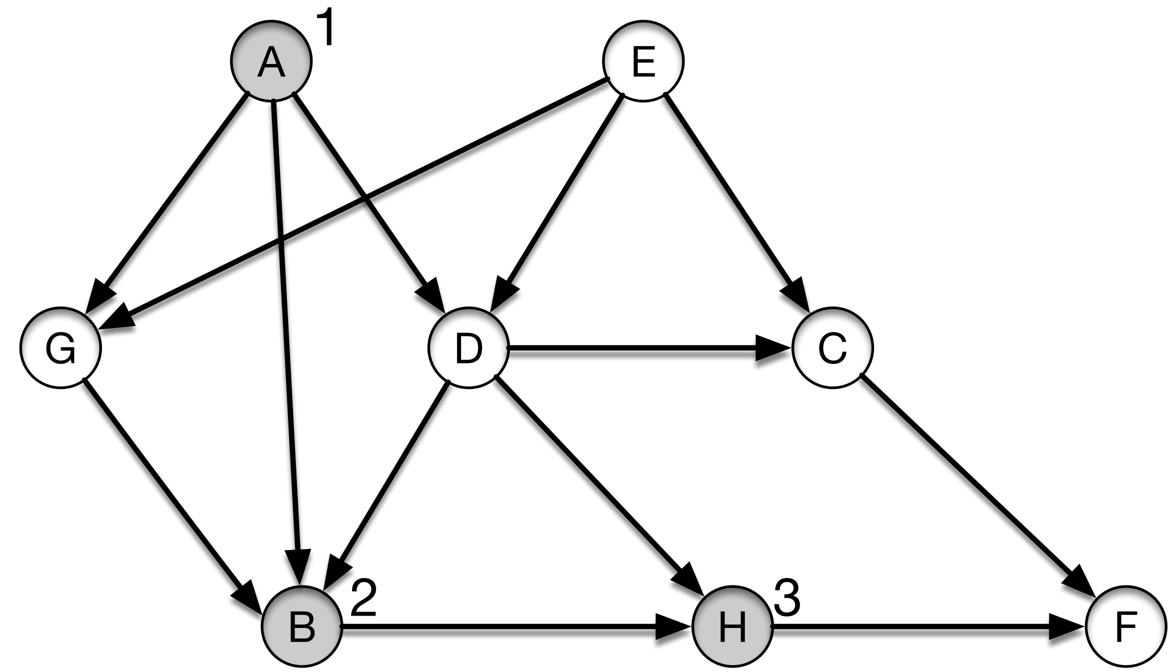



```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```

```
visit(B)
visit(A)
```

Topological Sort

- Visit H

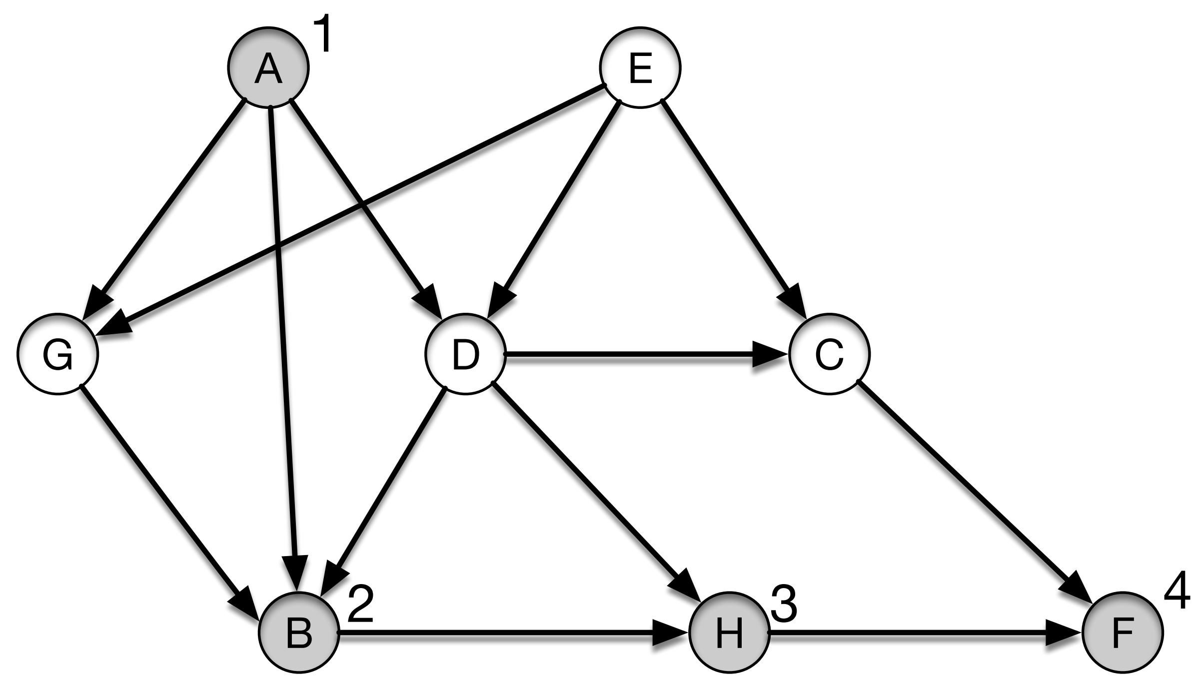



```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```

```
visit(H)
visit(B)
visit(A)
```

Topological Sort

- Visit F

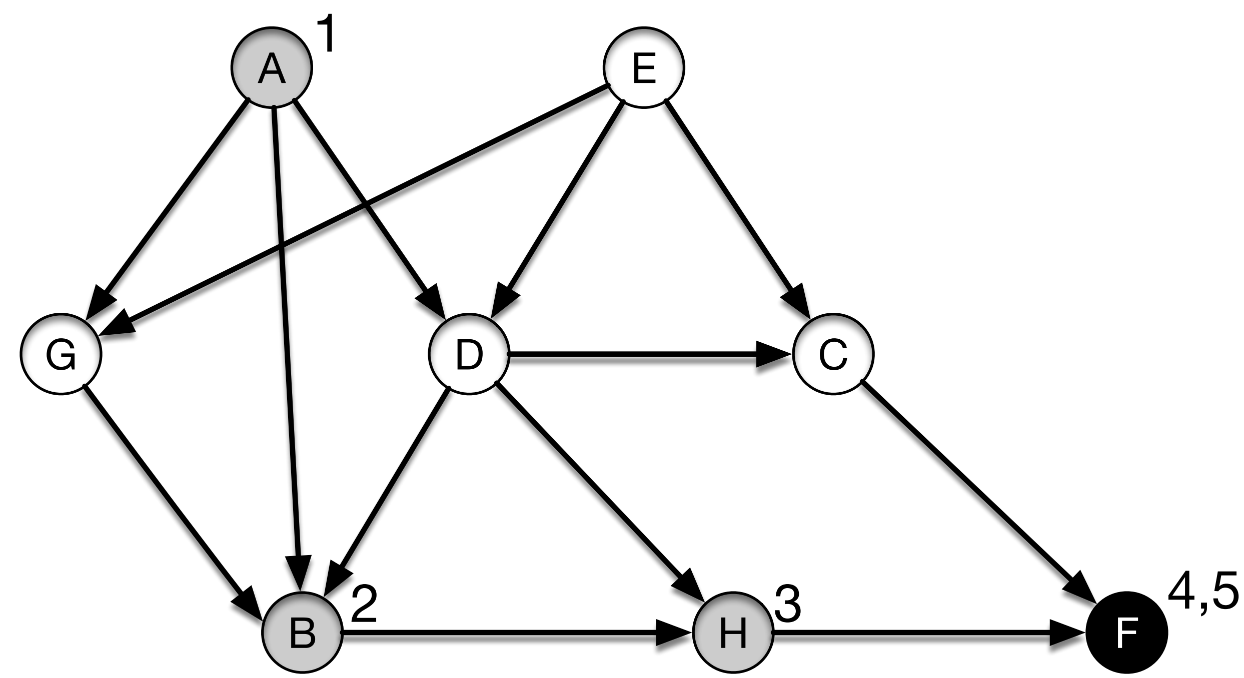



```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```

```
visit(F)
visit(H)
visit(B)
visit(A)
```

Topological Sort

- Finish F
- Push F at front: [F]

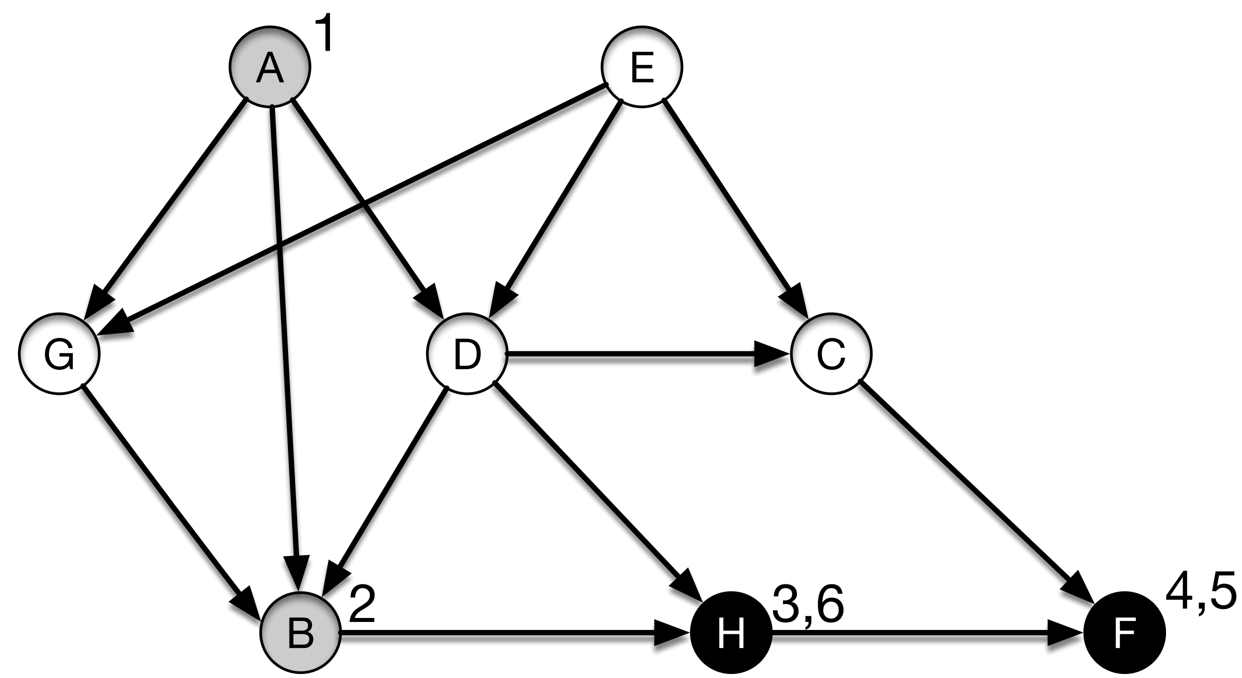



```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```

```
visit(F)
visit(H)
visit(B)
visit(A)
```

Topological Sort

- Finish H
- Push H at front: [H, F]

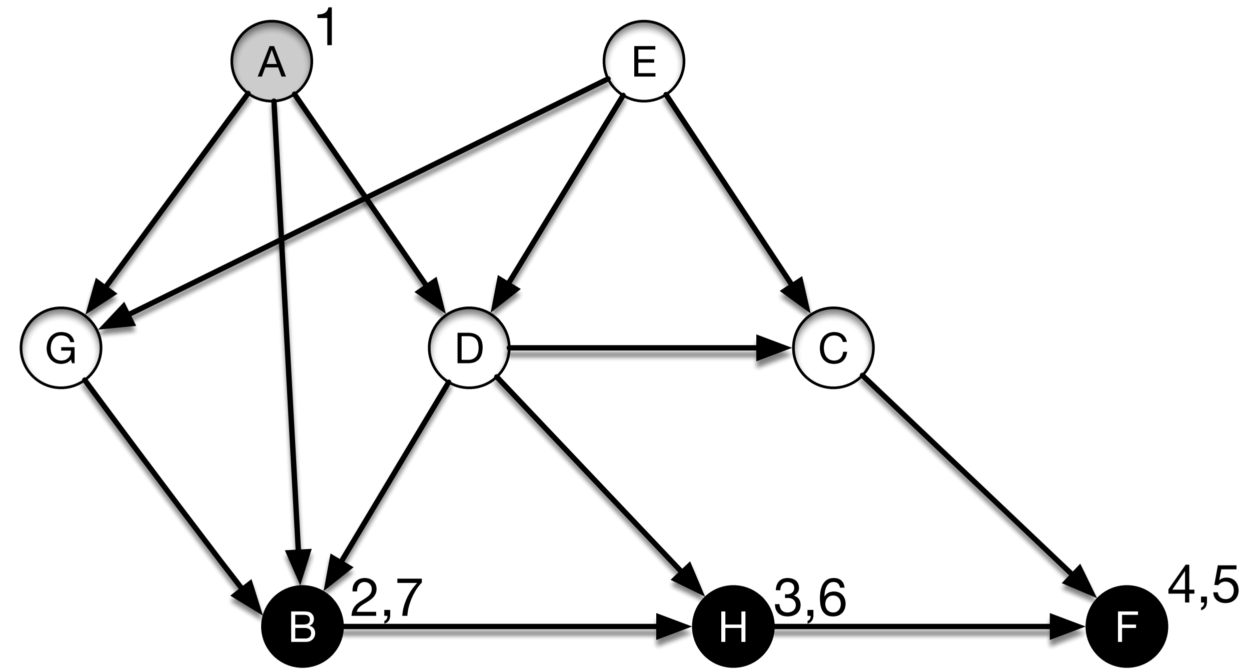



```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```

```
visit(H)
visit(B)
visit(A)
```

Topological Sort

- Finish B
- Push B at front: [B,H, F]

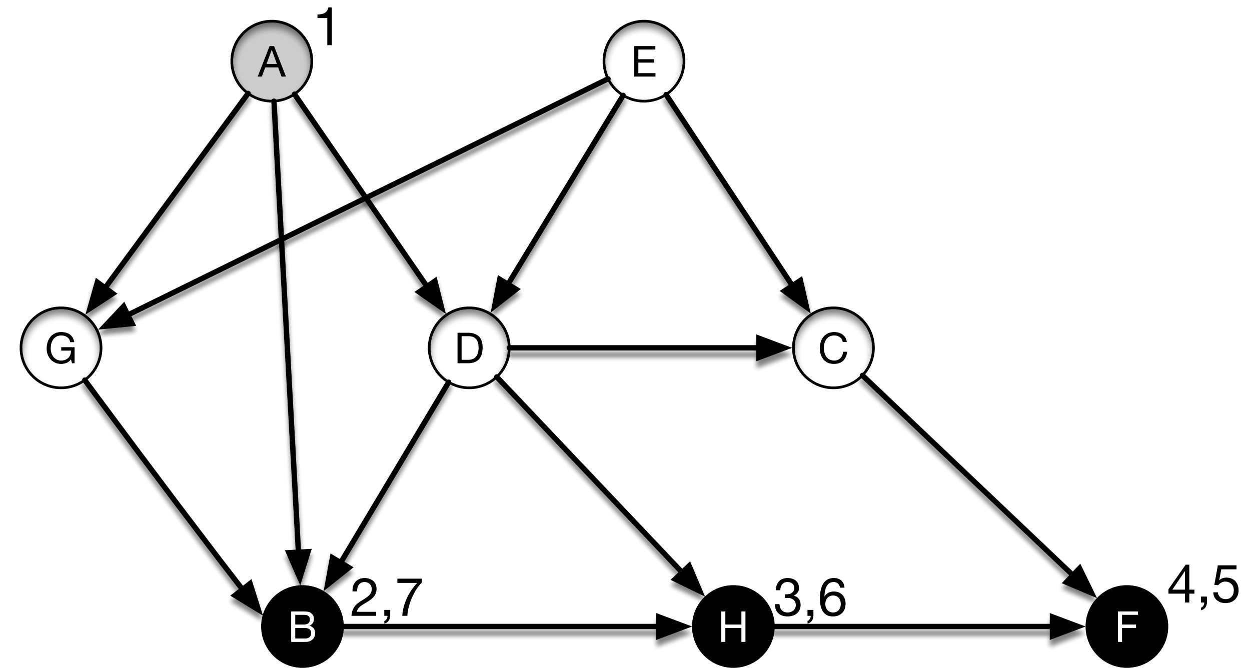



```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```

```
visit(B)
visit(A)
```

Topological Sort

- Go back to visit A
- [B,H, F]

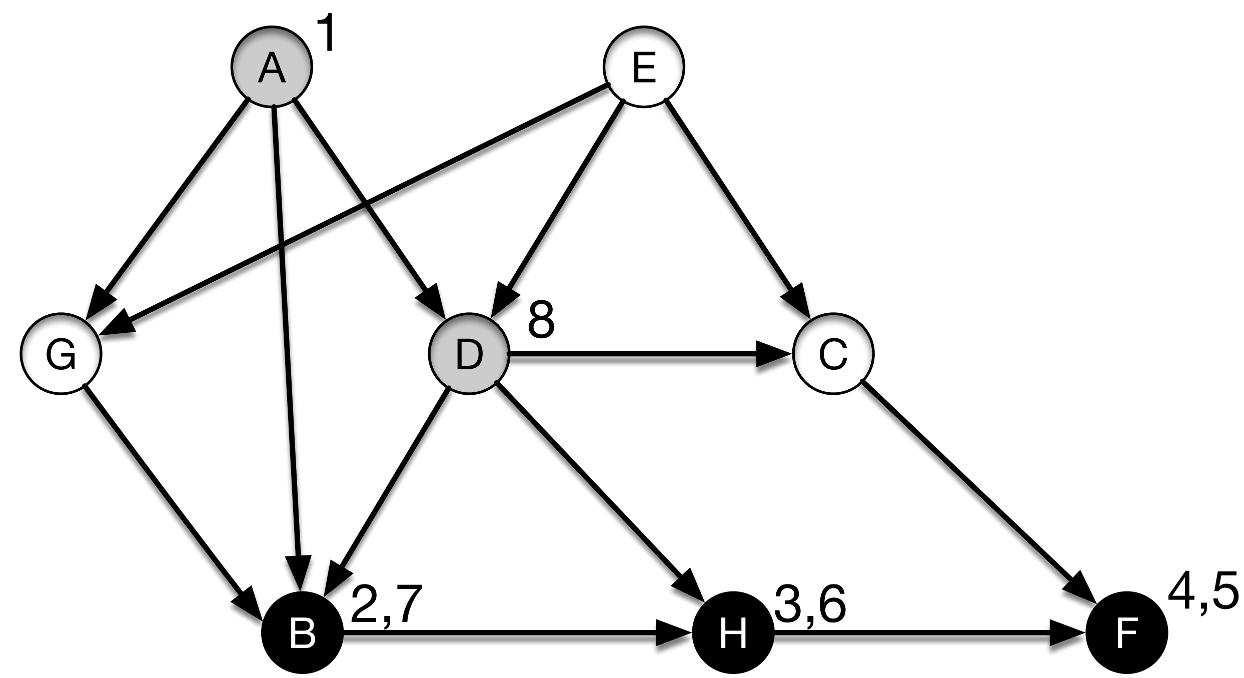



```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```

`visit(A)`

Topological Sort

- Visit D
- [B,H, F]



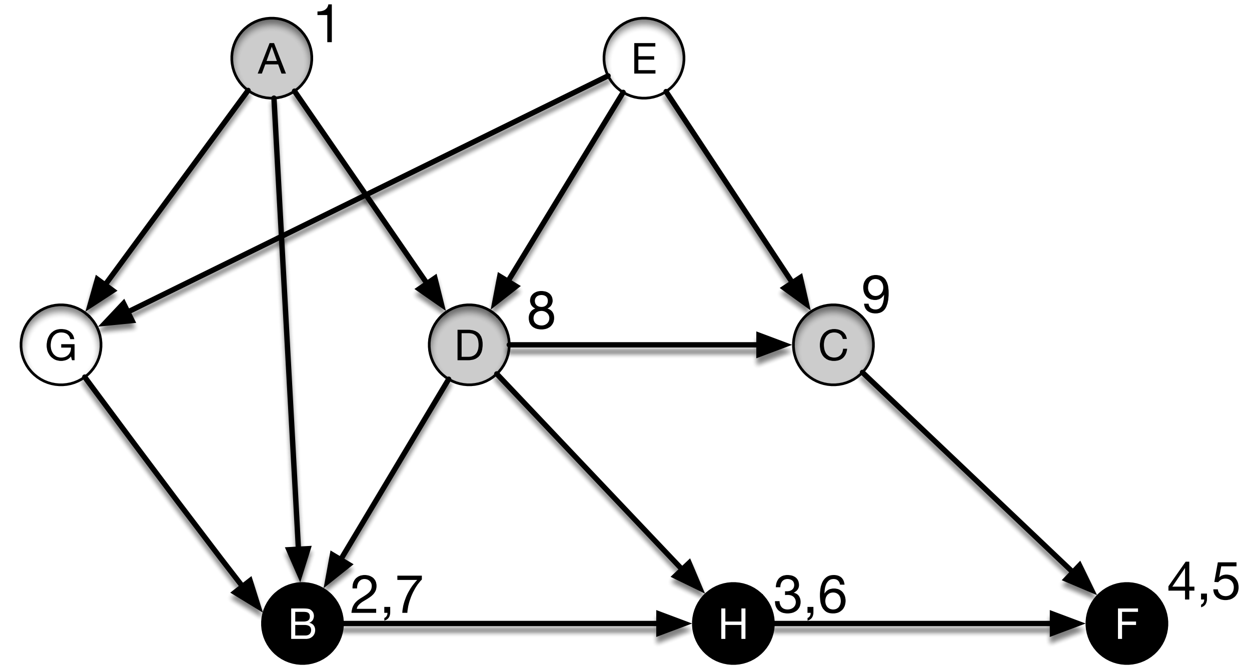

```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```

```
visit(D)
visit(A)
```

Topological Sort

- Visit C
- [B,H, F]

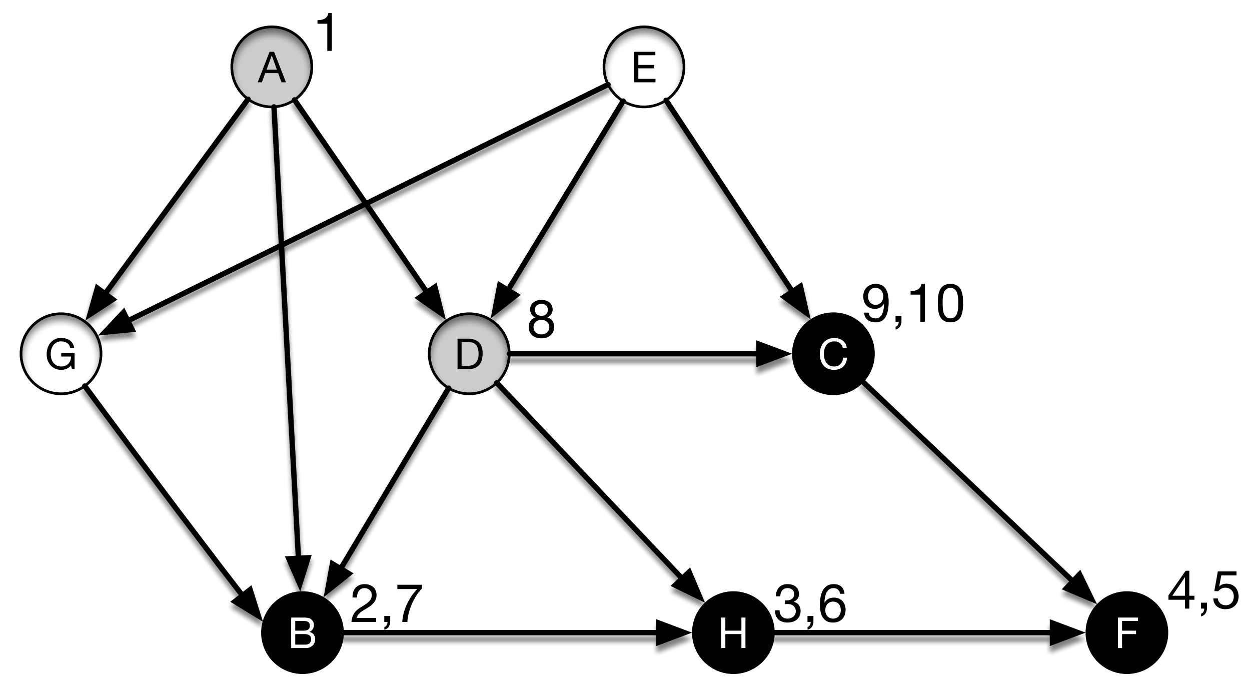
```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```




```
visit(C)
visit(D)
visit(A)
```

Topological Sort

- Finish C
- [C, B,H, F]

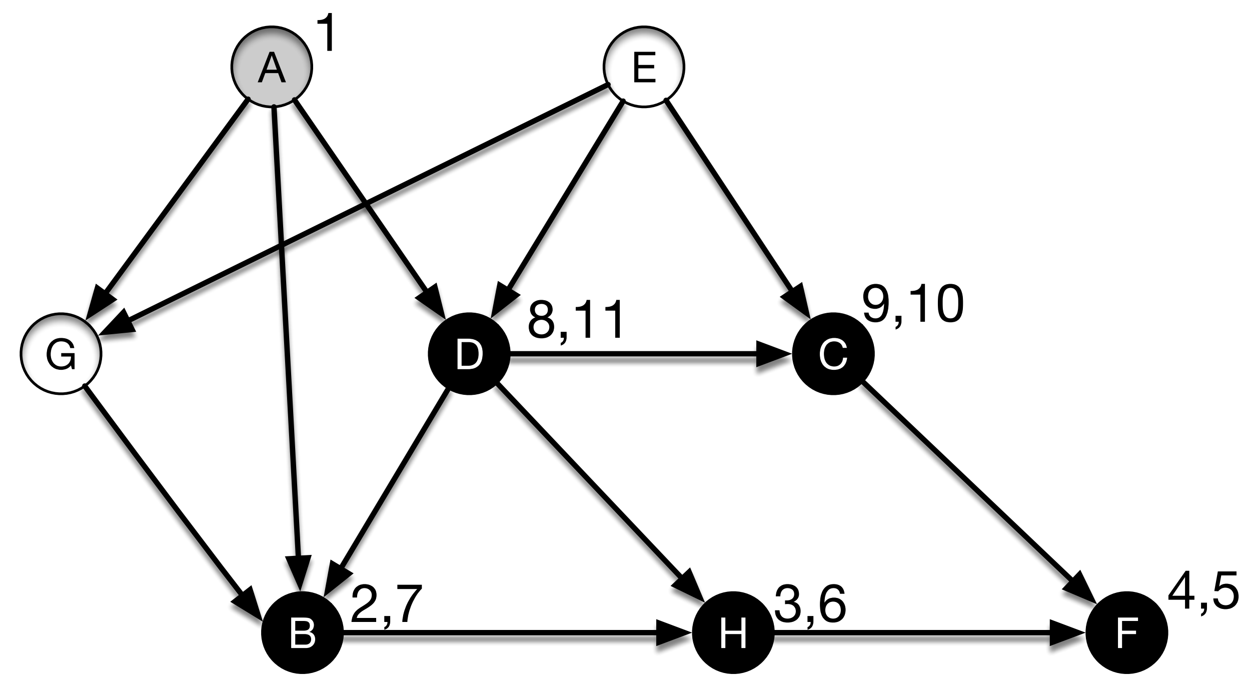



```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```

```
visit(C)
visit(D)
visit(A)
```

Topological Sort

- Go back and finish D
- [D, C, B, H, F]



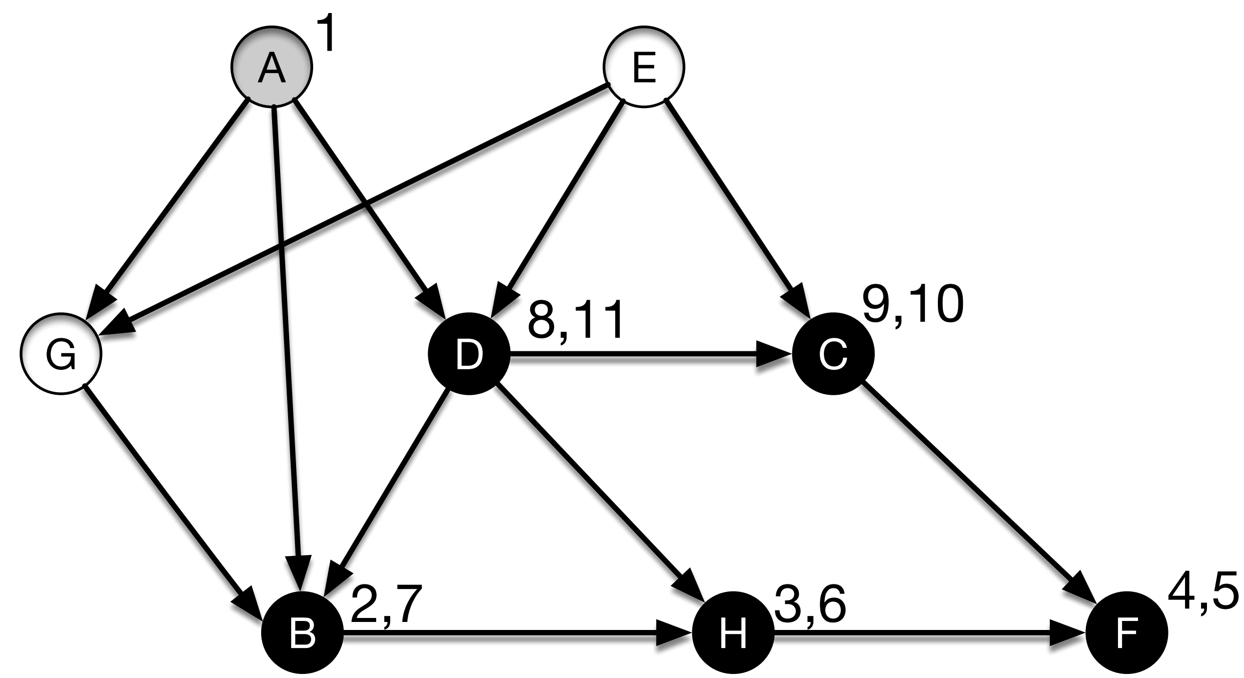

```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```

```
visit(D)
visit(A)
```

Topological Sort

- We are back to visit A
- Next node is G
- [D, C, B, H, F]

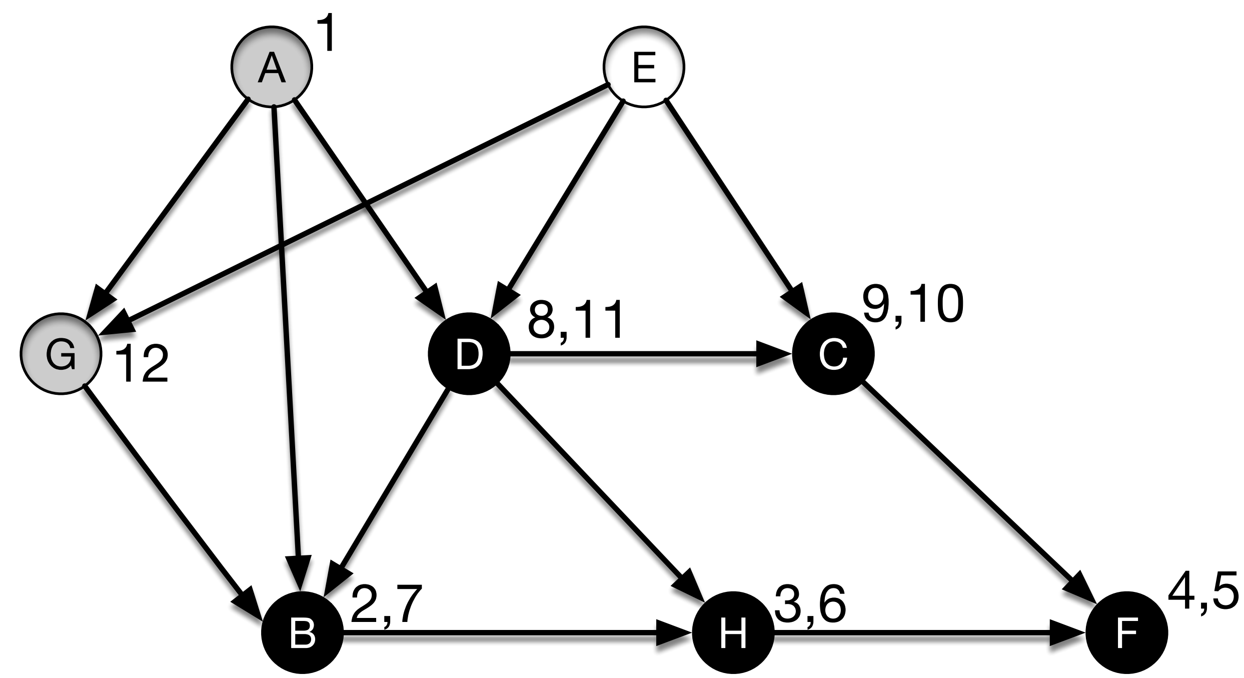
```
A:  B,D,G
B:  H
C:  F
D:  B,H
E:  C,D,G
F:
G:  B
H:  F
```



`visit(A)`

Topological Sort

- Visit G
- [D, C, B, H, F]

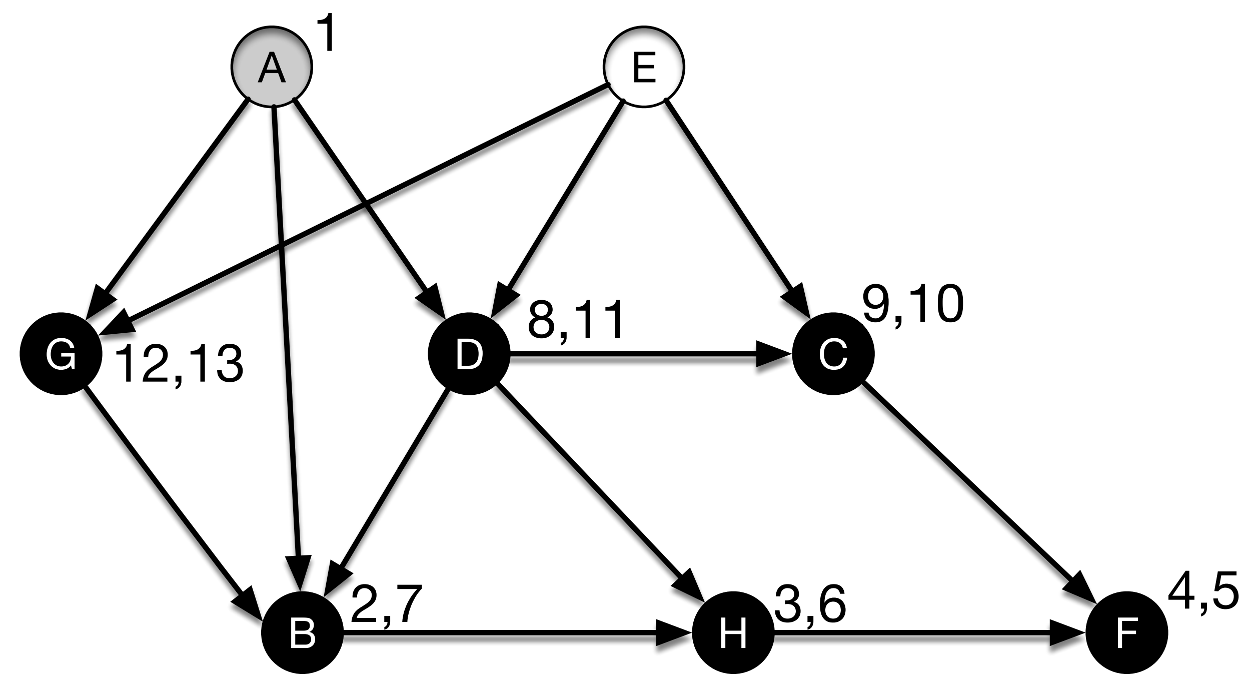



```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```

```
visit(G)
visit(A)
```

Topological Sort

- Finish G
- [G, D, C, B, H, F]



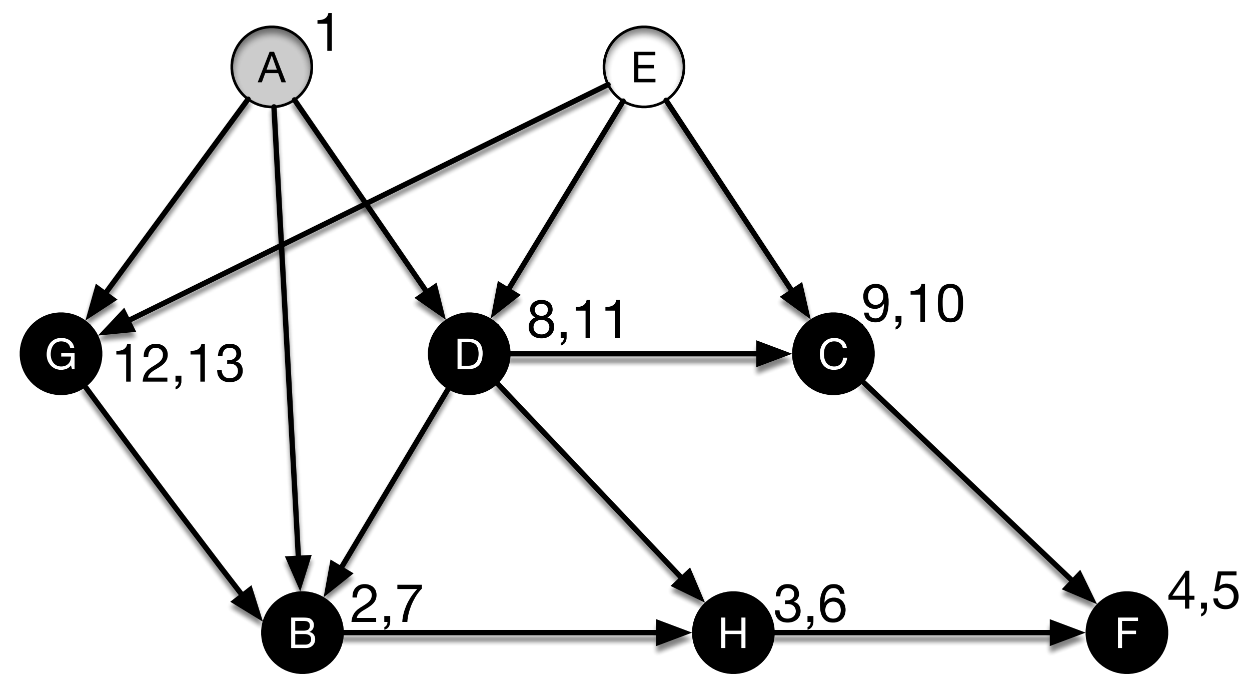

```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```

```
visit(G)
visit(A)
```

Topological Sort

- Go back to A
- [G, D, C, B, H, F]

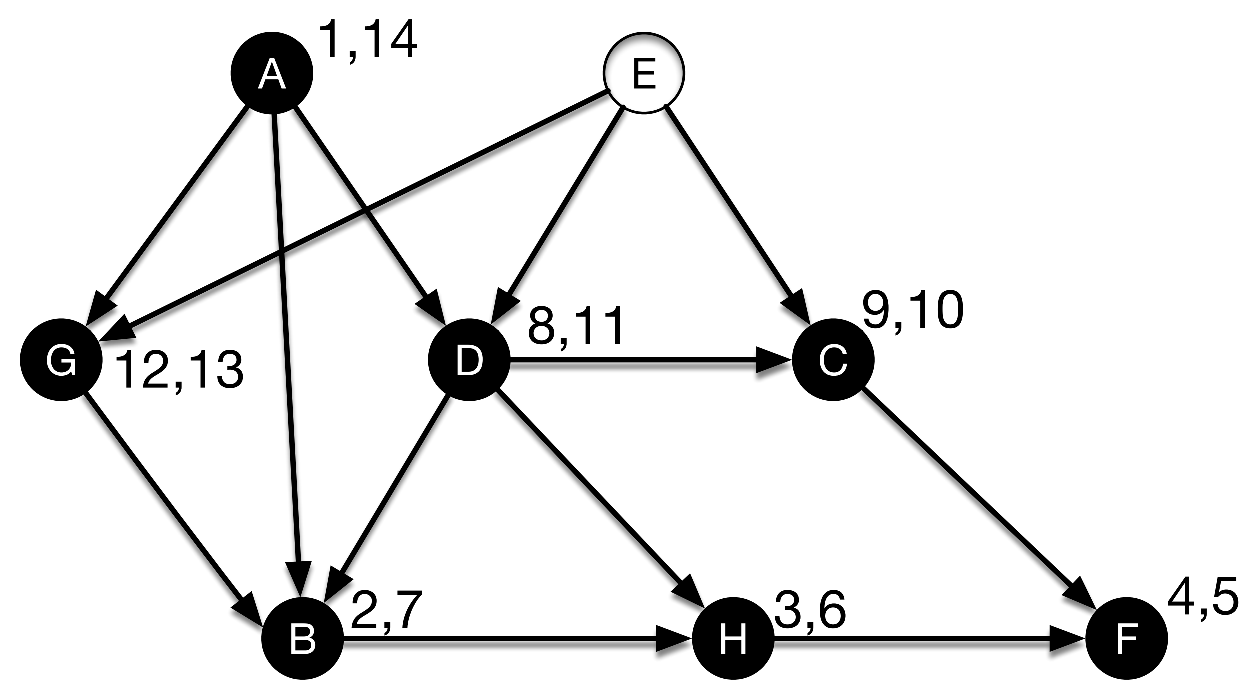
```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```



`visit(A)`

Topological Sort

- Finish A
- [A, G, D, C, B, H, F]

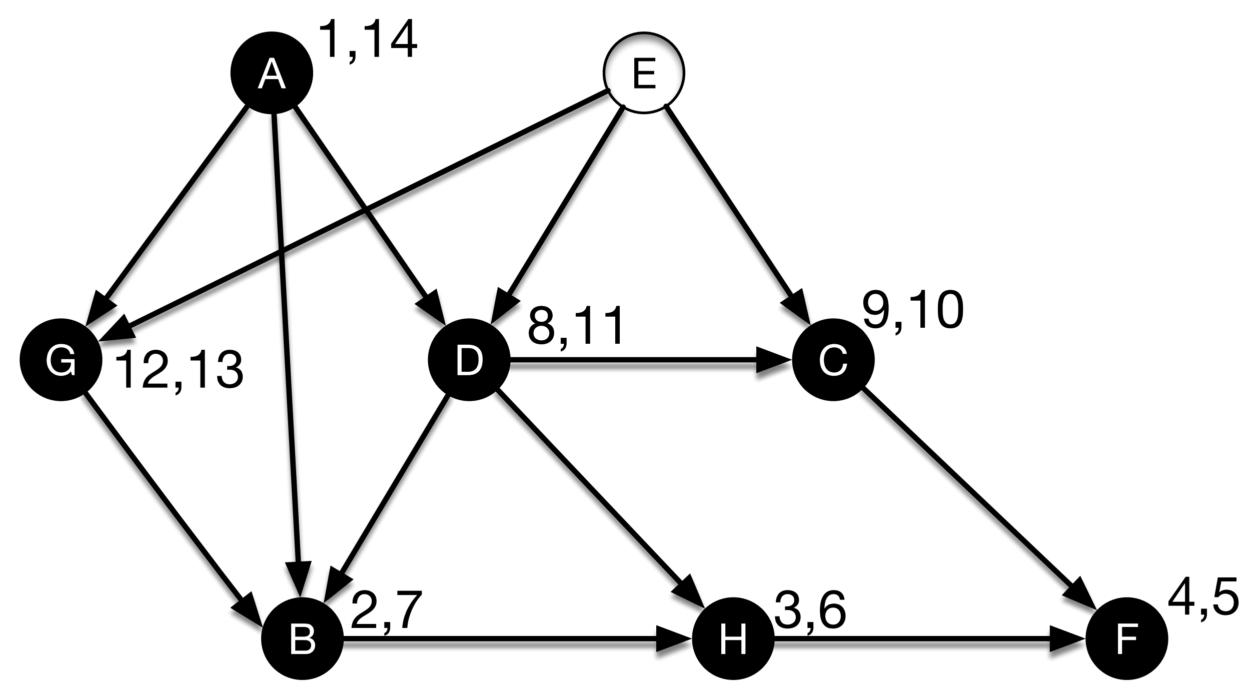



```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```

∅

Topological Sort

- Done with visit(A)
- [A, G, D, C, B, H, F]



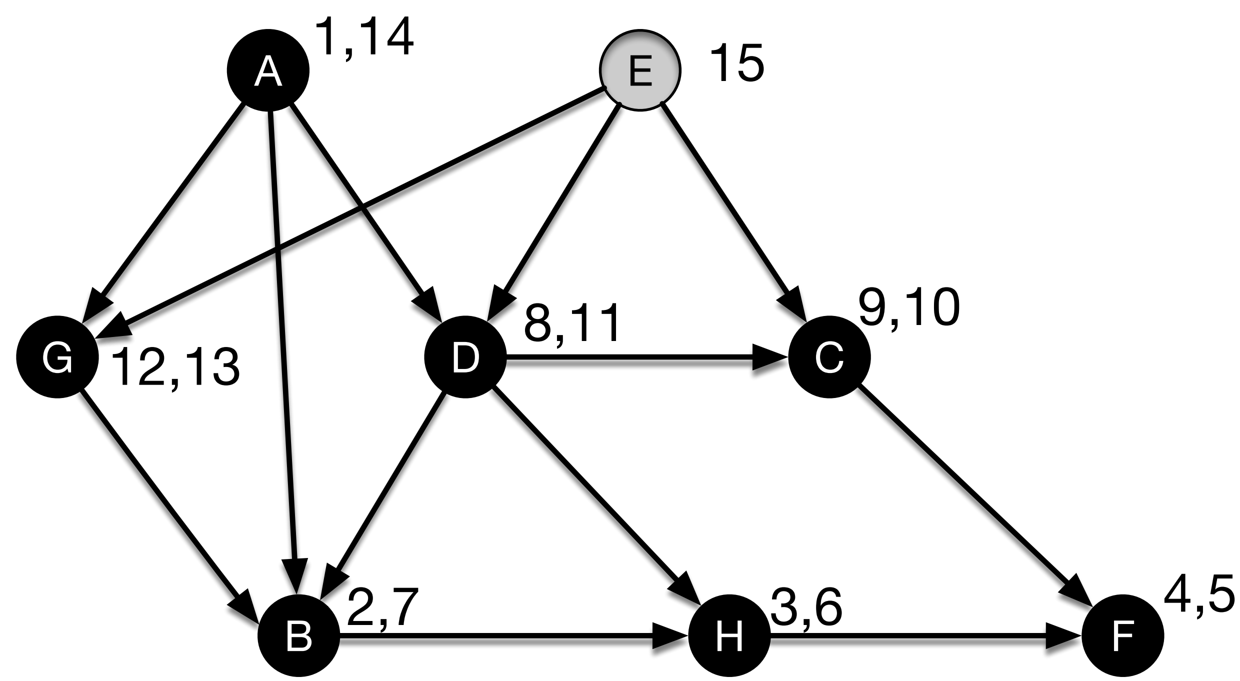

```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```

∅

Topological Sort

- One white node left: E
- Visit E
- [A, G, D, C, B, H, F]

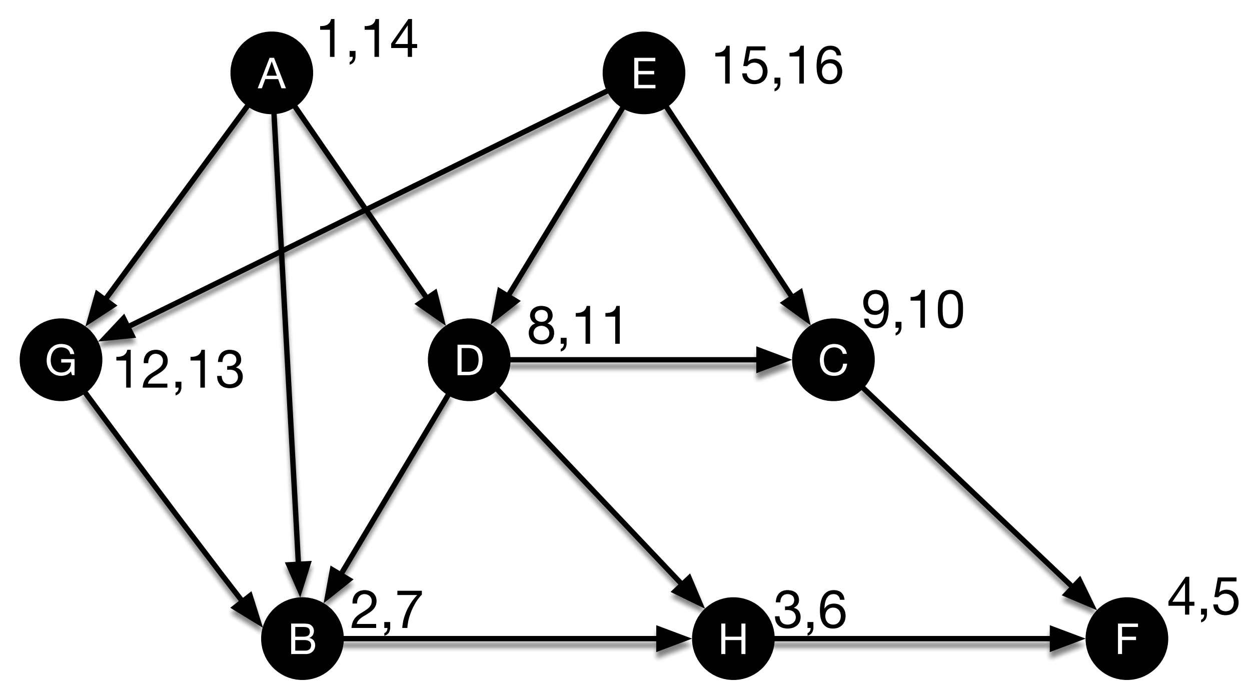
```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```



`visit(E)`

Topological Sort

- Finish E
- [E, A, G, D, C, B, H, F]




```
A: B,D,G
B: H
C: F
D: B,H
E: C,D,G
F:
G: B
H: F
```

Topological Sort

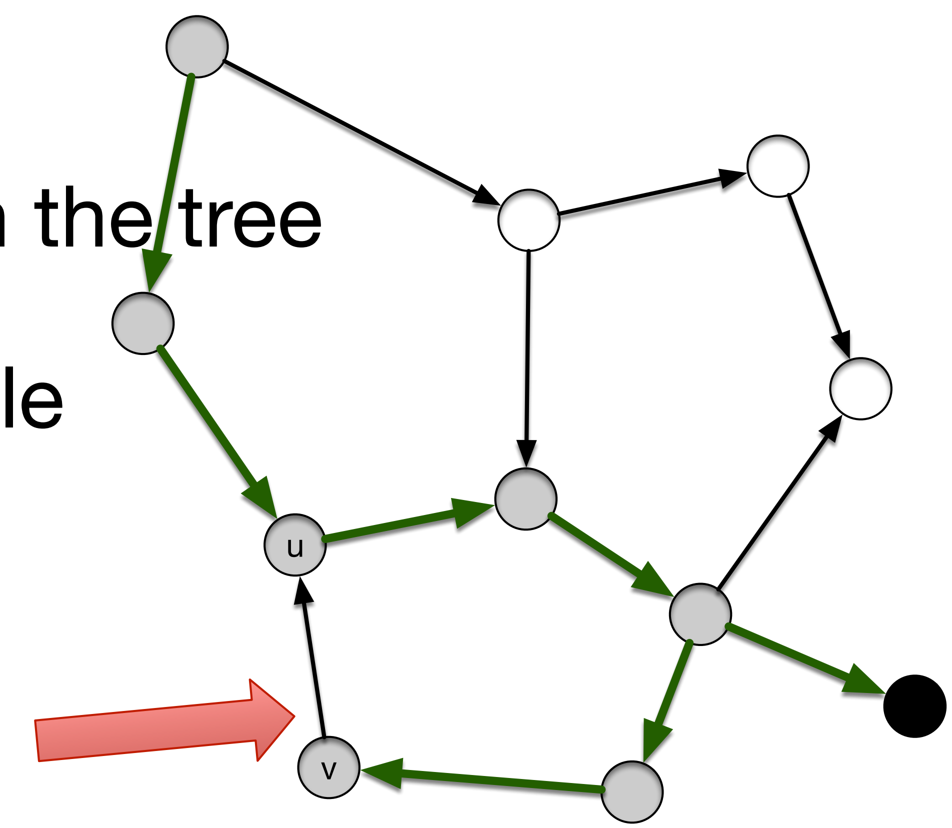
- Key observation from the examples:
  - We have a cycle if we ever try to visit a gray node

Topological Sort

- Lemma: A directed graph $G = (V, E)$ is acyclic if and only if a DFS of G yields no back edges

Topological Sort

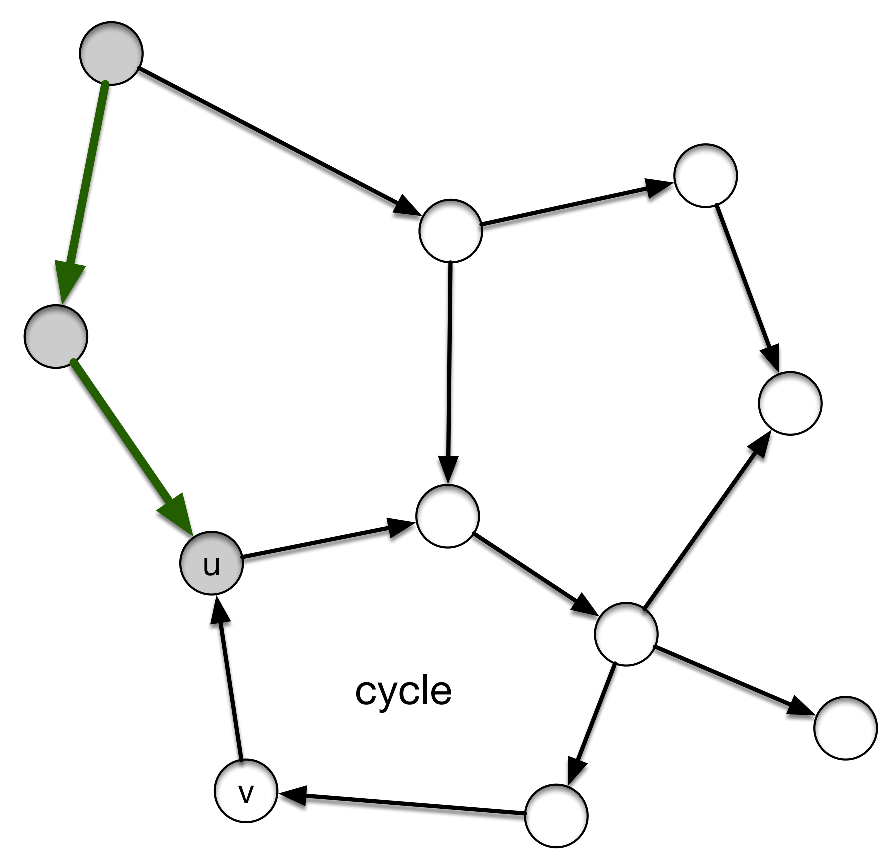
- Proof: "$\Rightarrow$"
  - If DFS produces a back-edge (v, u) then u is an ancestor of v
  - There is a path from u to v in the tree
  - The edge (v, u) closes a cycle
    - from u to v back to u



visiting v and discovering a gray node

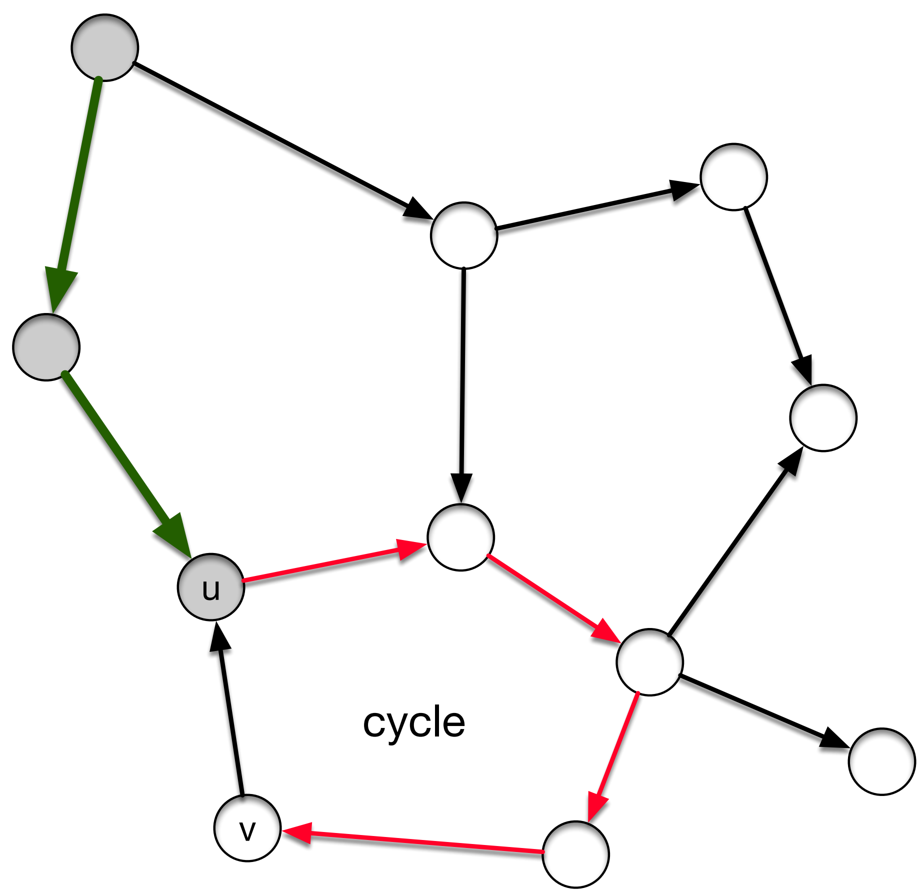
Topological Sort

- Proof: "$\Leftarrow$"
  - Suppose G has a cycle
  - Let u be the first vertex in the cycle to be discovered



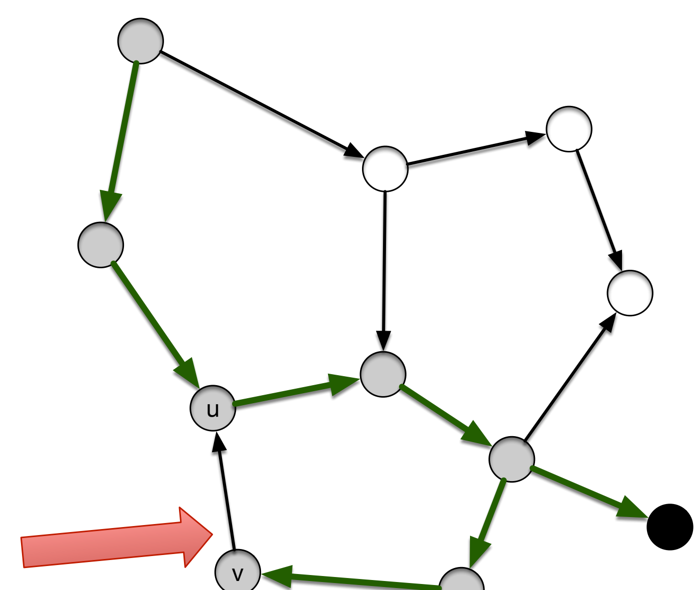
Topological Sort

- All other vertices in the cycle are white and there is a white-path to the node v just in front of u



Topological Sort

- By the white-path theorem:
  - We will discover v from u
    - (Though not necessarily through the cycle since there might be more cycles)
  - Thus, (v, u) is a back edge



visiting v and discovering a gray node

Topological Sort

- Theorem: DFS gives a topological sort or discovers a cycle
- Proof:
  - Need to show:
    - If DFS does not discover a cycle, then for each edge (u, v), we have $u.f > v.f$

Topological Sort

- Proof:
  - At the time that we are first looking at (u, v):
    - v cannot be gray, because then we would have a back-edge

Topological Sort

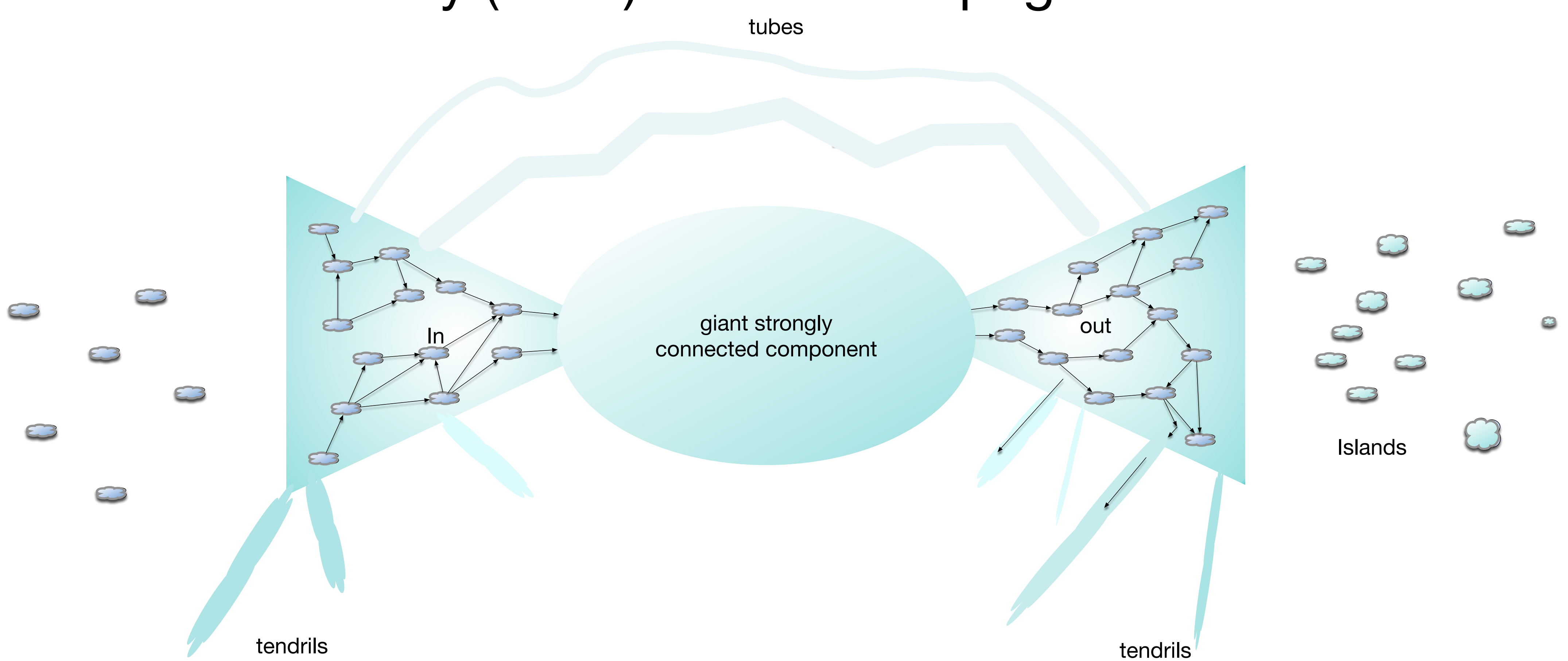
- At the time that we are first looking at (u, v):
  - If v is white:
    - Then by the white path theorem, u becomes an ancestor of v
    - By the parenthesis theorem $v.f < u.f$

Topological Sort

- Proof:
  - At the time that we are first looking at (u, v):
    - If v is black, then u is still be visited, so
    - u is not yet black
    - so, $u.f > v.f$
    - qed

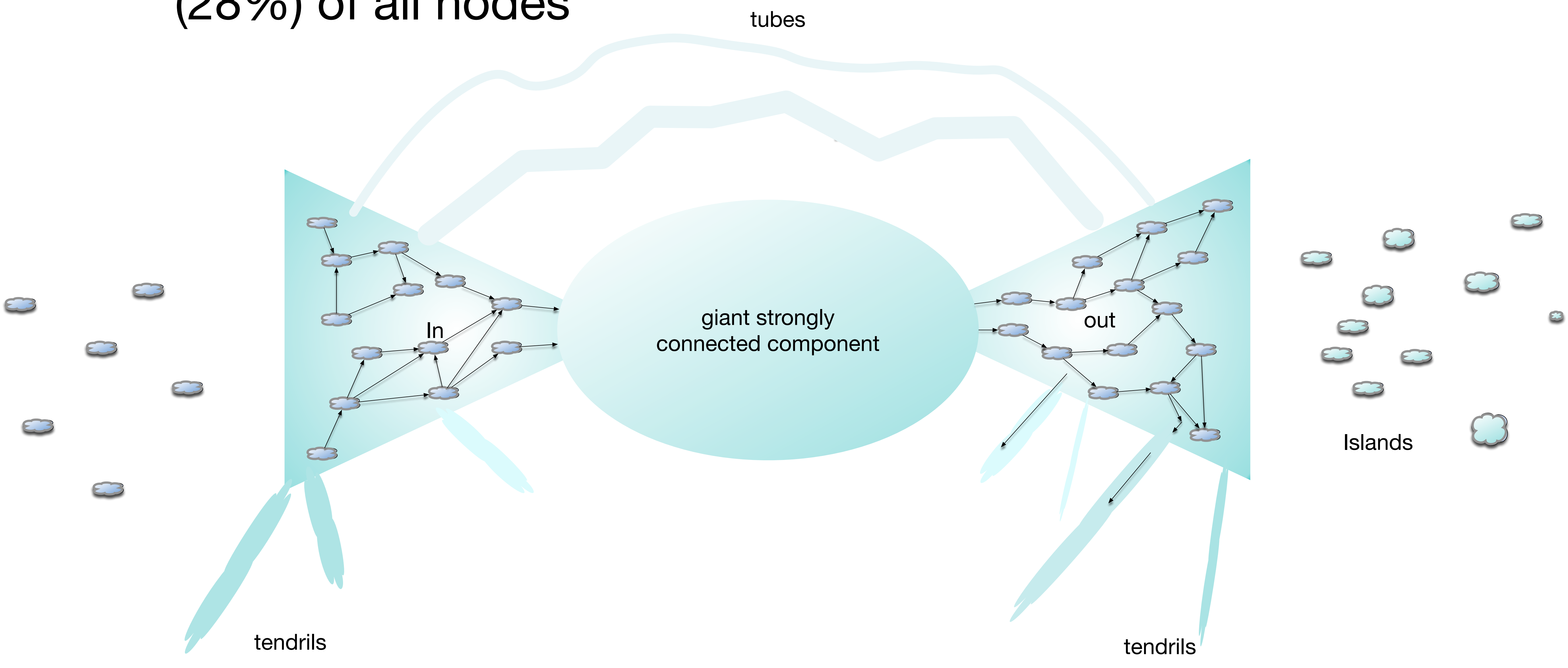
Strongly Connected Components

- WWW graph:
  - Nodes: pages
  - Edges: links from one page to another page
- Broder et al. study (2000): 200 million pages and 1.5 billion links



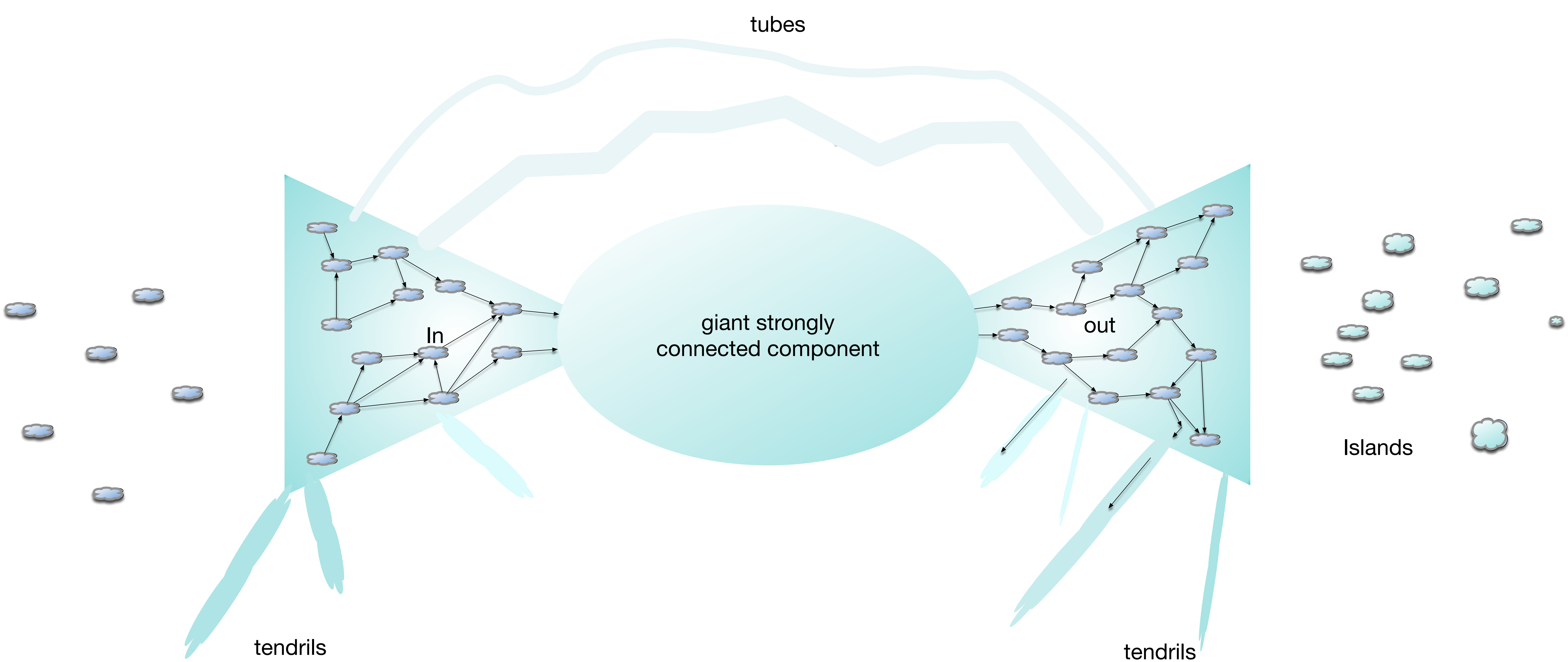
Strongly Connected Components

- Bowtie:
- Strongly connected component at the center of the WWW (28%) of all nodes



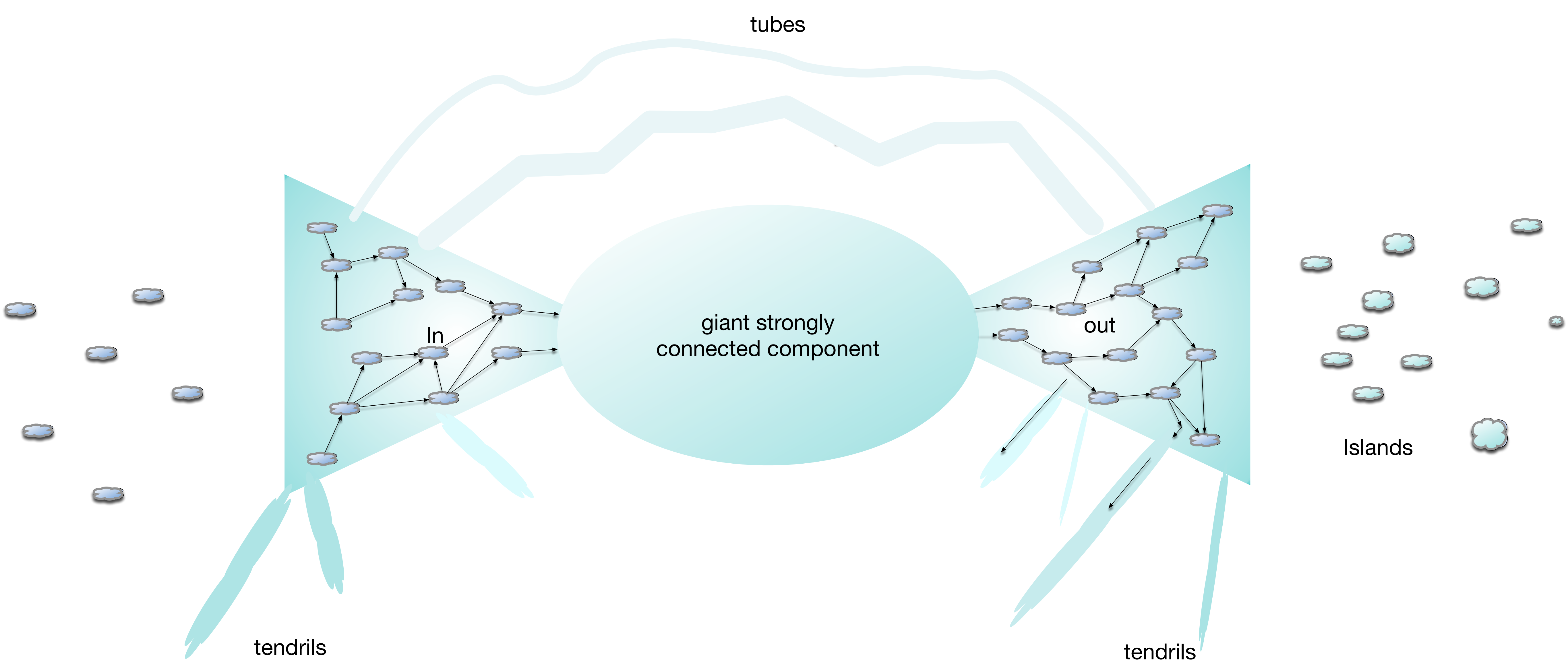
Strongly Connected Components

- Islands: Isolated areas of the web



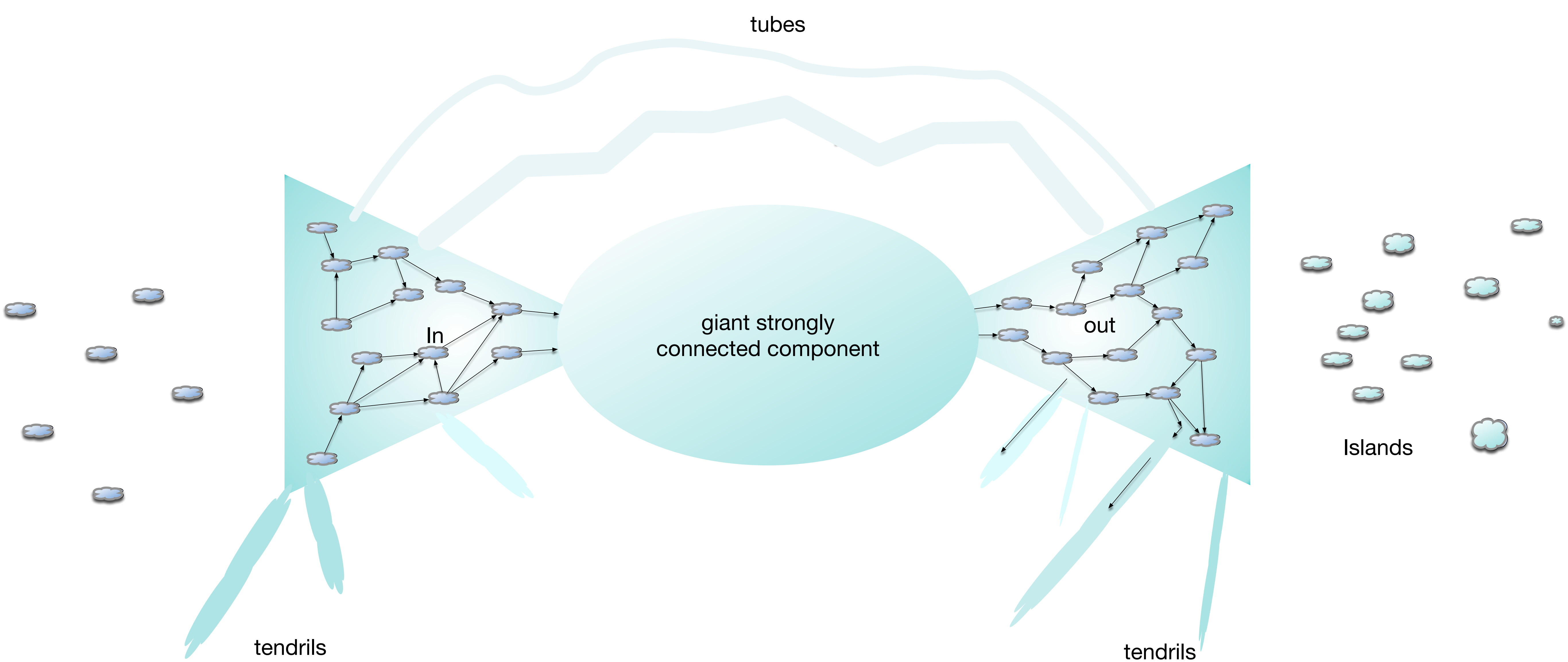
Strongly Connected Components

- In: Possible to reach the giant
- Out: Reachable from the giant



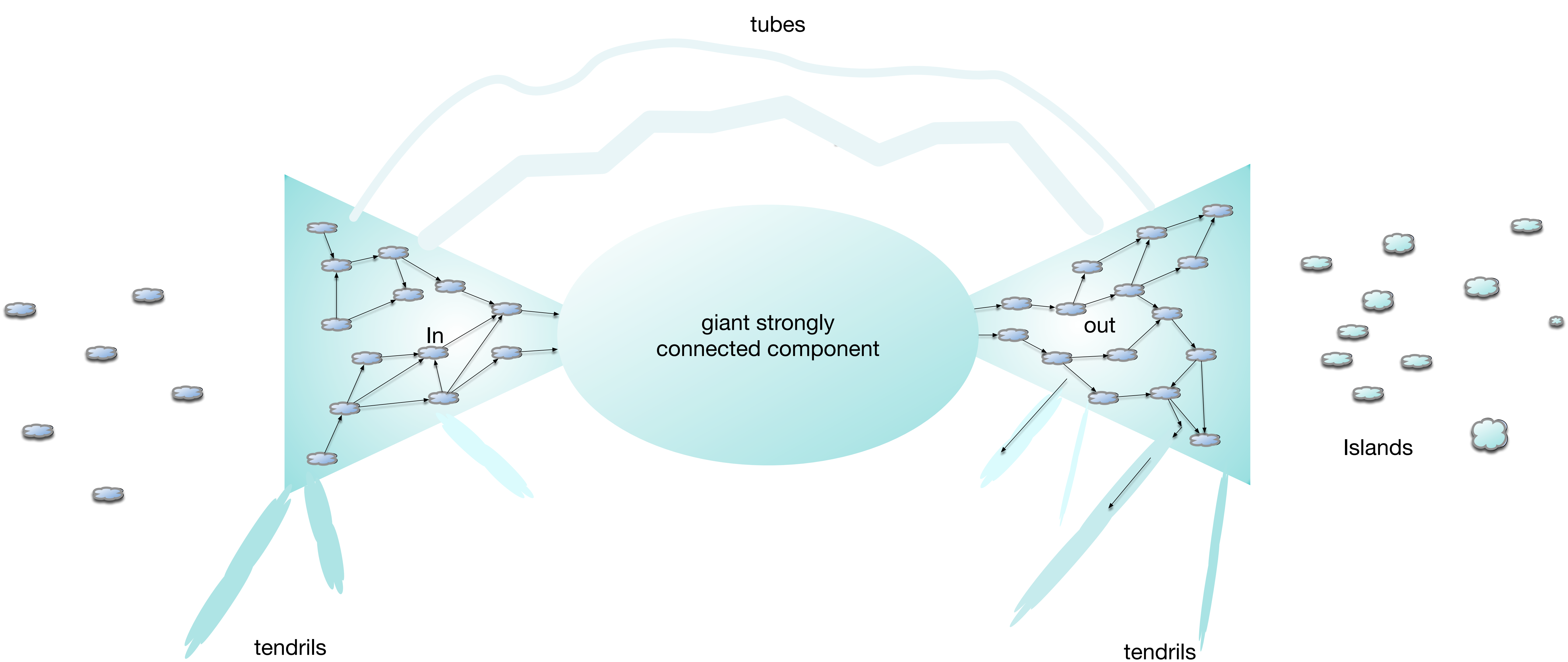
Strongly Connected Components

- Weird stuff: Tubes that move from In to Out bypassing the giant



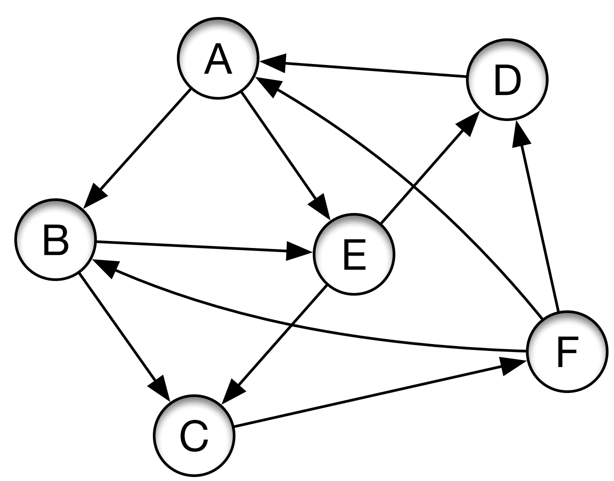
Strongly Connected Components

- Weird stuff: Tendrils to In and tendrils to Out



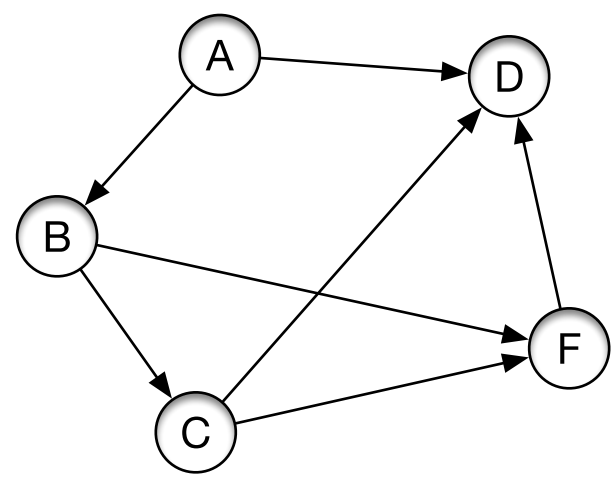
Strongly Connected Components

- Strongly connected component:
    - Can reach any vertex from any other vertex



Strongly Connected Components

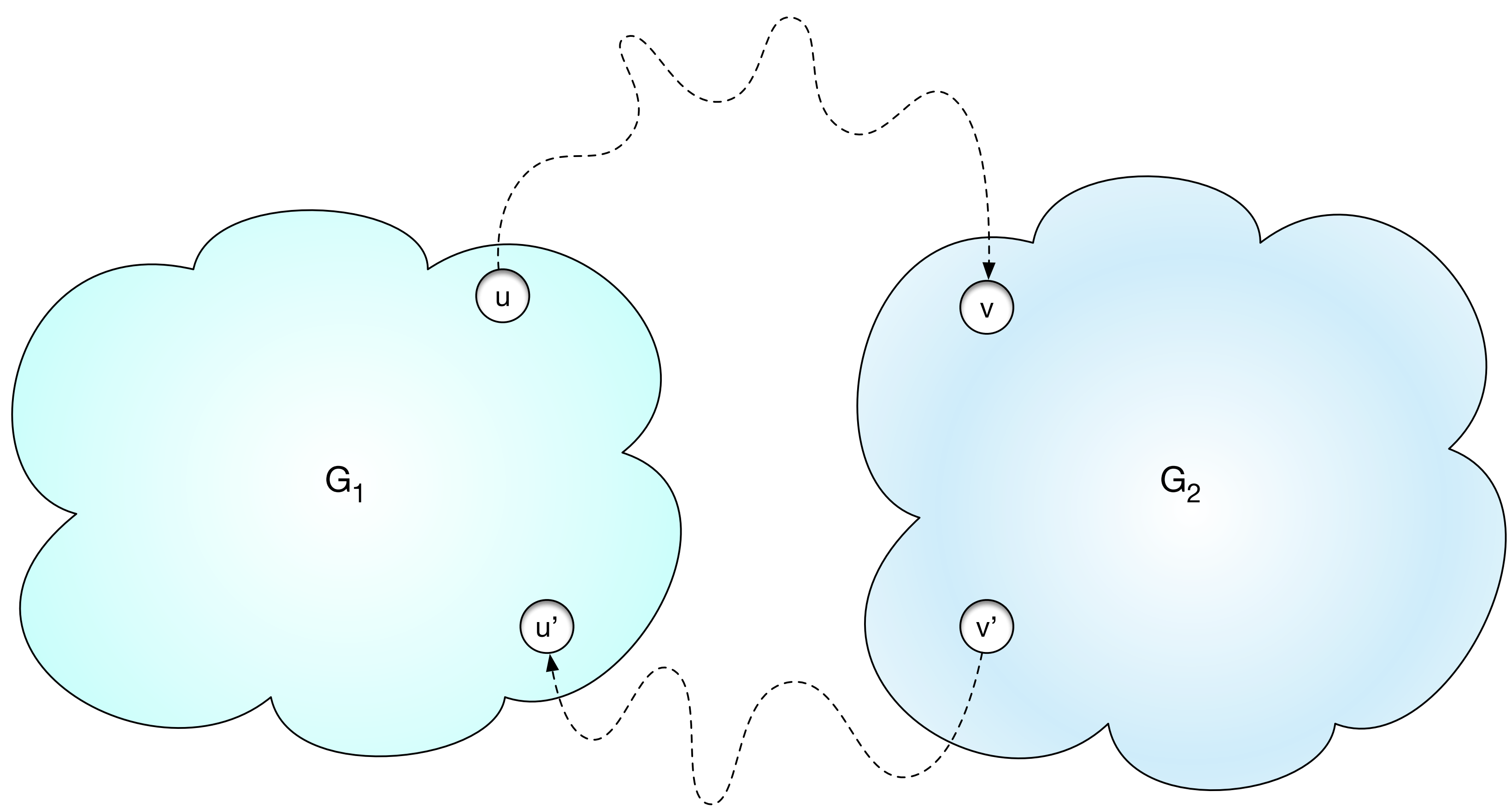
- Strongly connected component
  - This is **NOT** strongly connected



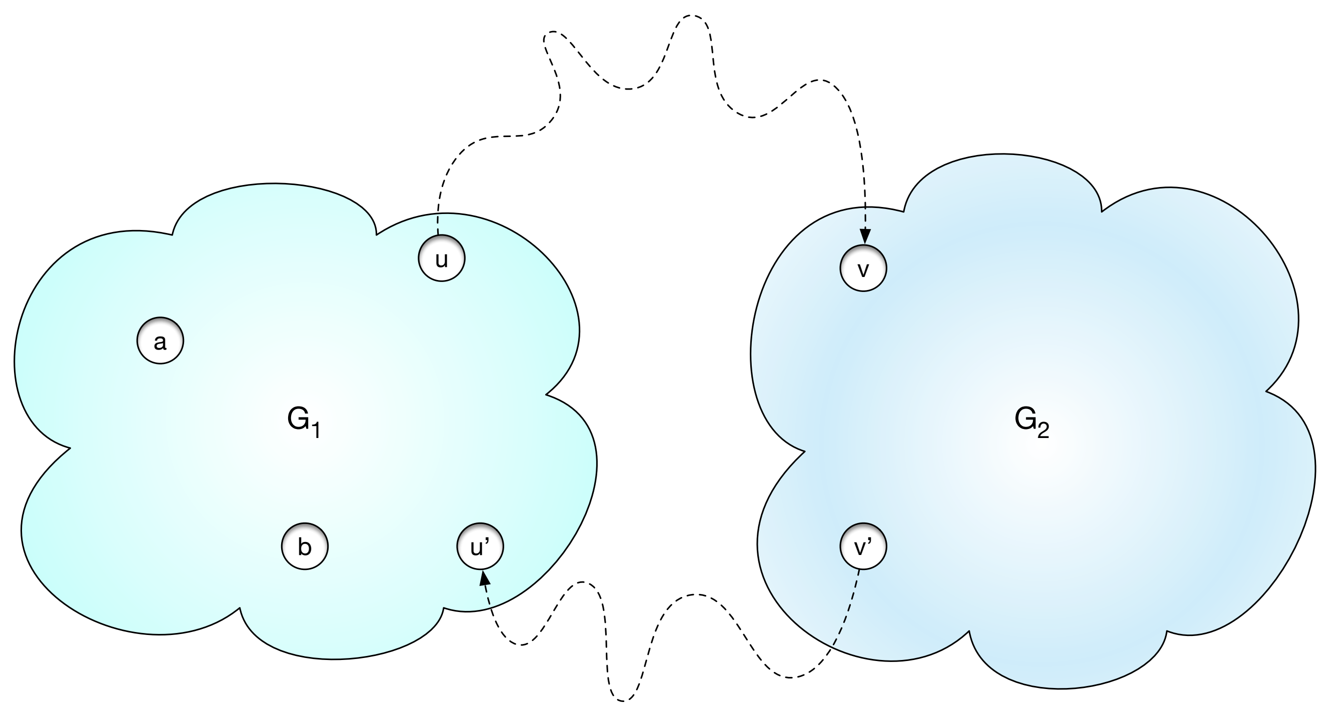
  - There is no way to get from D to A

Strongly Connected Components

- Lemma: Let G_1 and G_2 be two strongly connected subgraphs of a graph G and assume that there is a path from a vertex in G_1 to a vertex in G_2 and also a path from a vertex of G_2 to G_1, then $G_1 \cup G_2$ is strongly connected

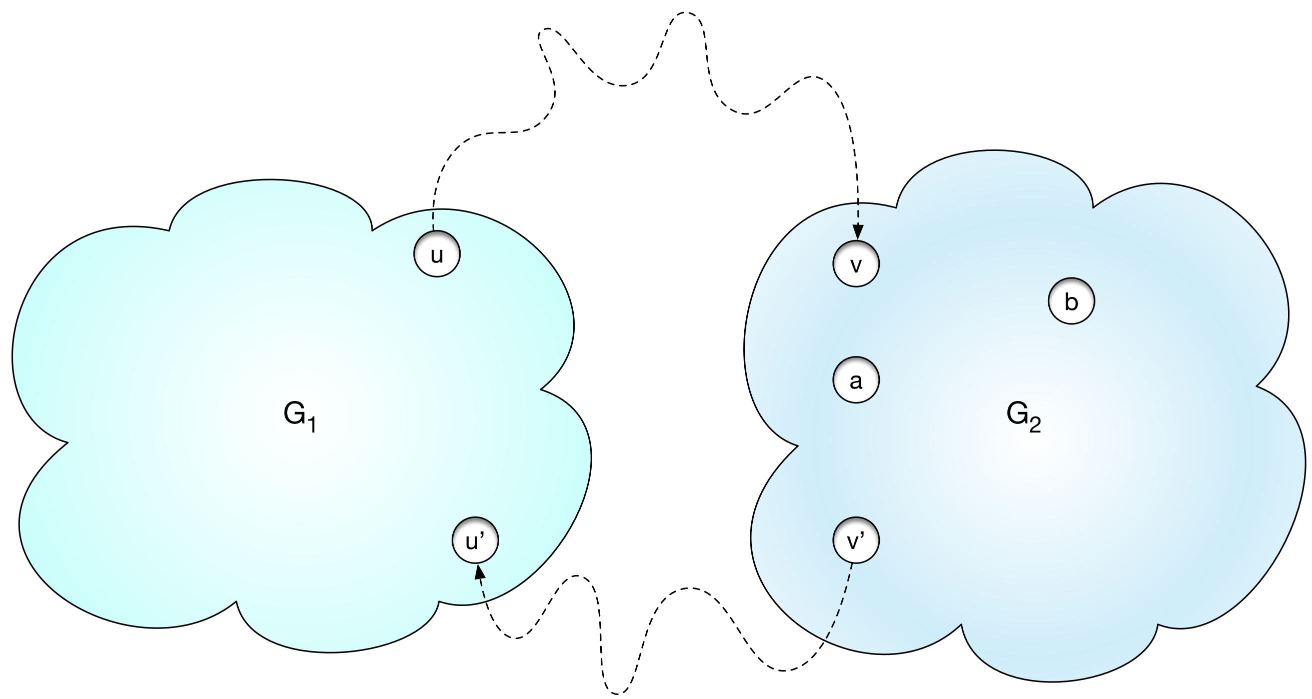


Strongly Connected Components



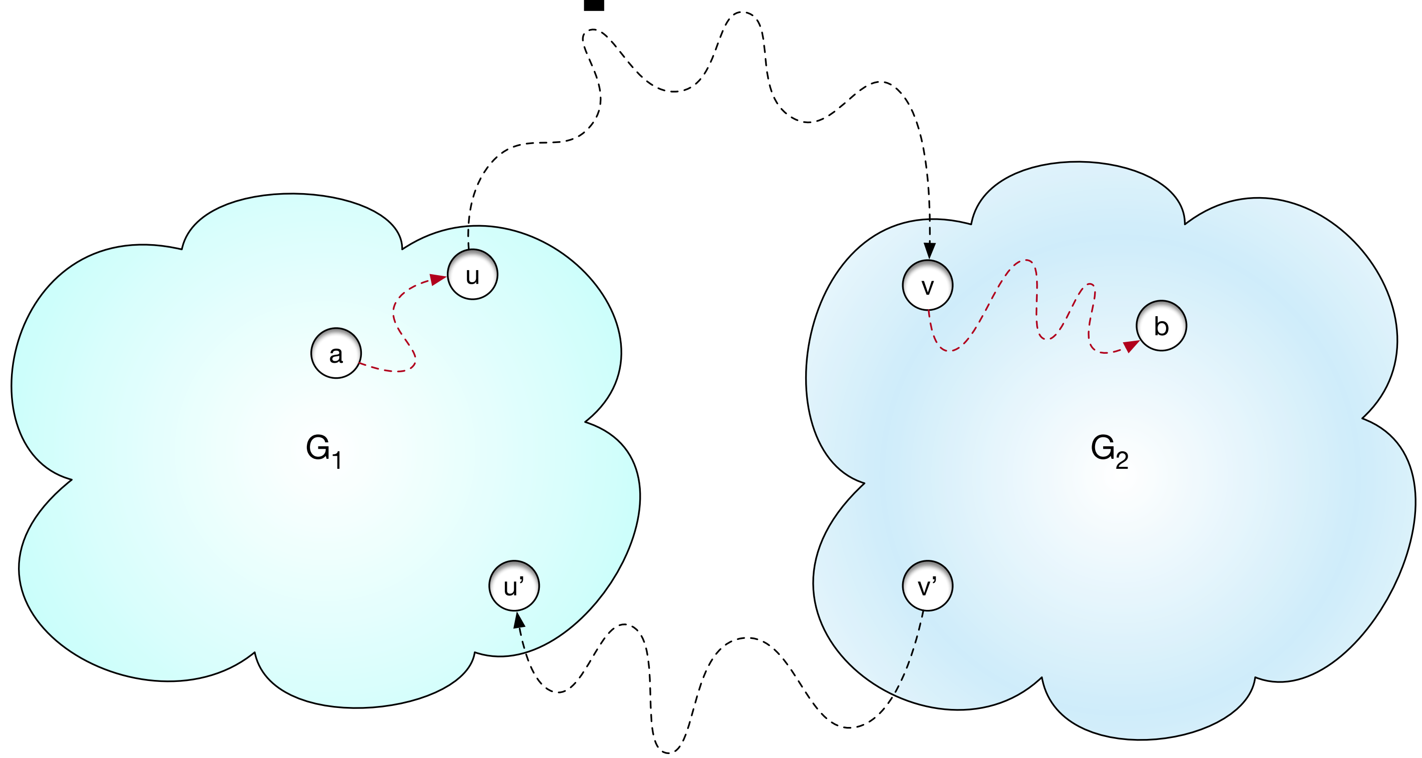
- Proof: Take two nodes a and b in $G_1 \cup G_2$.
  - If both are in G_1 then there is a path between a and b because they are in G_1

Strongly Connected Components



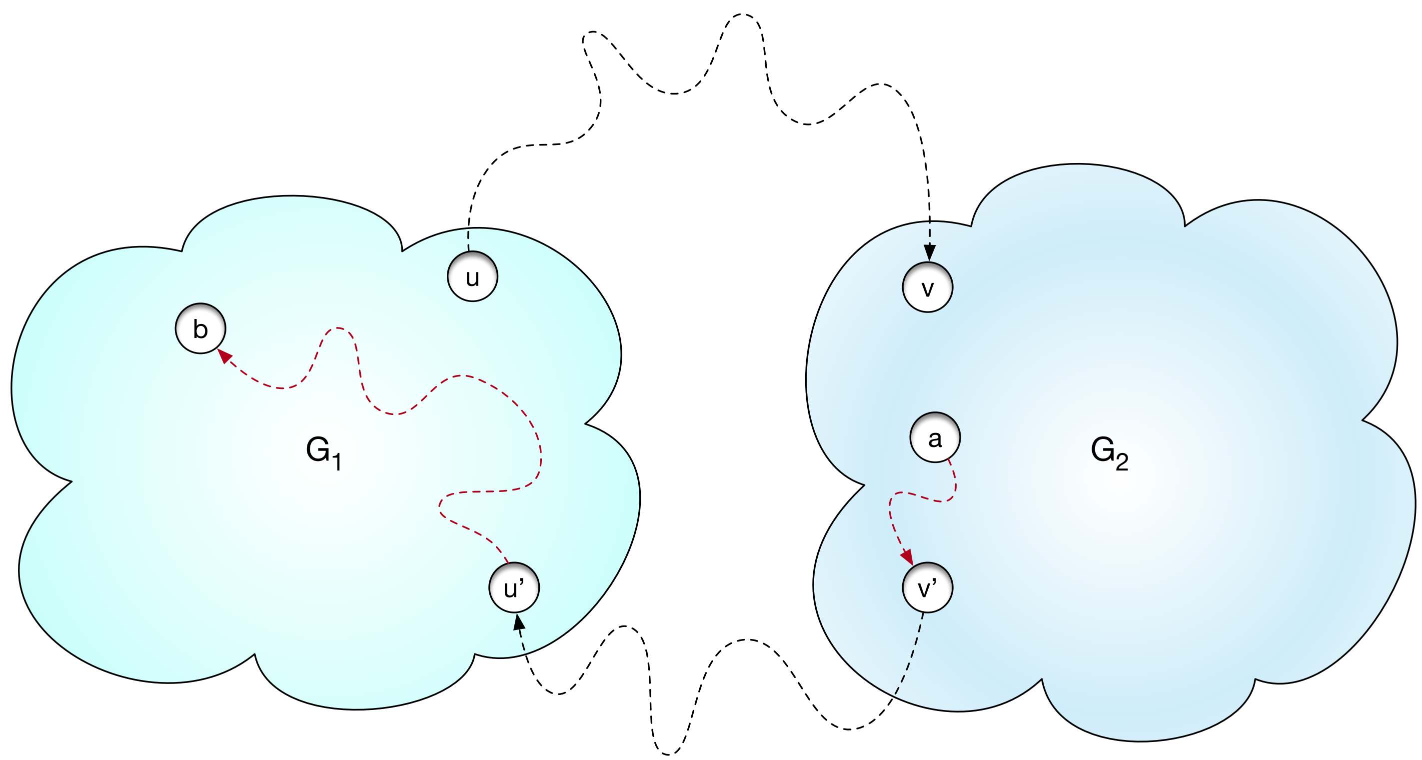
- Proof: Take two nodes a and b in $G_1 \cup G_2$.
  - If both are in G_2 then there is a path between a and b because they are in G_2

Strongly Connected Components



- If $a \in V(G_1)$ and $b \in V(G_2)$, then we can move from a to u and from u to v and then from v to b.
- After removing cycles, this is now a path from a to b

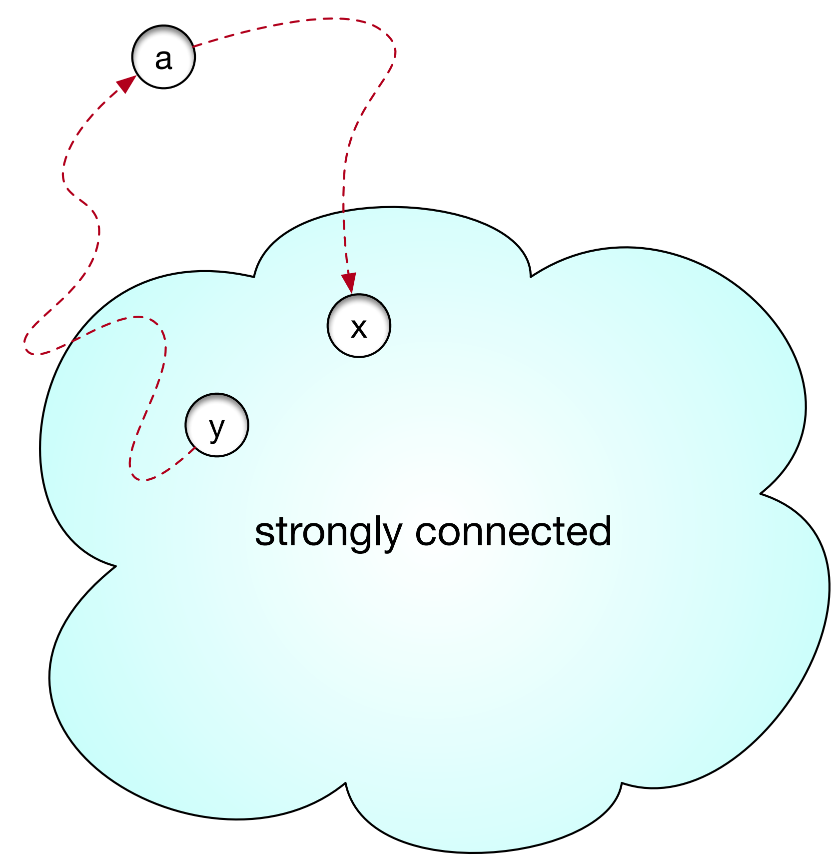
Strongly Connected Components



- Similarly, if $a \in V(G_2)$ and $b \in V(G_1)$, then we can move from a to v' and from v' to u' and then from u' to b.
- After removing cycles, this is now a path from a to b

Strongly Connected Components

- A single node is a strongly connected subgraph
- For each strongly connected subgraph, we can try to grow by adding other nodes
- If a node a has a path to and from a strongly connected subgraph, then by the lemma, we can add the node and get a bigger strongly connected subgraph

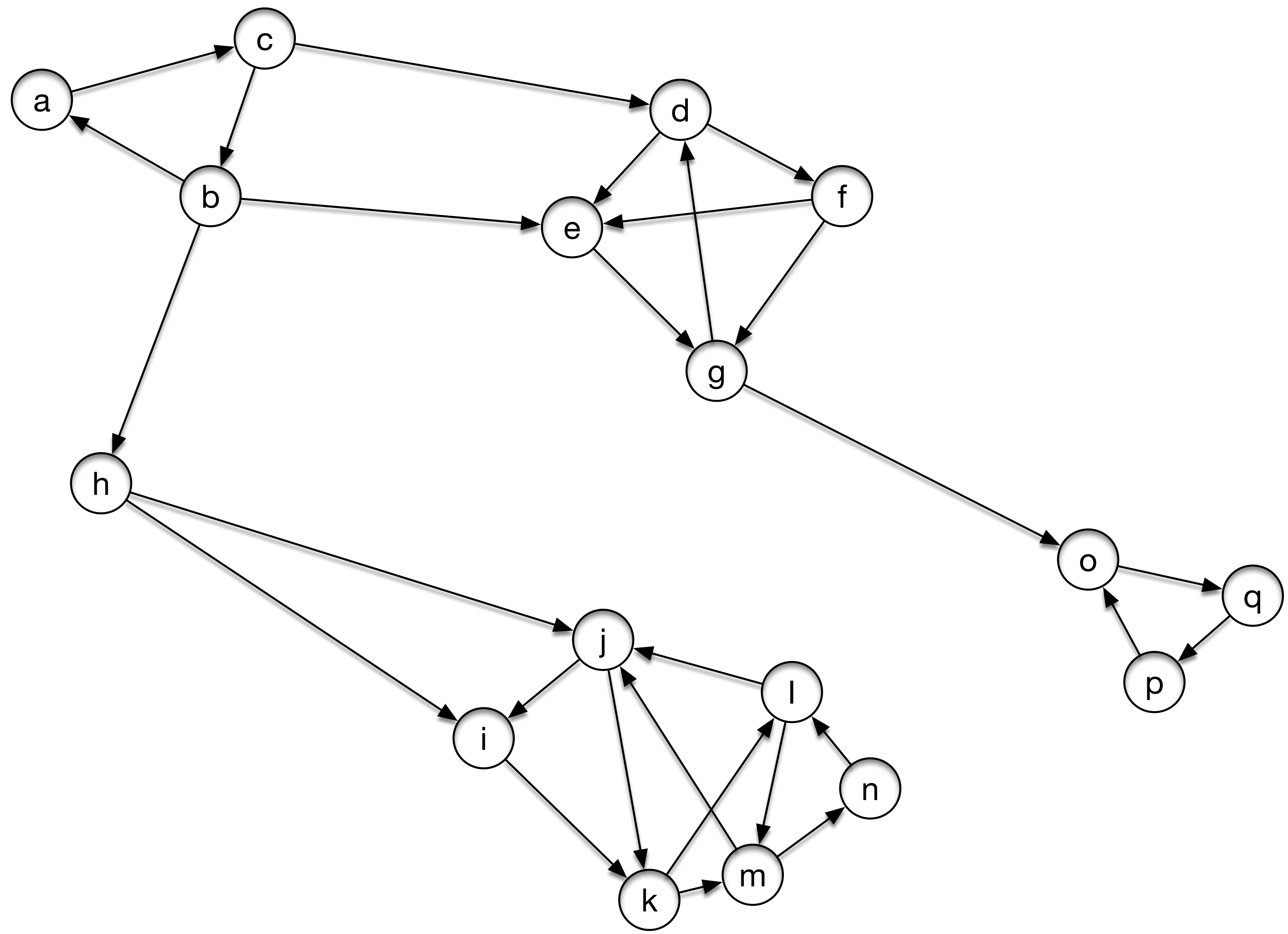


Strongly Connected Components

- Strongly connected component : A maximal strongly connected subgraph
- The nodes of any directed graph can be divided into strongly connected components

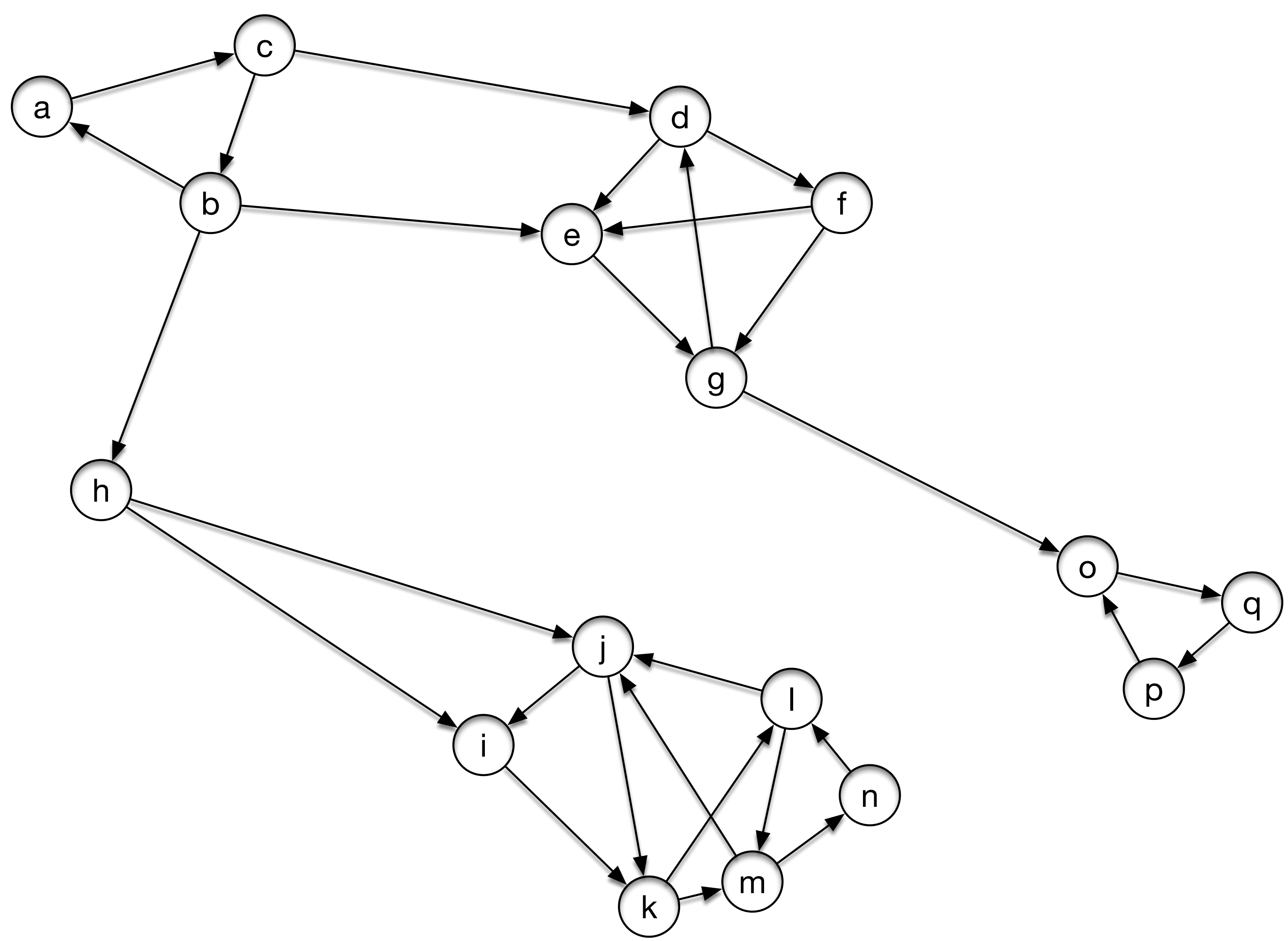
Strongly Connected Components

- Example:



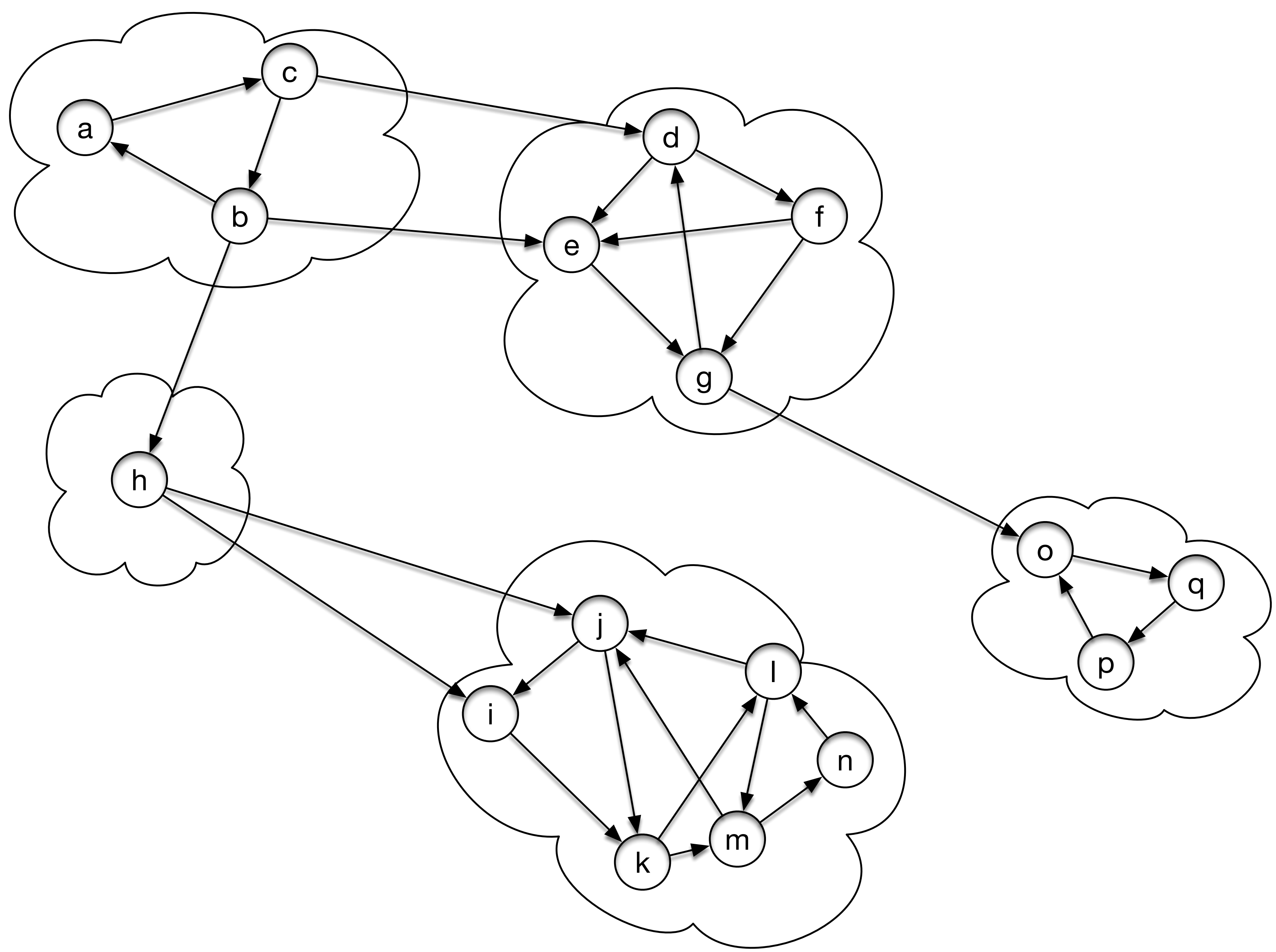
Strongly Connected Components

- Try it out by growing from individual nodes



Strongly Connected Components

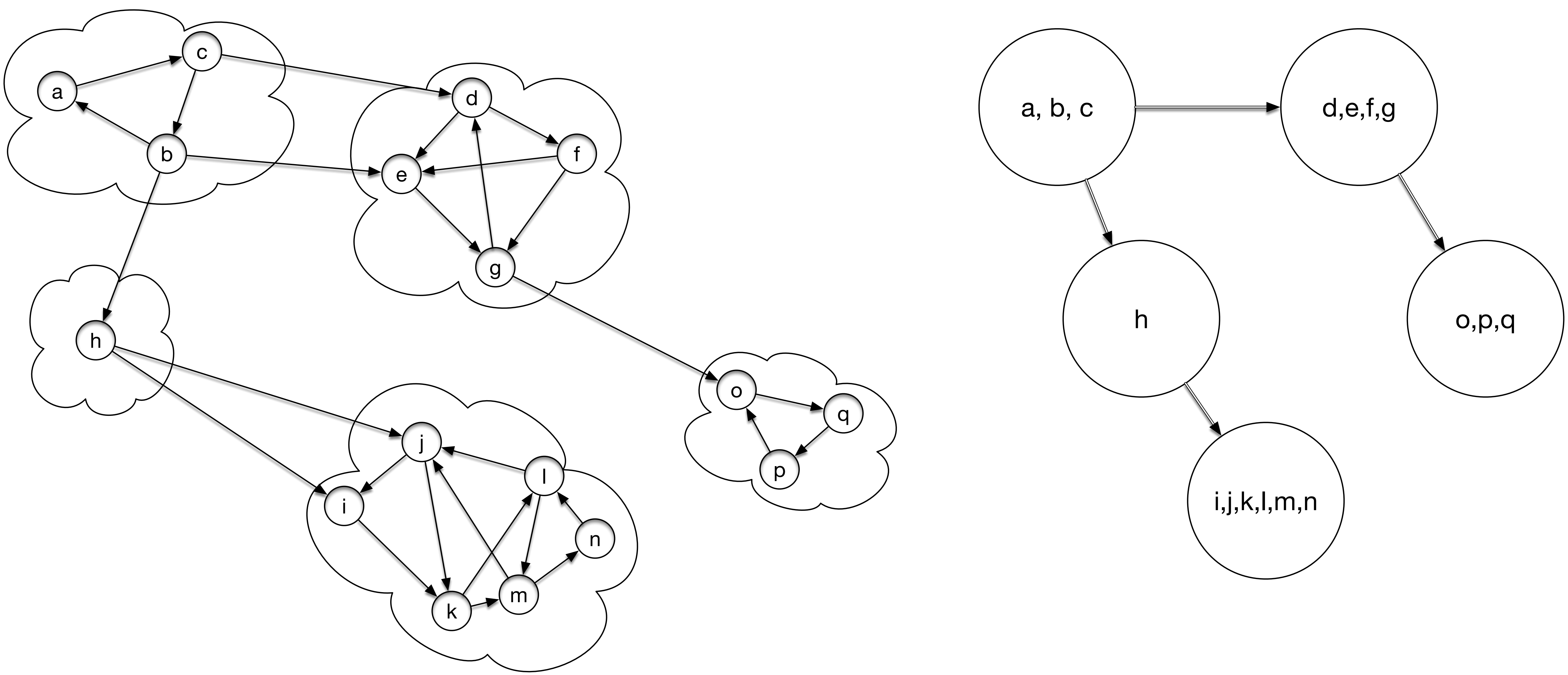
- Result:



Strongly Connected Components

- If we only look at the connected components we get the SCC metagraph
    - Nodes are the strongly connected components
    - Edges represent the existence of an edge from one component to the next

Strongly Connected Components

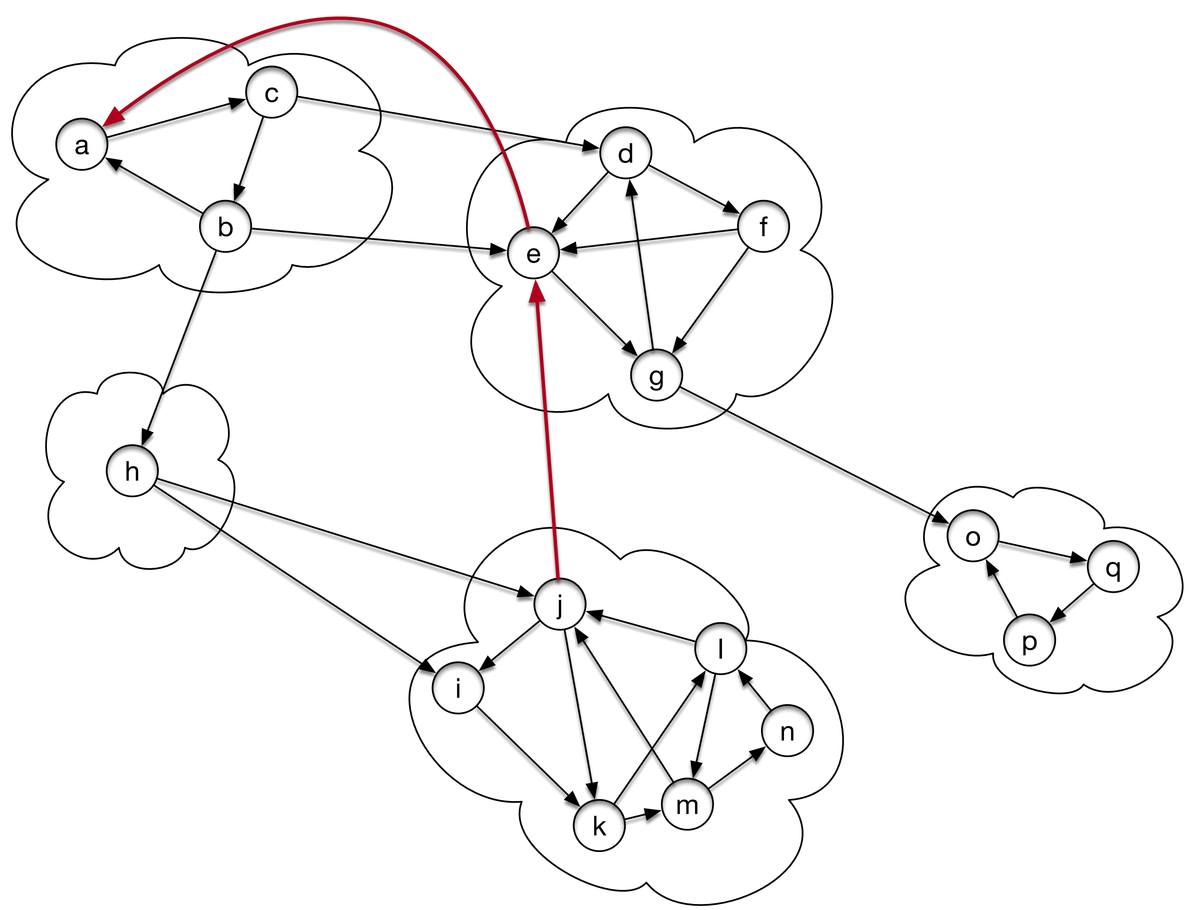


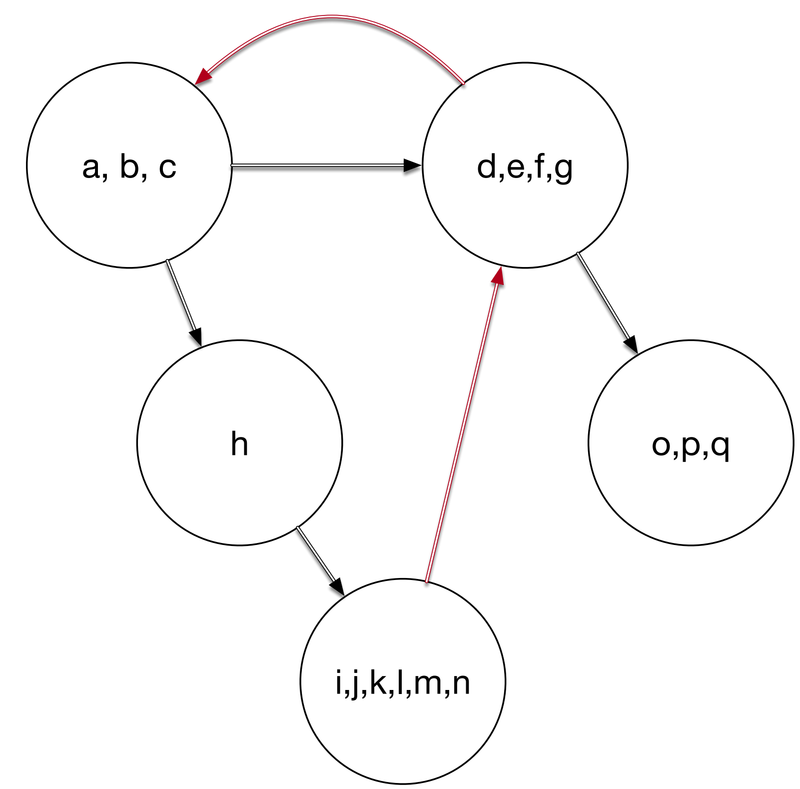
Strongly Connected Components

- The resulting metagraph has to be acyclic
  - If there is a cycle in the metagraph, then by the lemma, the metanodes can be merged into bigger strongly connected subgraphs

Strongly Connected Components

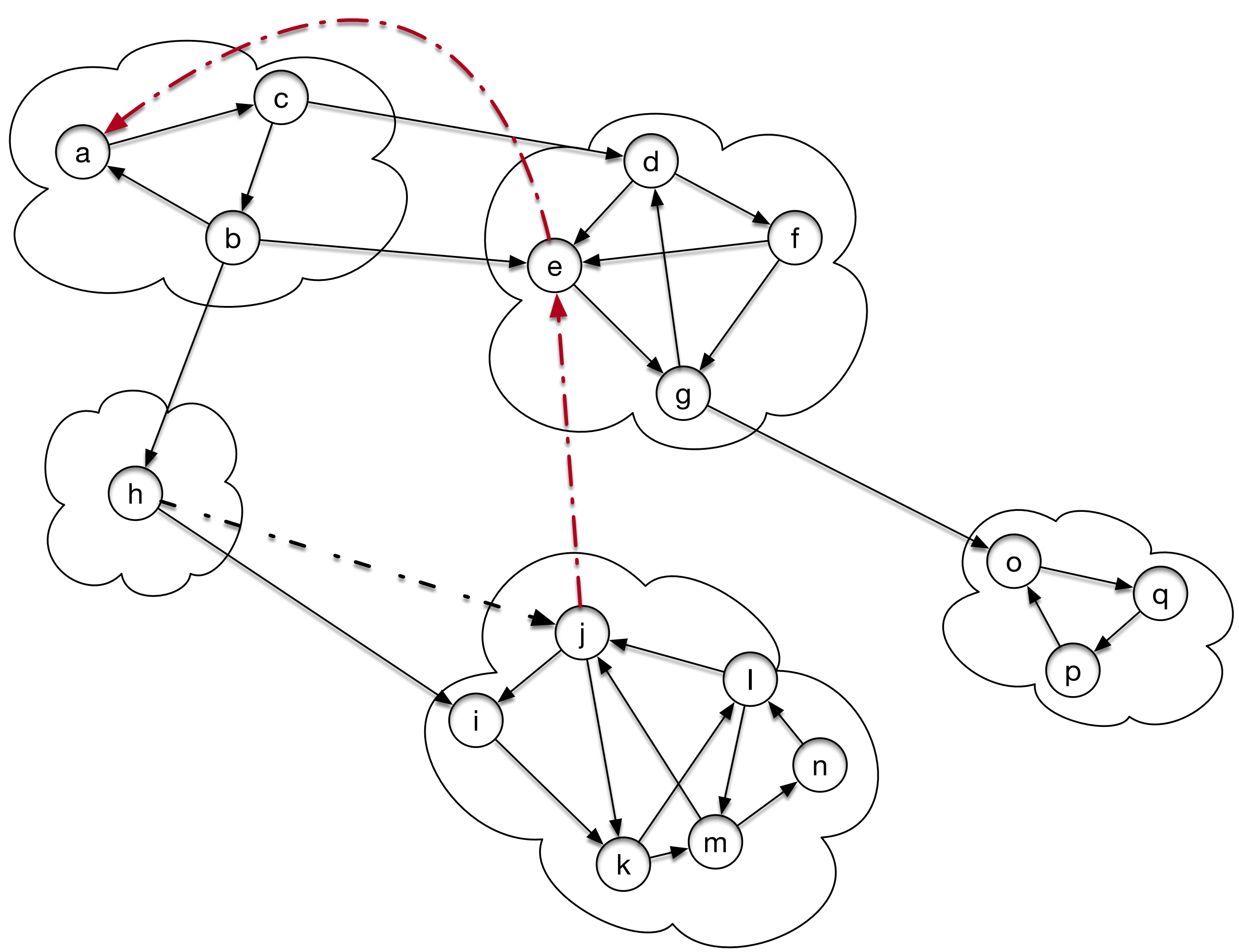
- Example: Add two edges

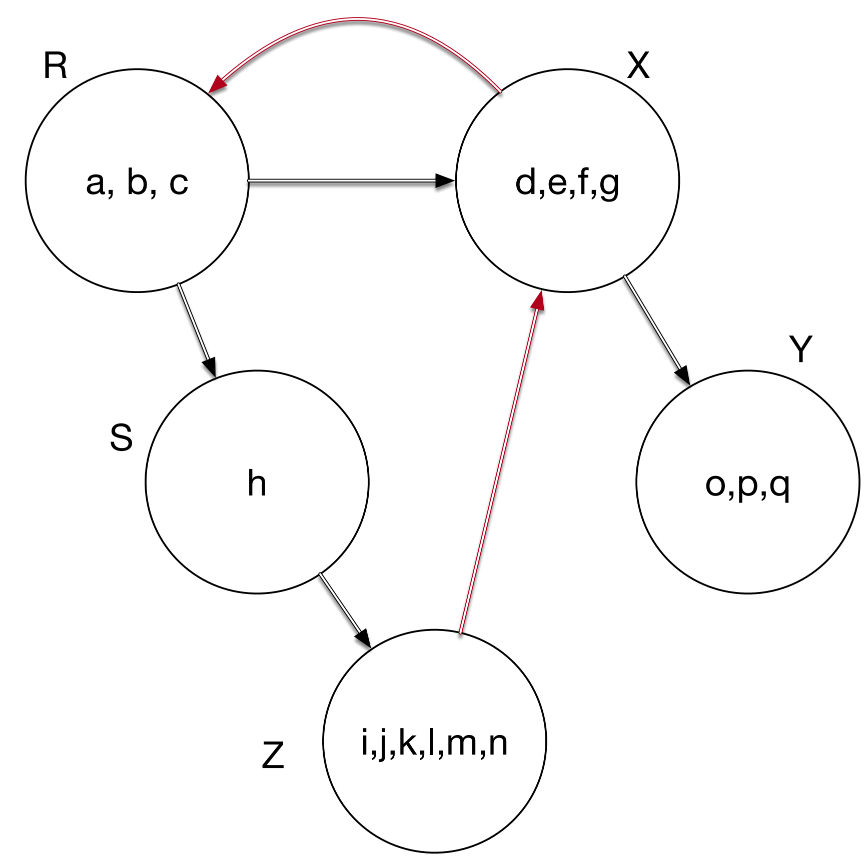




Strongly Connected Components

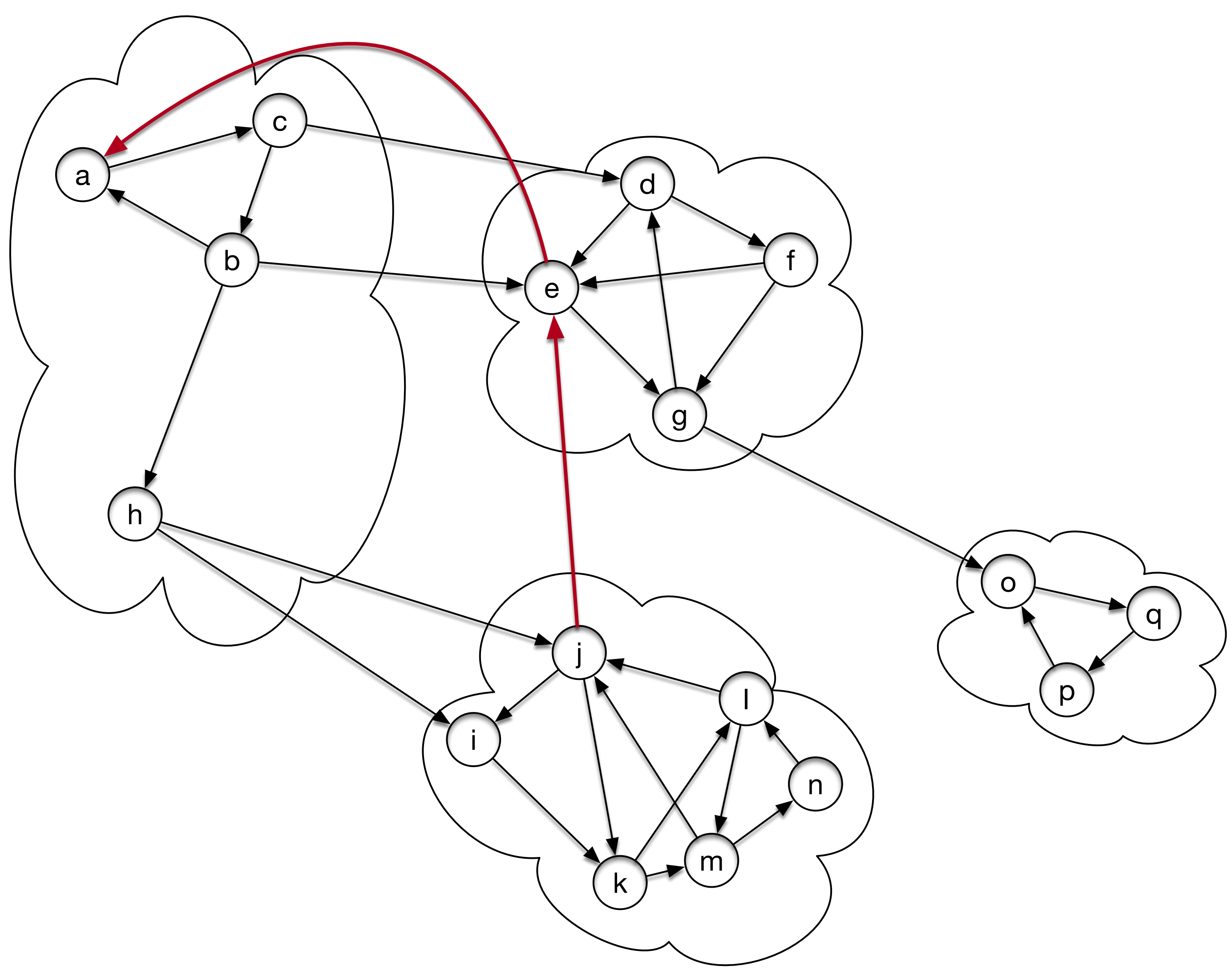
- Now we can start merging via the Lemma
- There is a path from components S to R and vice versa

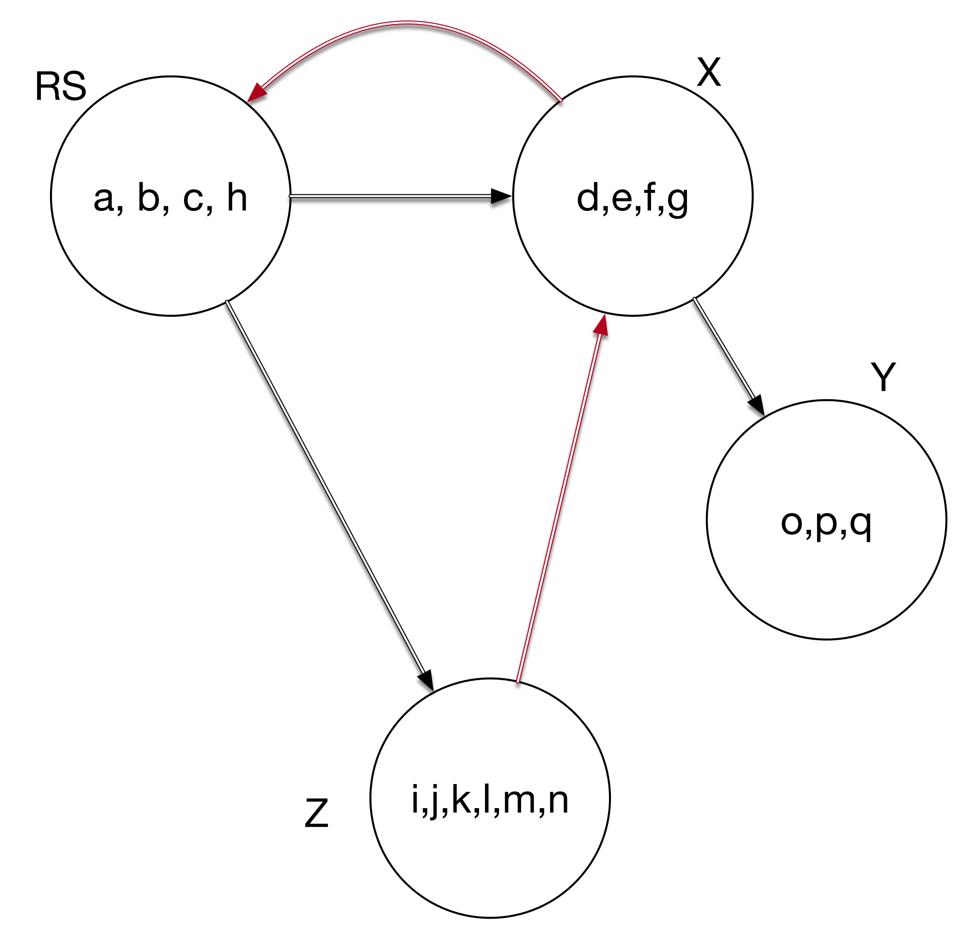




Strongly Connected Components

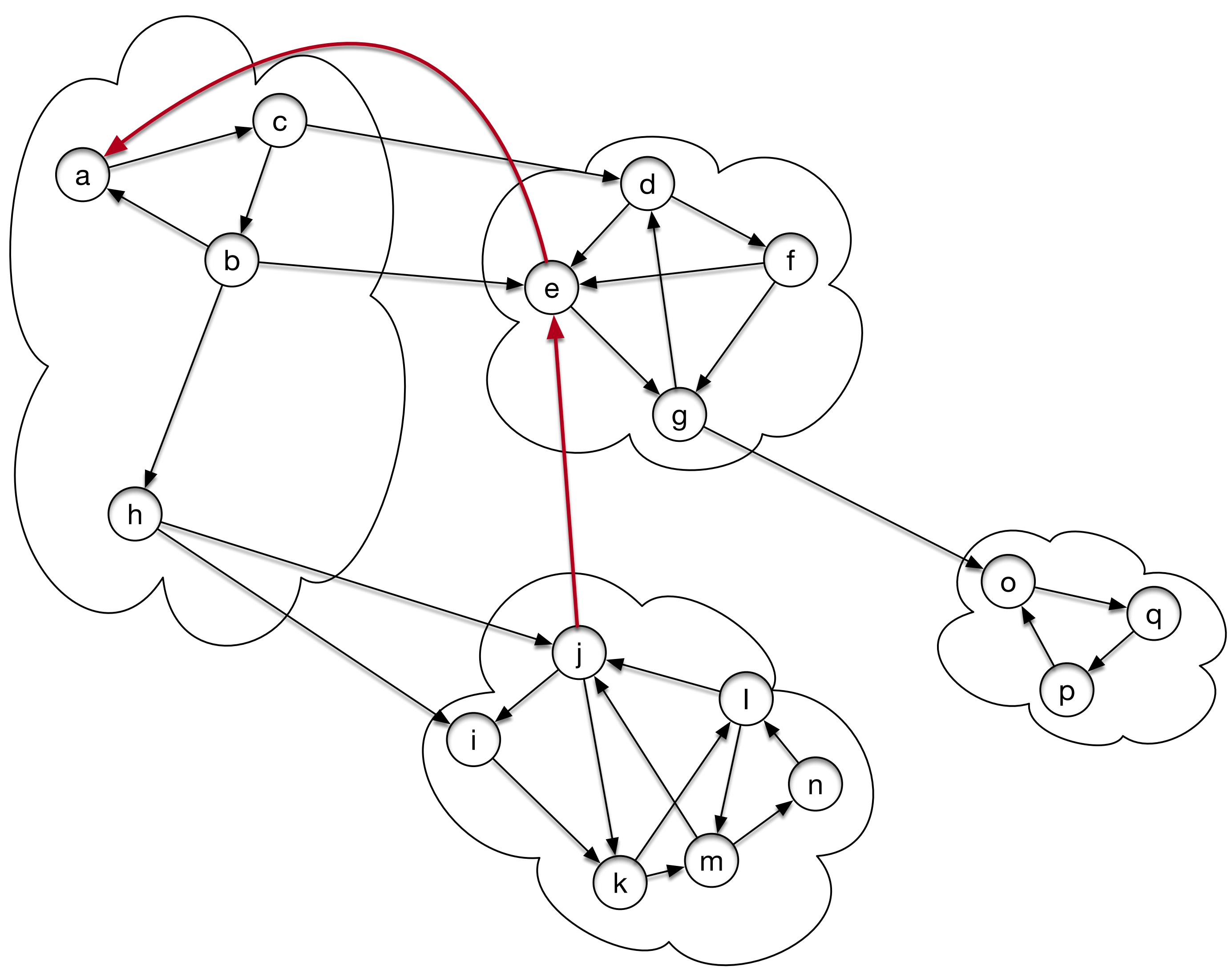
- So we merge

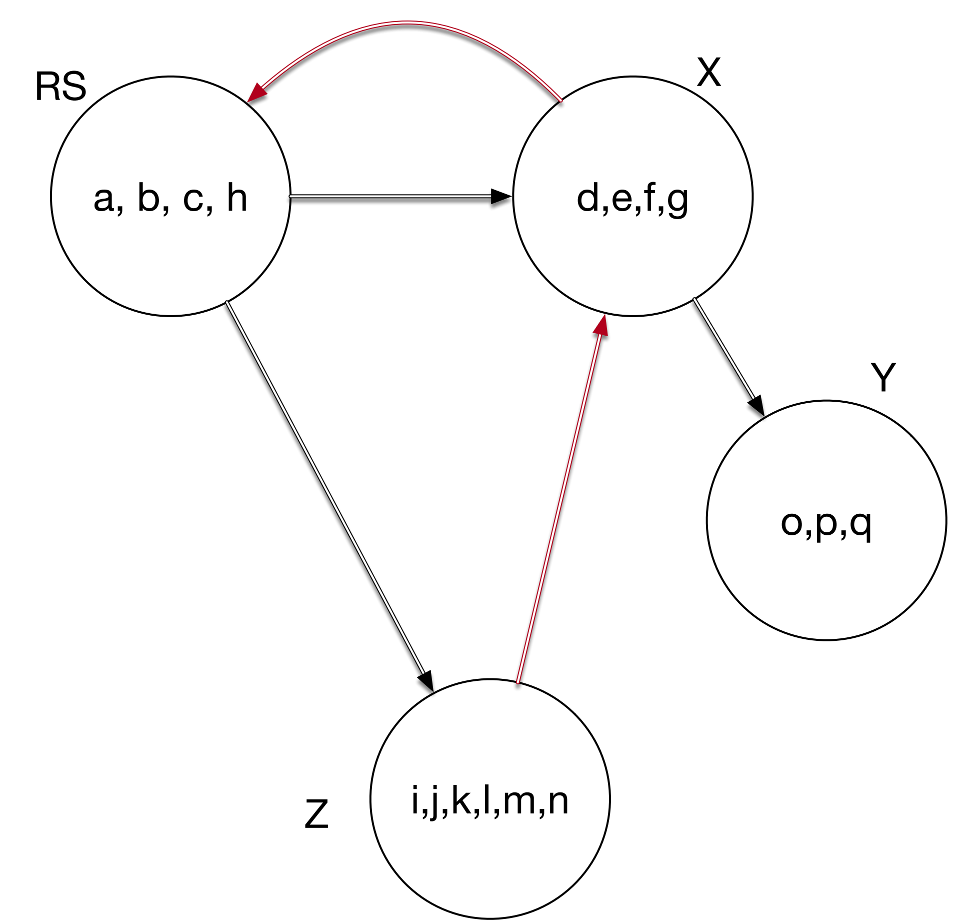




Strongly Connected Components

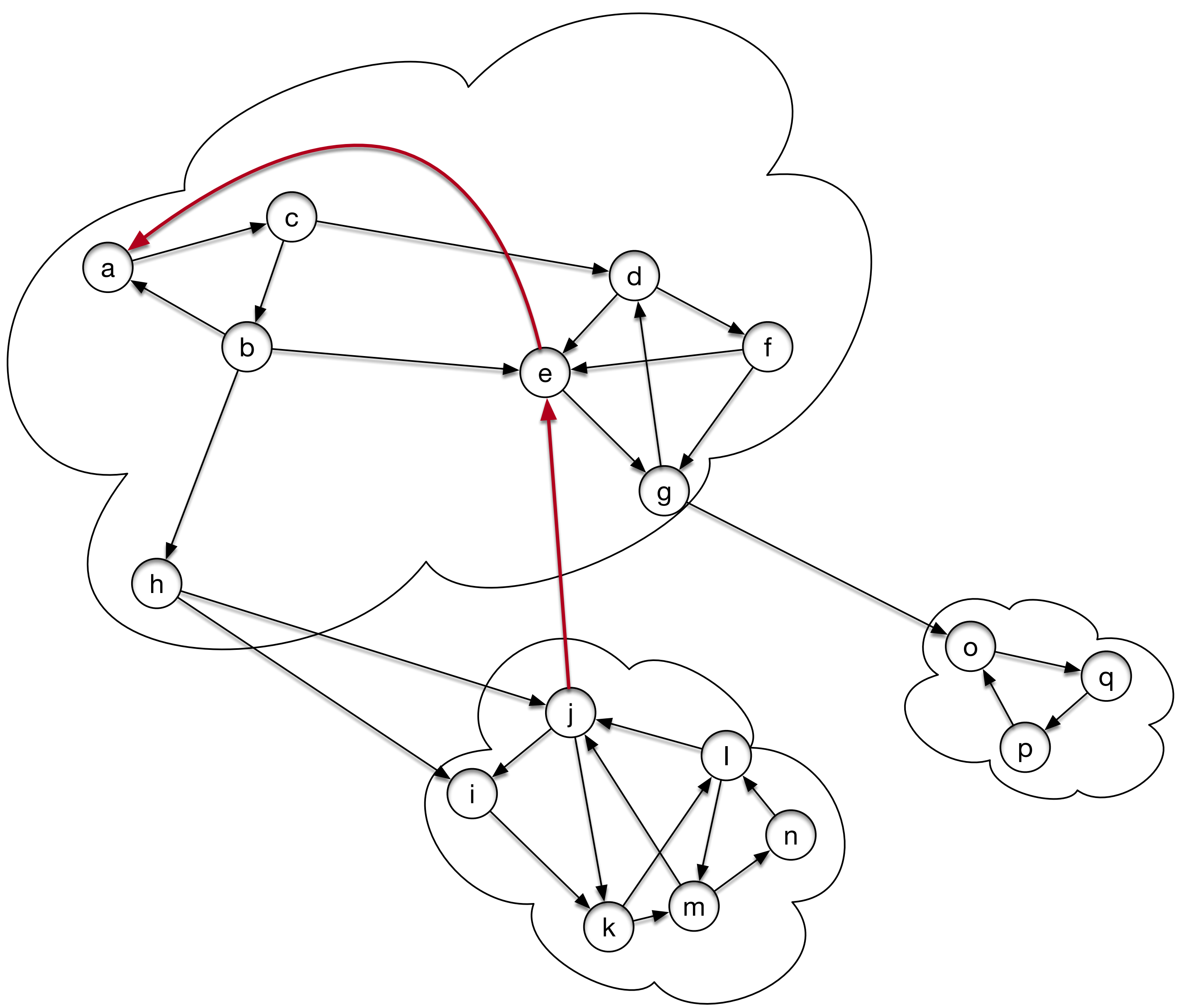
- There is a path from RS to X and vice versa:

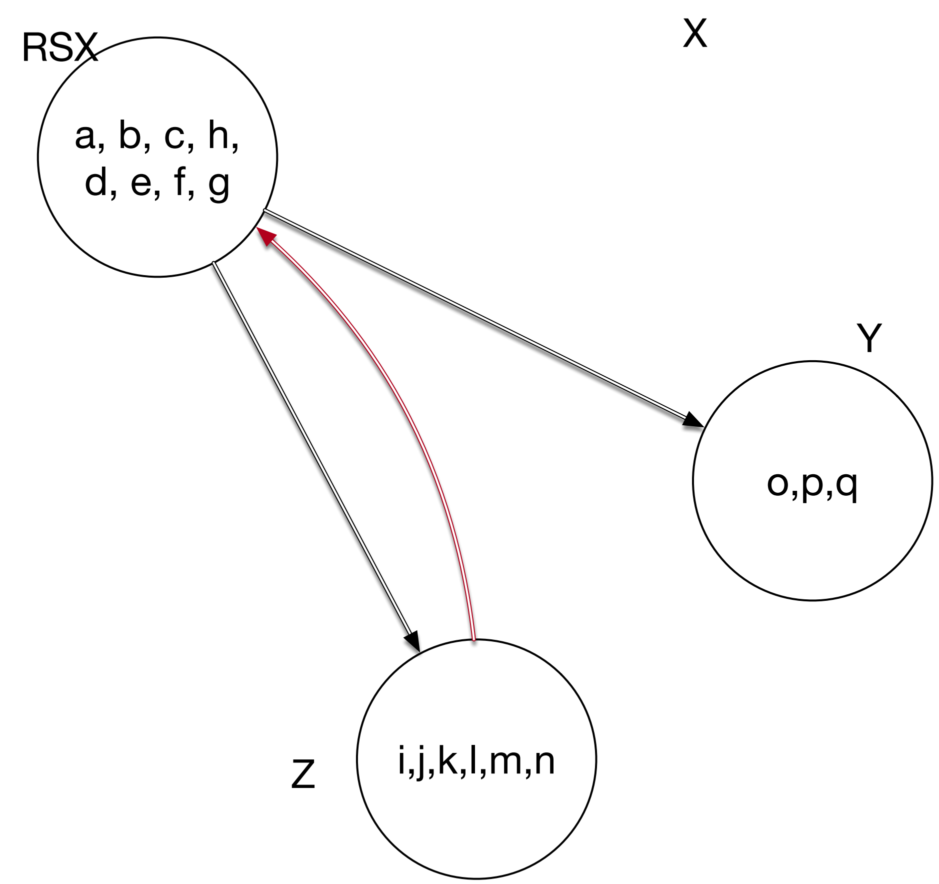




Strongly Connected Components

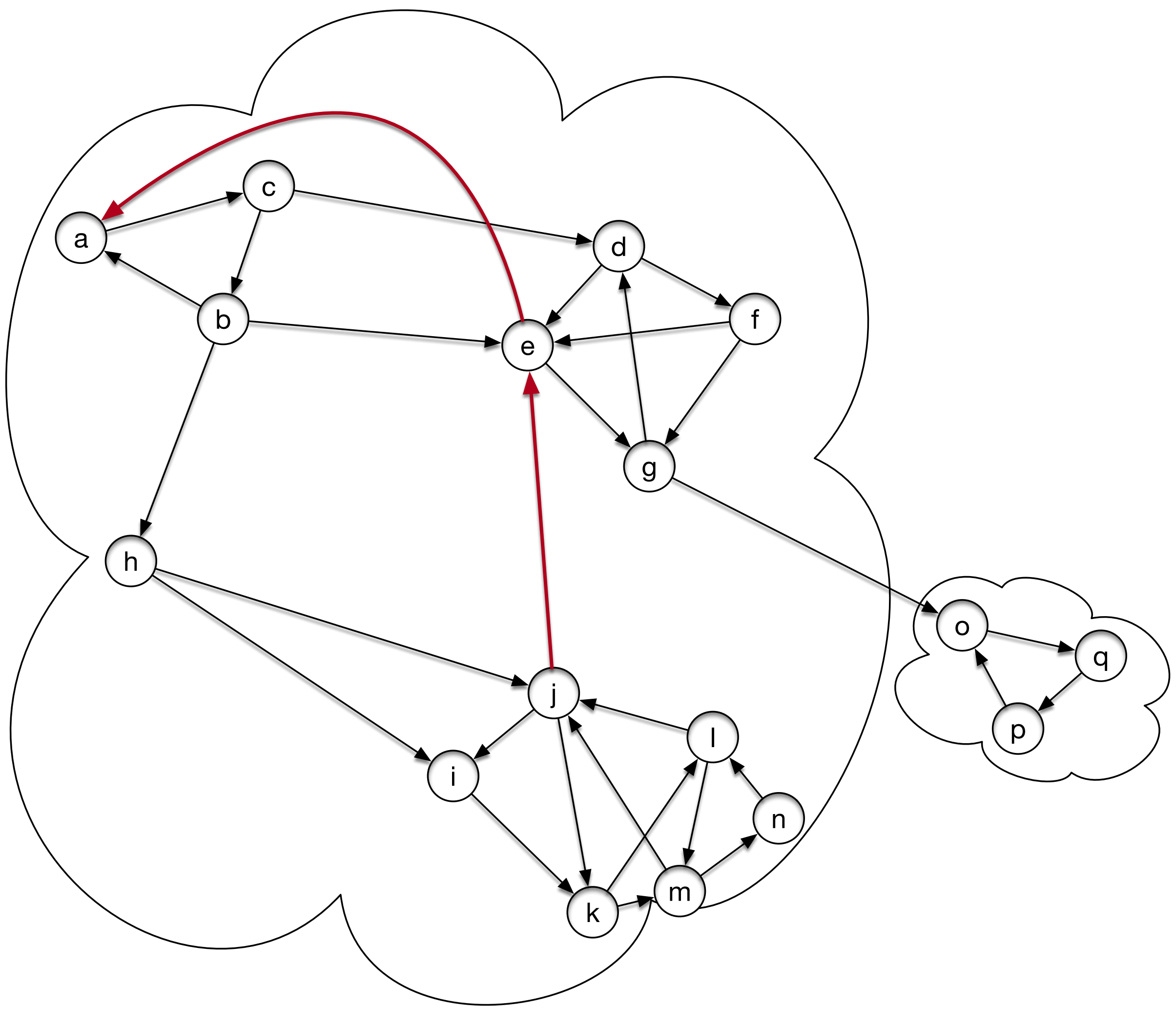
- We can merge

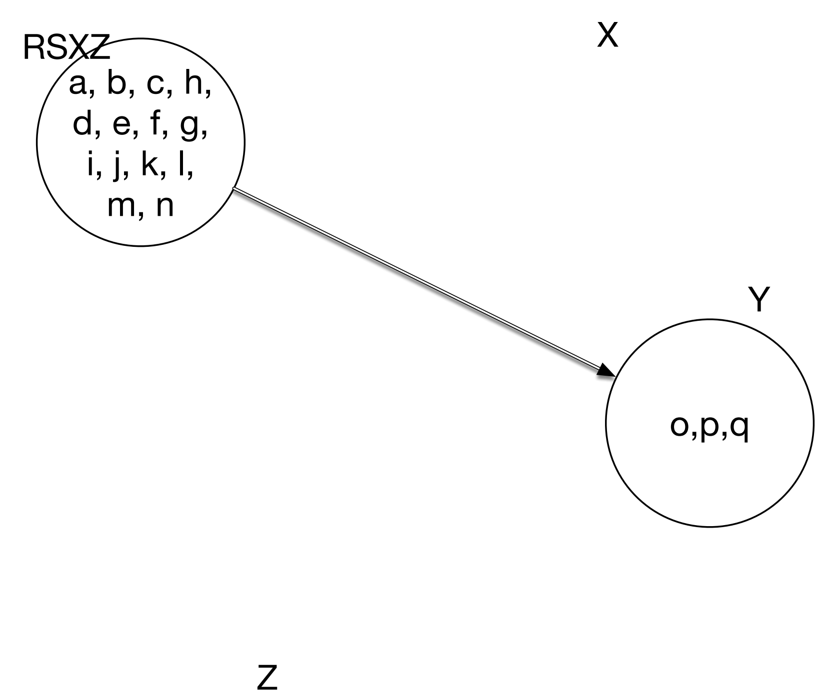




Strongly Connected Components

- Finally, we can merge Z with the new supernode





Strongly Connected Components

- This can be generalized:
  - Theorem: The metagraph is acyclic

Strongly Connected Components

- How can we apply DFS to the problem of determining connected components?
  - The WWW graph in 2000 would have been to big for anything but linear time algorithms

Strongly Connected Components

- Answer:
  - Use DFS several times
  - Including indirectly on the metagraph

Strongly Connected Components

Strongly Connected Components