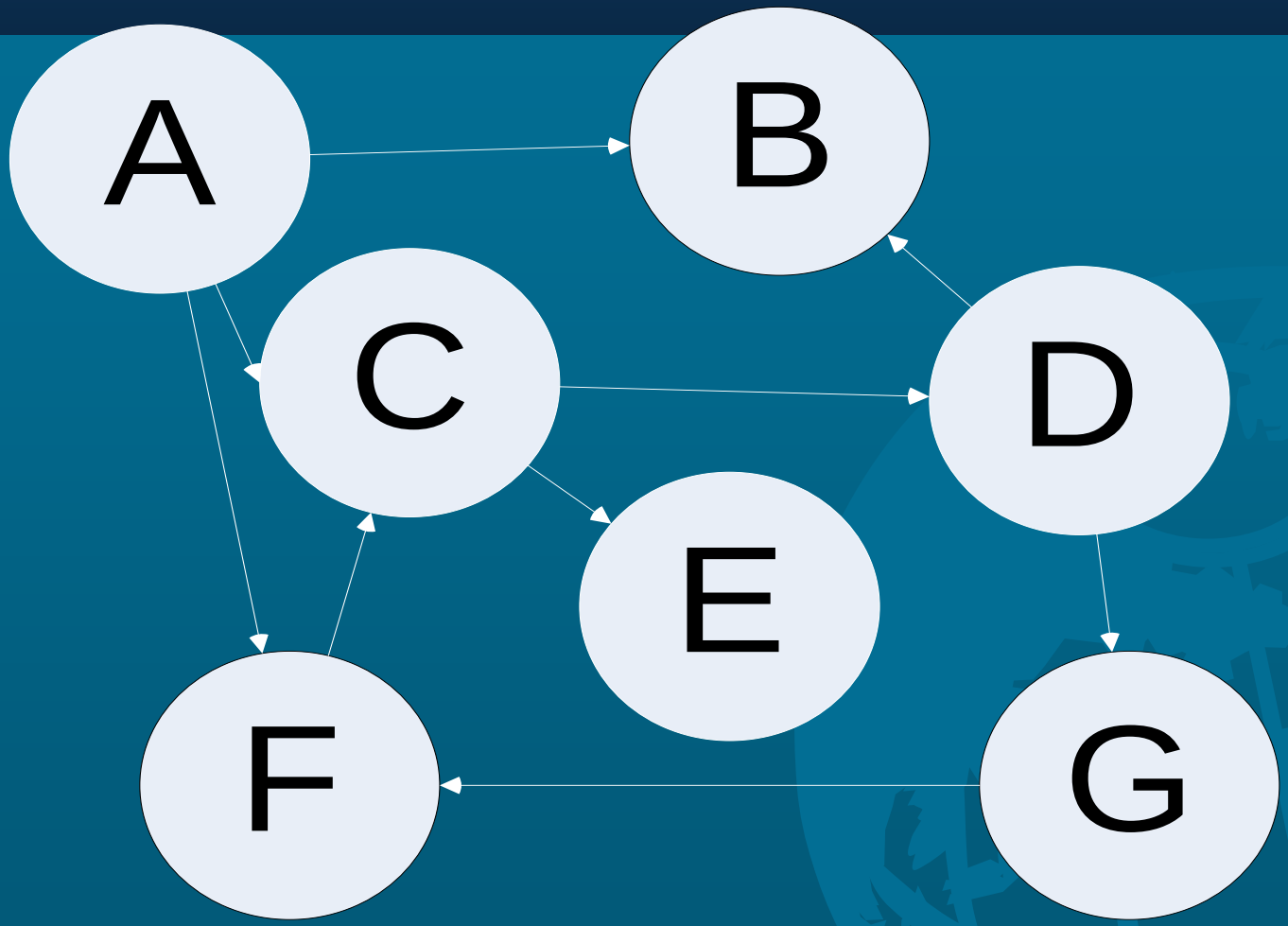
Company
LOGO

Strongly Connected Components Detection

Strongly Connected Components

- A directed graph is called **strongly connected** if there is a path from each vertex in the graph to every other vertex.

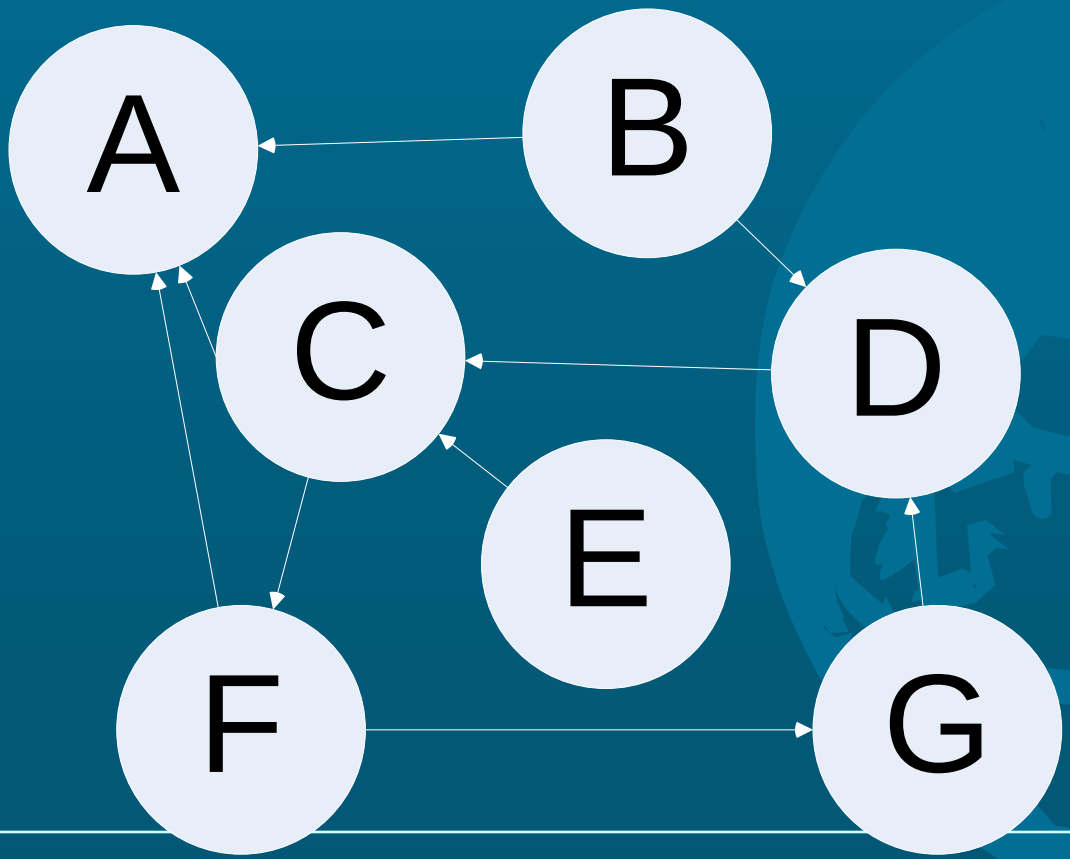
- The **strongly connected components (SCC)** of a directed graph are its maximal strongly connected subgraphs.



- CDGF is the only strongly connected component in the graph

Transport of Graph

- Transport of Graph: G^T is Graph with all edges reversed.



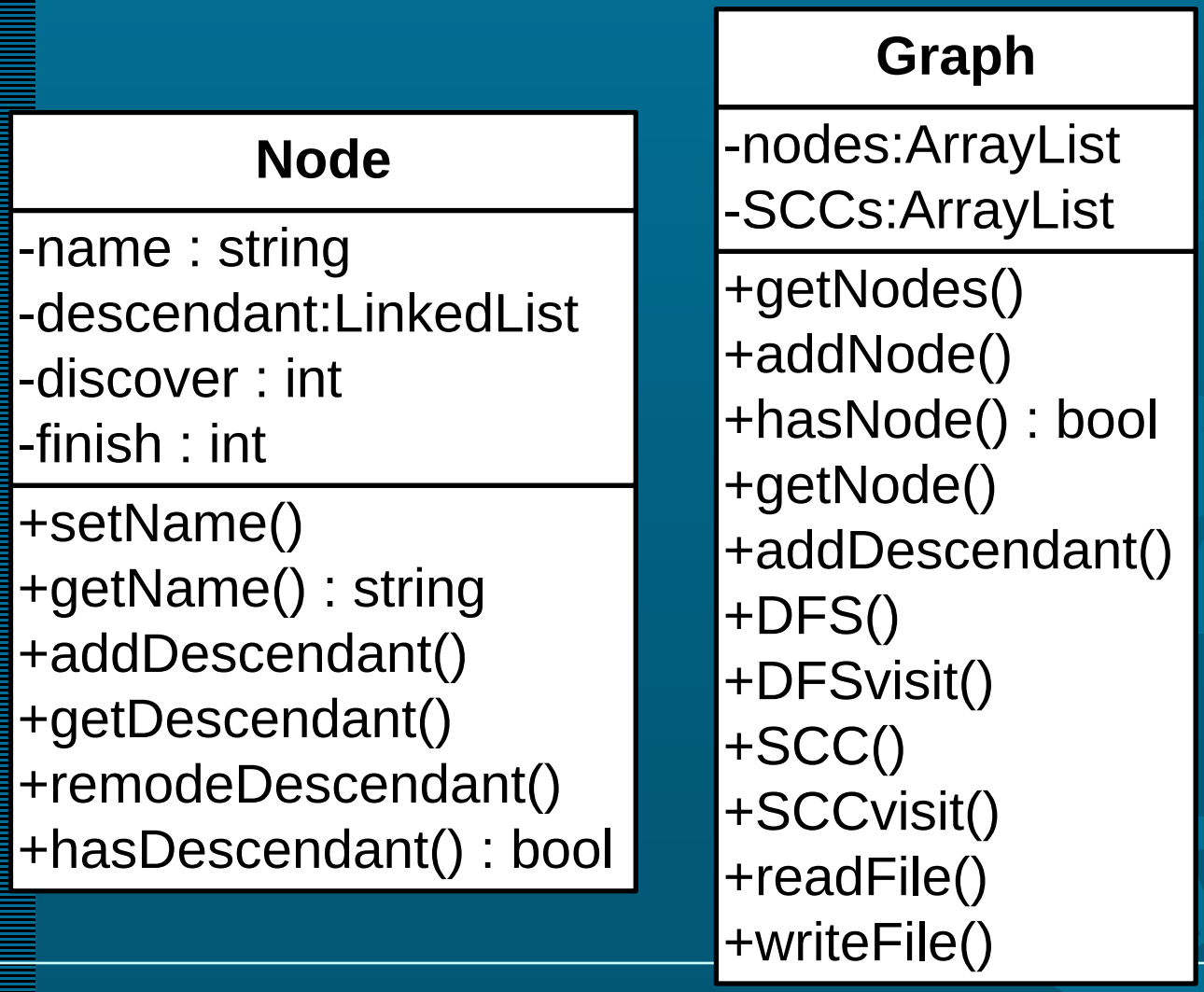
Algorithm to detect SCCs

1. call DFS(G) to compute finishing times $f[u]$ for all u
2. compute G^T
3. call DFS(G^T), but in the main loop, consider vertices in order of decreasing $f[u]$ (as computed in first DFS)
4. output the vertices in each tree of the depth-first forest formed in second DFS as a separate SCC

Object Oriented Design

- A Node has name, time stamps, descendants in LinkedList format.
- A Graph has many Nodes in an Array
- All manipulations are done by methods of the objects

Object Oriented Design



Programming Basics

- Graph aGraph = new Graph(); //instance of Graph
- public void readFile(String fileName)

  Add information to Graph from a file. For example “aGraph.readFile(“c:\graph.txt”);”
- public void SCC()

  Find Strongly Connected Components in aGraph by using “aGraph.SCC ();”

Programming Details

- public void addNode(Node node)

  Add a node to aGraph by using ”aGraph.addNode (node);”. The program will also add all descendants to the graph.
- public void addDescendant(String node, String dNode)

  Add nodes with specific names to the graph and the second node is the descendant of first node.

Programming Details

- public String writeFile(String fileName)

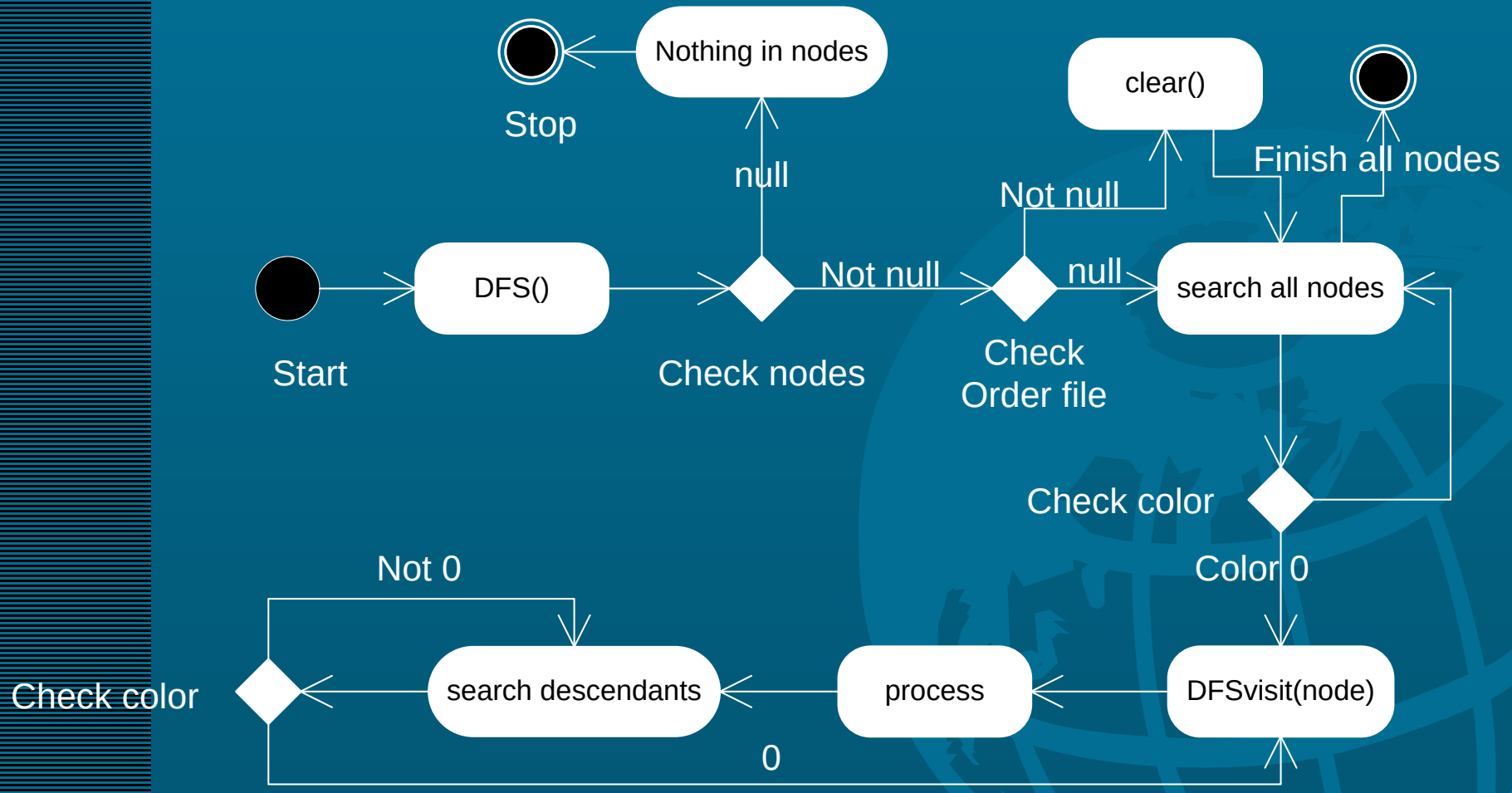
  Store all information of the graph to a file. For example “aGraph.wrieteFile(“c:\graph.txt”);”

Depth First Searching

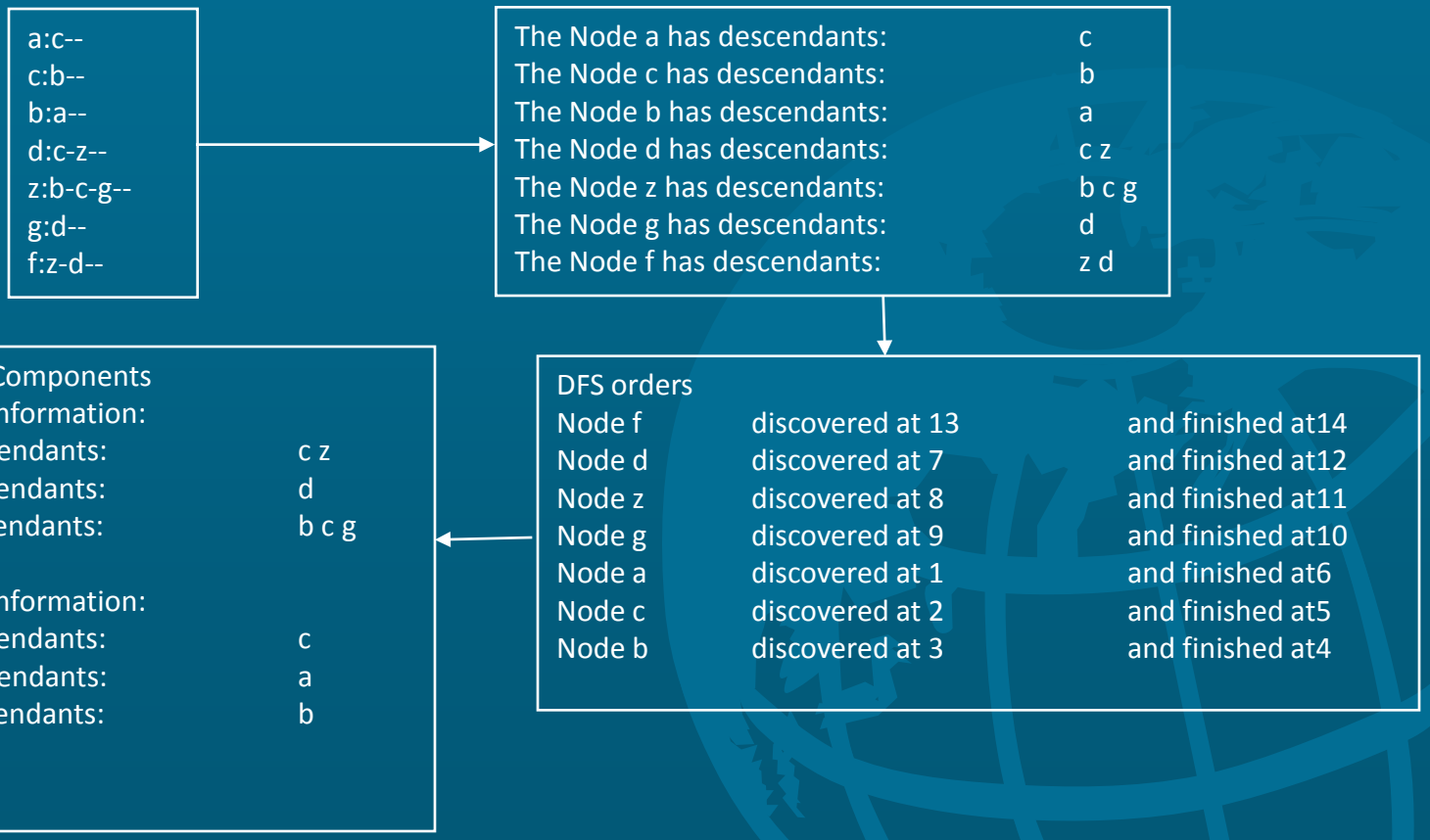
```
public void DFS()
{
if (nodes != null)
{
    if (DFSorder != null && DFSorder.Count>0)
    {
        DFSorder.Clear();   //Clean the order file before search
    }
    foreach (Node aNode in nodes)
    {
        aNode.color = 0;    //set all nodes to color 0
    }
    count = 0;
    foreach (Node aNode in nodes)
    {   //find all nodes with color 0 and visit them
        if (aNode.color == 0) DFSvisit(aNode);
    }
}
}
```

```
public void DFSvisit(Node aNode)
{
    aNode.color = 1;    //change color
    count++;
    aNode.discover = count; //set discover count
    if (aNode.getDescendant() != null)
    {
        foreach (Node bNode in aNode.getDescendant())
        {
            if (bNode.color == 0)
            {
                DFSvisit(bNode);    //recursive method
            }
        }
    }
    aNode.color = 2;    //finish count
    count++;
    aNode.finish = count;
    if (DFSorder == null) DFSorder = new ArrayList();
    DFSorder.Insert(0, aNode);  //add node to order array
}
```

Processing


Nothing in nodes
Stop
null
clear()
Finish all nodes
Not null
DFS()
Not null
null
search all nodes
Start
Check nodes
Check
Order file
Check color
Not 0
Color 0
Check color
search descendants
process
DFSvisit(node)
0

Program working order



User Interface