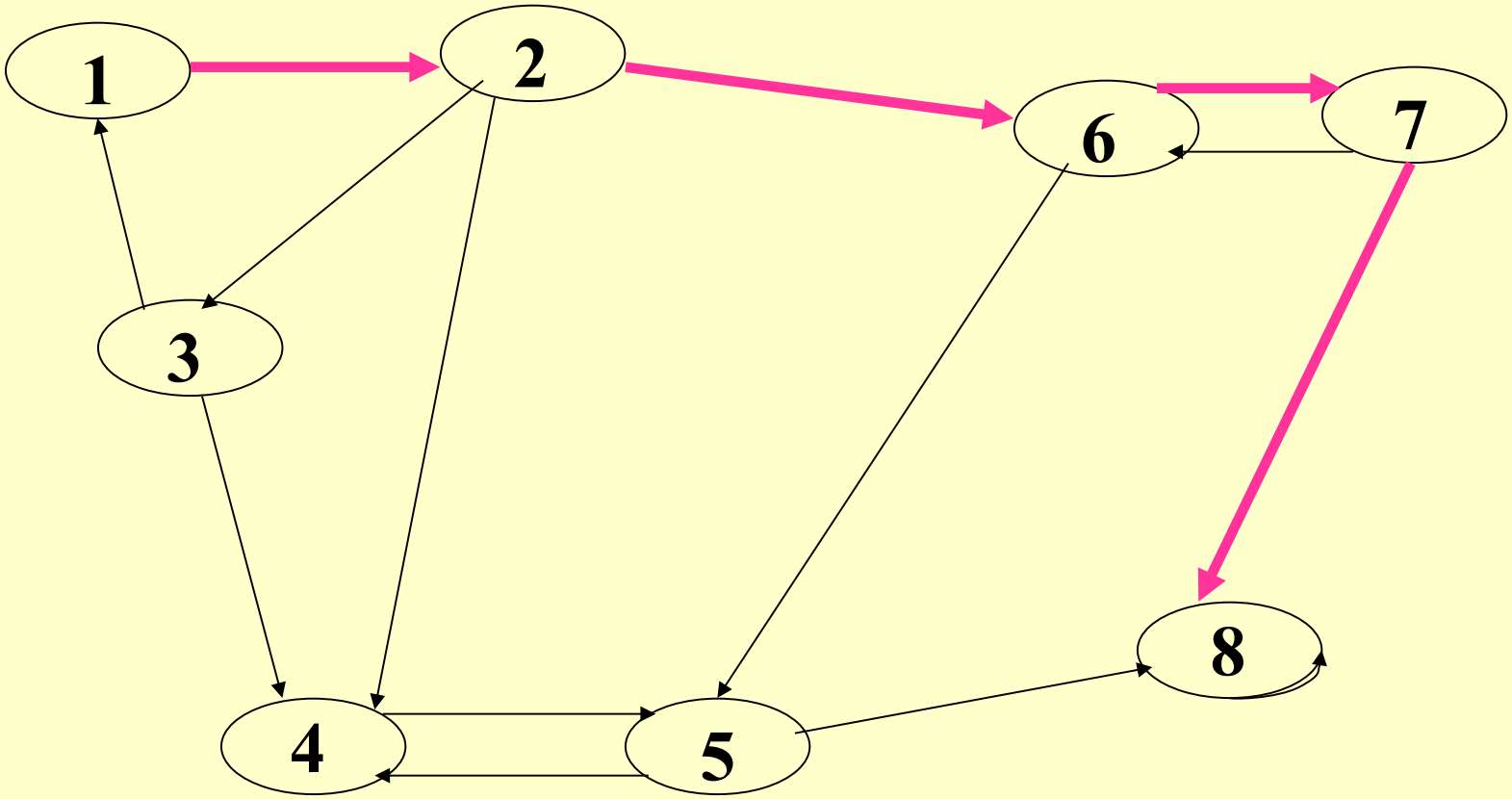
A Fully Dynamic Reachability Algorithm for Directed Graphs with an Almost Linear Update Time

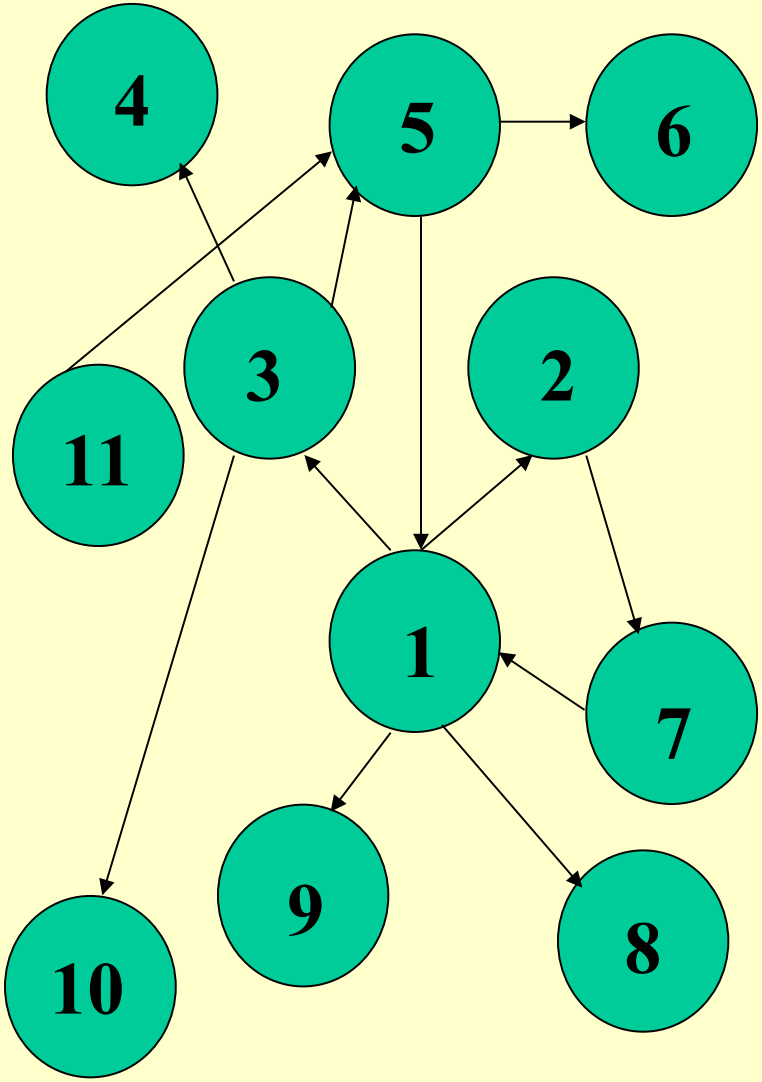
by Liam Roditty and Uri Zwick at STOC 04

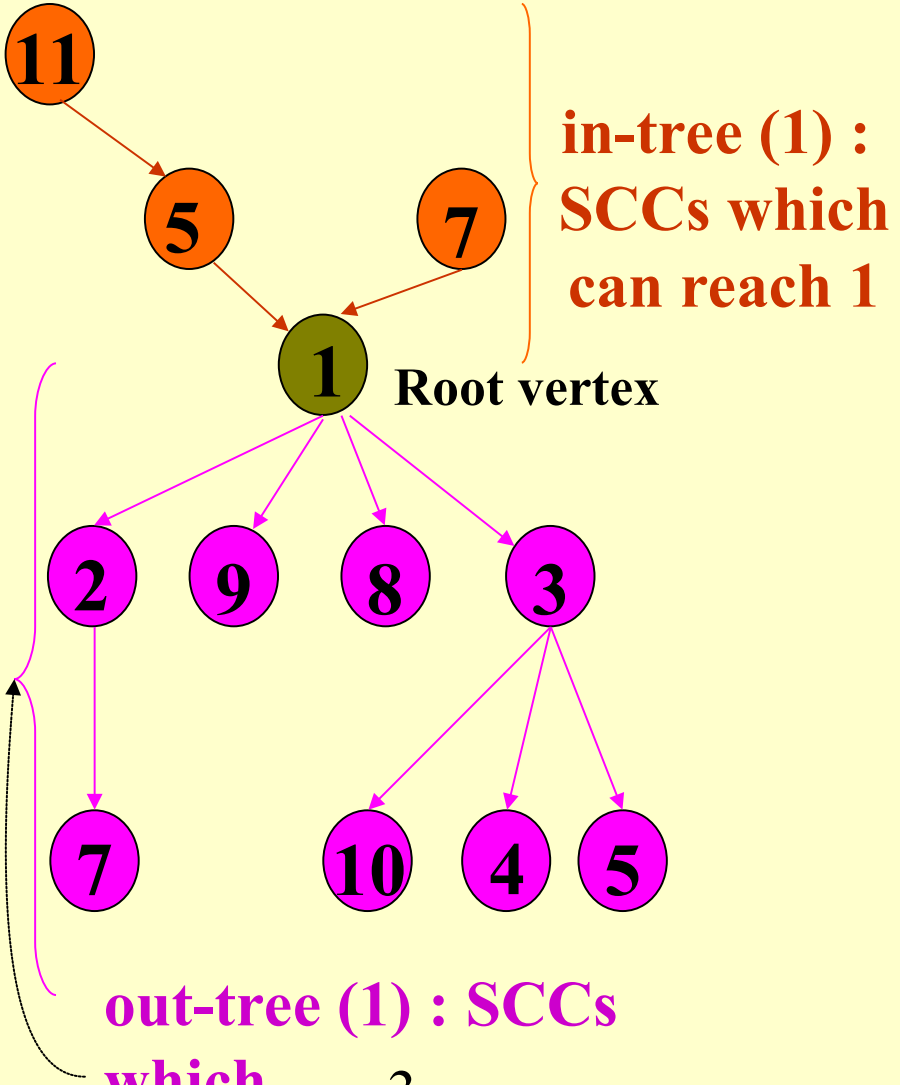
presented by R. Inkulu

Reachability

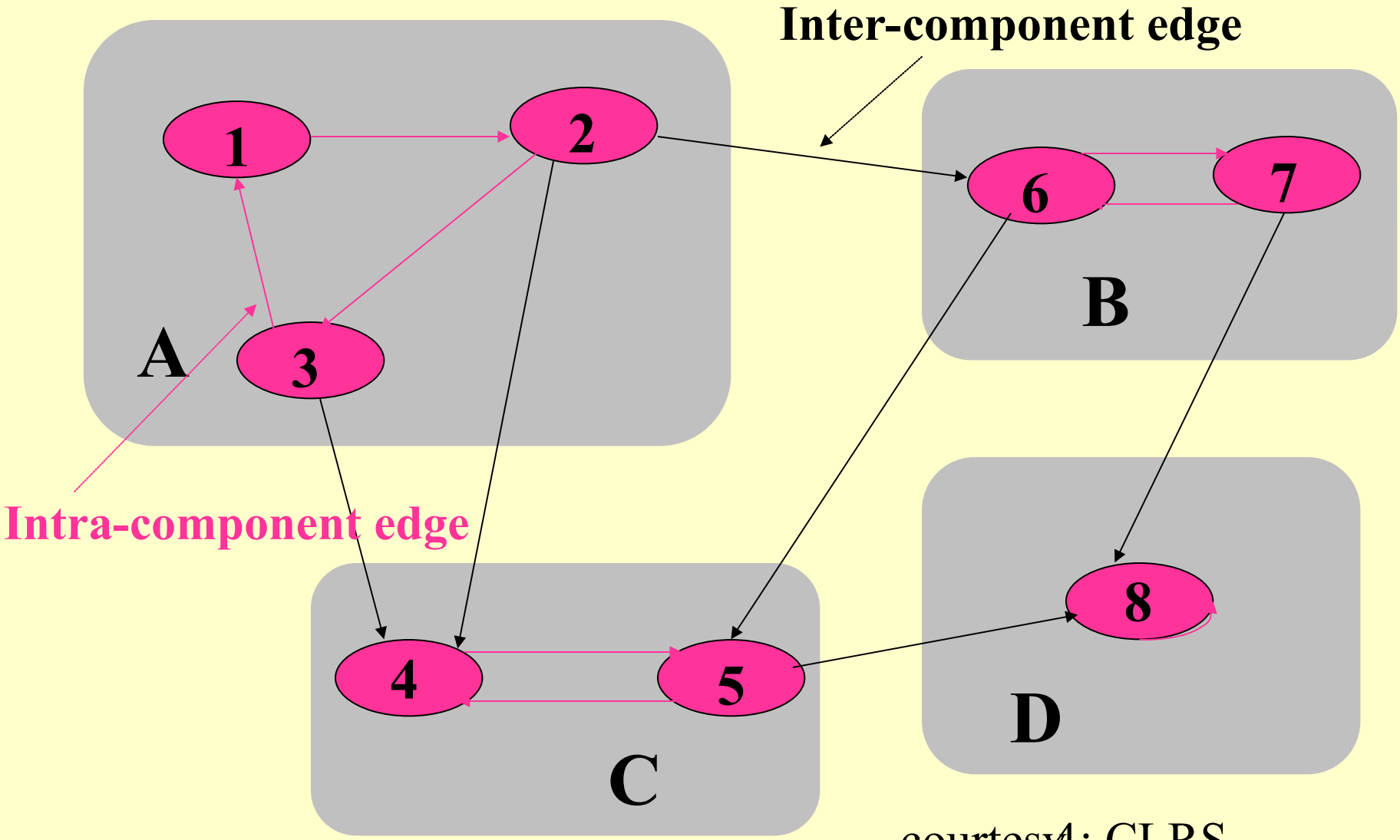


Reachability Tree from a Vertex



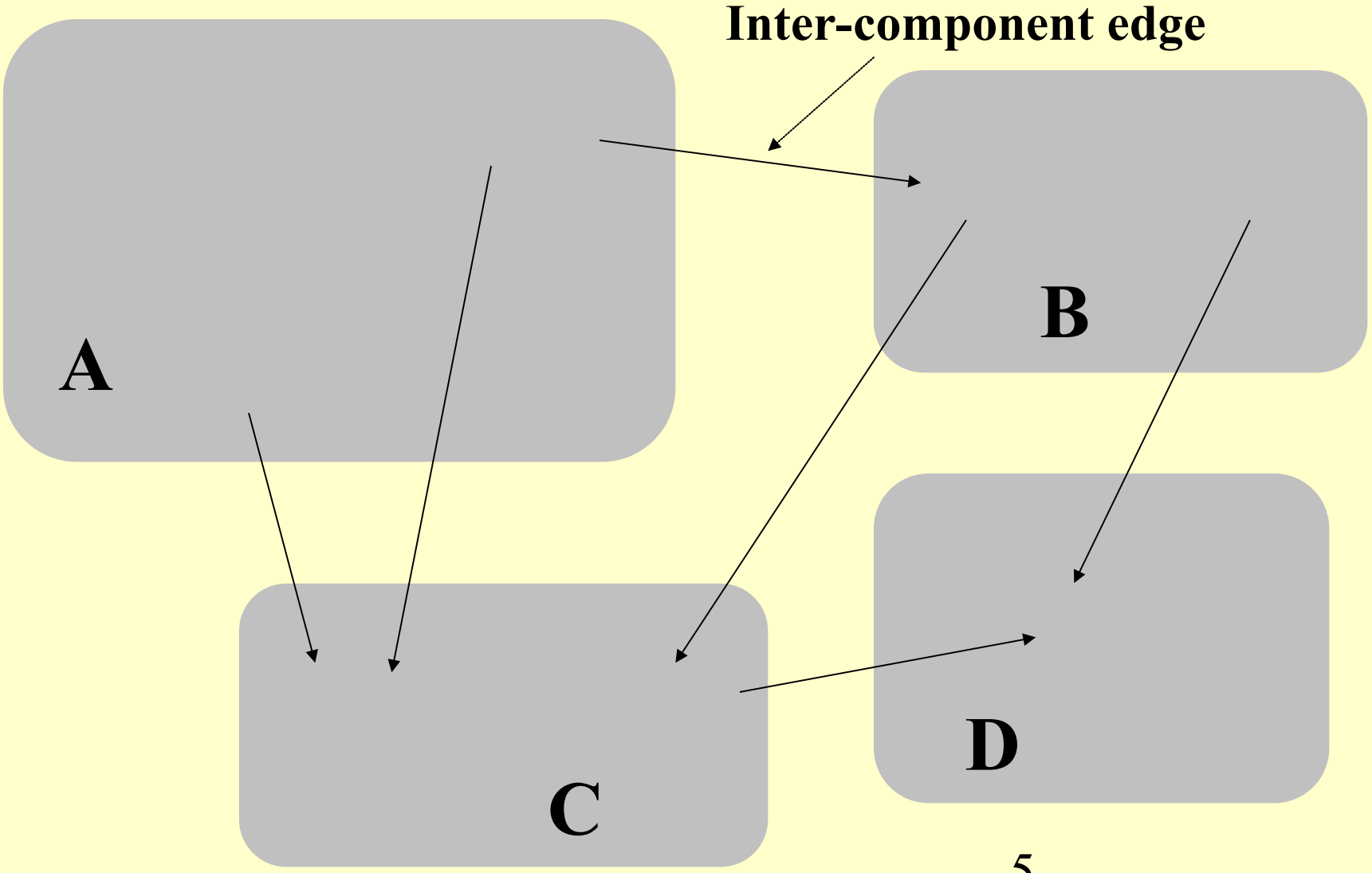


Strongly Connected Component (SCC)

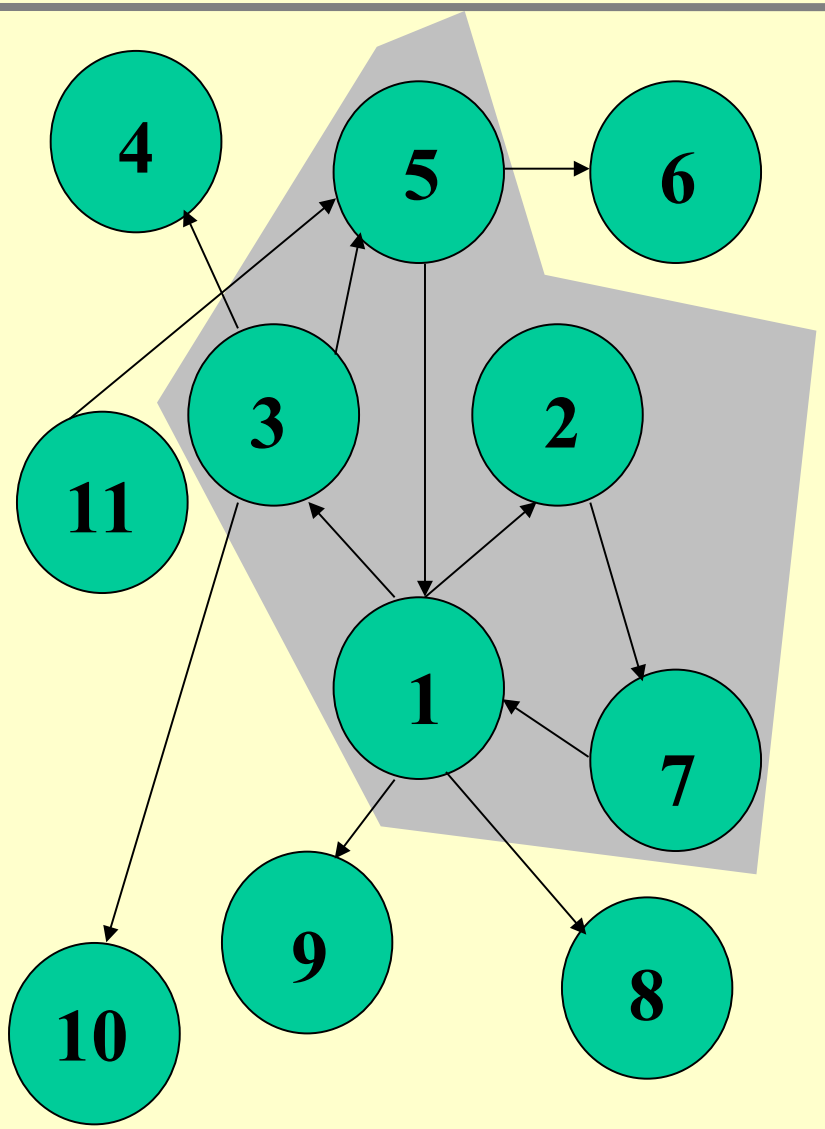


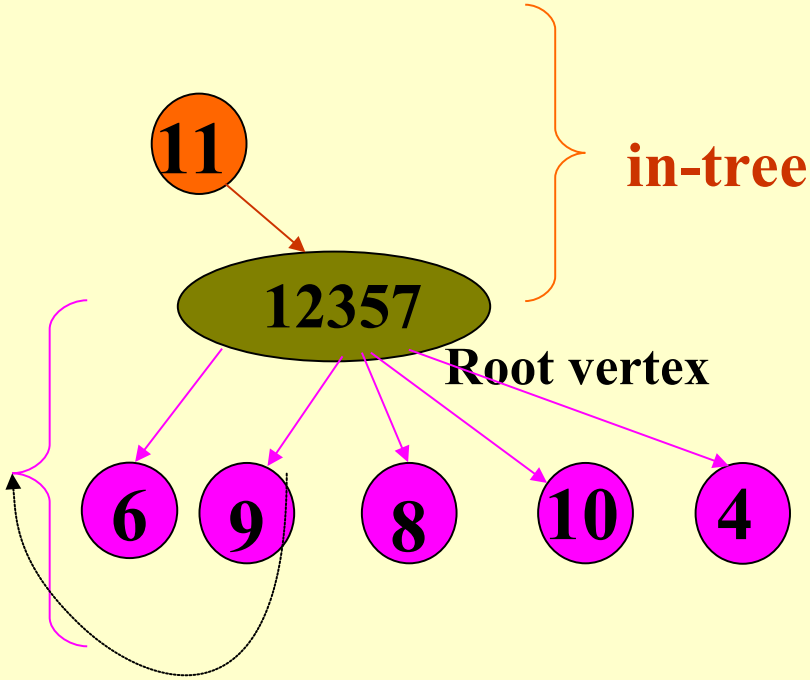
courtesy: CLRS

Strong Component Graph



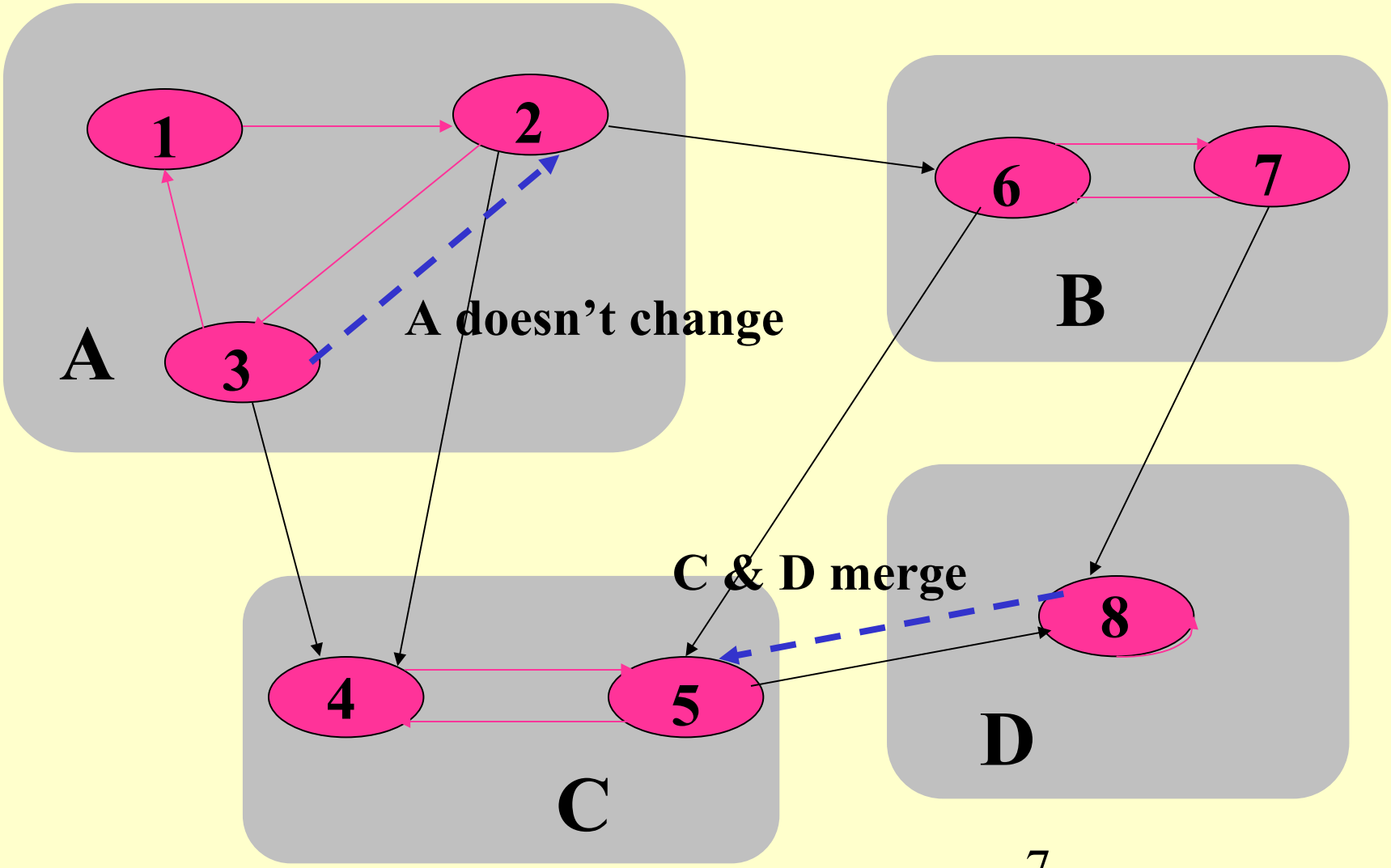
Reachability Tree among SCCs



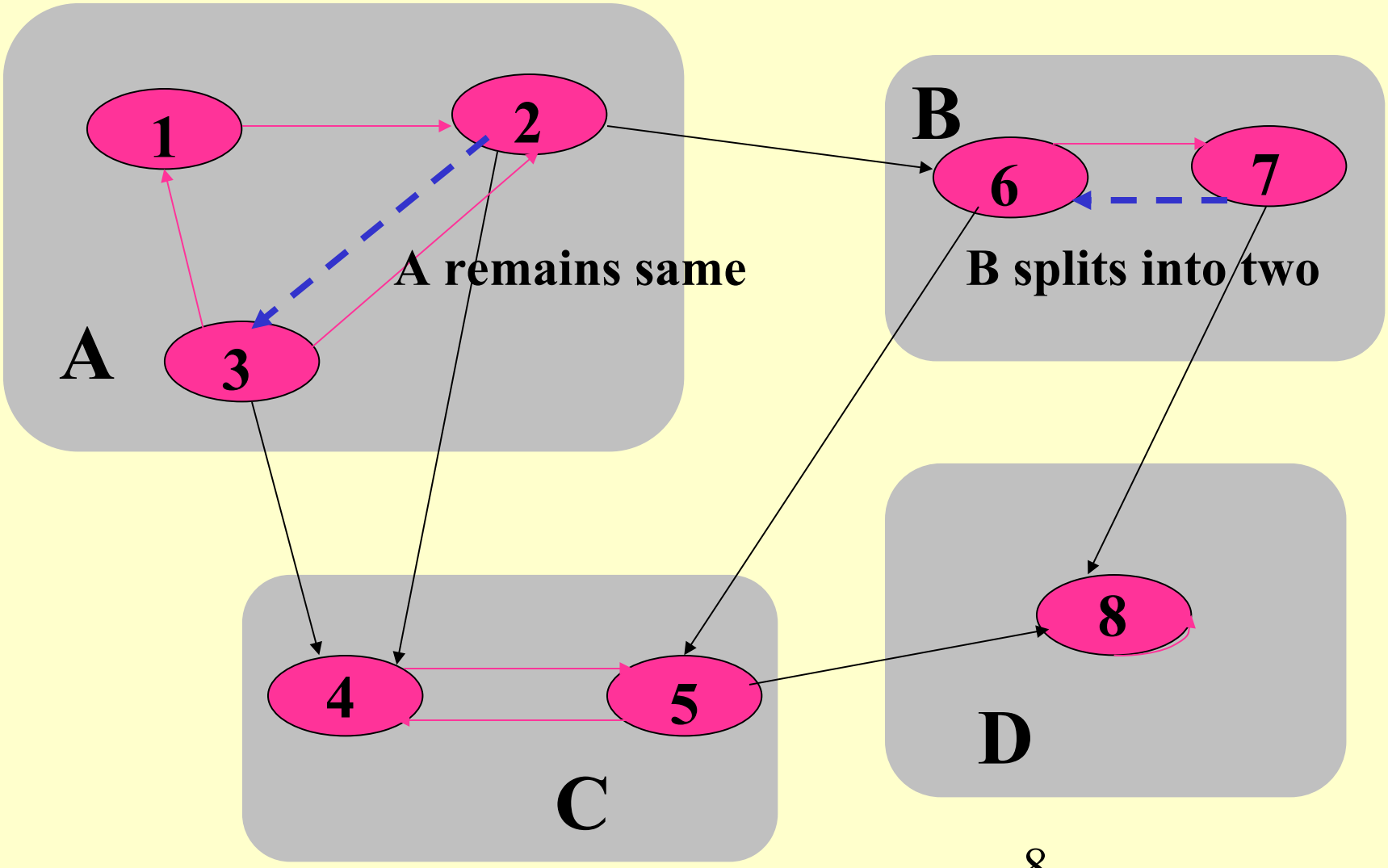


out-tree (1)

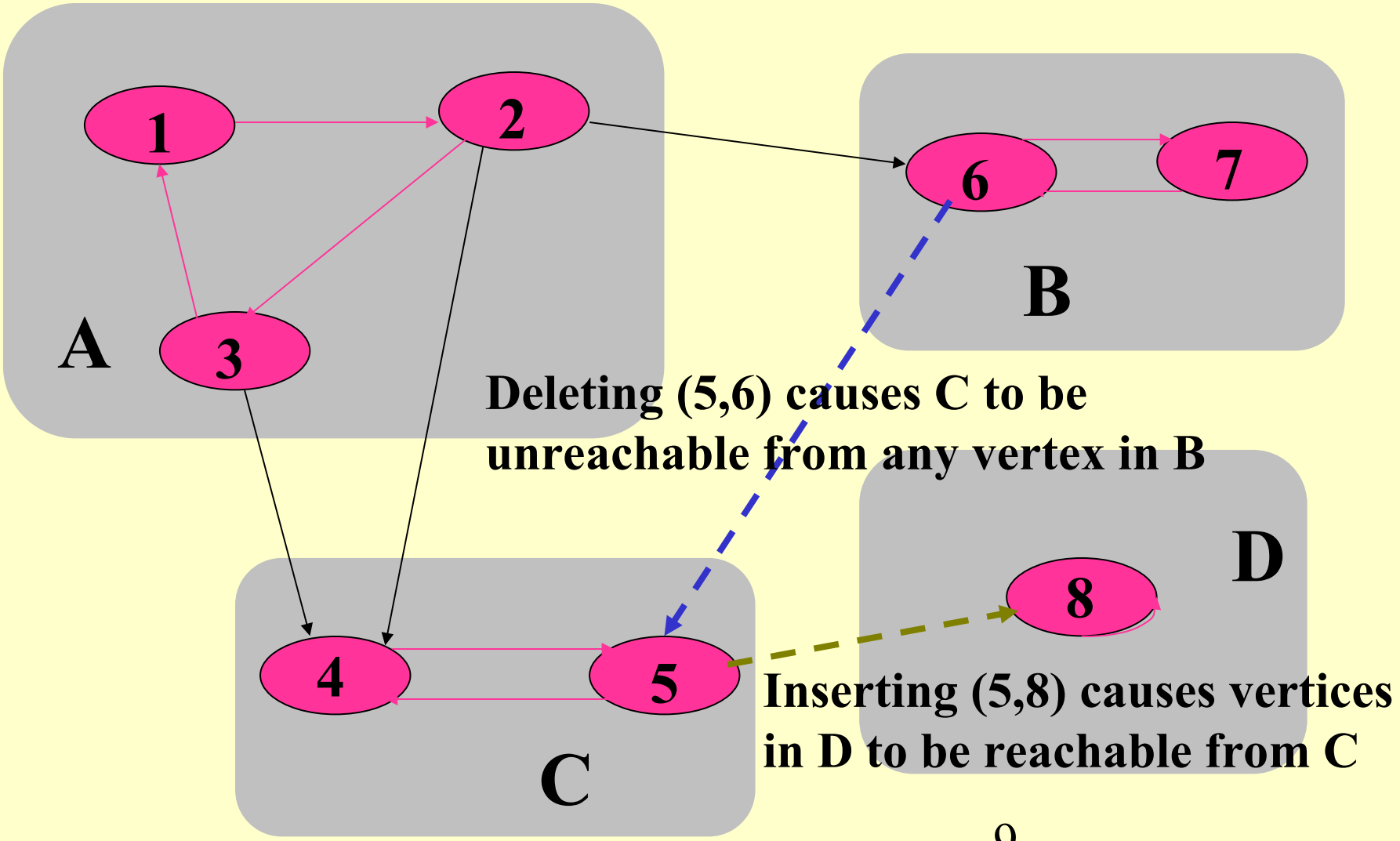
SCCs under Arc Insertions



SCCs under Arc Deletions

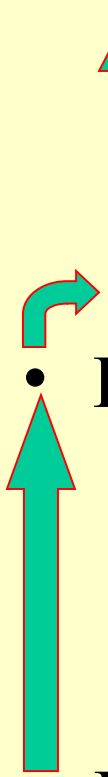


Reachability under Arc Insertions and Deletions



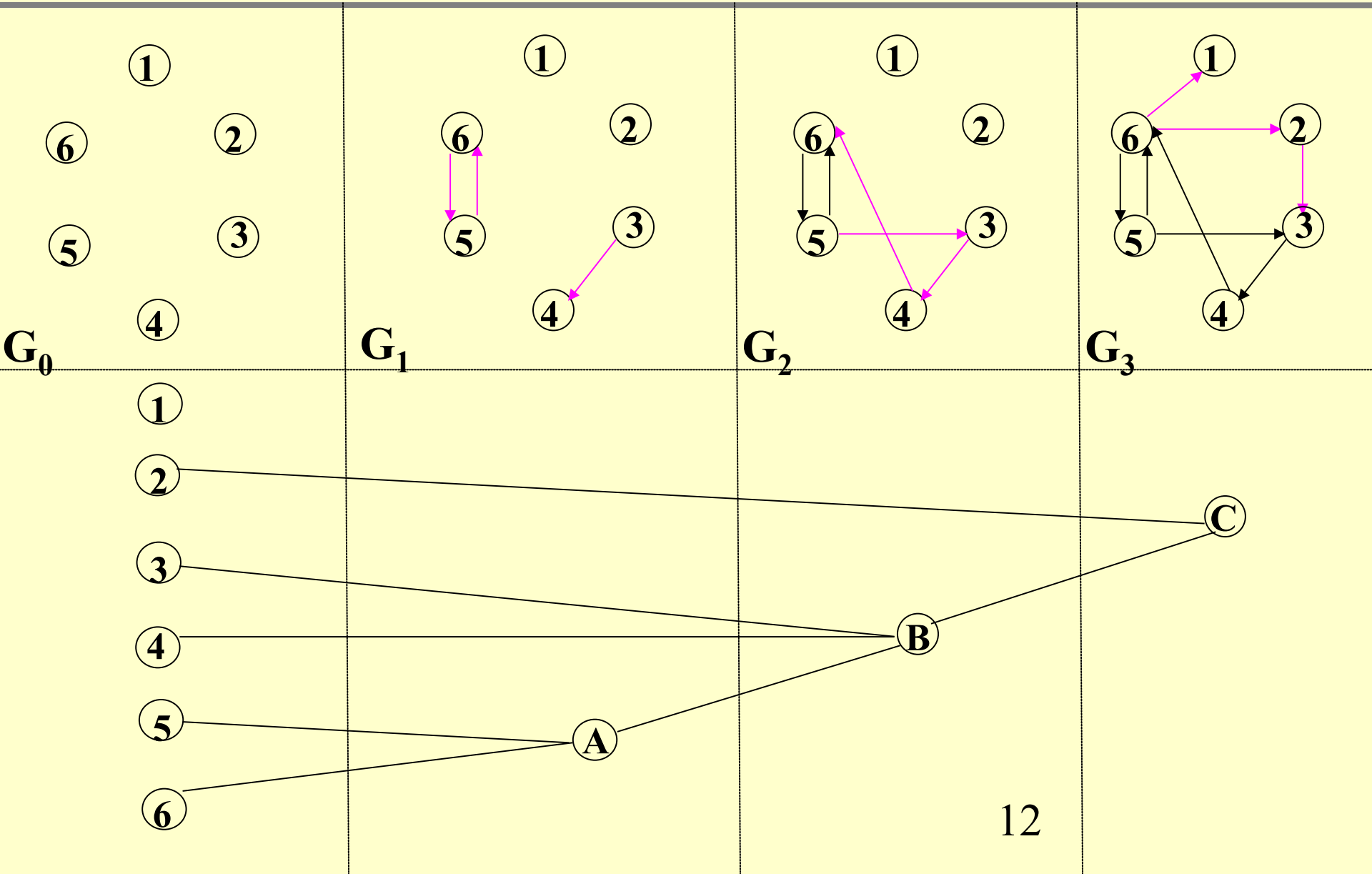
Overview of the Algorithm

- **Fully Dynamic Strong Connectivity**
  - **Maintaining SCCs under arc insertions and deletions using Component Forest (side effect : persistency)**
  - **SCC splits/merges from Component Forest**
- **Decremental Maintenance of Reachability Trees**
  - **SCC splitting and updating the inter-component Arcs**
  - **Reconnecting the possibly unconnected tree**
- **Fully Dynamic Reachability Algorithm**

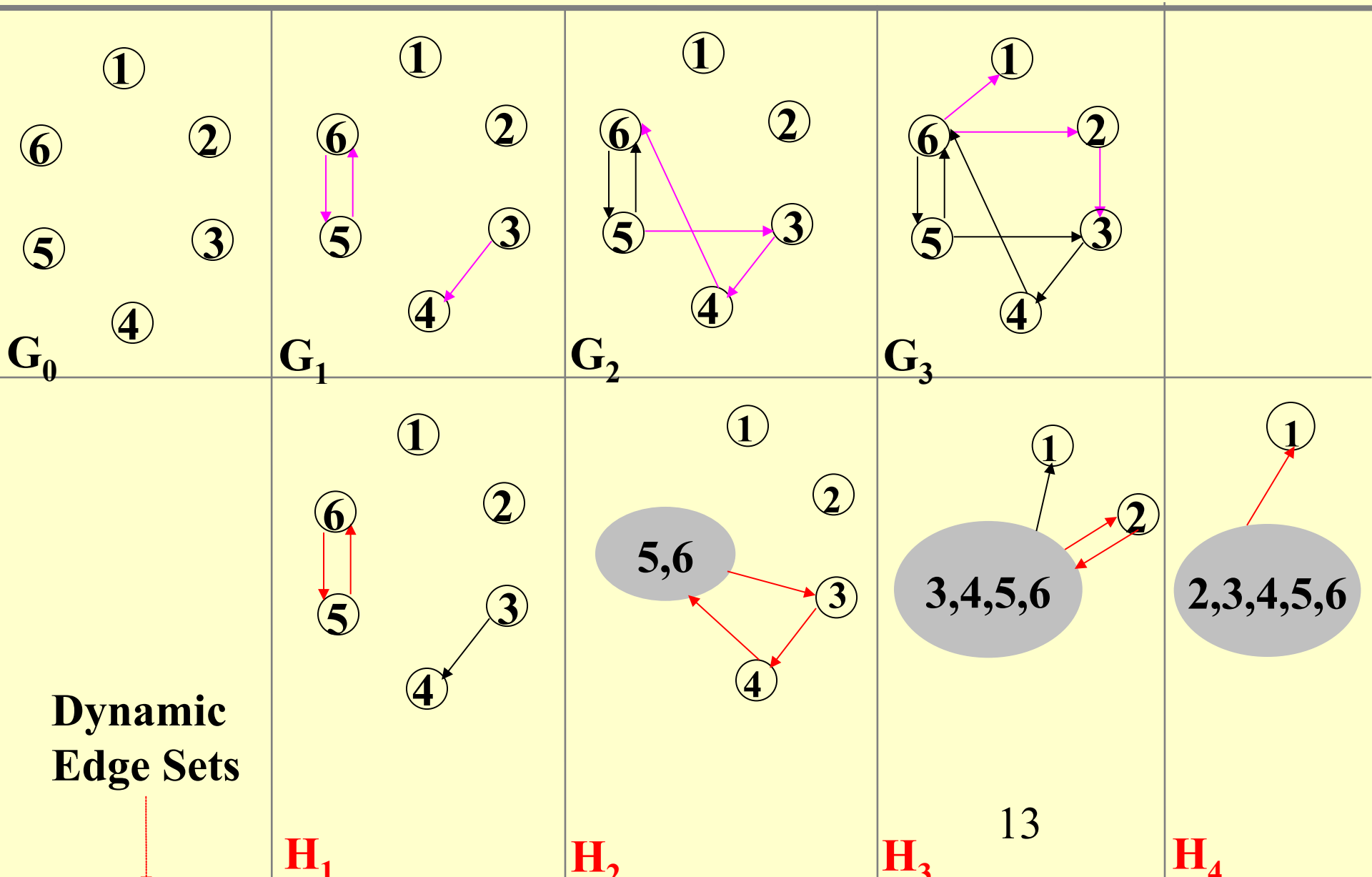


Fully Dynamic Strong Connectivity

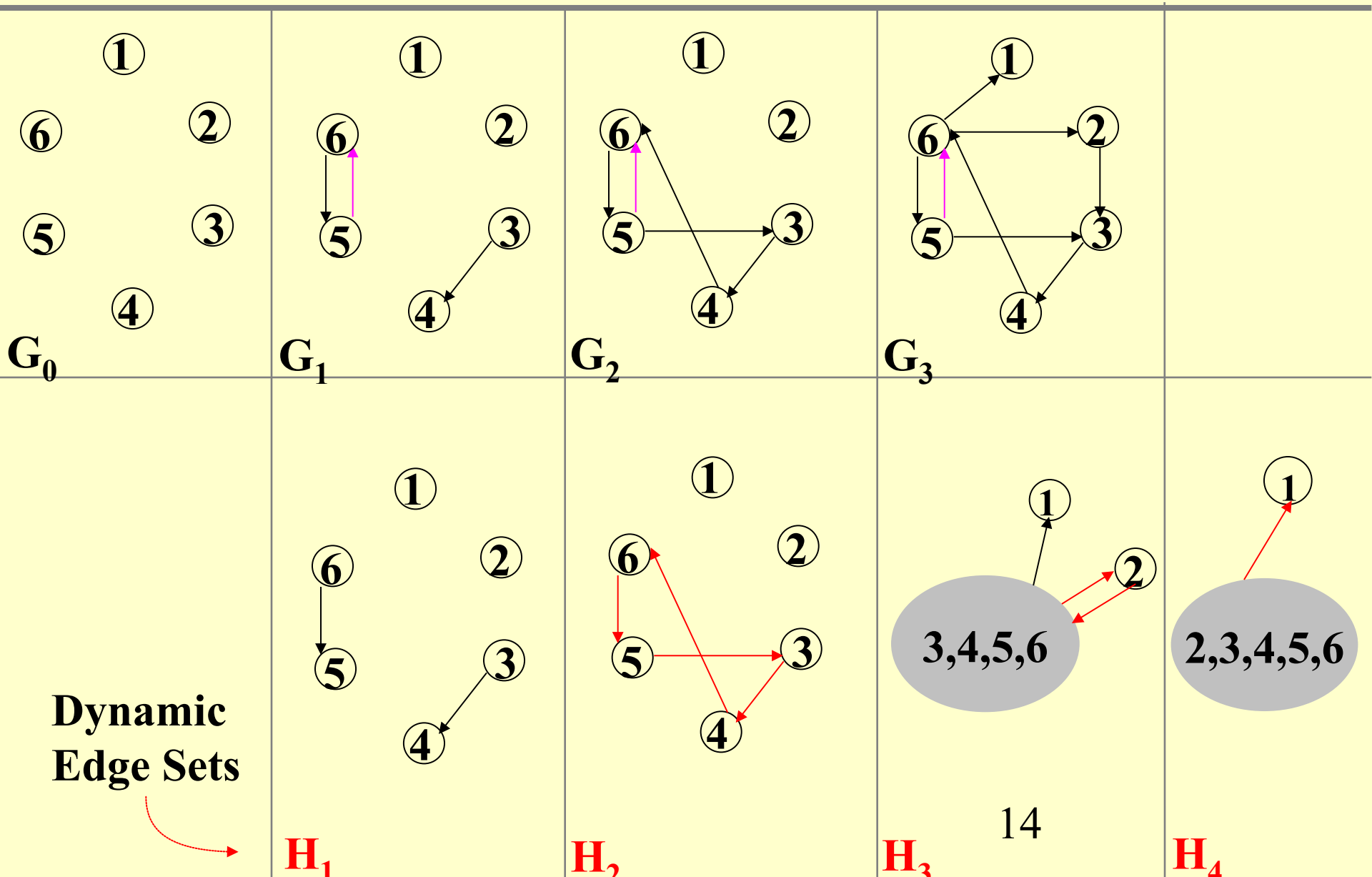
Component Forest



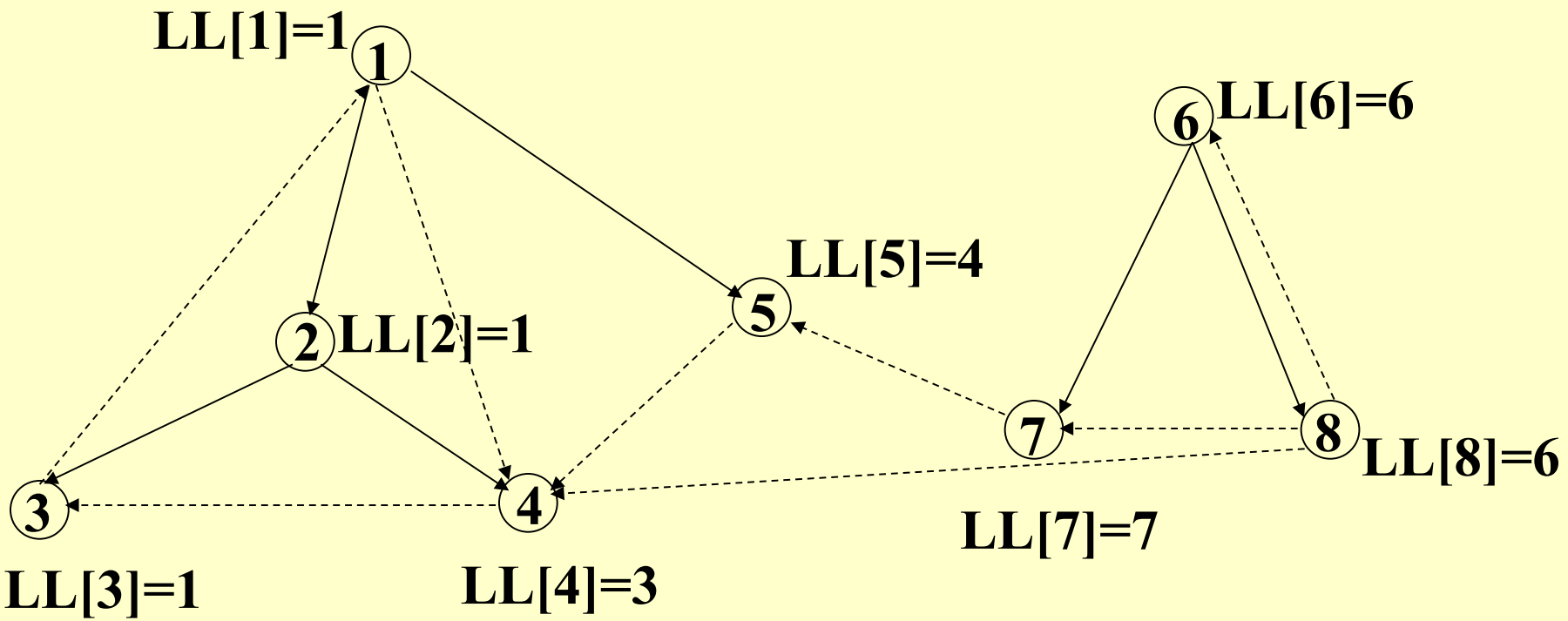
Maintaining SCCs under Insertions



Maintaining SCCs under Deletions



Finding SCCs in the given Graph



LL[v] – lowest dfs numbered vertex in the same SCC as 'v' reachable via a sequence of 0 or more tree Arcs and 0 or 1 cross or back Arcs

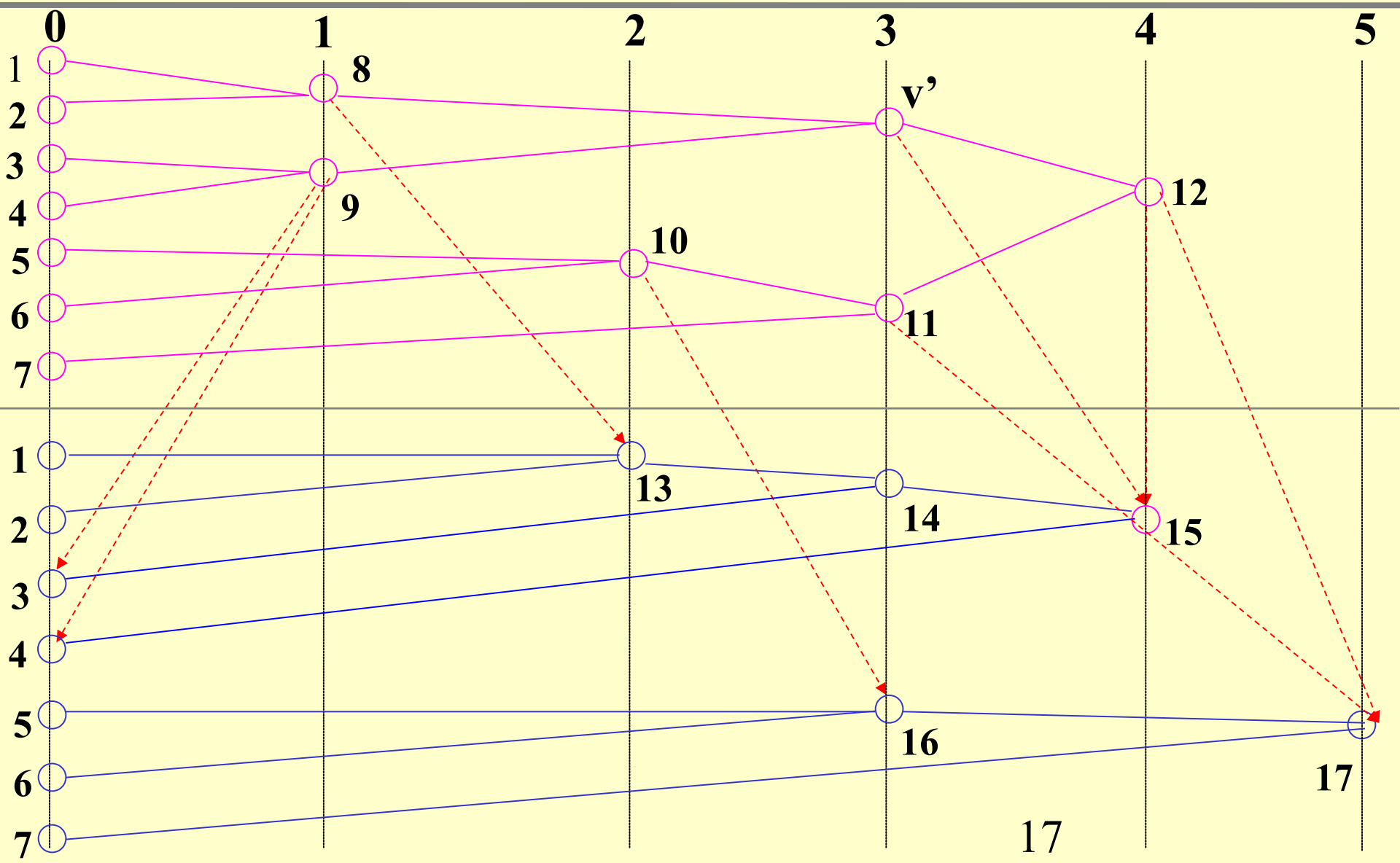
courtesy : The design and analysis of computer algorithms by Aho, Hopcroft, Ullman

Analysis of Fully Dynamic SCC Maintenance

(m : # of Arcs n : # of vertices)

- **O(mα(m, n)) worst-case insert operation**
- **O(mα(m, n)) amortized deletion operation**
- **O(1) to query whether two vertices belong to the same SCC in a given version of the graph (using LCA algorithm)**
- **O(|SCC|) to list the vertices of a SCC**
- **O(m+n) space**

Decompositions of SCCs

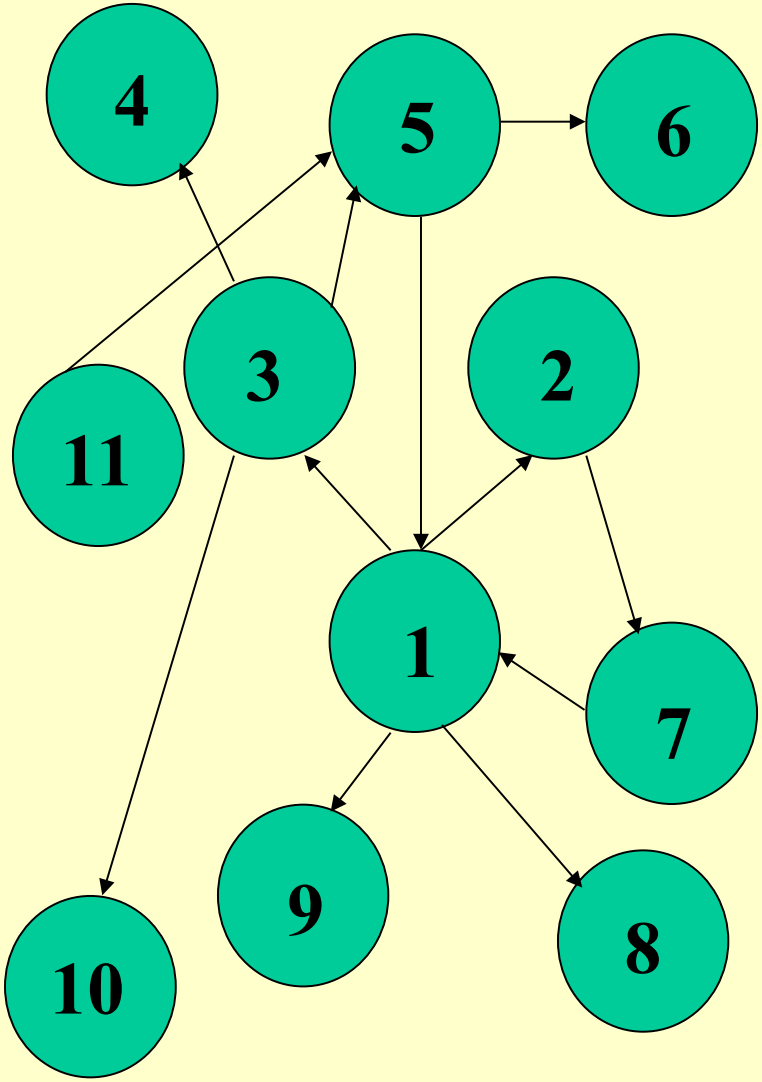


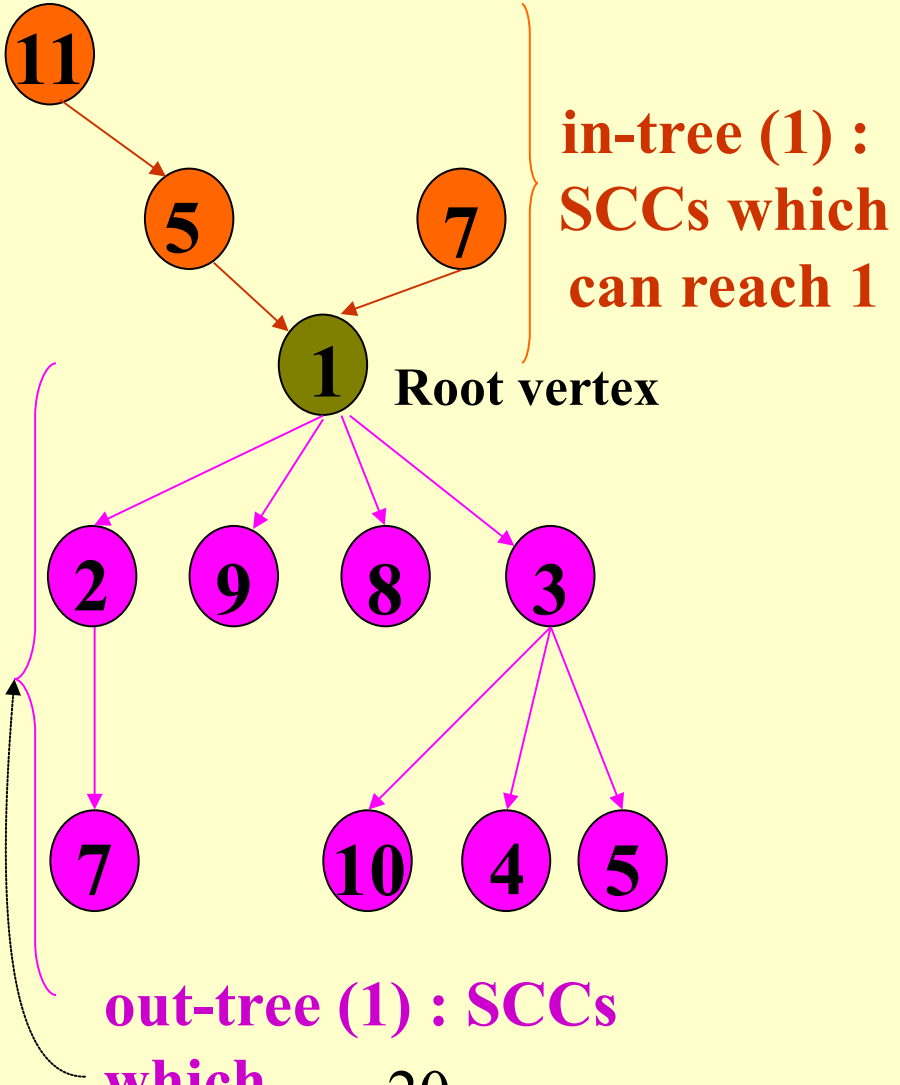
Analysis of Component Forest

- **O(n) amortized time to maintain the pointers from the old to new forest**
- **O(1) worst-case time to find the SCCs to which one SCC from the previous version got split**

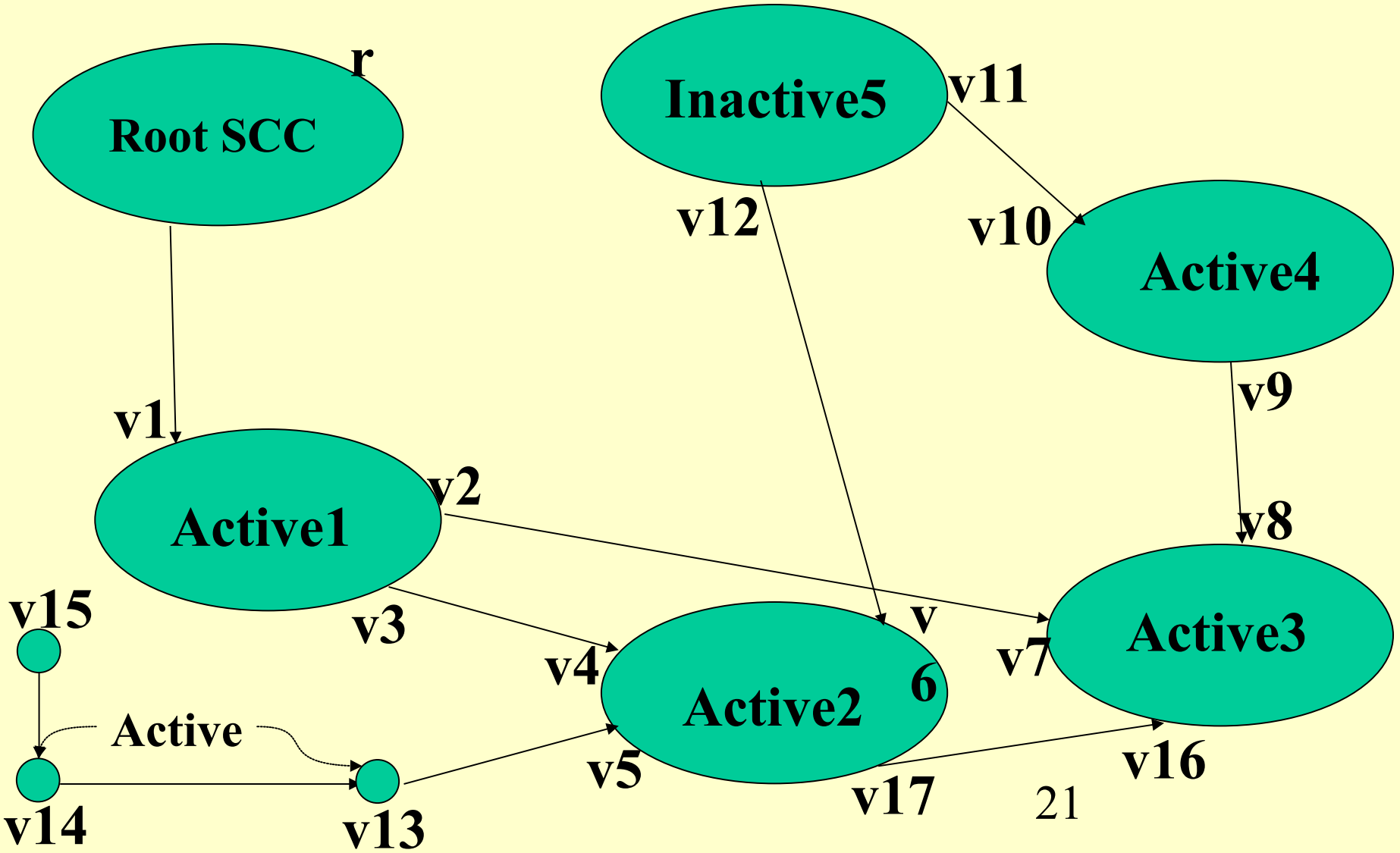
Decremental Maintenance of Reachability Trees

Reachability Tree from a Vertex

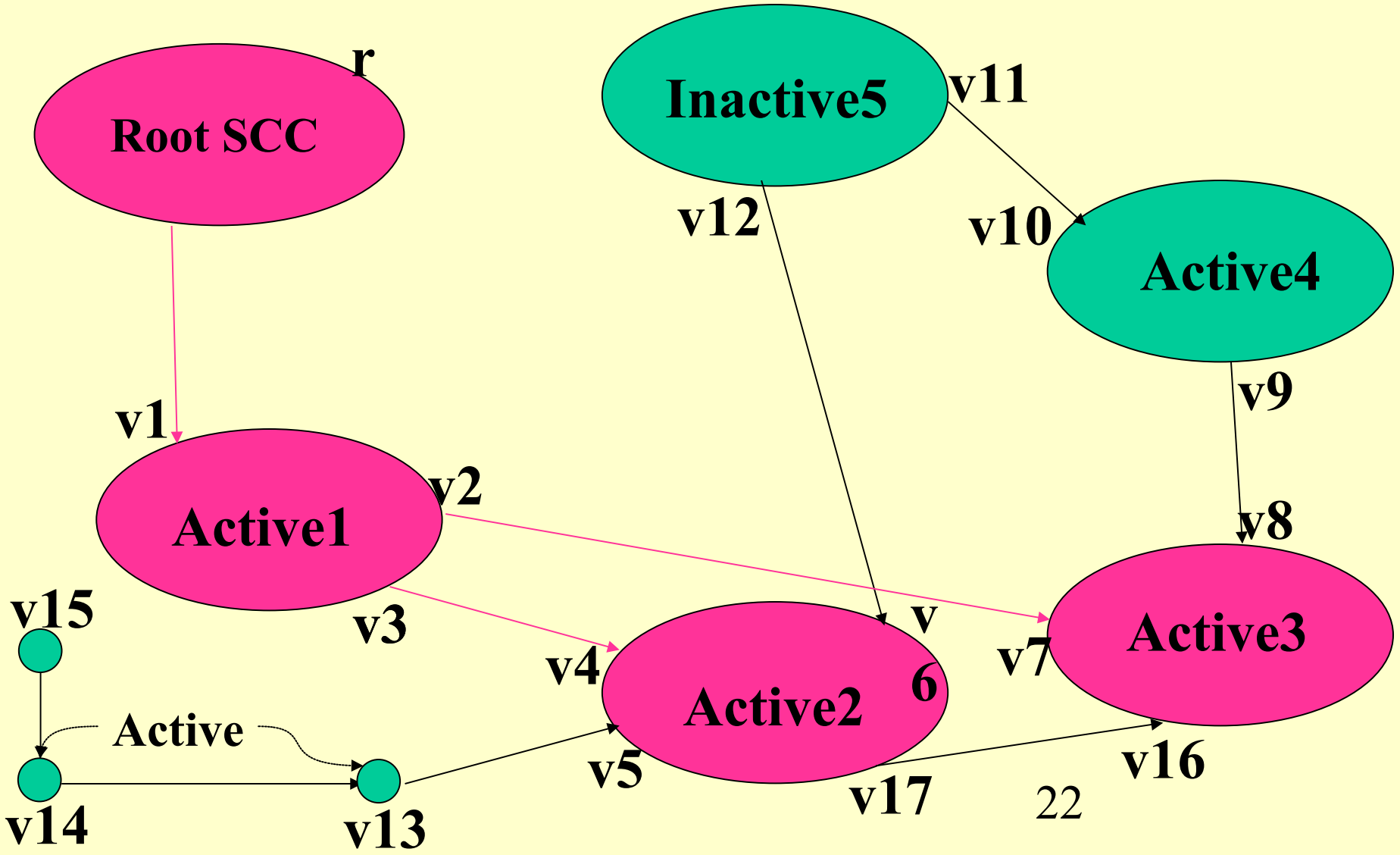




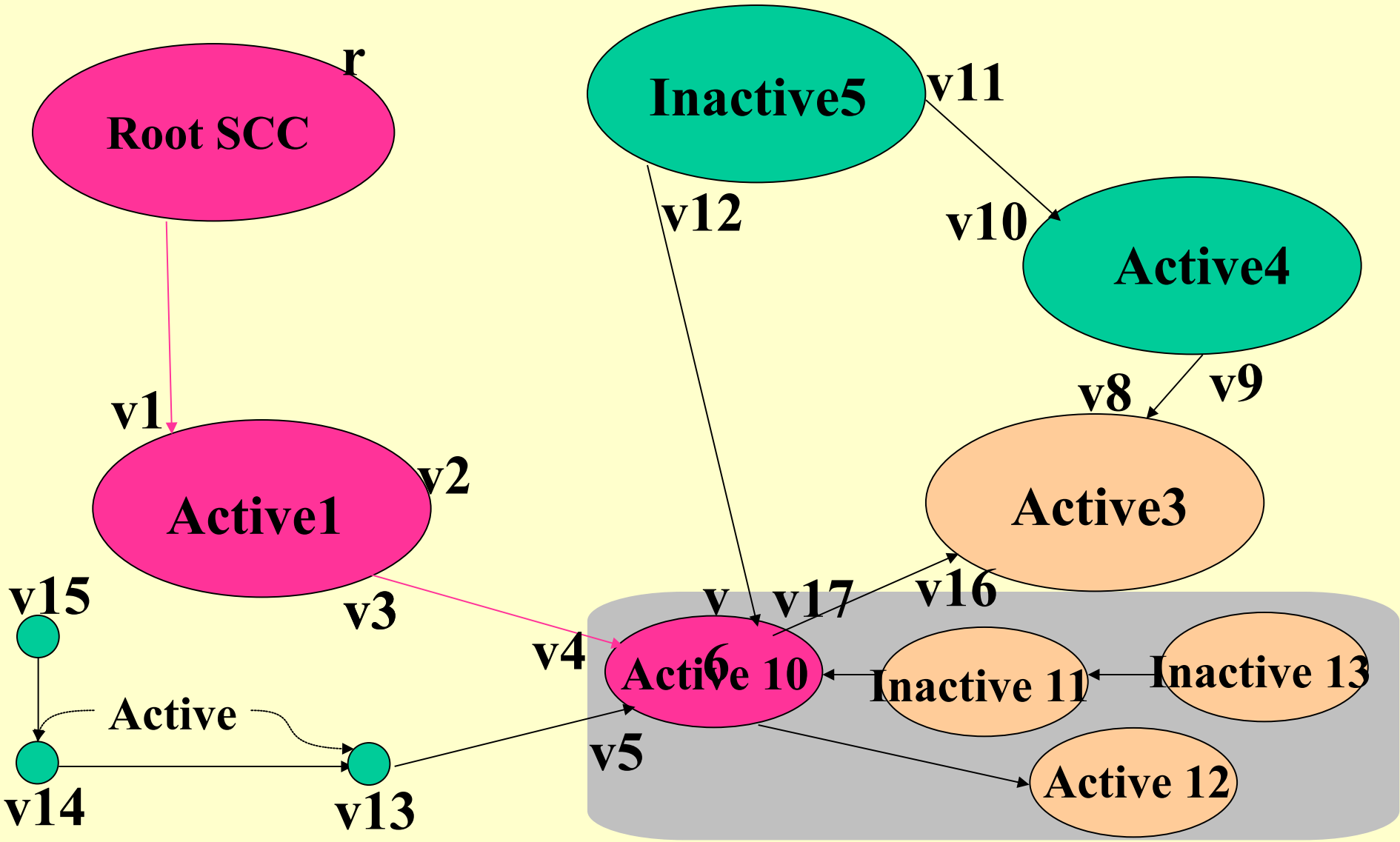
Determining Reachability Out Tree



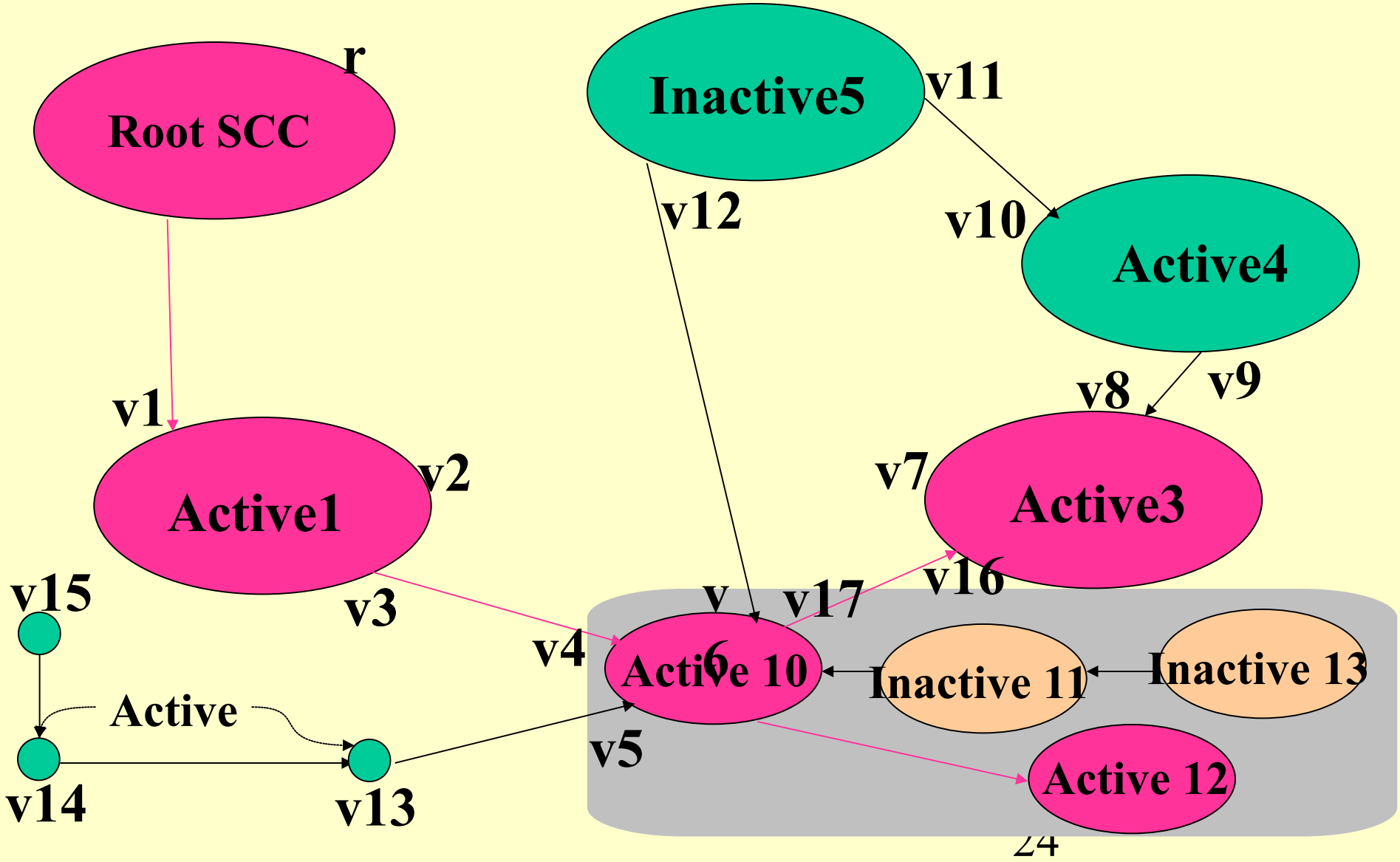
Resulting Reachability Out Tree



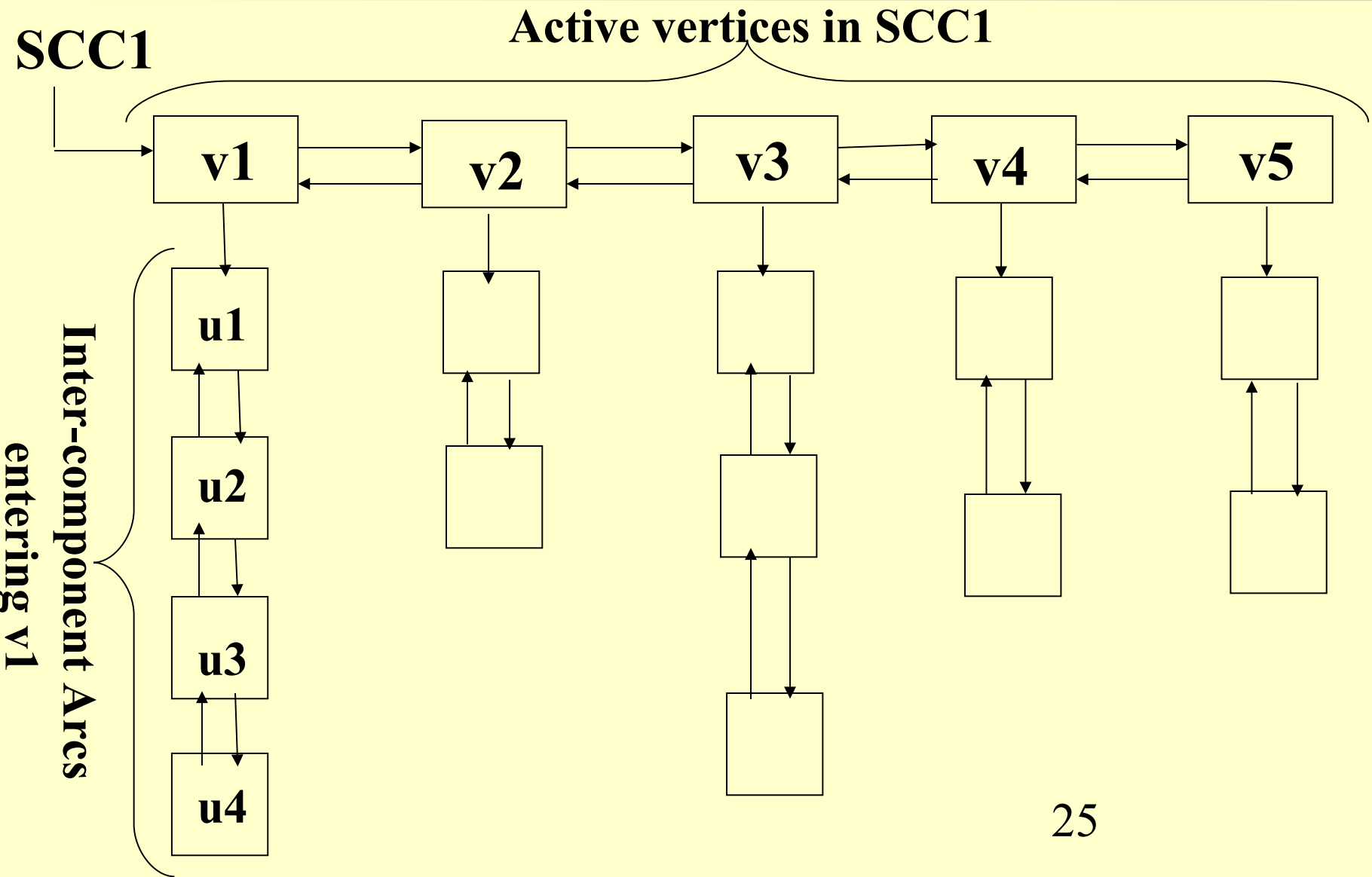
SCC decomposition after Arc Deletions



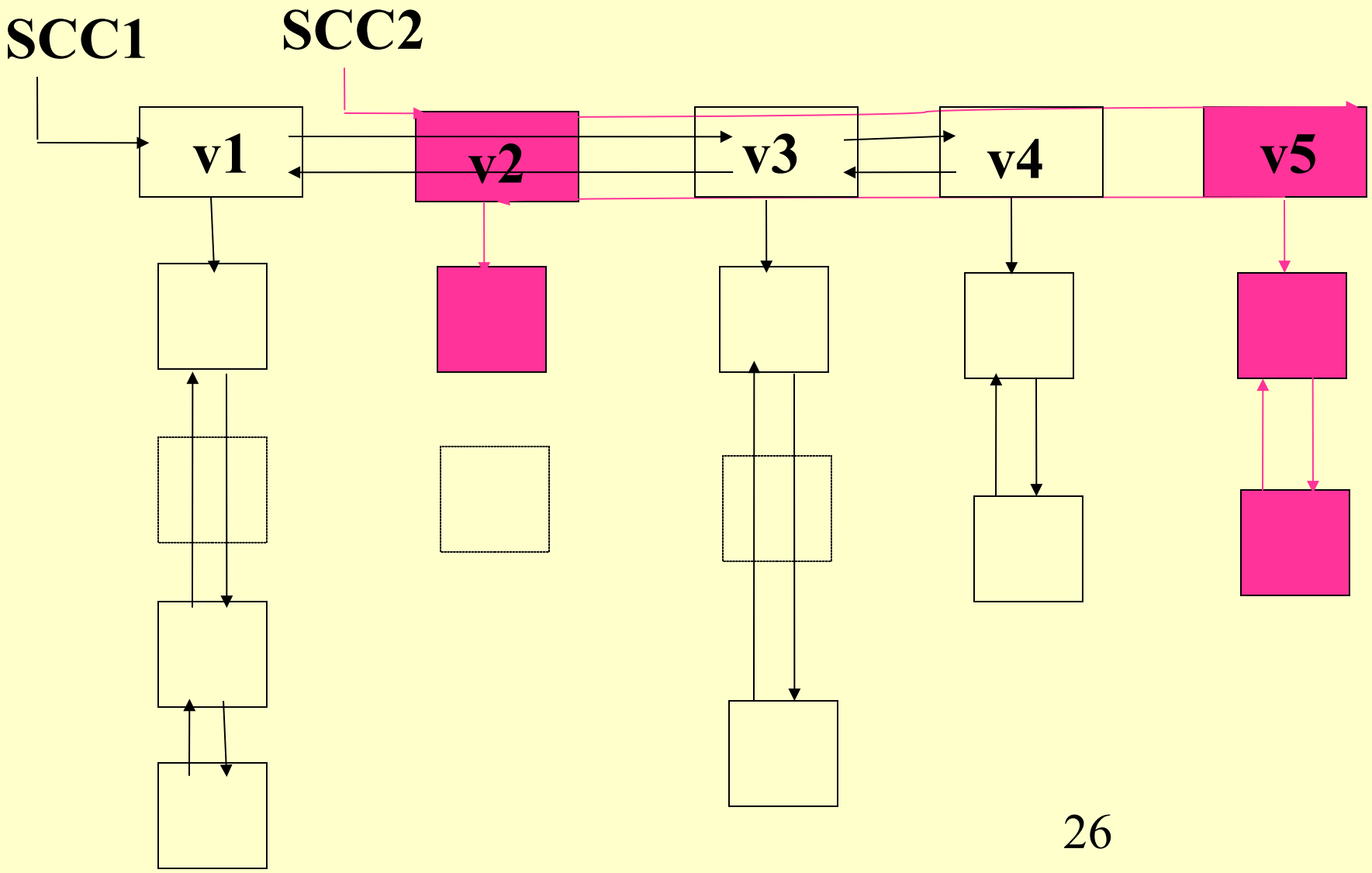
Reconnecting the disconnected Out Tree



SCC in Decremental Reachability Algorithm



SCC Split in the Decremental Maintenance

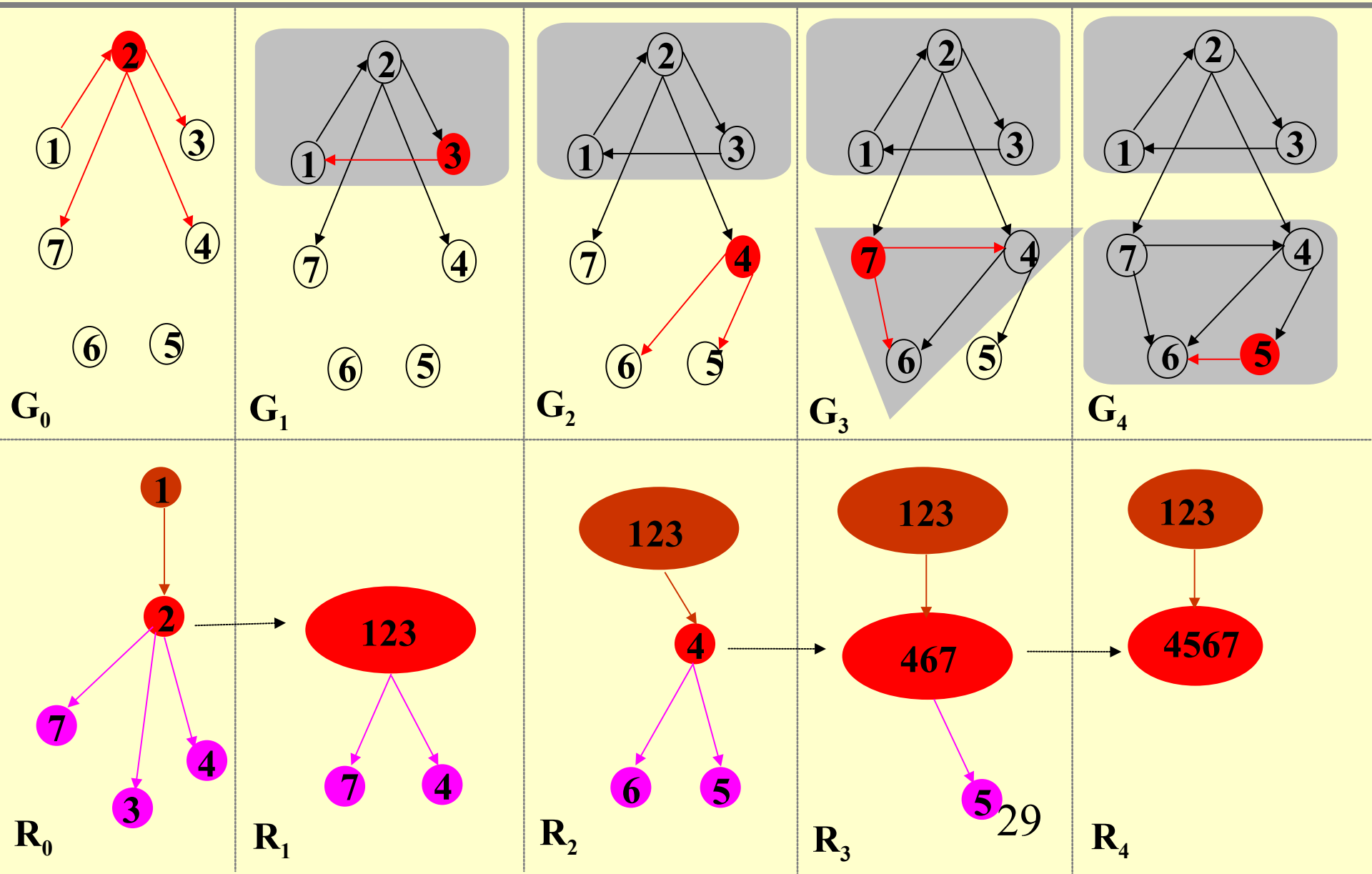


Analysis of Decremental Maintenance of Reachability Tree

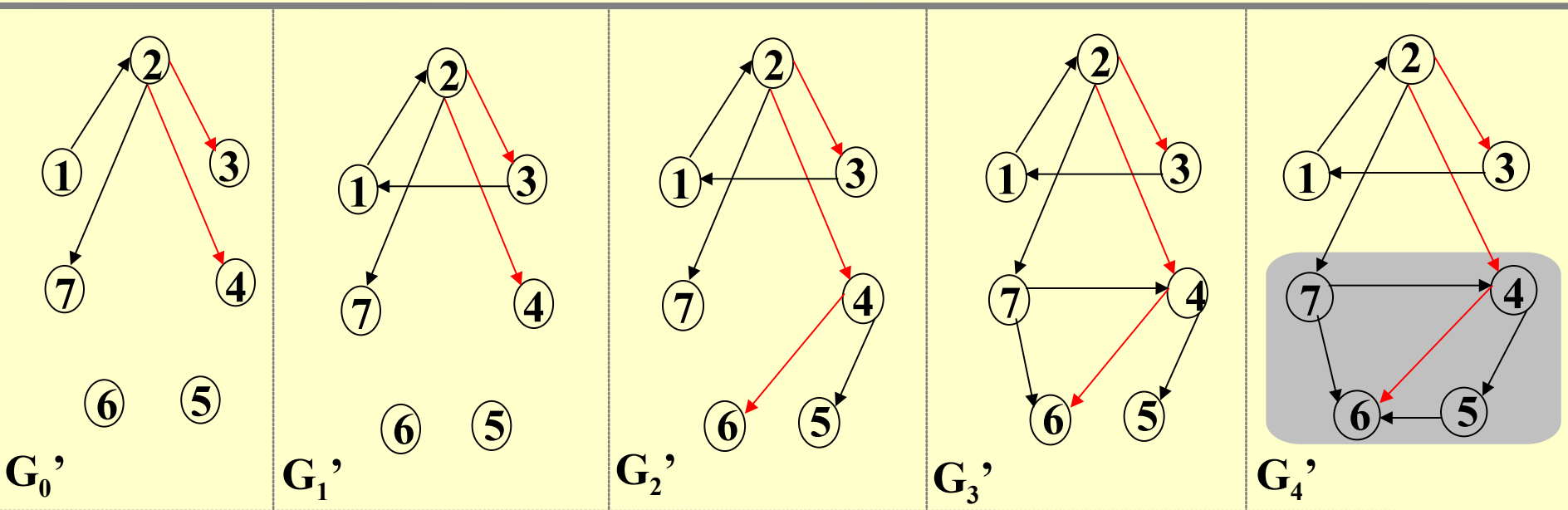
- **O (m + n log n) in the worst-case**
  - **O (m + n log n) in worst-case to maintain the SCC data structures under edge deletions**
  - **O(m) in worst-case to reconnect the possibly disconnected tree by inspecting every edge only once in each direction**
- **O(1) worst-case time to check whether the root vertex is a witness to reach vertex v from vertex u**

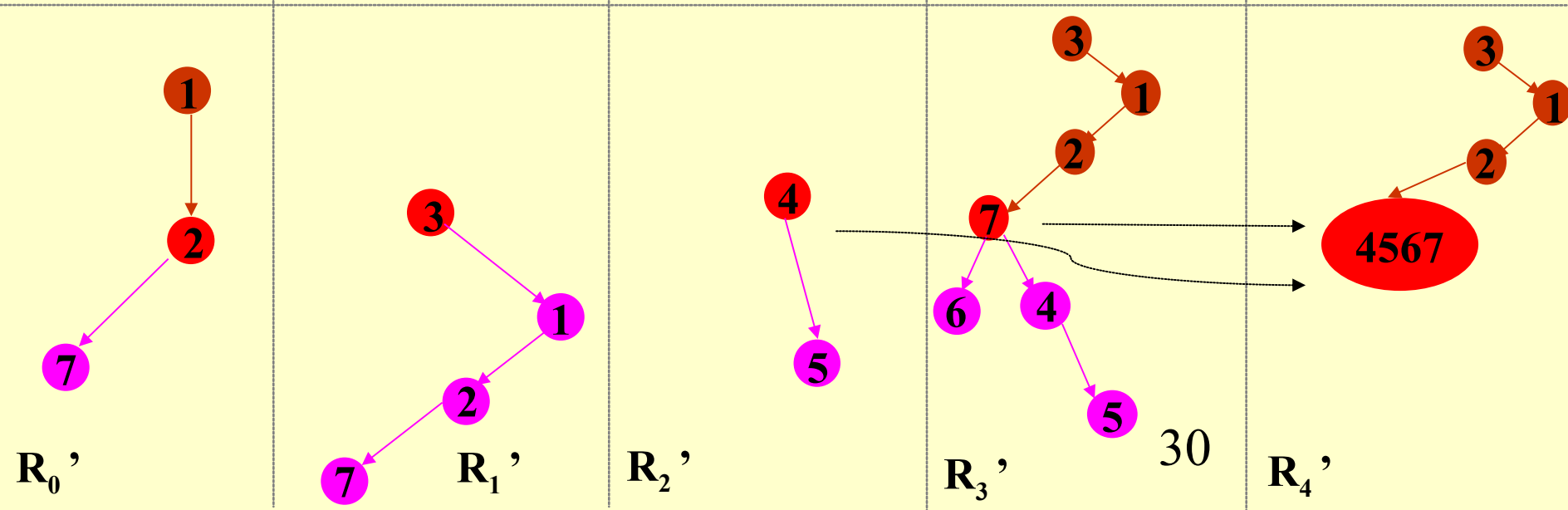
Fully Dynamic Reachability Algorithm

In and Out Trees under Insertions



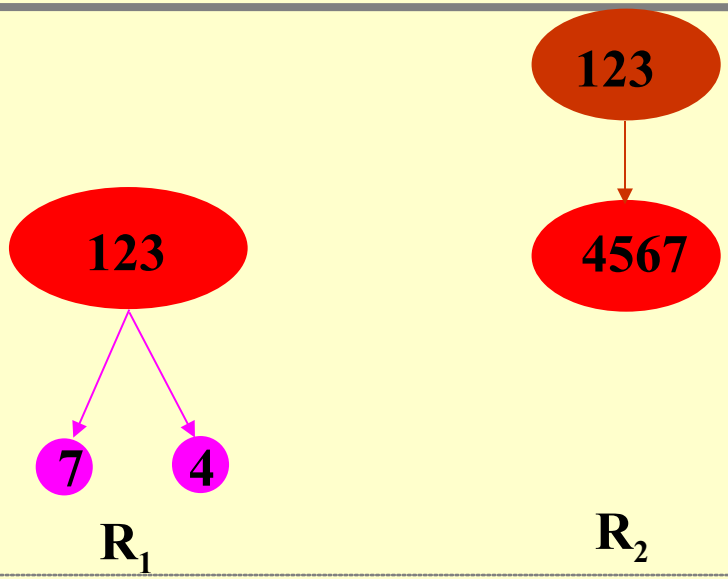
In and Out Trees under Deletions



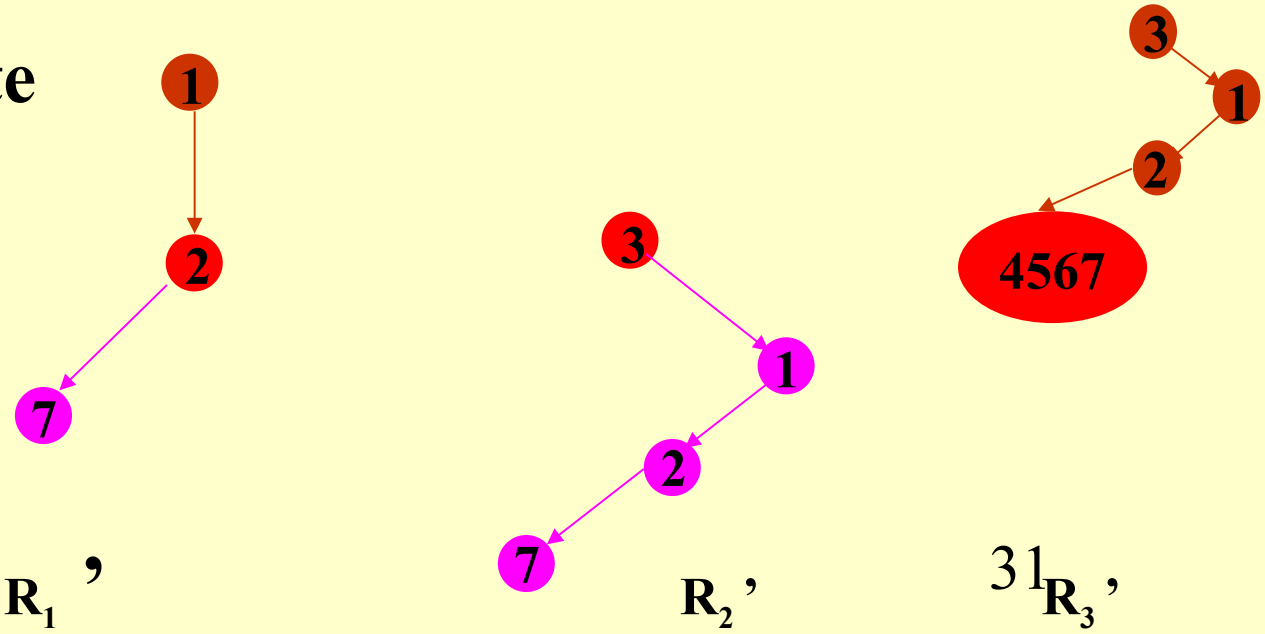


Reachability Trees after recent Insert

Before Delete



After Delete



Analysis of Fully Dynamic Reachability Algorithm

- **O(m + n log n) amortized update time**
- **O(n) worst-case reachability query time**

Comparison with the other Deterministic Fully Dynamic Reachability Algorithms

Authors	Amortized Update Time	Reachability Query Time
[C.Demetrescu, D.Leiserson] [L.Roditty]	$O(n^2)$	$O(1)$
[L.Roditty, U.Zwick]	$O(m\sqrt{n})$	$O(\sqrt{n})$
This Paper	**$O(m + n \log n)$**	**$O(n)$**

Conclusion

- **Concluding Remarks from the paper**
  - **How to reduce the update time to O(m)**
- **Possible Improvements**
  - **After finding SCCs under edge insertions/deletions, how about maintaining transitive closure matrices on those SCCs**
  - **Merge the information from all the reachability trees by constructing a reachability matrix to get better query time (useful when the number of queries after each update operation is relatively large)**
  - **Rather than maintaining at most n reachability trees, is there any way to be selective in choosing the root vertices so that the total number of reachability trees are reduced without disturbing the correctness of the algorithm**
  - **Is this applicable in determining bi-connected components??**

Resources Referred

- H. Gabow. Path-based depth-first search for strong and bi-connected components. Information Processing Letters, 74(3-4):107-114, 2000.
- S. Baswana, R.Hariharan, and S.Sen. Improved decremental algorithms for transitive closure and all-pairs shortest paths. In Proceedings of the 34th Annual ACM Symposium on Theory of Computing, Montreal, Canada, pages 117-123, 2002.
- L.Roditty and U.Zwick. Improved dynamic reachability algorithms for directed graphs. In Proceedings of FOCS'02, pages 679-689, 2002.
- L.Roditty. A faster and simpler fully dynamic transitive closure. In Proceedings of SODA'03, pages 404-412, 2003.
- S.Even and Y.Shiloach. An on-line Arc-detection problem. Journal of the ACM, 28(1):1-4, 1981.
- D.Harel and R.Tarjan. Fast Algorithms for Finding Nearest Common Ancestors. SIAM Journal on Computing, 13:338-355, 1984