Graphs
Depth-First Search, Timestamp Structure

(Class 31)

From Book’s Page Number xx (Chapter 20)

DFS - Timestamp Structure

- As we traverse the graph in DFS order, we will associate two numbers with each vertex.
- When we first discover a vertex u, store a counter in $d[u]$.
- When we are finished processing a vertex, we store a counter in $f[u]$.
- These two numbers are *time stamps*.

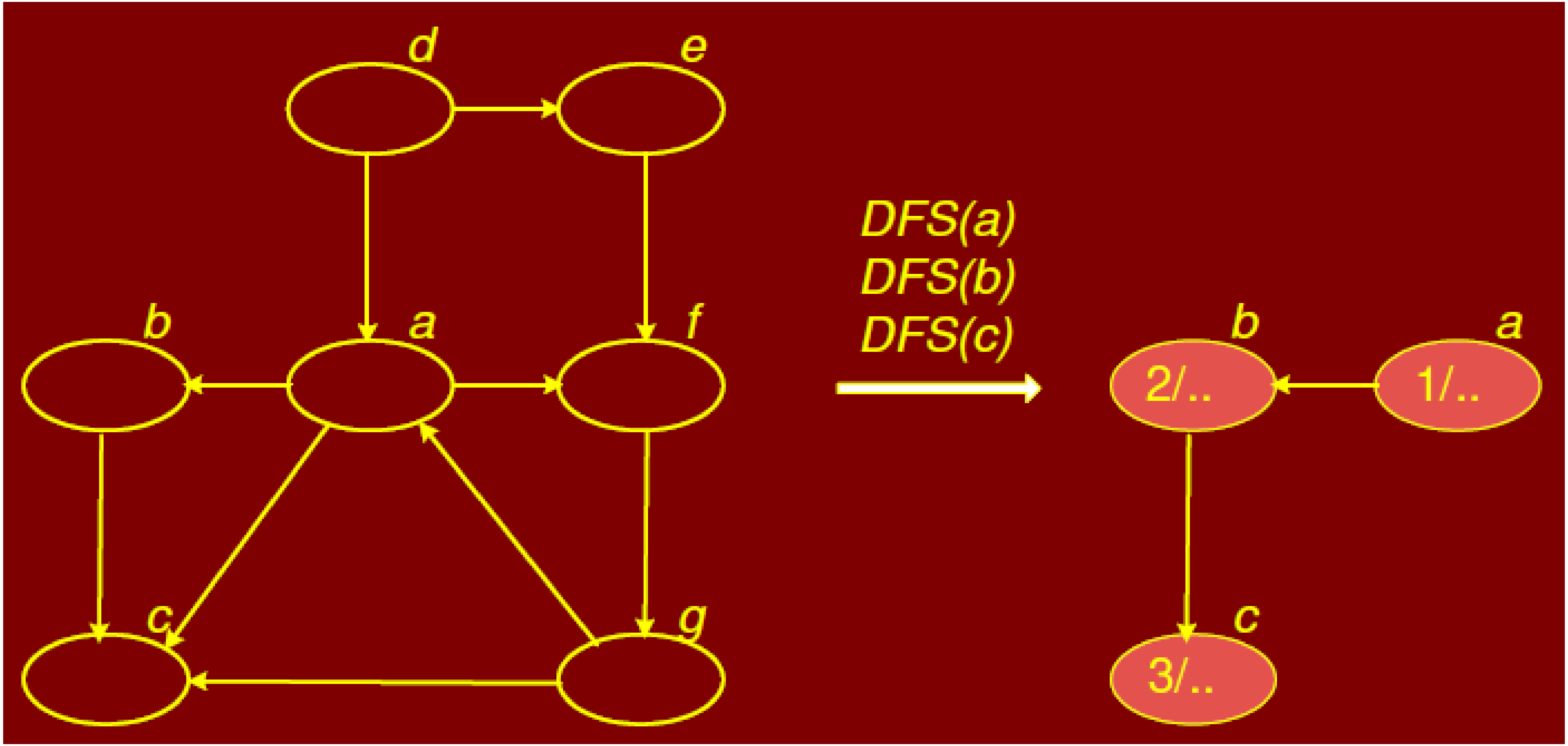
- Consider the *recursive* version of depth-first traversal:

```
DFS(G)
for (each u ∈ V)
    color[u] ← white
    pred[u] ← null
time ← 0
for (each u ∈ V)
    if (color[u] = white)
        DFSVISIT(u)
```

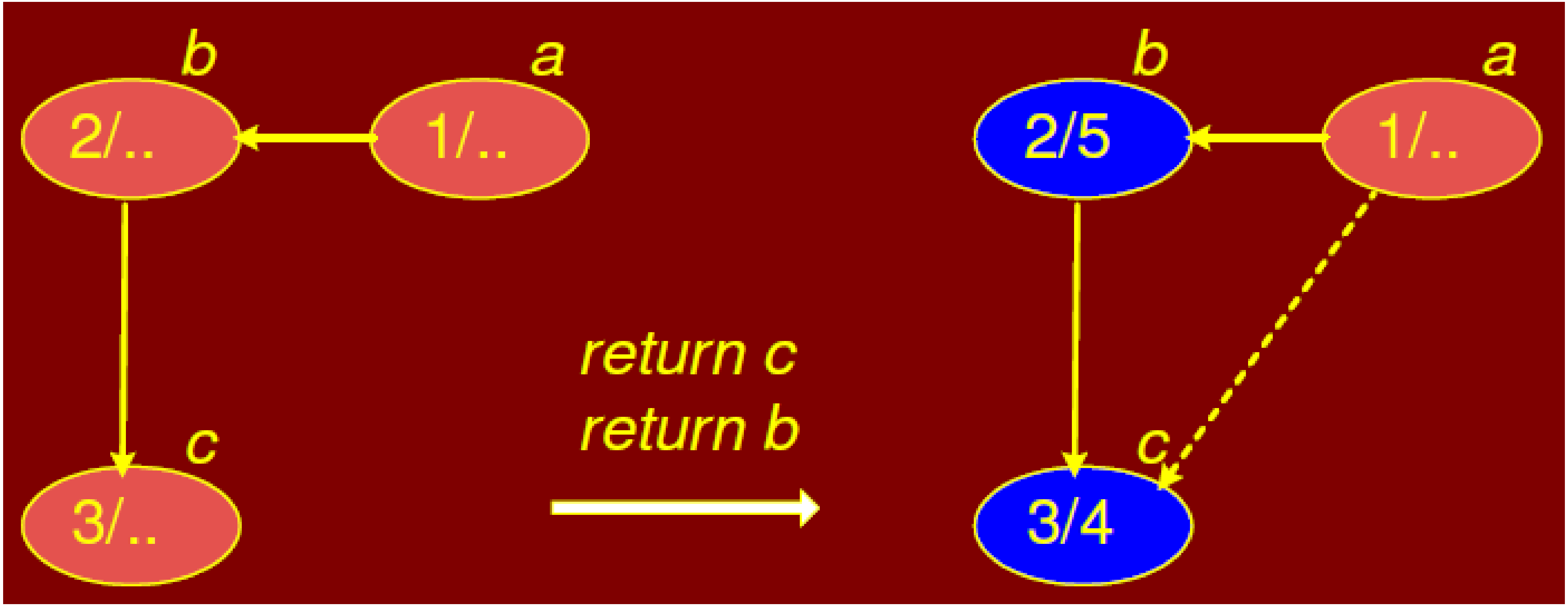
- The DFSVISIT routine is as follows:

```
DFSVISIT(u)
color[u] ← gray; // mark u visited
d[u] ← ++time
for (each u ∈ Adj[u])
    if (color[v] = white)
        pred[v] ← u
        DFSVISIT(v)
color[u] ← black; // we are done with u
f[u] ← ++time;
```

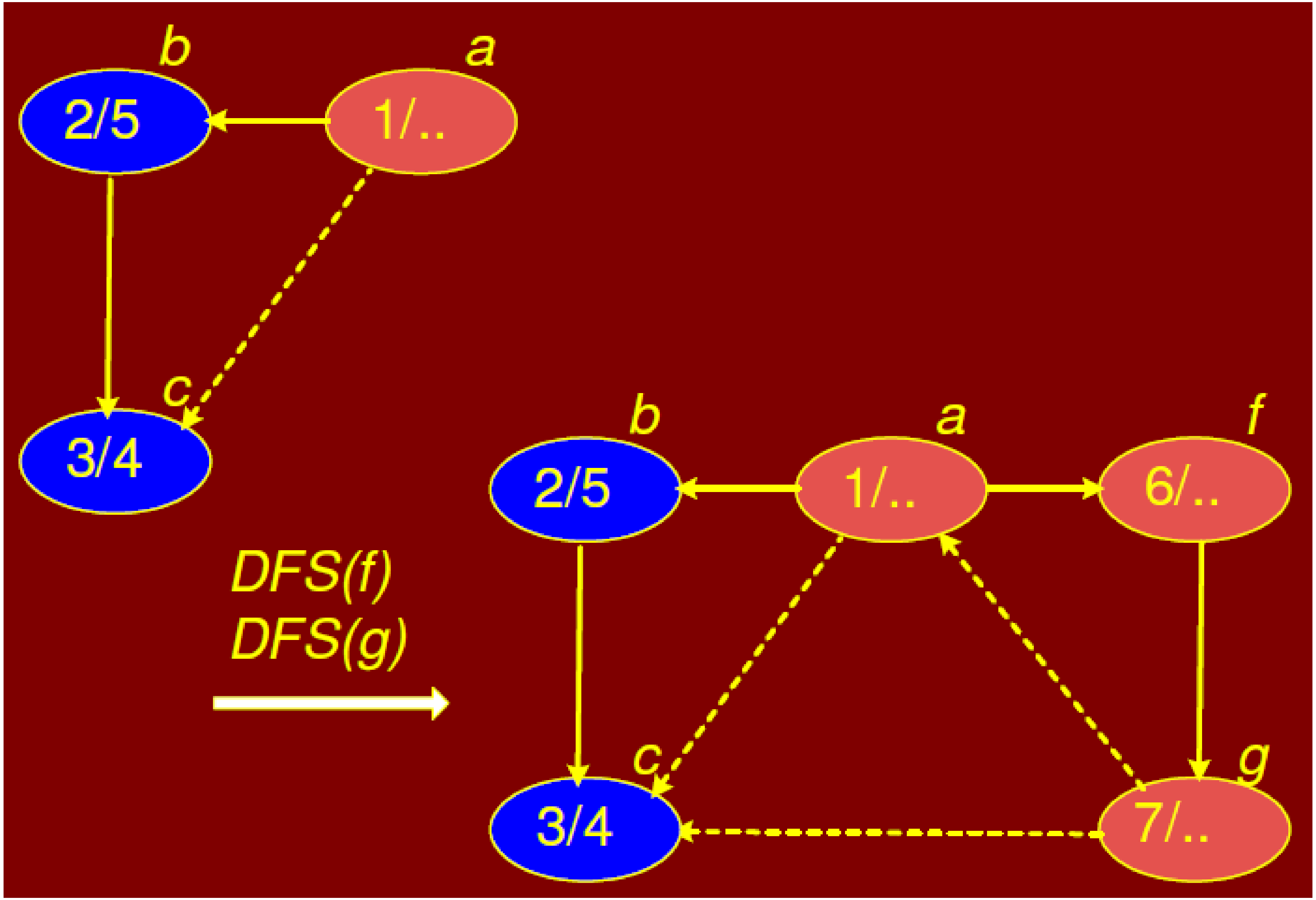
- The figures below present a trace of the execution of the time stamping algorithm.
- Terms like “2/5” indicate the value of the counter (time).
- The number before the “/” is the time when a vertex was discovered (colored gray).
- And the number after the “/” is the time when the processing of the vertex finished (colored black).



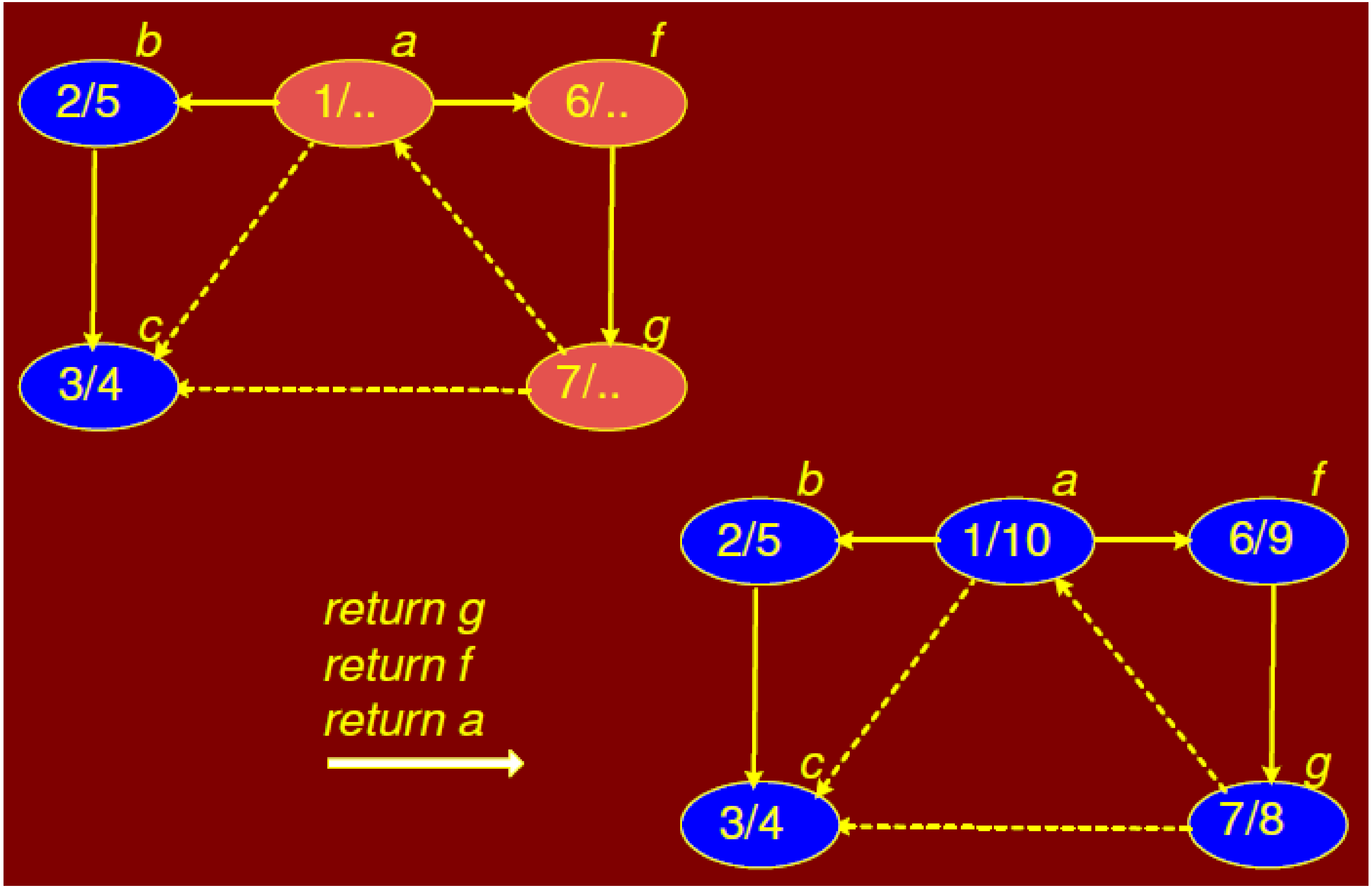
DFS with time stamps: recursive calls initiated at vertex ‘a’



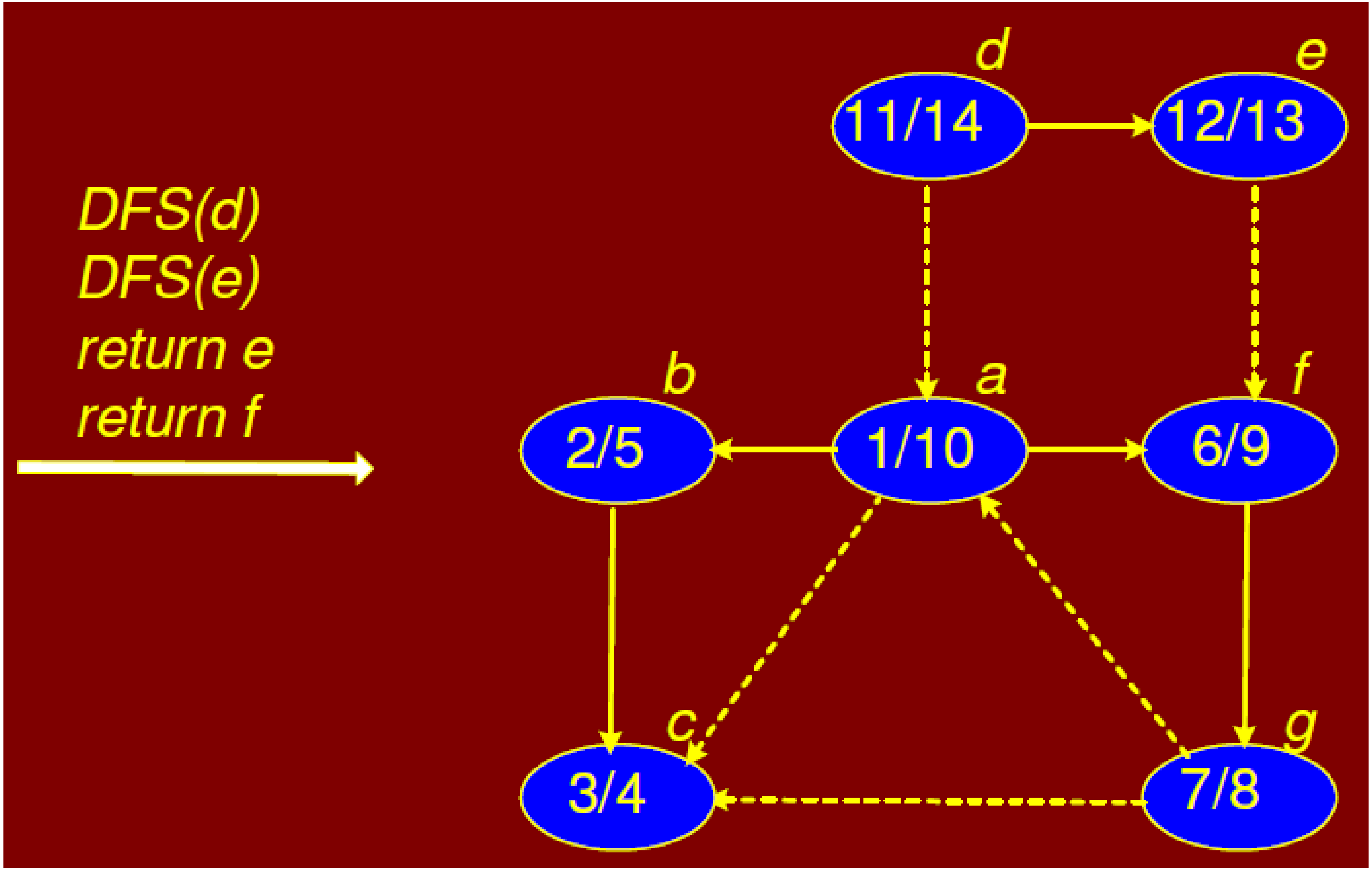
DFS with time stamps: processing of ‘b’ and ‘c’ completed



DFS with time stamps: recursive processing of ‘f’ and ‘g’



DFS with time stamps: processing of ‘f’ and ‘g’ completed



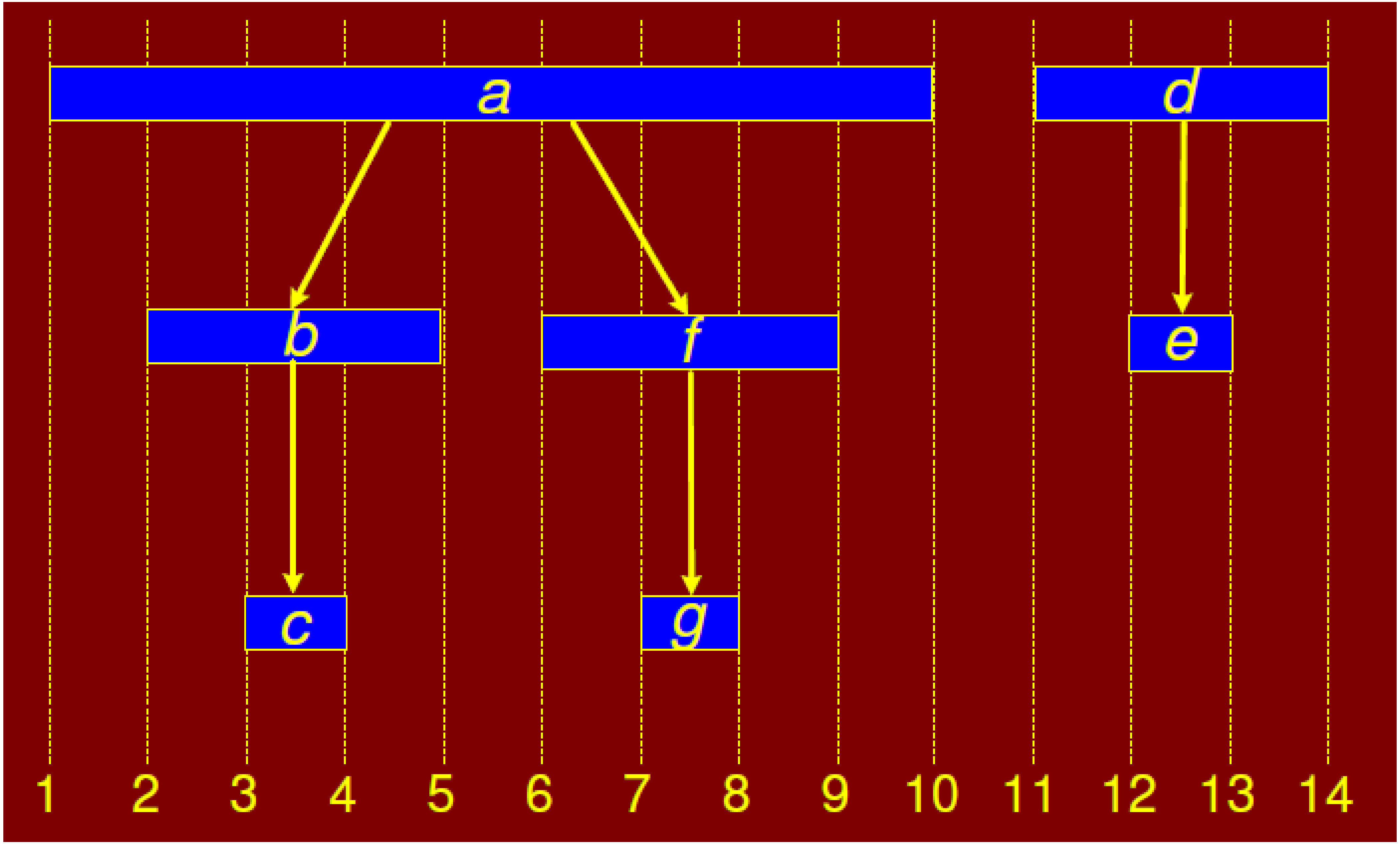
DFS with time stamps: processing of 'd' and 'e'

- Notice that the DFS tree structure (actually a collection of trees, or a forest) on the structure of the graph is just the recursion tree, where the edge (u, v) arises when processing vertex u we call DFSVISIT(V) for some neighbor v.
- For *directed graphs* the edges that are not part of the tree (indicated as dashed edges in the figures) edges of the graph can be classified as follows:

- **Back edge:** (u, v) where v is an ancestor of u in the tree.
- **Forward edge:** (u, v) where v is a proper descendent of u in the tree.
- **Cross edge:** (u, v) where u and v are not ancestor or descendent of one another. In fact, the edge may go between different trees of the forest.

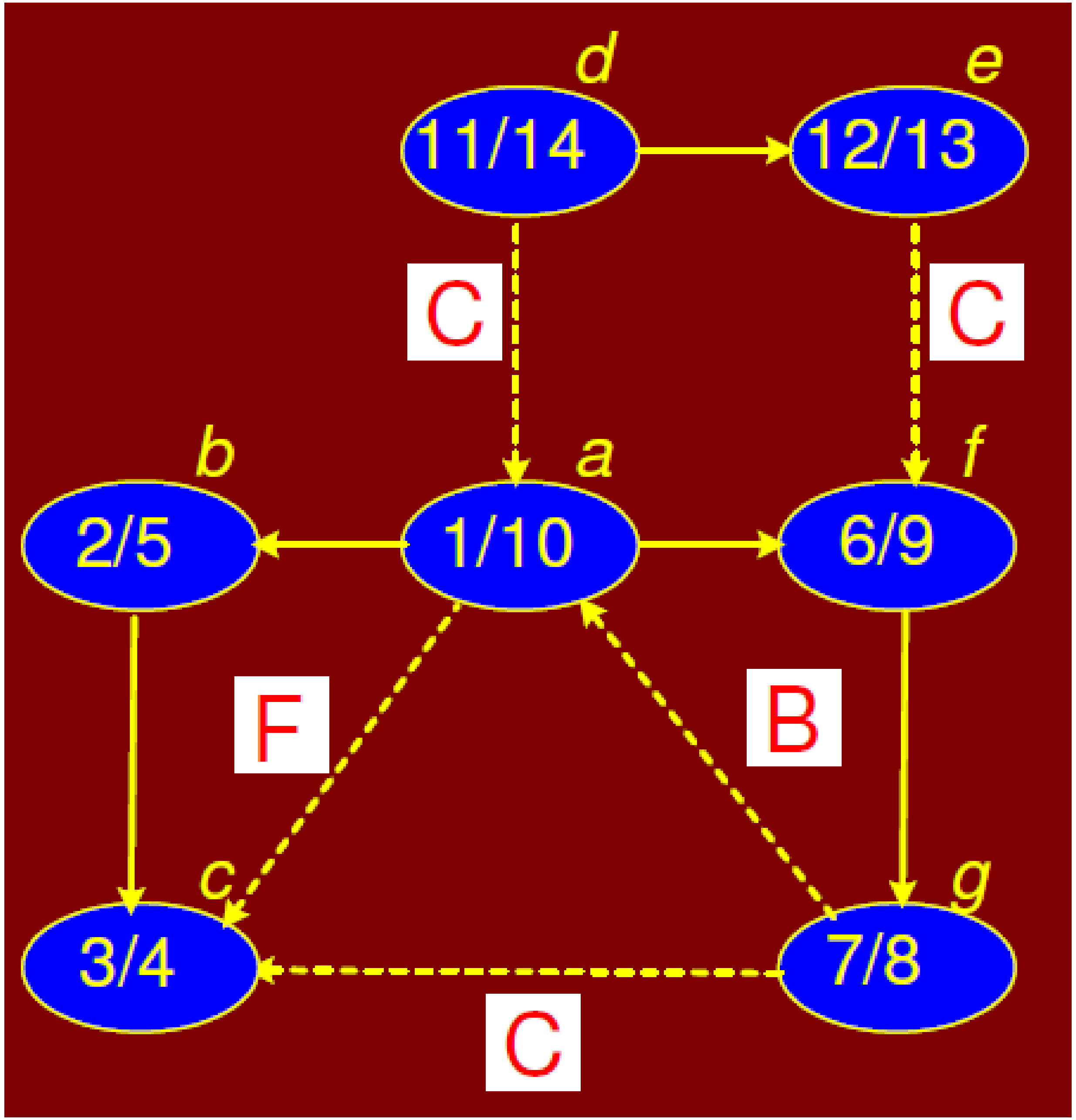
- The ancestor and descendent relation can be nicely inferred by the parenthesis lemma.
- u is a descendent of v if and only if $[d[u], f[u]] \subseteq [d[v], f[v]]$.
- u is an ancestor of v if and only if $[d[u], f[u]] \supseteq [d[v], f[v]]$.
- u is unrelated to v if and only if $[d[u], f[u]]$ and $[d[v], f[v]]$ are disjoint.
- The is shown in the figure below.

- The width of the rectangle associated with a vertex is equal to the time the vertex was discovered till the time the vertex was completely processed (colored black).
- Imagine an opening parenthesis ‘(’ at the start of the rectangle and closing parenthesis ‘)’ at the end of the rectangle.
- The rectangle (parentheses) for vertex ‘b’ is completely enclosed by the rectangle for ‘a’.
- The rectangle for ‘c’ is completely enclosed by vertex ‘b’ rectangle.



Parenthesis lemma

- The figure below shows the classification of the non-tree edges based on the parenthesis lemma.
- Edges are labelled ‘F’, ‘B’ and ‘C’ for forward, back and cross edge respectively.



Classfication of non-tree edges in the DFS tree for a graph

- For *undirected graphs*, there is no distinction between forward and back edges.
- By convention they are all called back edges.
- Furthermore, there are no cross edges.
- We can use this timestamp algorithm to detect the loops in the graphs.