



POLITECNICO  
DI TORINO



e-Lite



# Graph Visits

Tecniche di Programmazione – A.A. 2017/2018



# Summary

---

- ▶ Graph visits
- ▶ Visits in JGraphT



# Graph visits

Representing and visiting graphs

# Visit Algorithms

---

- ▶ **Visit =**
  - ▶ Systematic exploration of a graph
  - ▶ Starting from a ‘source’ vertex
  - ▶ Reaching all reachable vertices
- ▶ **Main strategies**
  - ▶ Breadth-first visit (“in ampiezza”)
  - ▶ Depth-first visit (“in profondità”)

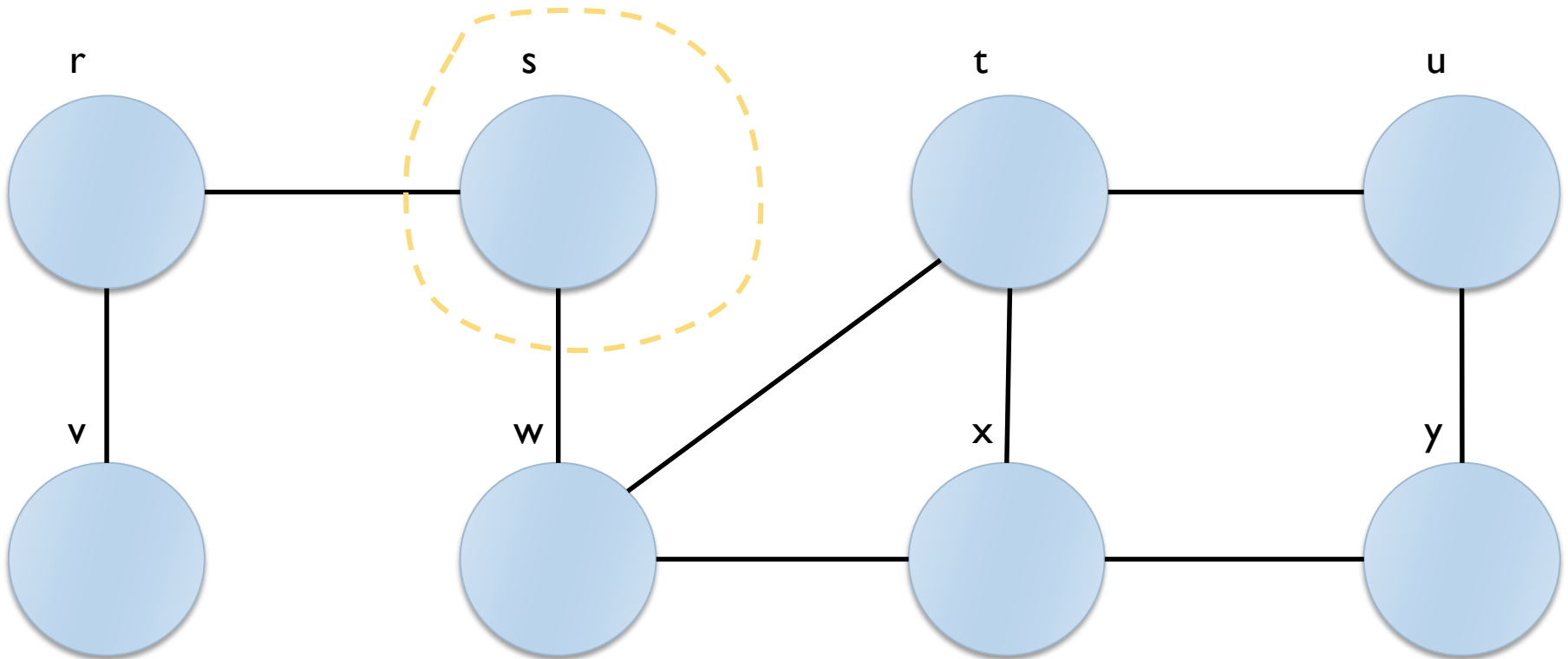
# Breadth-First Visit

---

- ▶ Also called Breadth-first search (BFV or BFS)
- ▶ All reachable vertices are visited “by levels”
  - ▶  $L$  – level of the visit
  - ▶  $S_L$  – set of vertices in level  $L$
  - ▶  $L=0, S_0 = \{ v_{\text{source}} \}$
  - ▶ Repeat while  $S_L$  is not empty:
    - ▶  $S_{L+1}$  = set of all vertices:
      - not visited yet, and
      - adjacent to at least one vertex in  $S_L$
    - ▶  $L=L+1$

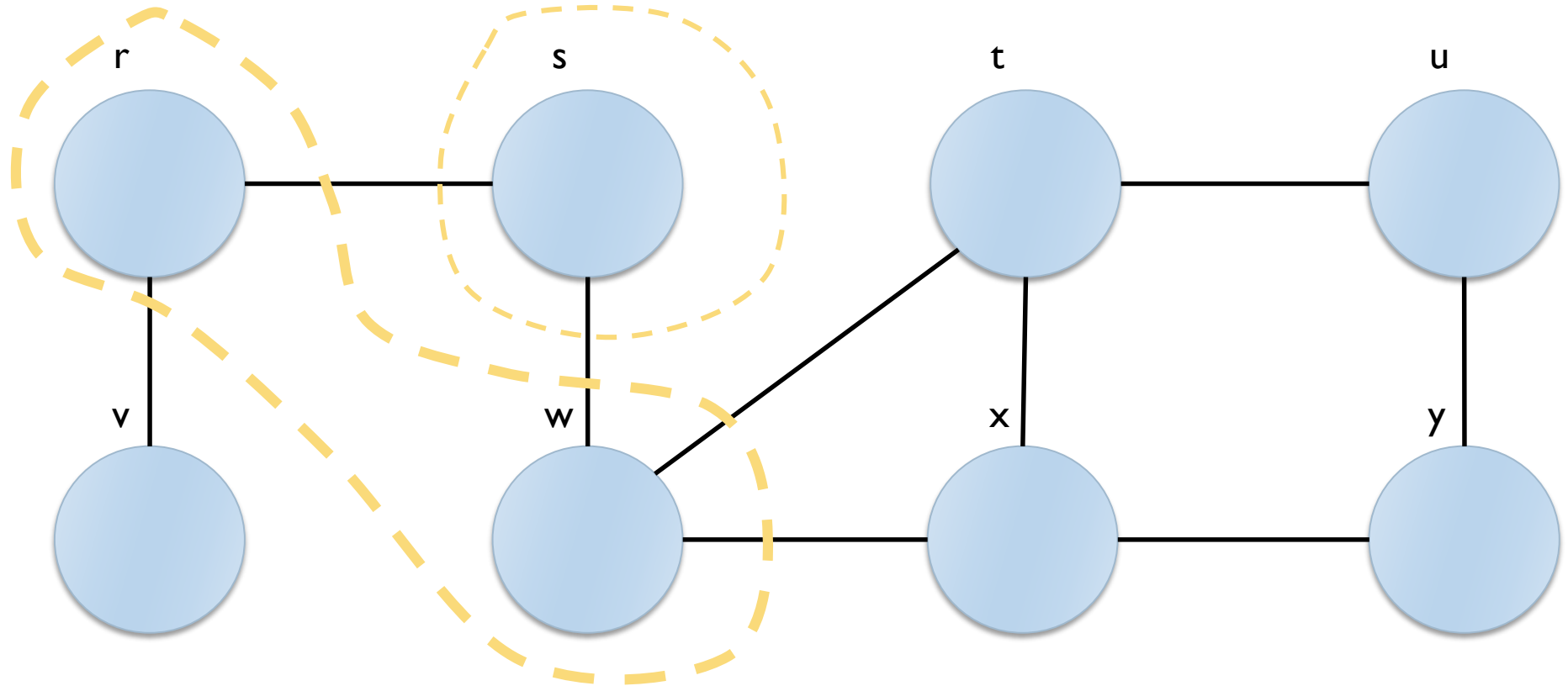
# Example

Source = s  
L = 0  
 $S_0 = \{s\}$



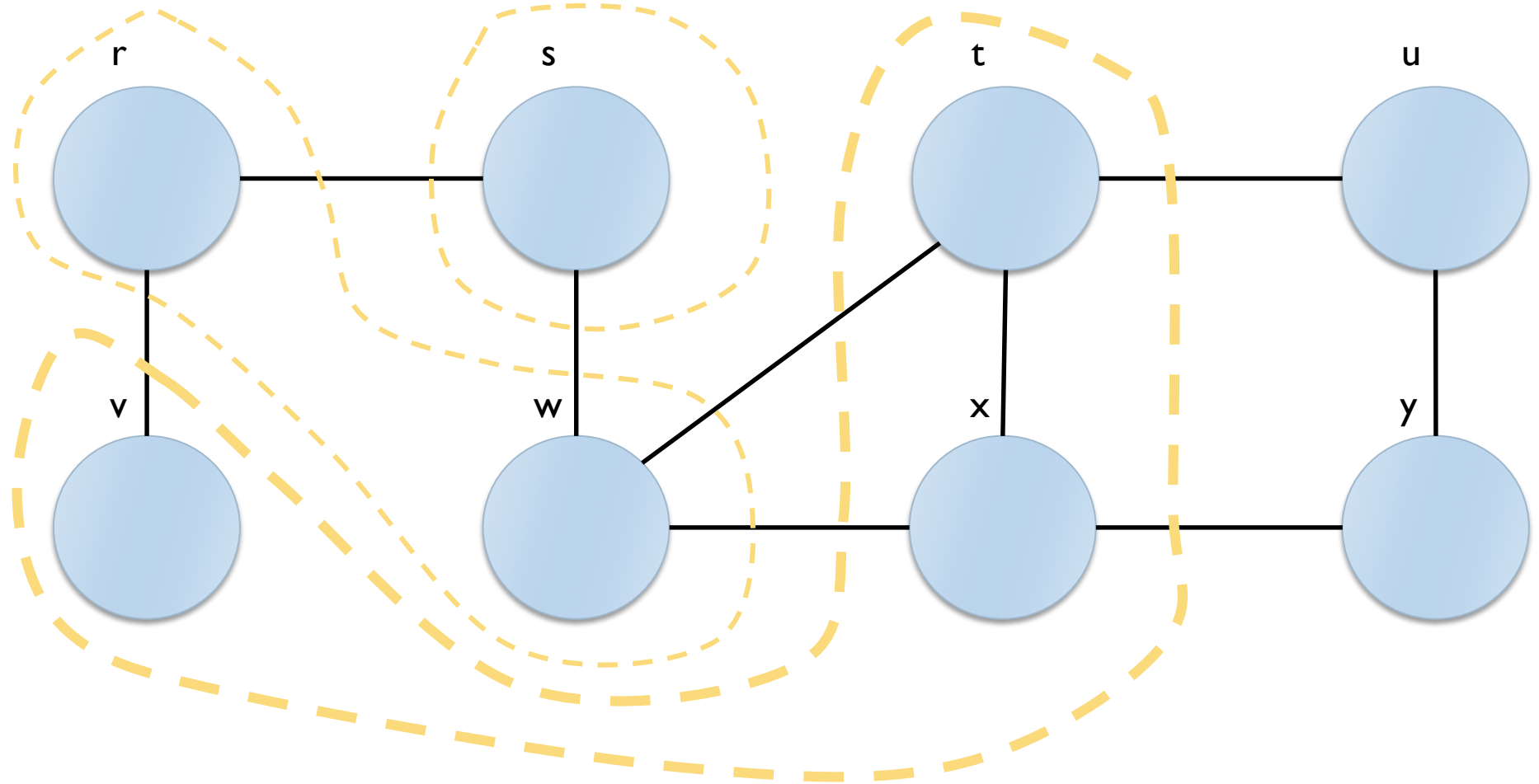
# Example

$L = I$   
 $S_0 = \{s\}$   
 $S_1 = \{r, w\}$



# Example

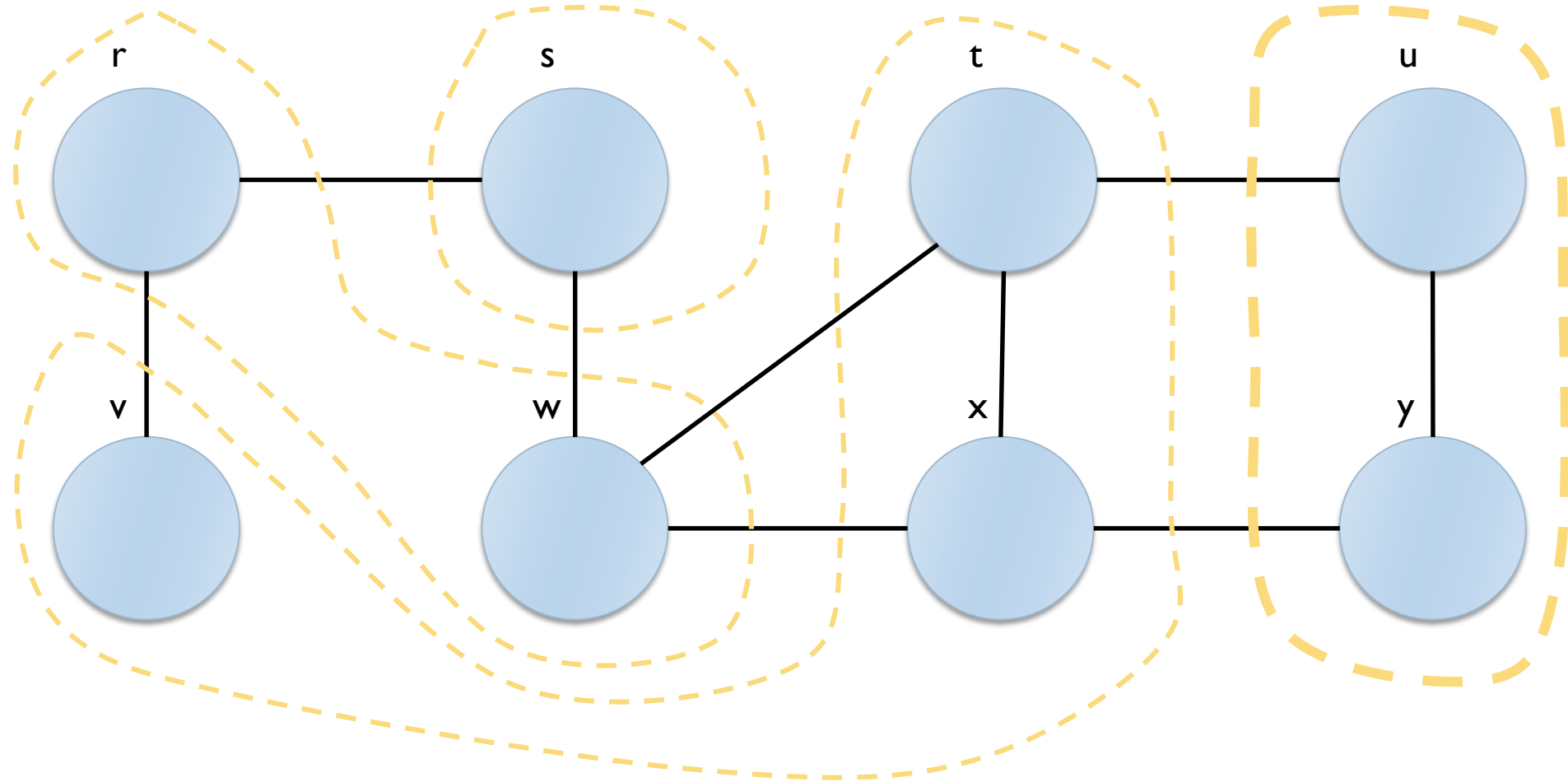
$L = 2$   
 $S_1 = \{r, w\}$   
 $S_2 = \{v, t, x\}$





# Example

$L = 3$   
 $S_2 = \{v, t, x\}$   
 $S_3 = \{u, y\}$



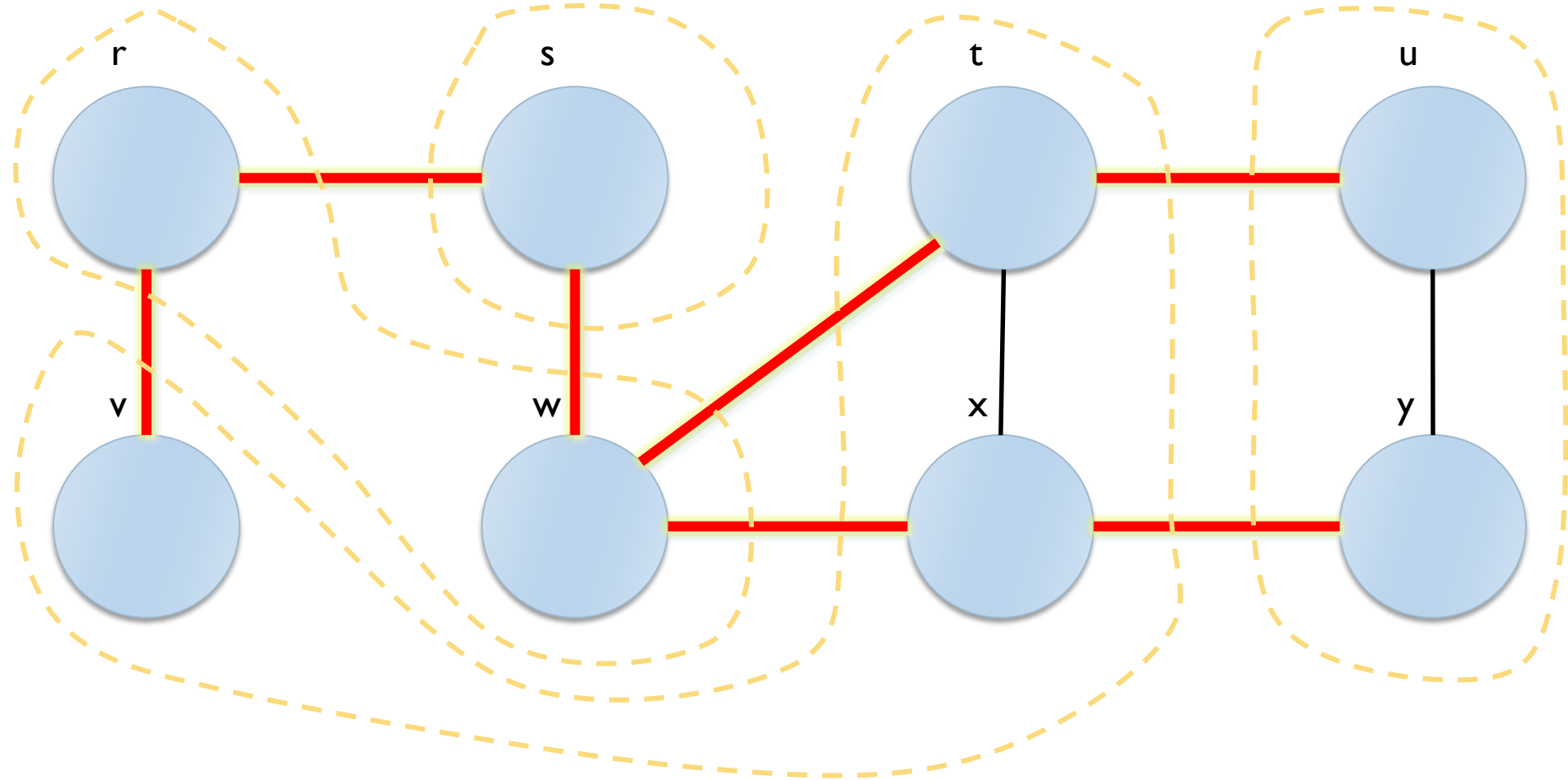
# BFS Tree

---

- ▶ The result of a BFV identifies a “visit tree” in the graph:
  - ▶ The tree root is the source vertex
  - ▶ Tree nodes are all graph vertices
    - ▶ (in the same connected component of the source)
  - ▶ Tree are a subset of graph edges
    - ▶ Those edges that have been used to “discover” new vertices.

# BFS Tree

---



# Minimum (shortest) paths

---

- ▶ Shortest path: the minimum number of edges on any path between two vertices
- ▶ The BFS procedure computes all minimum paths for all vertices, starting from the source vertex
- ▶ NB: unweighted graph : path length = number of edges

# Depth First Visit

---

- ▶ Also called Depth-first search (DFV or DFS)
- ▶ Opposite approach to BFS
- ▶ At every step, visit one (yet unvisited) vertex, adjacent to the last visited one
- ▶ If no such vertex exist, go back one step to the previously visited vertex
- ▶ Lends itself to recursive implementation
  - ▶ Similar to tree visit procedures

# DFS Algorithm

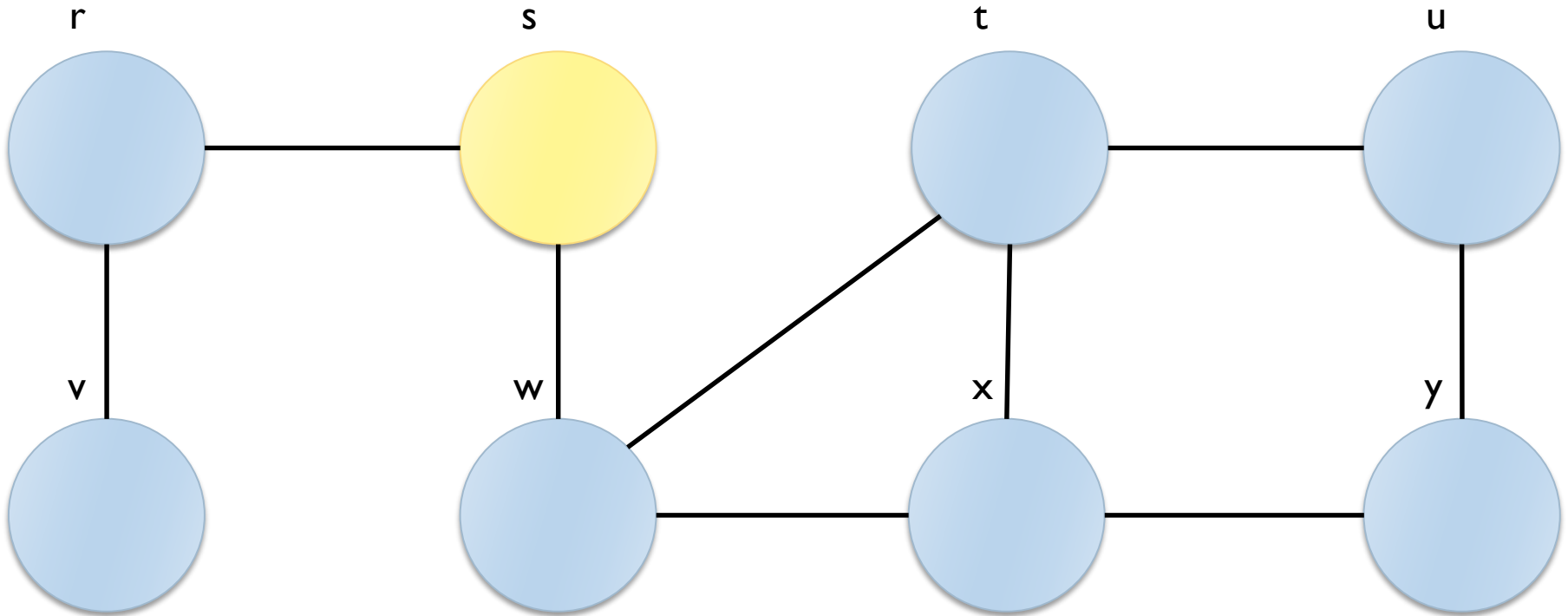
---

- ▶ DFS(Vertex  $v$ )
  - ▶ For all (  $w : \text{adjacent\_to}(v)$  )
    - ▶ If( not visited ( $w$ ) )
      - Visit ( $w$ )
      - DFS( $w$ )
  
- ▶ Start with: DFS(source)

# Example

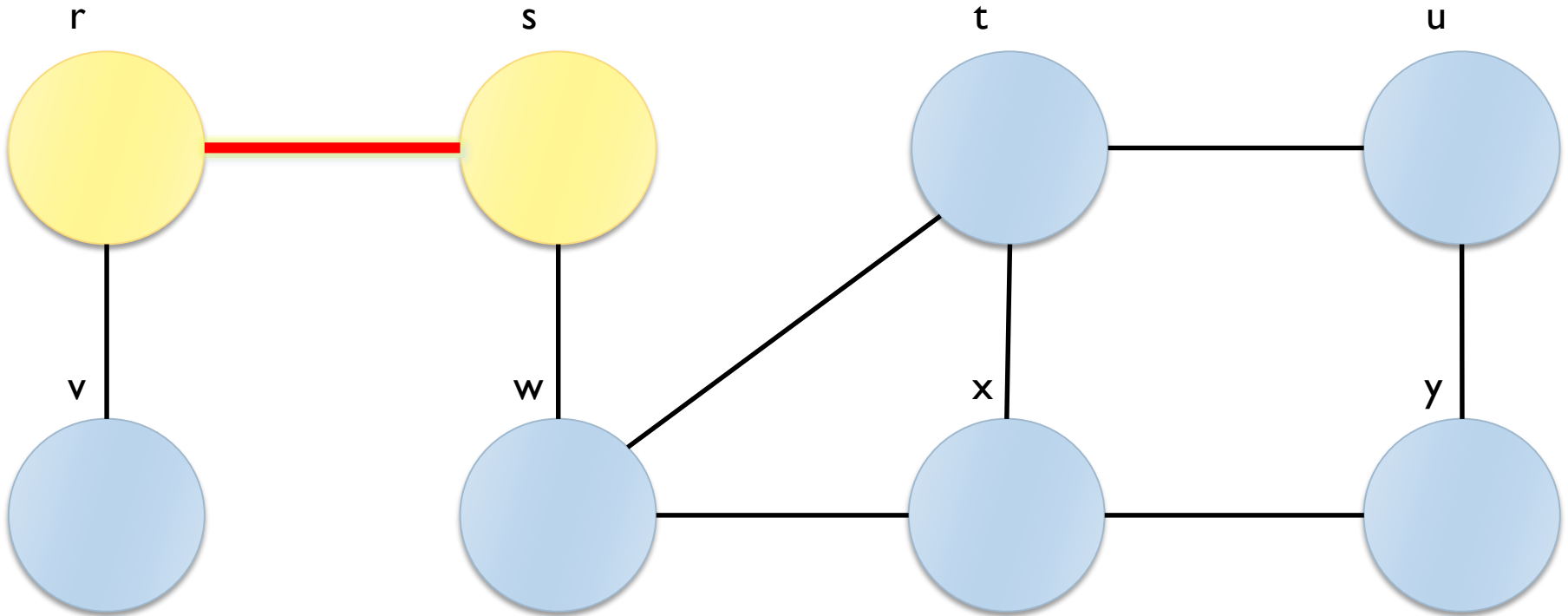
---

Source = s



# Example

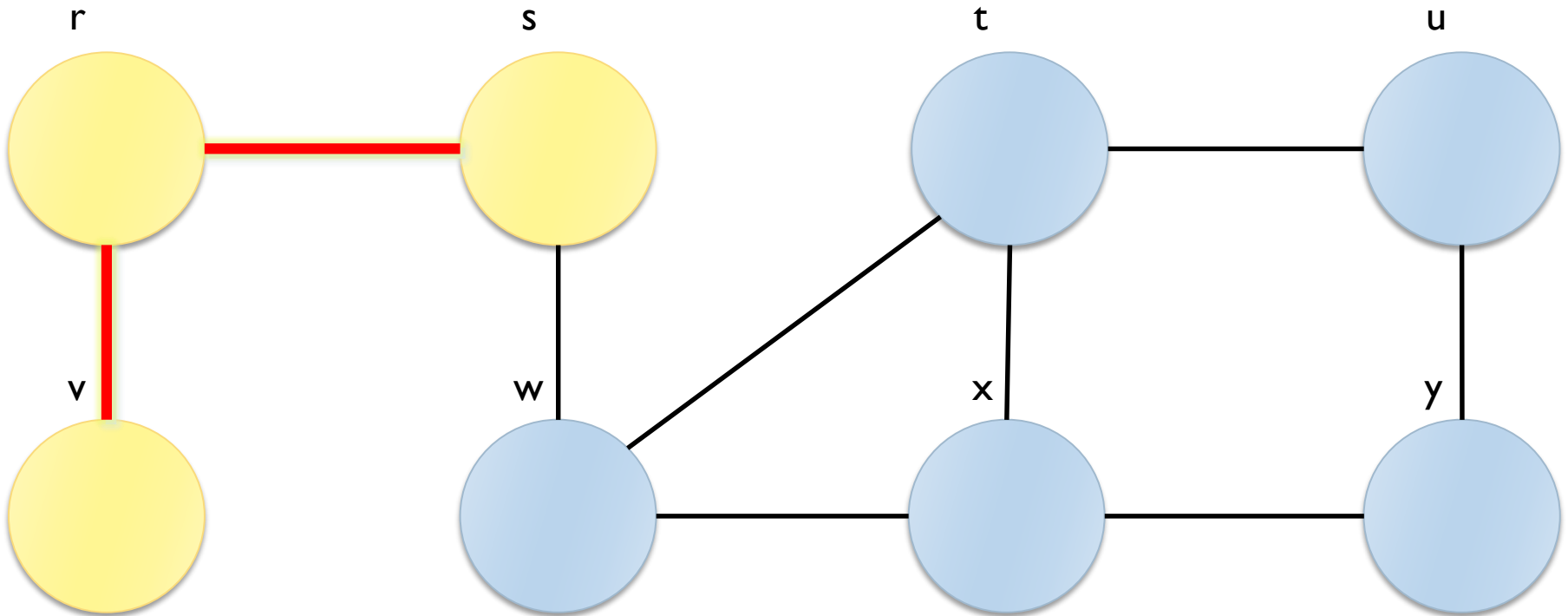
Source = s  
Visit r





# Example

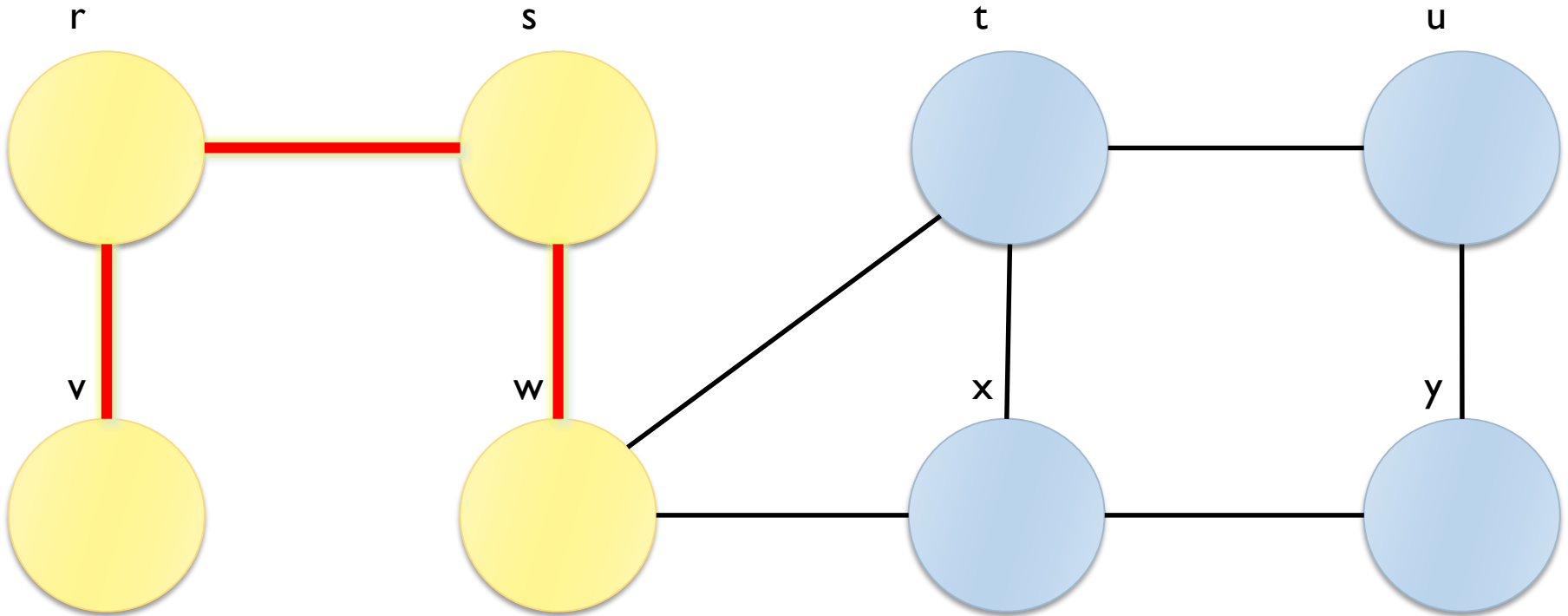
Source = s  
Visit r  
Visit v



# Example

---

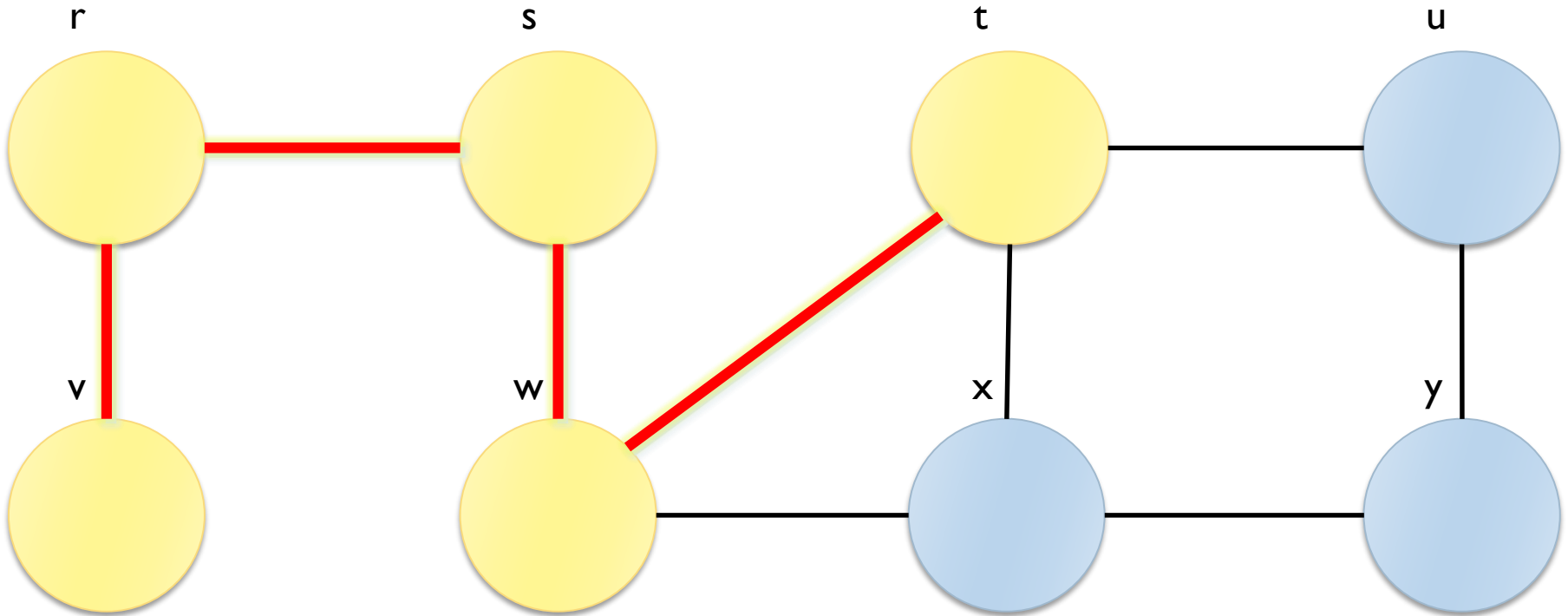
Source = s  
Back to r  
Back to s  
Visit w



# Example

---

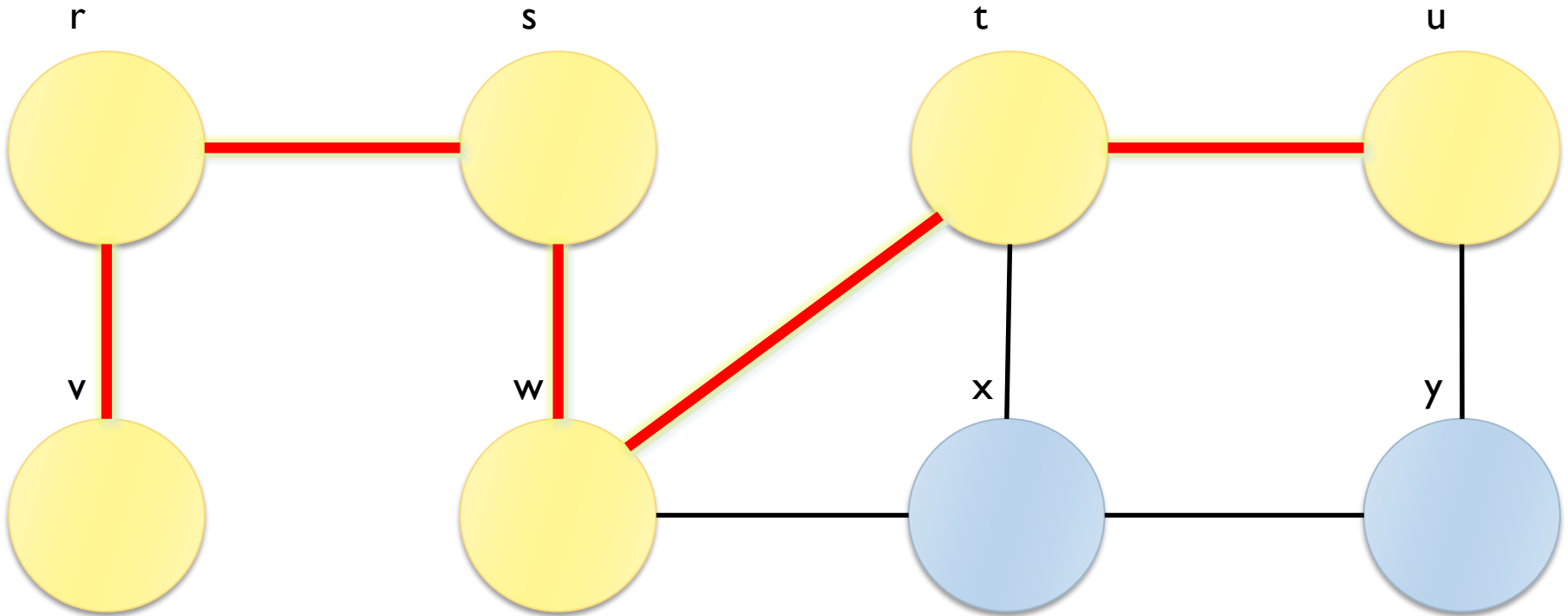
Source = s  
Visit w  
Visit t



# Example

---

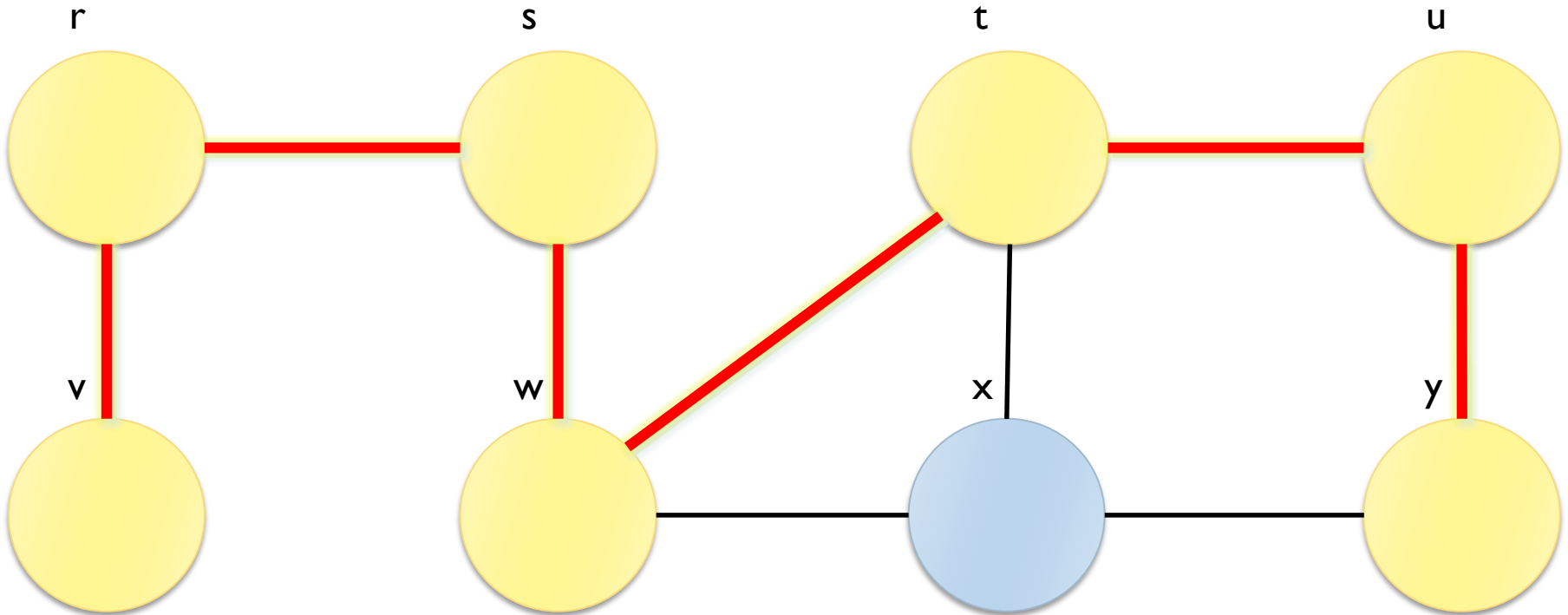
Source = s  
Visit w  
Visit t  
Visit u



# Example

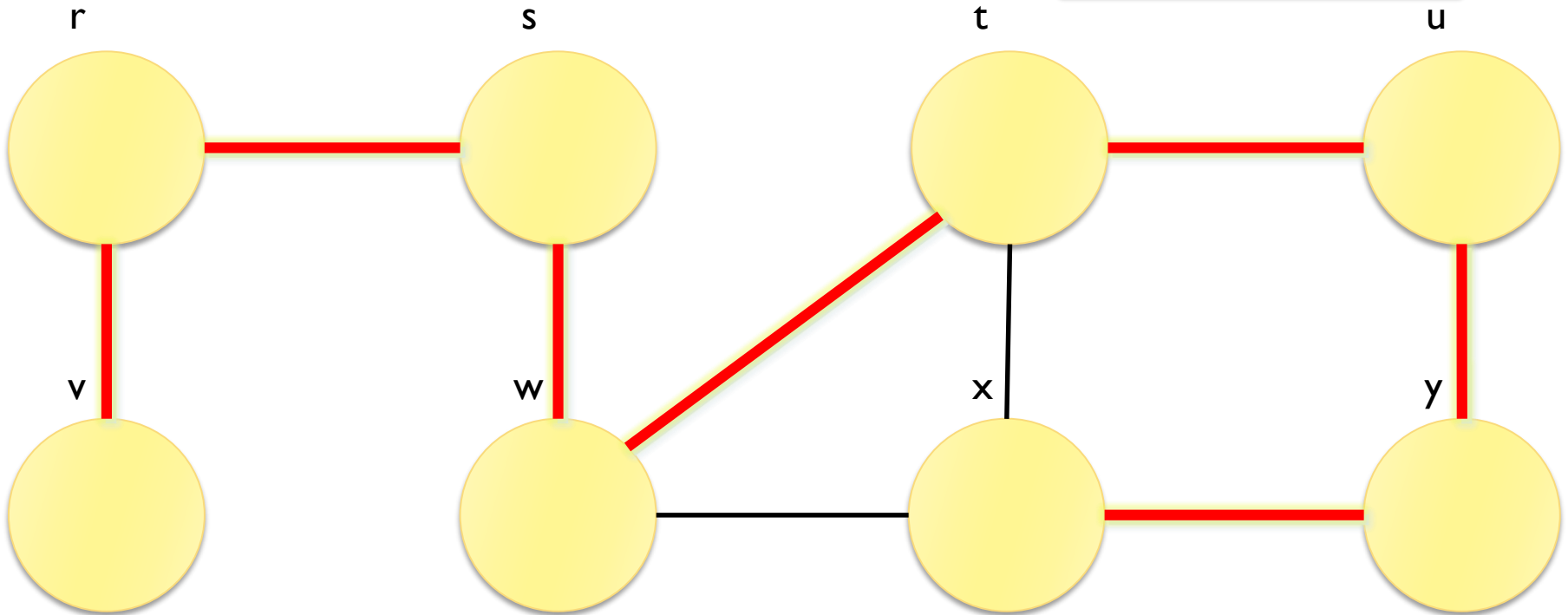
---

Source = s  
Visit w  
Visit t  
Visit u  
Visit y

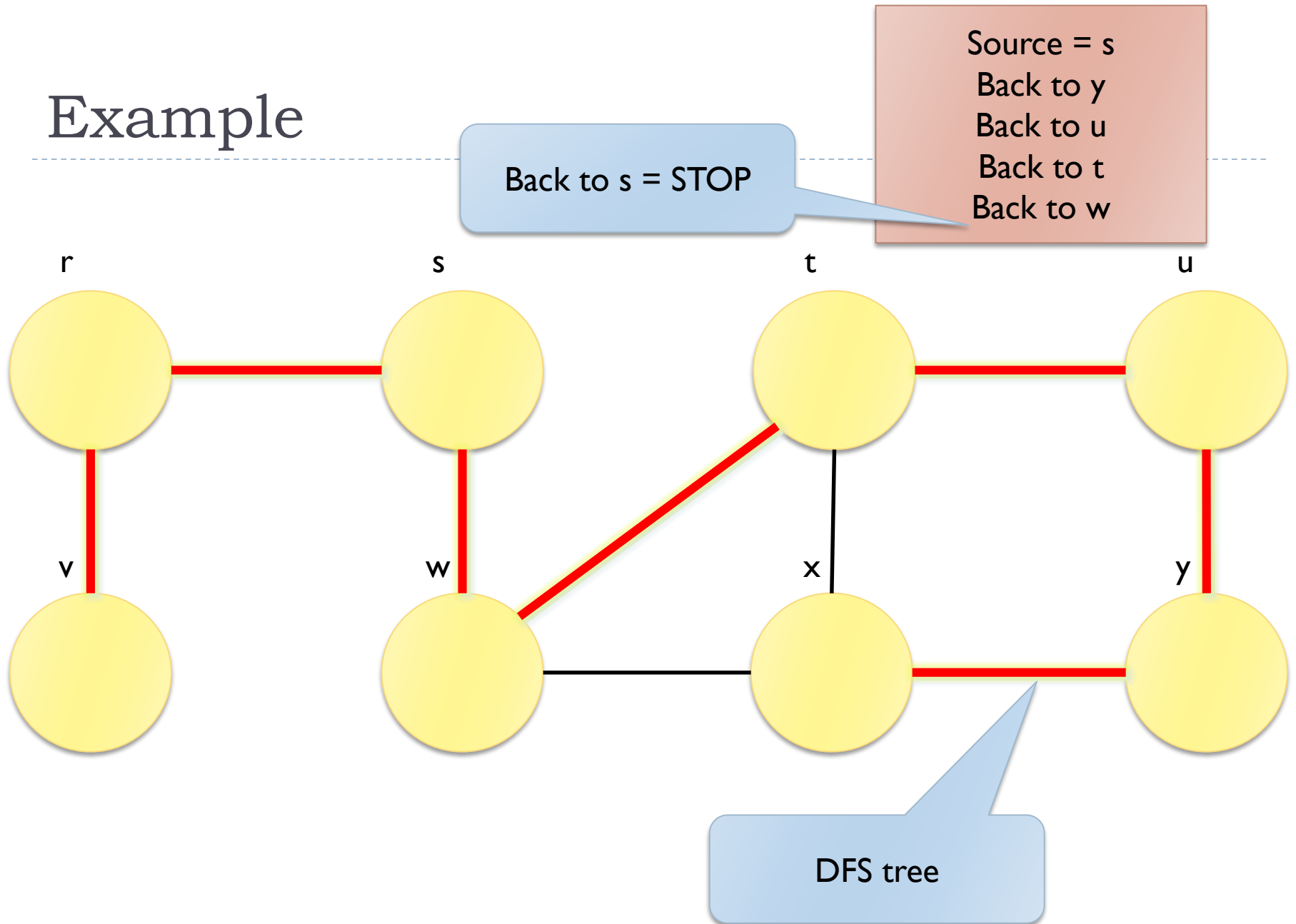


# Example

Source = s  
Visit w  
Visit t  
Visit u  
Visit y  
Visit x



# Example



# Edge classification

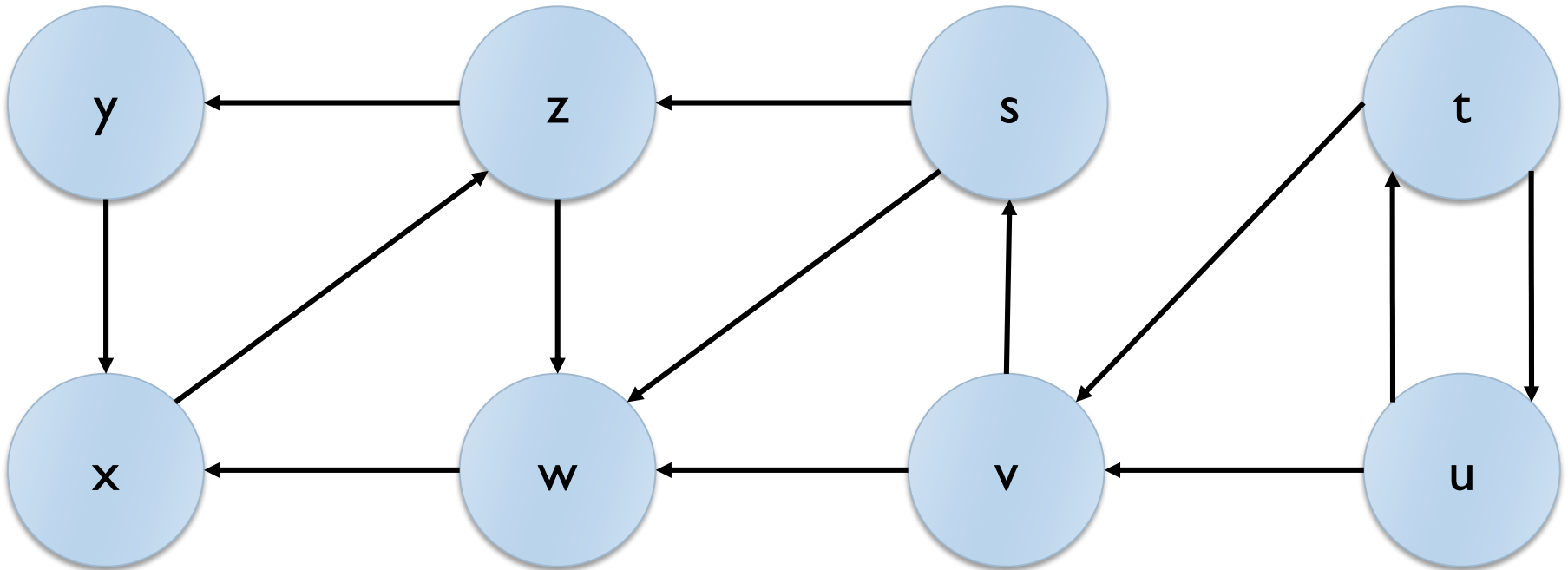
---

- ▶ In an directed graph, after a DFS visit, all edges fall in one of these 4 categories:
  - ▶ T: **Tree** edges (belonging to the DFS tree)
  - ▶ B: **Back** edges (not in T, and connect a vertex to one of its ancestors)
  - ▶ F: **Forward** edges (not in T and B, and connect a vertex to one of its descendants)
  - ▶ C: **Cross** edges (all remaining edges)



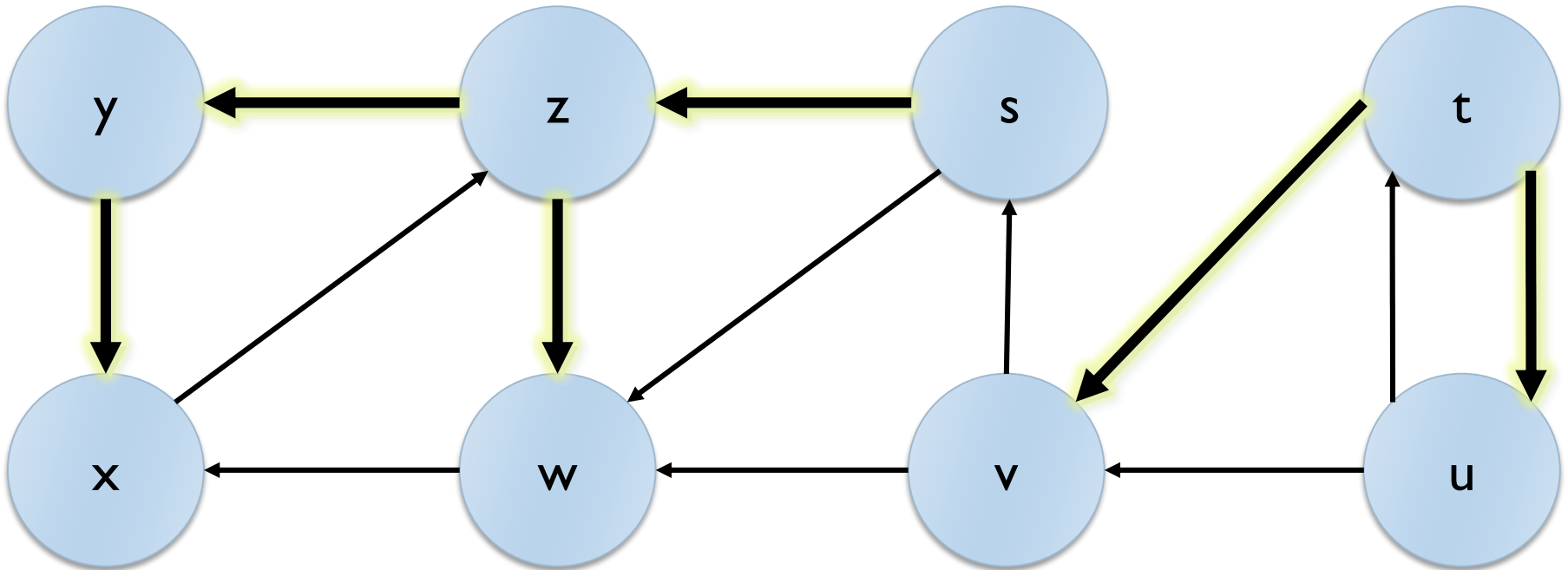
# Example

Directed graph



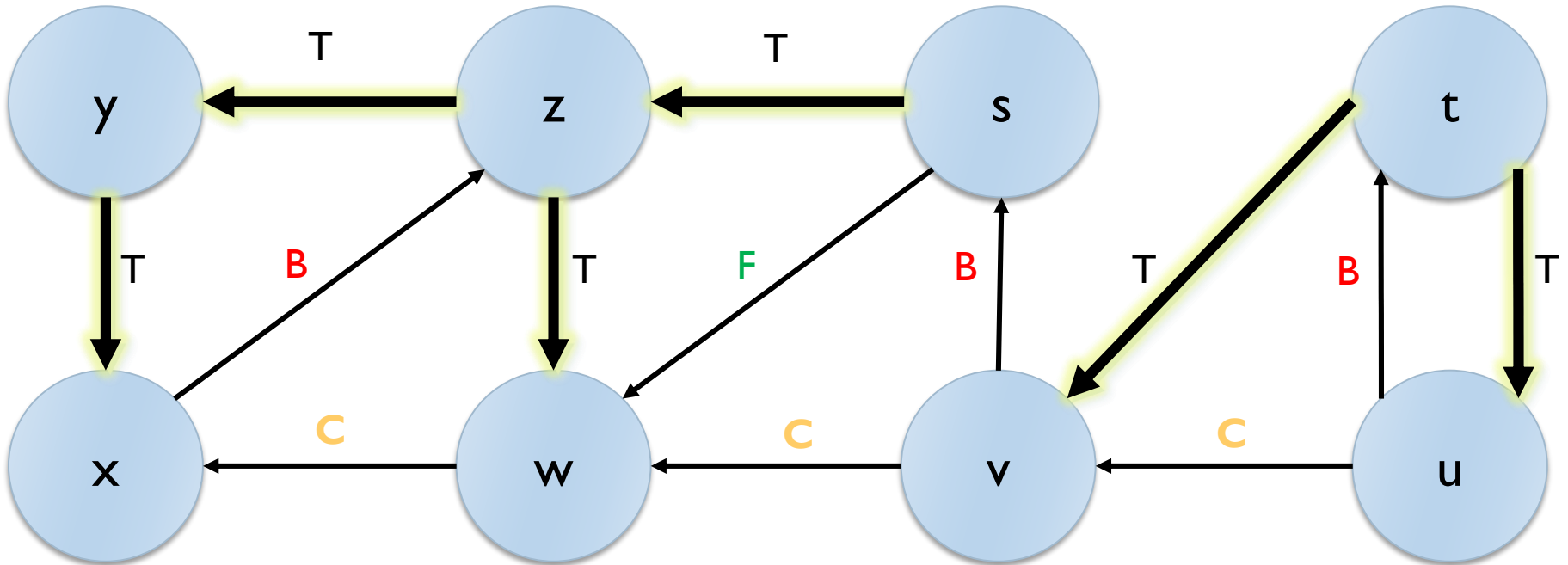
# Example

DFS visit  
(sources: s, t)



# Example

Edge classification



# Cycles

---

- ▶ Theorem:
- ▶ A directed graph is acyclic if and only if a depth-first visit does not produce any B edge

# Complexity

---

- ▶ Visits have linear complexity in the graph size
  - ▶ BFS :  $O(V+E)$
  - ▶ DFS :  $\Theta(V+E)$
- ▶ N.B. for dense graphs,  $E = O(V^2)$

# Resources

---

- ▶ Maths Encyclopedia: <http://mathworld.wolfram.com/>
- ▶ Basic Graph Theory with Applications to Economics  
<http://www.isid.ac.in/~dmishra/mpdoc/lecgraph.pdf>
- ▶ Application of Graph Theory in real world  
<http://prezi.com/tsehIwvpves-/application-of-graph-theory-in-real-world/>

# Resources

---

- ▶ Open Data Structures (in Java), Pat Morin, <http://opendatastructures.org/>
- ▶ Algorithms Course Materials, Jeff Erickson, <http://www.cs.uiuc.edu/~jeffe/teaching/algorithms/>
- ▶ Graphbook - A book on algorithmic graph theory, David Joyner, Minh Van Nguyen, and David Phillips, <https://code.google.com/p/graphbook/>





# JGraphT and visits

---

- ▶ Visits are called “traversals”
- ▶ Implemented through **Iterator** classes
- ▶ Package **org.jgrapht.traverse**

# Graph traversal classes

---

## Package org.jgrapht.traverse

Graph traversal means.

See: [Description](#)

### Interface Summary

Interface	Description
<code>GraphIterator&lt;V,E&gt;</code>	A graph iterator.

### Class Summary

Class	Description
<code>AbstractGraphIterator&lt;V,E&gt;</code>	An empty implementation of a graph iterator to minimize the effort required to implement graph iterators.
<code>BreadthFirstIterator&lt;V,E&gt;</code>	A breadth-first iterator for a directed or undirected graph.
<code>ClosestFirstIterator&lt;V,E&gt;</code>	A closest-first iterator for a directed or undirected graph.
<code>CrossComponentIterator&lt;V,E,D&gt;</code>	Provides a cross-connected-component traversal functionality for iterator subclasses.
<code>DegeneracyOrderingIterator&lt;V,E&gt;</code>	A degeneracy ordering iterator.
<code>DepthFirstIterator&lt;V,E&gt;</code>	A depth-first iterator for a directed or undirected graph.
<code>RandomWalkIterator&lt;V,E&gt;</code>	A random walk iterator for a directed or undirected graph.
<code>TopologicalOrderIterator&lt;V,E&gt;</code>	A topological ordering iterator for a directed acyclic graph.

# Graph iterators

---

- ▶ Usual hasNext() and next() methods

# Processing during traversal

---

- ▶ May register event listeners to traversal steps
  - ▶ void **addTraversalListener**(TraversalListener<V,E> l)
- ▶ TraversalListeners may react to:
  - ▶ Edge traversed
  - ▶ Vertex traversed
  - ▶ Vertex finished
  - ▶ Connected component started
  - ▶ Connected component finished

# Types of traversal iterators

---

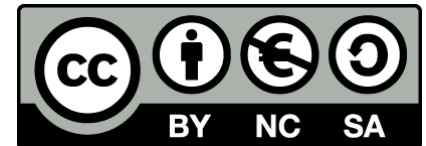
- ▶ **BreadthFirstIterator**
- ▶ **DepthFirstIterator**
- ▶ **ClosestFirstIterator**
  - ▶ The metric for *closest* here is the path length from a start vertex. `Graph.getEdgeWeight(Edge)` is summed to calculate path length. Optionally, path length may be bounded by a finite radius.
- ▶ **TopologicalOrderIterator**
  - ▶ A topological sort is a permutation  $p$  of the vertices of a graph such that an edge  $\{i,j\}$  implies that  $i$  appears before  $j$  in  $p$ . Only directed acyclic graphs can be topologically sorted.






# Resources

---

- ▶ JGraphT Library: <http://jgrapht.org/>

# Licenza d'uso



- ▶ Queste diapositive sono distribuite con licenza Creative Commons “Attribuzione - Non commerciale - Condividi allo stesso modo (CC BY-NC-SA)”
- ▶ Sei libero:
  - ▶ di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare quest'opera 
  - ▶ di modificare quest'opera 
- ▶ Alle seguenti condizioni:
  - ▶ **Attribuzione** — Devi attribuire la paternità dell'opera agli autori originali e in modo tale da non suggerire che essi avallino te o il modo in cui tu usi l'opera. 
  - ▶ **Non commerciale** — Non puoi usare quest'opera per fini commerciali. 
  - ▶ **Condividi allo stesso modo** — Se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con una licenza identica o equivalente a questa. 
- ▶ <http://creativecommons.org/licenses/by-nc-sa/3.0/>