

# CMSC5733 Social Computing

## Tutorial 2: NetworkX & Graphviz

Baichuan Li

The Chinese University of Hong Kong

[bcli@cse.cuhk.edu.hk](mailto:bcli@cse.cuhk.edu.hk)

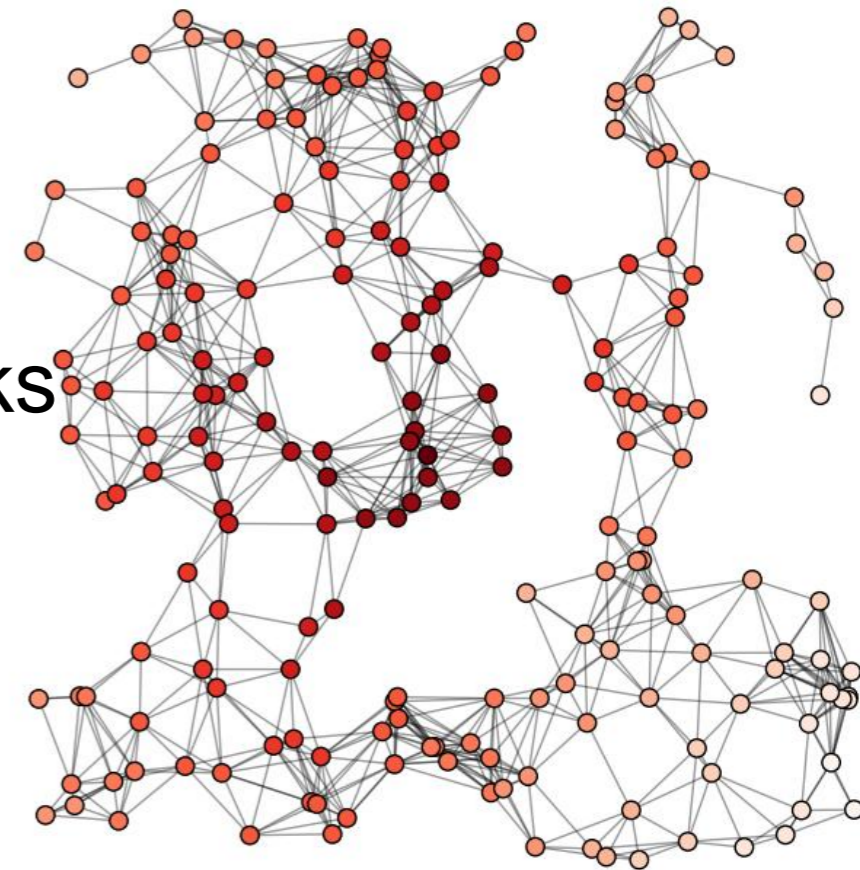
# Outline

- NetworkX
  - Creating A Graph
  - Adding Attributes
  - Directed Graphs
  - Graph Generators
  - Analyzing Graphs
  - Drawing Graphs
- Graphviz
  - Dot Language
  - Sample Graphs

**NETWORKX**

# Introduction

- A Python language software package for the creation, manipulation, and study of the structure, dynamics, and functions of **complex networks**
- What can it do?
  - Load and store networks
  - Generate random and classic networks
  - Analyze network structure
  - Build network models
  - Draw networks
  - ...



# Installing

- Quick install
  - Python Egg: <http://pypi.python.org/pypi/networkx>
  - **easy\_install networkx** or **pip install networkx**
- Current version: **1.8**
- Requirement: Python version 2.6 or later
- Optional packages
  - NumPy, SciPy, Matplotlib, Pyparsing, etc.

# Creating A Graph

- Create an empty graph with no nodes and no edges

```
>>> import networkx as nx
>>> G=nx.Graph()
```

- In NetworkX, nodes can be any hashable object e.g. a text string, an image, an XML object, another Graph, a customized node object, etc.

# Nodes

- Add one node at a time

```
>>> G.add_node(1)
```

- Add a list of nodes

```
>>> G.add_nodes_from([2,3])
```

- Add *nbunch* of nodes

- An *nbunch* is any iterable **container of nodes** that is not itself a node in the graph
- E.g., a list, set, graph, file, etc.

```
>>> H=nx.path_graph(10)  
>>> G.add_nodes_from(H)
```

- Add a graph as a node

```
>>> G.add_node(H)
```

# Demolishing A Graph

- Functions
  - `remove_node()`
  - `remove_nodes_from()`
  - `remove_edge()`
  - `remove_edges_from()`
  - `clear()`



# Graph Properties

- Functions
  - `nodes()`
  - `number_of_nodes()`
  - `edges()`
  - `number_of_edges()`
  - `neighbors()`
  - `degree()`
  - ...

# Adding Attributes

- Attributes
- Weights, labels, colors, etc.

```
>>> G = nx.Graph(day="Friday")
>>> G.graph
{'day': 'Friday'}
```

```
>>> G.graph['day'] = 'Monday'
>>> G.graph
{'day': 'Monday'}
```

# Node and Edge Attributes

```
>>> G.add_node(1, time='5pm')
>>> G.add_nodes_from([3], time='2pm')
>>> G.node[1]
{'time': '5pm'}
>>> G.node[1]['room'] = 714
```

```
>>> G.add_edge(1, 2, weight=4.7 )
>>> G.add_edges_from([(3,4),(4,5)], color='red')
>>> G.add_edges_from([(1,2,{'color':'blue'}), (2,3,{'weight':8})])
>>> G[1][2]['weight'] = 4.7
>>> G.edge[1][2]['weight'] = 4
```

# Directed Graph

- Additional functions
  - `out_edges()`, `in_degree()`
  - `predecessors()`, `successors()`

```
>>> DG=nx.DiGraph()
>>> DG.add_weighted_edges_from([(1,2,0.5), (3,1,0.75)])
>>> DG.out_degree(1,weight='weight')
0.5
>>> DG.degree(1,weight='weight')
1.25
>>> DG.successors(1)
[2]
>>> DG.neighbors(1)
[2]
```

# Graph Generators

- Applying classic graph operations
  - `subgraph(G, nbunch)` - induce subgraph of `G` on nodes in `nbunch`
  - `union(G1,G2)` - graph union
  - `disjoint_union(G1,G2)` - graph union assuming all nodes are different
  - `cartesian_product(G1,G2)` - return Cartesian product graph
  - `compose(G1,G2)` - combine graphs identifying nodes common to both
  - `complement(G)` - graph complement
  - `create_empty_copy(G)` - return an empty copy of the same graph class
  - `convert_to_undirected(G)` - return an undirected representation of `G`
  - `convert_to_directed(G)` - return a directed representation of `G`

# Graph Generators (cont.)

- Calling graph generators

```
>>> petersen=nx.petersen_graph()  
>>> tutte=nx.tutte_graph()  
>>> maze=nx.sedgewick_maze_graph()  
>>> tet=nx.tetrahedral_graph()
```

```
>>> K_5=nx.complete_graph(5)  
>>> K_3_5=nx.complete_bipartite_graph(3,5)  
>>> barbell=nx.barbell_graph(10,10)  
>>> lollipop=nx.lollipop_graph(10,20)
```

```
>>> er=nx.erdos_renyi_graph(100,0.15)  
>>> ws=nx.watts_strogatz_graph(30,3,0.1)  
>>> ba=nx.barabasi_albert_graph(100,5)  
>>> red=nx.random_lobster(100,0.9,0.9)
```

Refer to <http://networkx.lanl.gov/reference/generators.html> for details

# Graph Generators (cont.)

- Loading from files
- Common graph formats: edge lists, adjacency lists, GML, GraphML, pickle, LEDA, etc.

```
>>> nx.write_gml(red, "path.to.file")  
>>> mygraph=nx.read_gml("path.to.file")
```

- Download *pyparsing* from <http://pypi.python.org/pypi/pyparsing/>

# Analyzing Graphs

```
>>> G=nx.Graph()
>>> G.add_edges_from([(1,2),(1,3)])
>>> G.add_node("spam")           # adds node "spam"
```

```
>>> nx.connected_components(G)
[[1, 2, 3], ['spam']]
```

```
>>> sorted(nx.degree(G).values())
[0, 1, 1, 2]
```

```
>>> nx.degree(G)
{1: 2, 2: 1, 3: 1, 'spam': 0}
```

Refer to <http://networkx.lanl.gov/reference/algorithms.html> for details



# Drawing Graphs

- First import **Matplotlib**'s plot interface

```
>>> import matplotlib.pyplot as plt
```

- Draw

```
>>> nx.draw(G)
>>> nx.draw_random(G)
>>> nx.draw_circular(G)
>>> nx.draw_spectral(G)
```

- Show the graph

```
>>> plt.show()
```

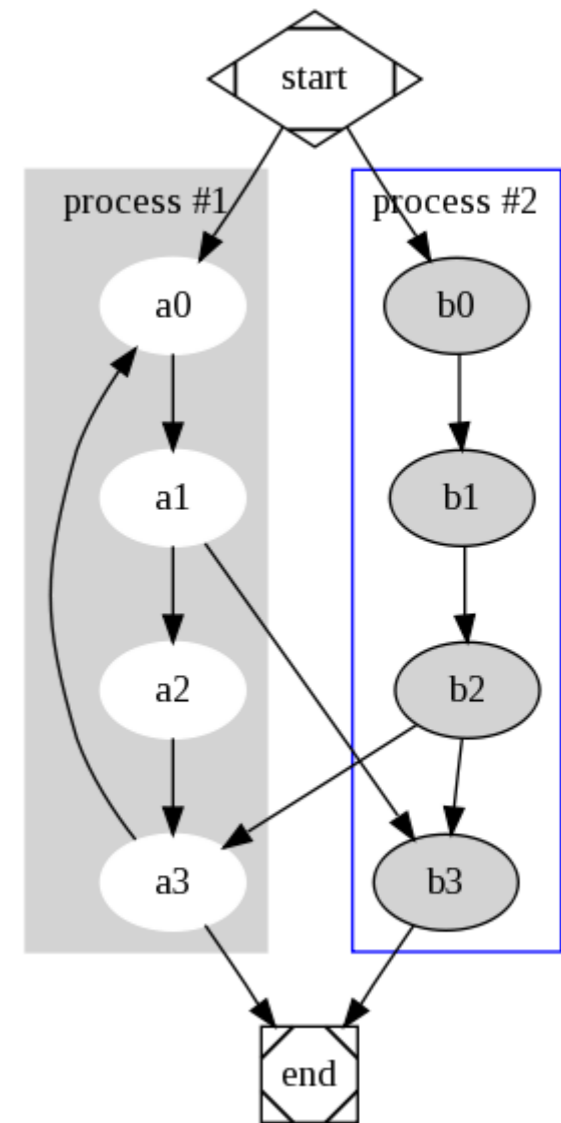
- Save

```
>>> plt.savefig("path.png")
```

**GRAPHVIZ**

# Introduction

- An open source **graph visualization** software
- Graph drawing
- Graph layout
- Features
  - Take descriptions of graphs in a simple text language
  - Make diagrams in useful formats
  - Colors, fonts, tabular node layouts, line styles, hyperlinks, etc.



# Installing

- Download at <http://www.graphviz.org/Download.php>
- Work on
  - FreeBSD
  - Linux
  - Mac
  - Solaris
  - Windows

# The Dot Language

```
graph: [ strict ] (graph | digraph) [ ID ] '{ stmt_list }'  
stmt_list: [ stmt [ ';' ] [ stmt_list ] ]  
stmt: node_stmt  
      | edge_stmt  
      | attr_stmt  
      | ID '=' ID  
      | subgraph  
attr_stmt: (graph | node | edge) attr_list  
attr_list: '[' [ a_list ] '[' attr_list ]  
a_list: ID [ '=' ID ] [ ',' ] [ a_list ]  
edge_stmt: (node_id | subgraph) edgeRHS [ attr_list ]  
edgeRHS: edgeop (node_id | subgraph) [ edgeRHS ]  
node_stmt: node_id [ attr_list ]  
node_id: ID [ port ]  
port: ':' ID [ ':' compass_pt ]  
      | ':' compass_pt  
subgraph: [ subgraph [ ID ] ] '{ stmt_list }'  
compass_pt: (n | ne | e | se | s | sw | w | nw | c | _)
```

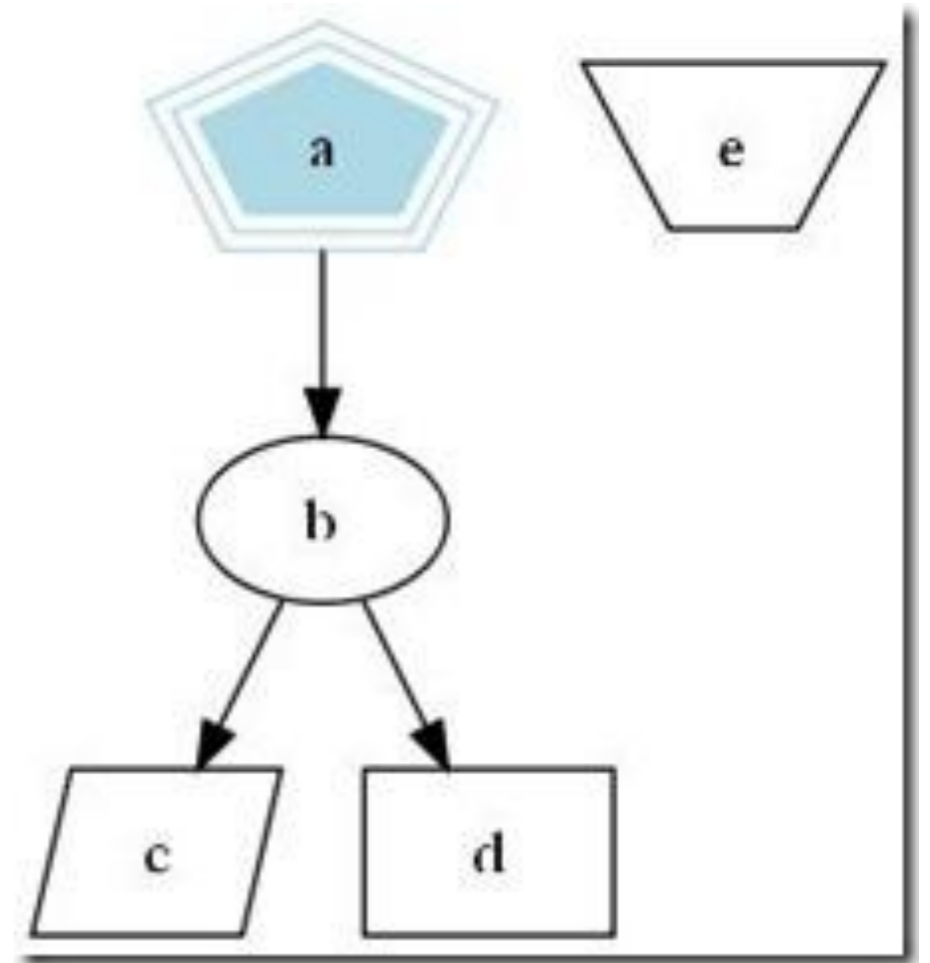
# Sample Graphs

```
digraph G{
  size = "4, 4"
  a->b->c;
  b->d;

  a[shape = polygon, sides = 5, peripheries=3,
  color = lightblue, style = filled];

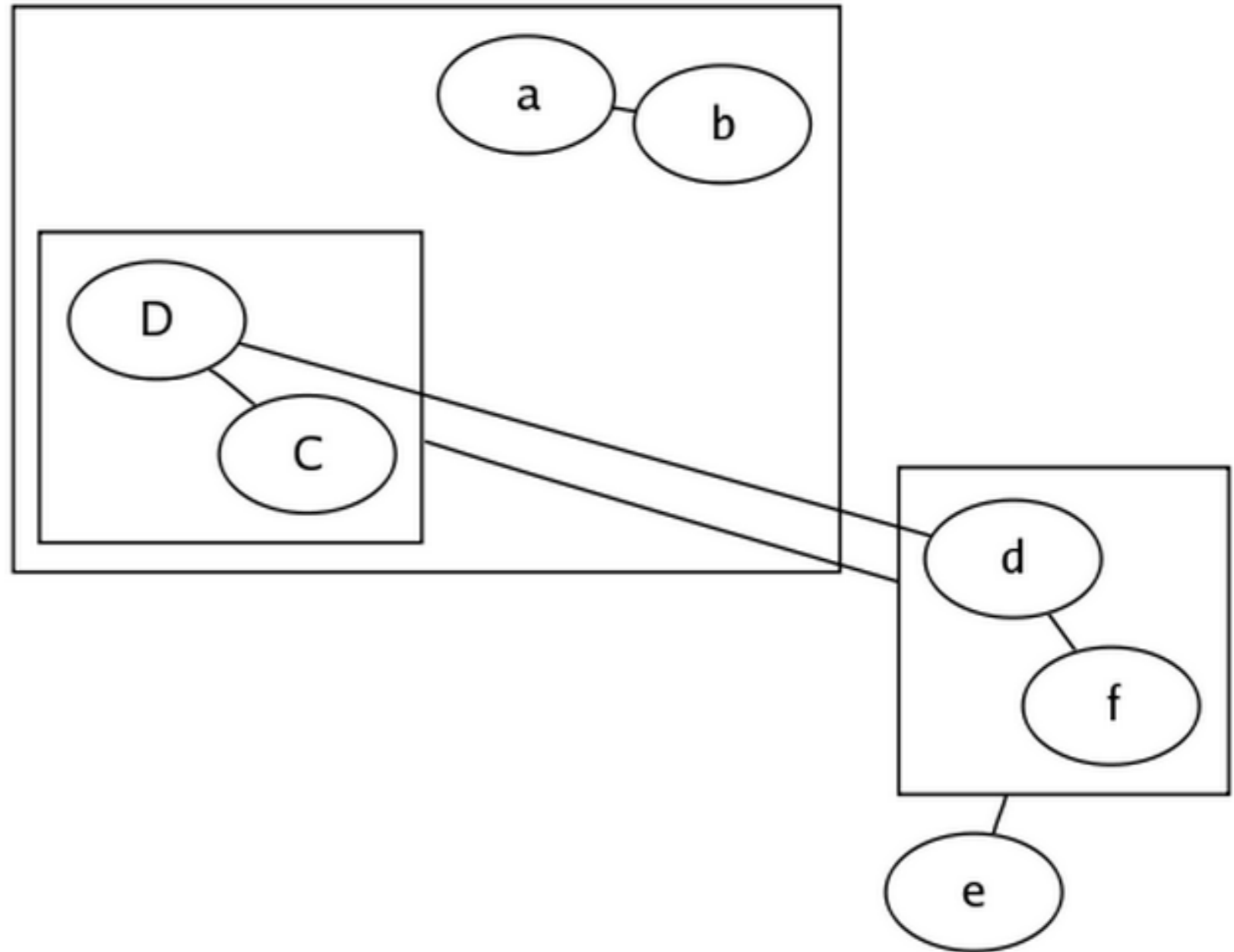
  c[shape = polygon, sides = 4, skew= 0.4,
  lable = "hello world"];

  d[shape = invtriange];
  e[shape = polygon, side = 4, distortion = .7];
}
```



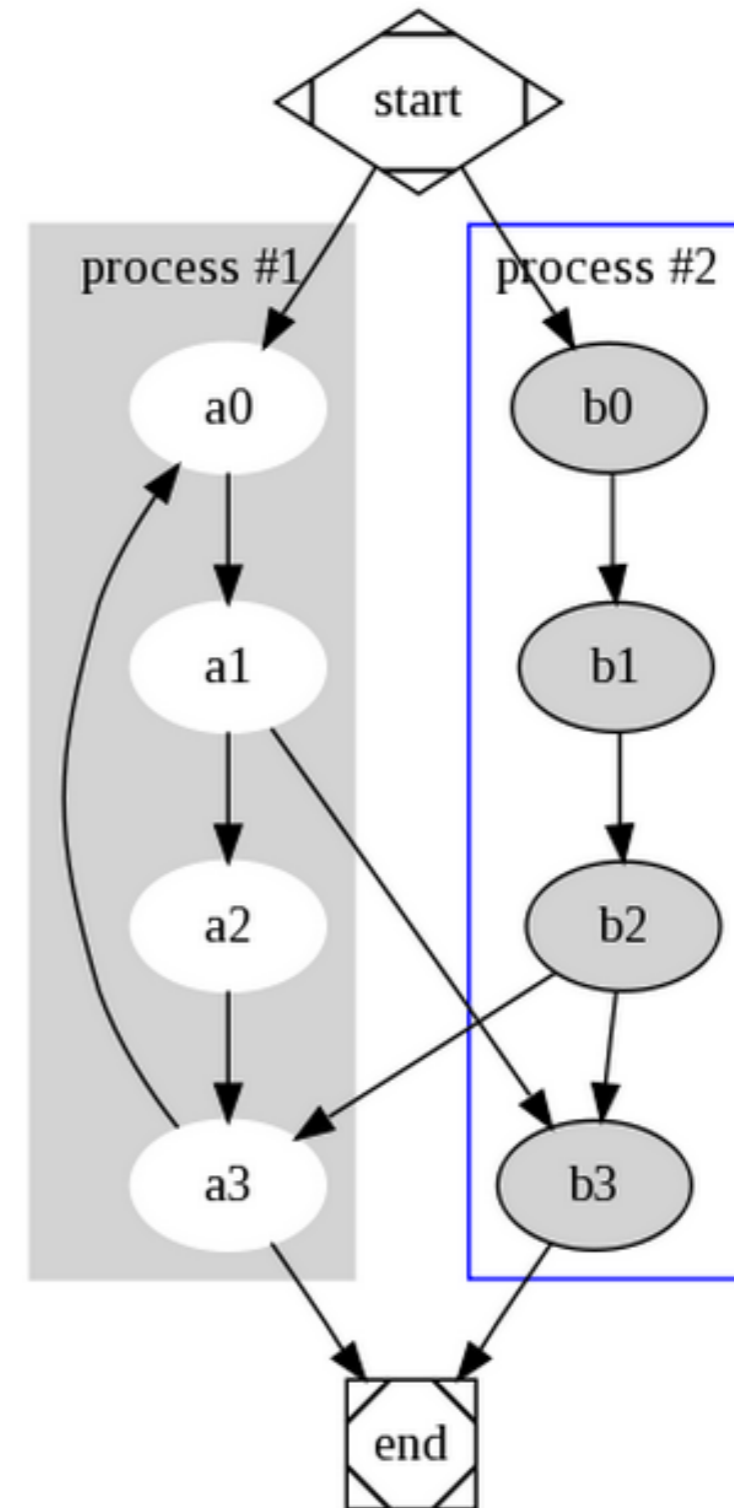
# Sample Graphs (cont.)

```
graph G {
  e
  subgraph clusterA {
    a -- b;
    subgraph clusterC {
      C -- D;
    }
  }
  subgraph clusterB {
    d -- f
  }
  d -- D
  e -- clusterB
  clusterC -- clusterB
}
```



# Sample Graphs (cont.)

```
digraph G {  
  
  subgraph cluster_0 {  
    style=filled;  
    color=lightgrey;  
    node [style=filled,color=white];  
    a0 -> a1 -> a2 -> a3;  
    label = "process #1";  
  }  
  
  subgraph cluster_1 {  
    node [style=filled];  
    b0 -> b1 -> b2 -> b3;  
    label = "process #2";  
    color=blue  
  }  
  
  start -> a0;  
  start -> b0;  
  a1 -> b3;  
  b2 -> a3;  
  a3 -> a0;  
  a3 -> end;  
  b3 -> end;  
  
  start [shape=Mdiamond];  
  end [shape=Msquare];  
}
```





# About the Project

- Project grouping deadline: Sep 23, 2013
  - Up to four students form a team to finish the project
  - Please send your team members' names, student ids, and email addresses to [cmssc5733@gmail.com](mailto:cmssc5733@gmail.com) with the email title "CMSC grouping"
- Proposal deadline: Sep 30, 2013
  - Use the proposal template on our course Wiki

# Declaration

- All graphs in this tutorial are from the following websites:
  - <http://networkx.lanl.gov/>
  - <http://www.graphviz.org>
  - <http://www.cnblogs.com/sld666666/archive/2010/06/25/1765510.html>

# References

- [http://networkx.lanl.gov/downloads/networkx\\_tutorial.pdf](http://networkx.lanl.gov/downloads/networkx_tutorial.pdf)
- <http://networkx.lanl.gov/contents.html>
- <http://www.graphviz.org/content/dot-language>
- <http://www.cnblogs.com/sld666666/archive/2010/06/25/1765510.html>