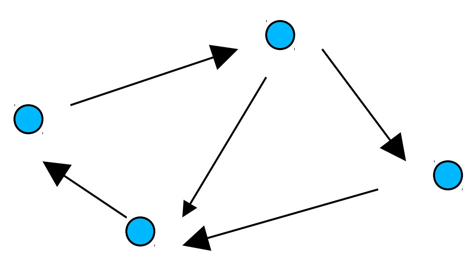
Computer Science 112
Data Structures

Lecture 21:
Graphs:
Depth First Search
Topsort

Review: Graphs

Generalization of trees

- Digraph (Directed Graph)
  - Like a tree but any vertex can point to any other

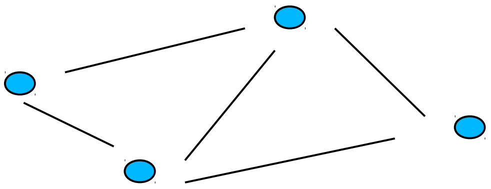


  - E.g., Twitter follows relationship

Graphs

Generalization of trees

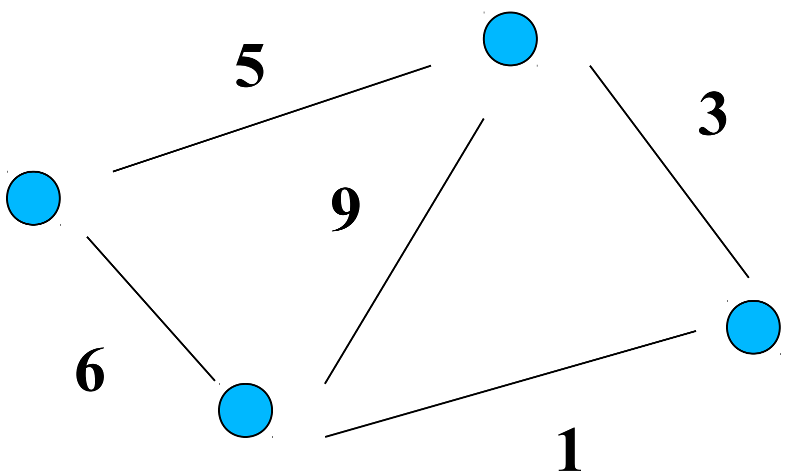
- **Graph**
  - **like digraph but arcs have no direction**



  - **E.g., Facebook friends relationship**

Graphs

- **Weighted Graph**
  - **Positive integer weights on each edge**

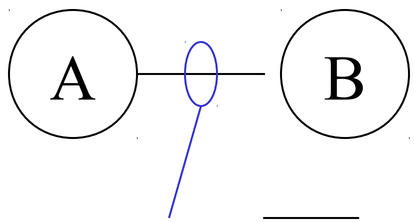


Applications

- **Lots**

Notation

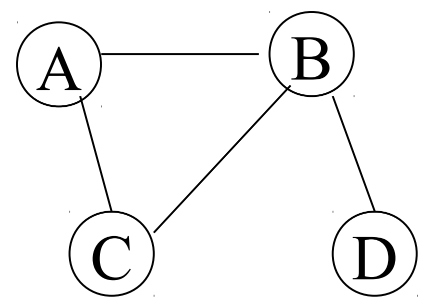
- **Arcs are named by the vertices they connect**



(B,A) or $\overline{\mathbf{BA}}$

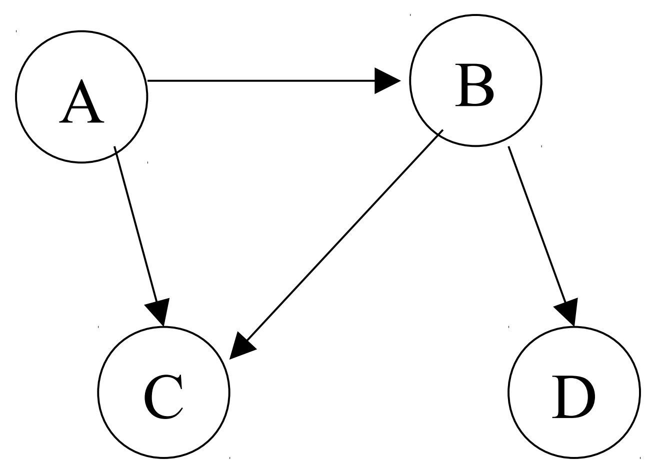
Graph Concepts

- **Neighbors of a vertex: vertices that it shares an arc with**
  - **Neighbors of A are B and C**
- **Degree of a vertex: number of neighbors**
  - **Degree of A is 2, degree of B is 3**



Graph Concepts

- **In degree (in a digraph): number of vertices that have arcs to this vertex**
  - **In degree of B is 1**
- **Out degree (in a digraph): number of vertices that have arcs from this vertex**
  - **Out degree of B is 2**



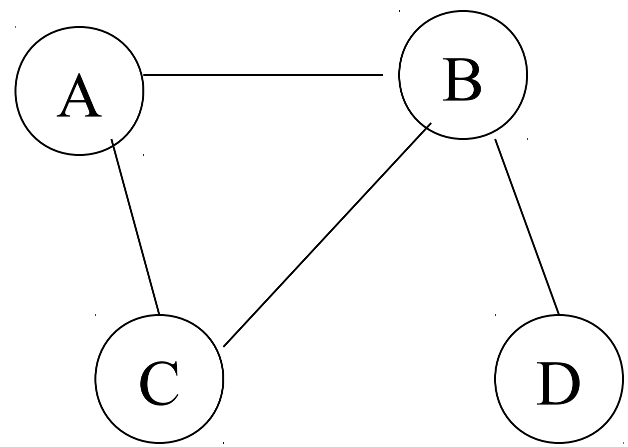
Graph Concepts

- **(Simple) Path**
  - **Sequence of arcs**

    (A,B),(B,C)
  - **May not revisit a vertex**

    ~~**(B,A),(A,C),(C,B),(B,D)**~~
  - **Except last vertex may = first**

    (B,A),(A,C),(C,B)
- **Vertex A is** reachable **from B if there is a path from B to A**

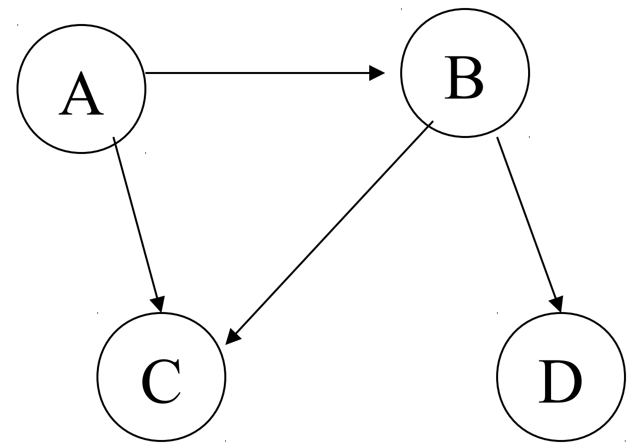


Graph Concepts

- **Path**
  - **On digraph must follow arc directions**

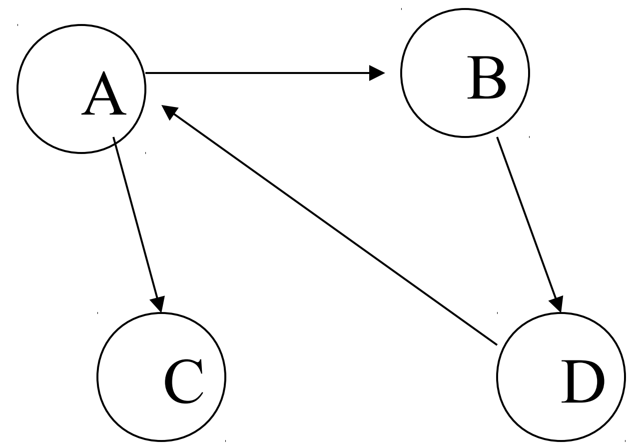
    (A,B),(B,D)

    ~~**(A,C),(C,B)**~~



Graph Concepts

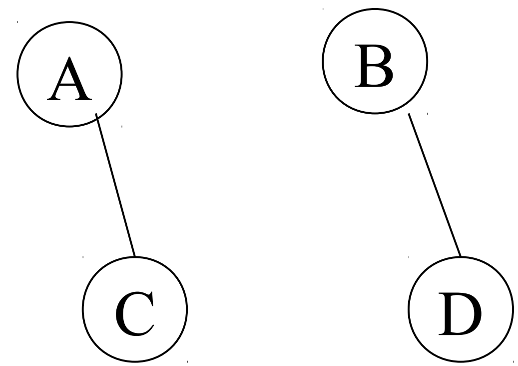
- **A cycle is a path from a node back to itself**
  - **(A, B)(B, D)(D, A)**
- **A graph with no cycles is called acyclic**



Graph Concepts

- **Connected Graph**

  For any two vertices X and Y there is a path from X to Y.



not connected

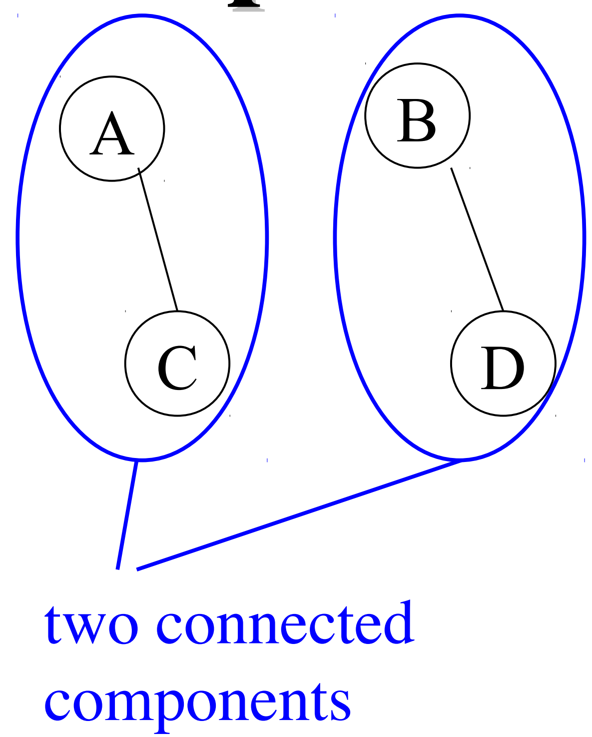
Graph Concepts

- **Connected Graph**

  For any two vertices X and Y there is a path from X to Y.

- **Connected Component**

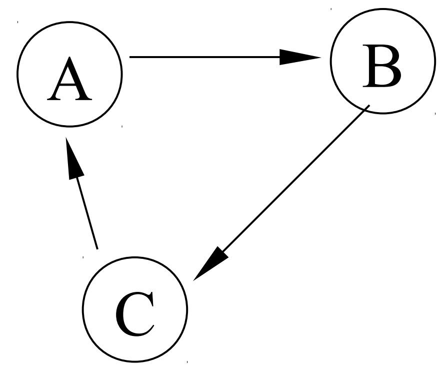
  A subset of vertices that is connected



Graph Concepts

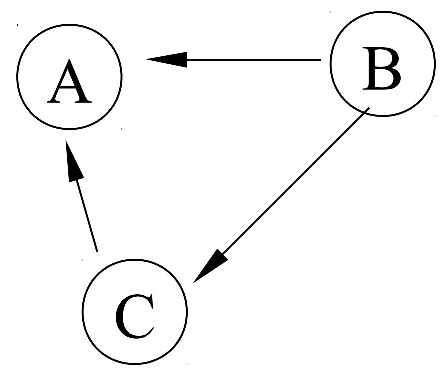
- **Strongly Connected Digraph**

  For any two vertices X and Y there is a path from X to Y. (Paths must follow arc directions)



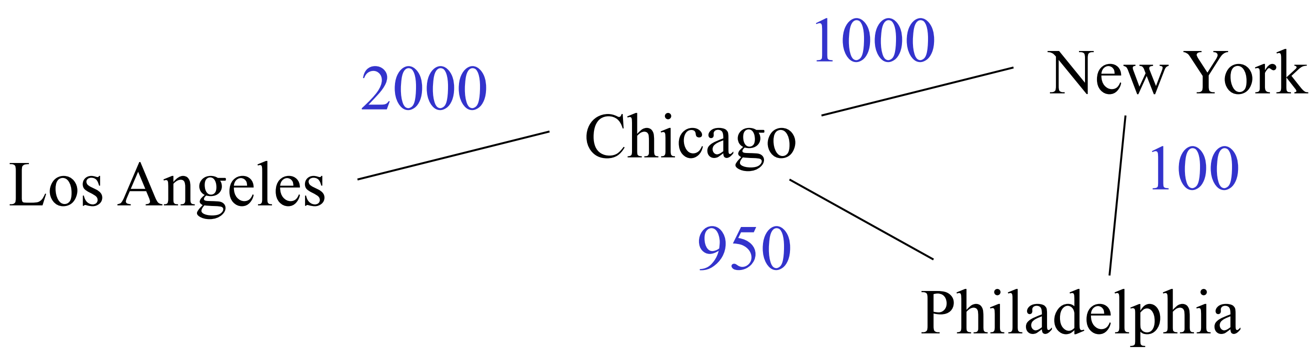
- **Weakly Connected Digraph**

  Corresponding graph is connected (i.e., ignoring arc direction)

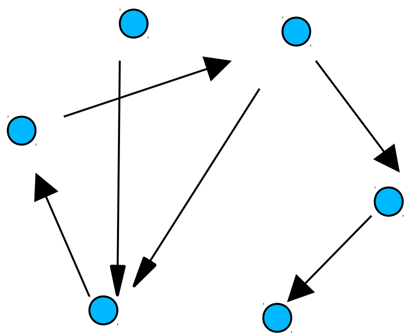


Graph Concepts

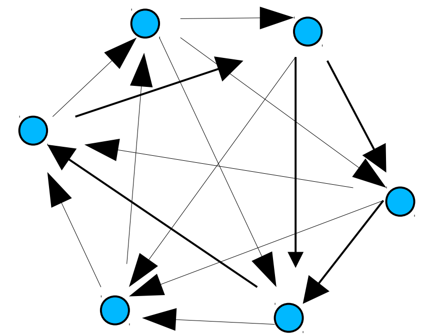
- **Weighted graph: each arc has a numerical weight**



Sparse vs Dense Graphs



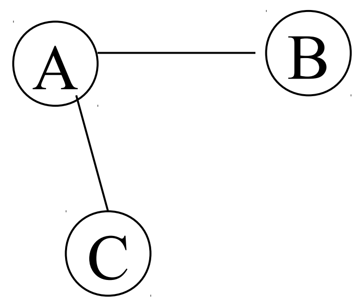
Sparse



Dense

Representing Graphs

- **Adjacency matrix**
  - **n x n boolean matrix: is there an arc?**

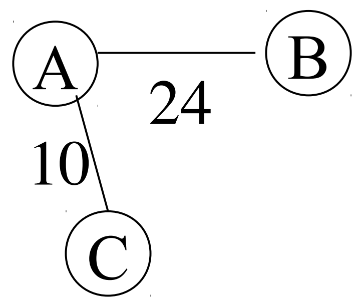


	name
0	**A**
1	**B**
2	**C**

	0	1	2
0		**T**	**T**
1	**T**		
2	**T**		

Representing Graphs

- **Adjacency matrix**
  - **n x n boolean matrix: is there an arc?**

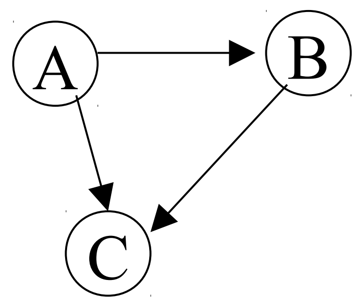


	name
0	A
1	B
2	C

	0	1	2
0	-1	24	10
1	24	-1	-1
2	10	-1	-1

Representing Graphs

- **Adjacency matrix**
  - n x n boolean matrix: is there an arc?

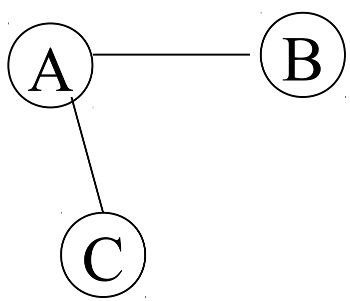


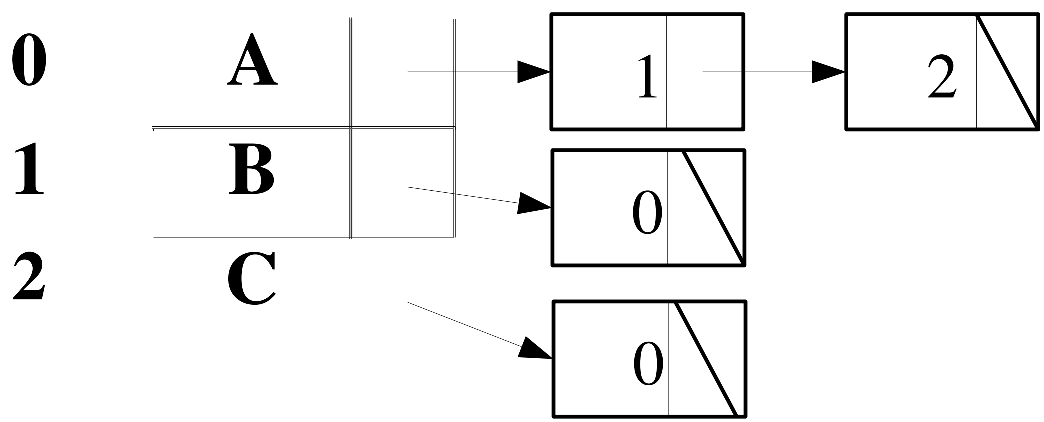
	name
0	A
1	B
2	C

	0	1	2
0		T	T
1			T
2			

Representing Graphs

- **Adjacency list**
  - **for each node, a linked list of edges that touch it**





Representing Graphs

- **Adjacency list**
  - **for each node, a linked list of edges that touch it**
  - **Space cost:**

    undirected v + 2 * 2 * e

    directed v + 2 * e

Costs, Worst case

	Space	Time: Is there and edge from i to j	Time: List the neighbors of i
Adjacency matrix	$O(v^2)$	O(1)	O(v)
Adjacency list	O(v + e)	O(d)	O(d)

d is degree (i.e., number of neighbors) of i

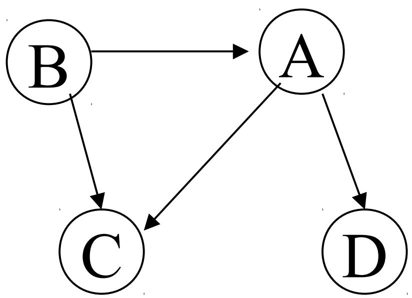
d<v

Graph Traversals

- Need to mark vertices as we see them to prevent infinite loops
- Need driver in case not connected
- Otherwise like tree traversals
- Depth first

```
dfsG(v):
    if (marked(v)) return;
    visit v;
    mark v;
    for each vn in neighbors(v)
      dfsG(vn)
```

Graph Traversals



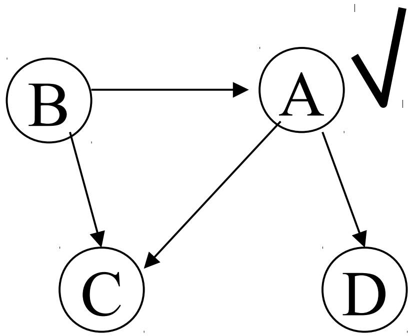
Driver

v = <A>

dfsG

v = <A>

Graph Traversals

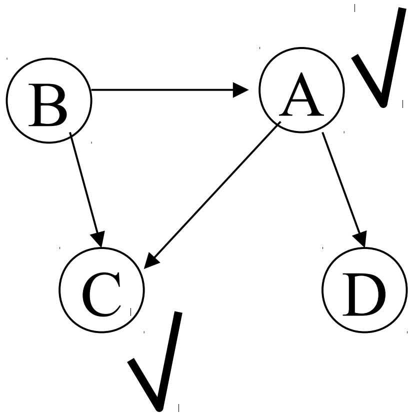


Driver
v = <A>

dfsG
v = <A>
vn = <C>

dfsG
v = <C>

Graph Traversals

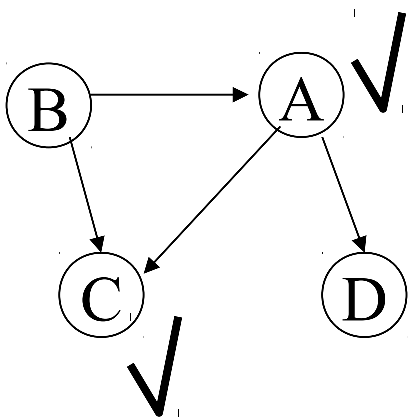


Driver
v = <A>

dfsG
v = <A>
vn = <C>

dfsG
v = <C>

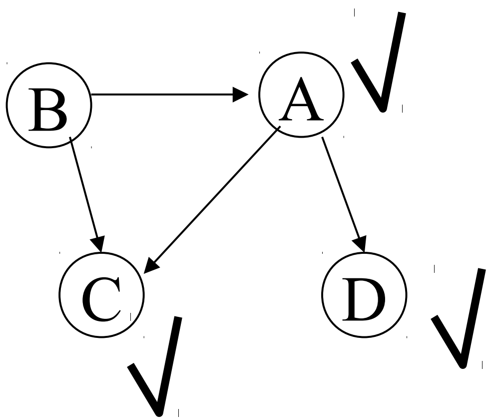
Graph Traversals



Driver
v = <A>

dfsG
v = <A>
vn = <D>

Graph Traversals

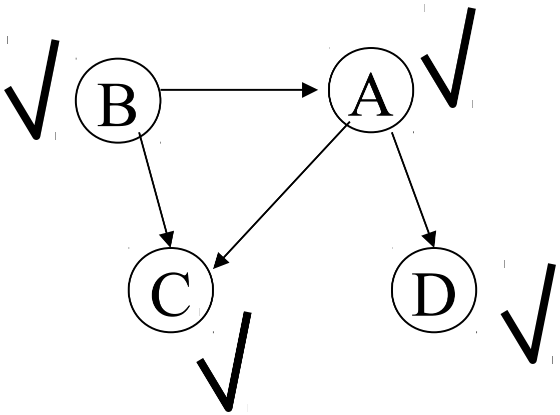


Driver
v = <A>

dfsG
v = <A>
vn = <D>

dfsG
v = <D>

Graph Traversals



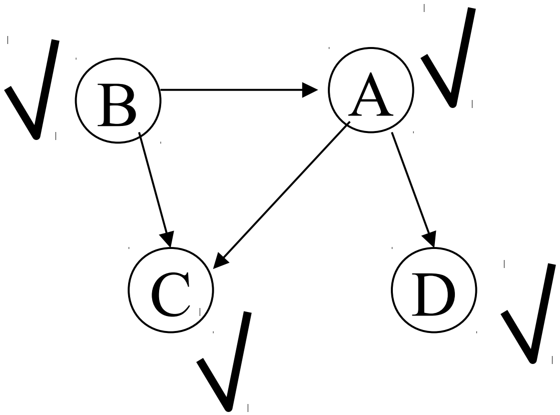
Driver

v = <B>

dfsG

v = <B>

Graph Traversals



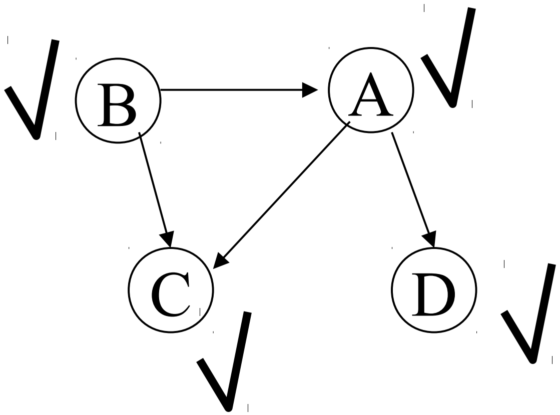
Driver

v = <C>

dfsG

v = <C>

Graph Traversals



Driver

v = <D>

dfsG

v = <D>

Graph Traversals

- **Time:**
  - **Visit each vertex**
  - **inspect each edge**

  O(n + e) n vertices, e edges

Uses of DFS Traversal

- Connected Components
  - See GraphCC.java
- Topsort
  - See GraphTS.java

New: Topological Sort

- **Acyclic Digraph <=> partial order**
- **Topsort: find total order consistent with partial order**

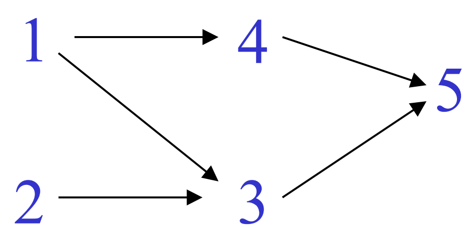
1 **a=1;**

2 **b=2;**

3 **c=a*b;**

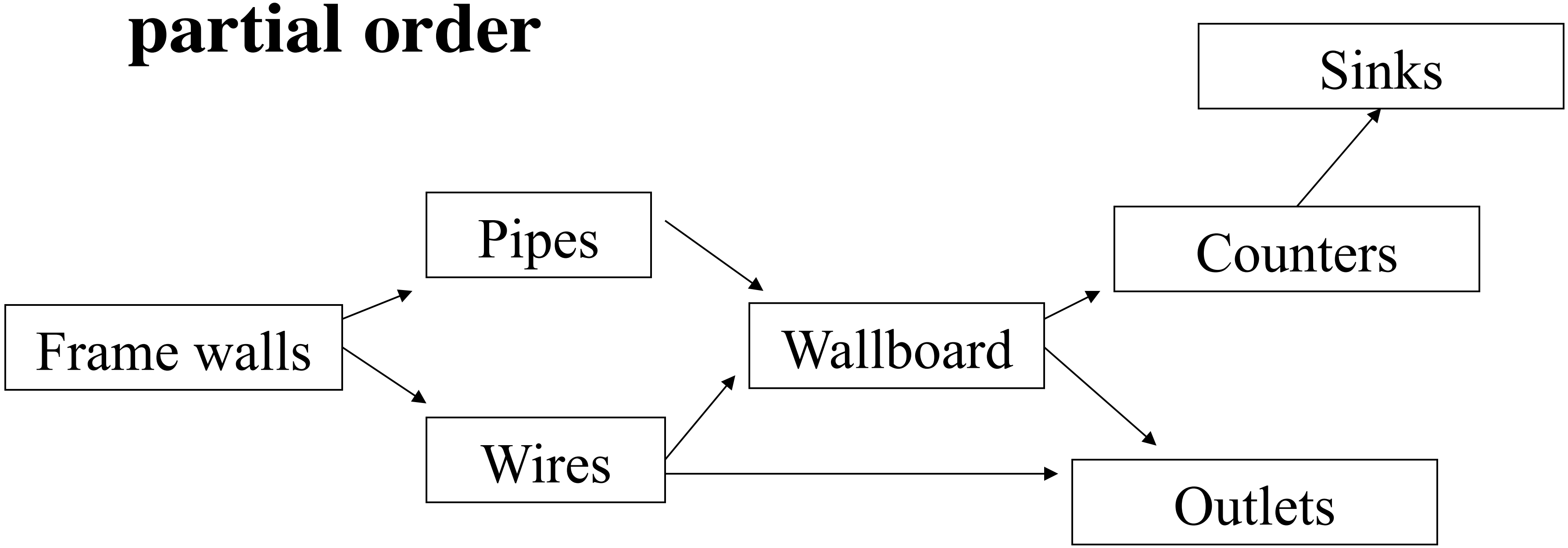
4 **d=a+4;**

5 **c=c+d**



Topological Sort

- **Acyclic Digraph <=> partial order**
- **Topsort: find total order consistent with partial order**



Topsort Algorithms

- **Most work by assigning numbers to vertices**
  - **vertex order = numerical order**
- **Depth first**
- **Breadth First**

DFS Topsort Algorithm

- **Algorithm:**
  - **Do DFS**
  - **Number vertices as you leave them**
- **Problem: leave vertex *after* leave reachable vertices, but needs number *smaller* than reachable vertices**
  - **Solution: number with decreasing numbers**
- **See GraphTS.java**