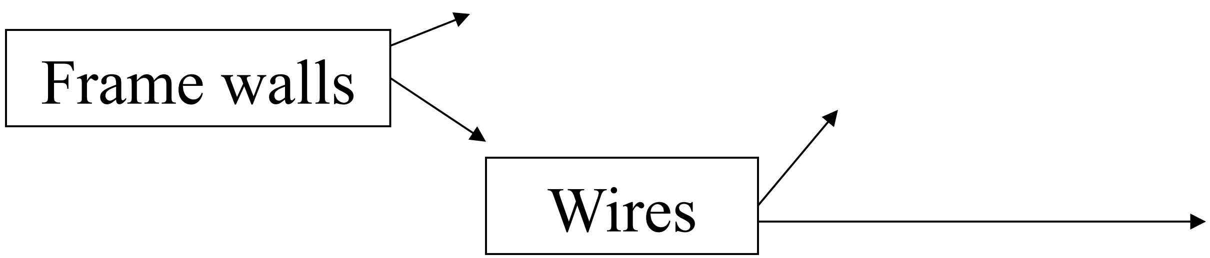
Computer Science 112
Data Structures

Lecture 23:
Shortest Path
Quicksort

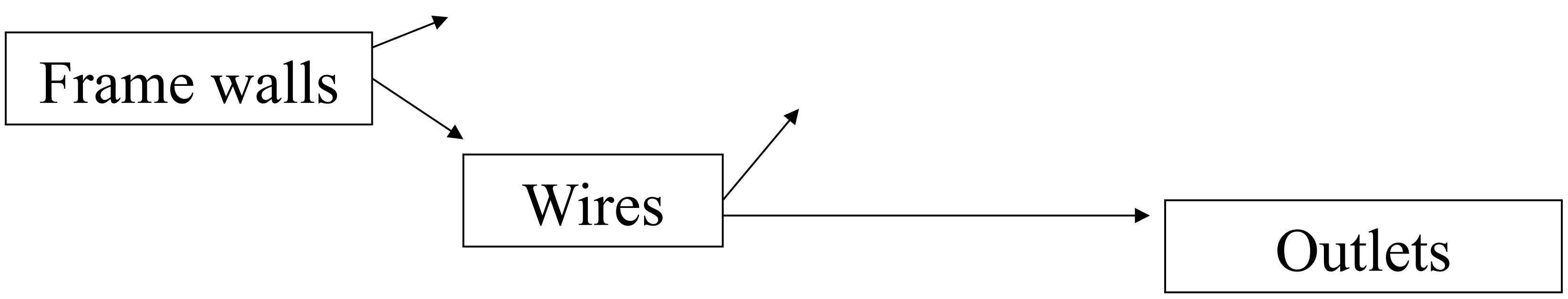
Review: Topsort Example

Frame walls

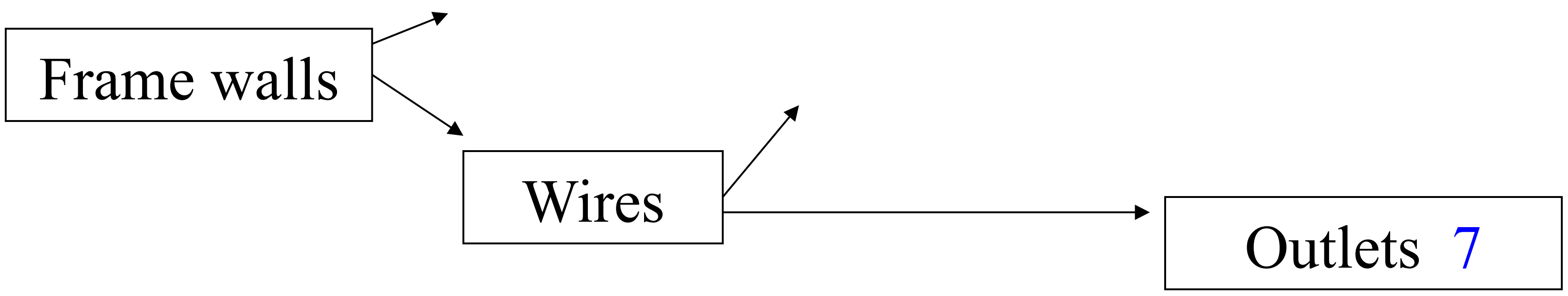
Topsort Example



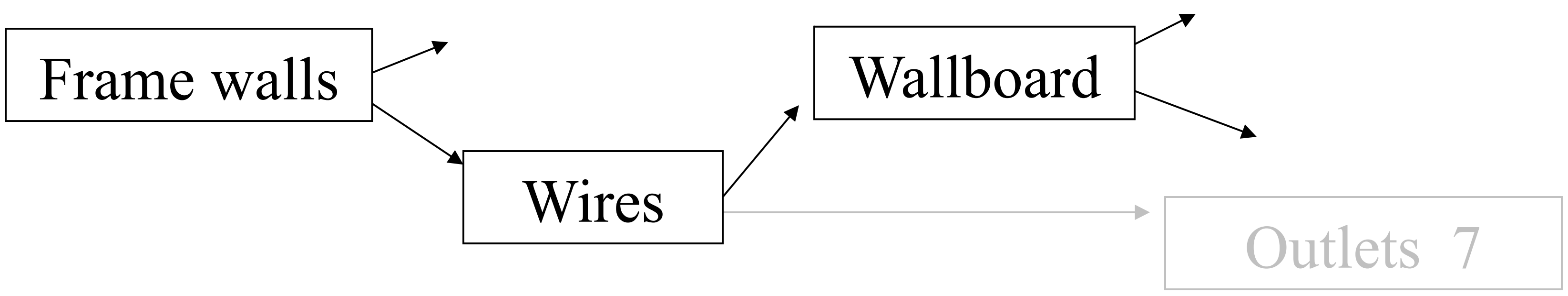
Topsort Example



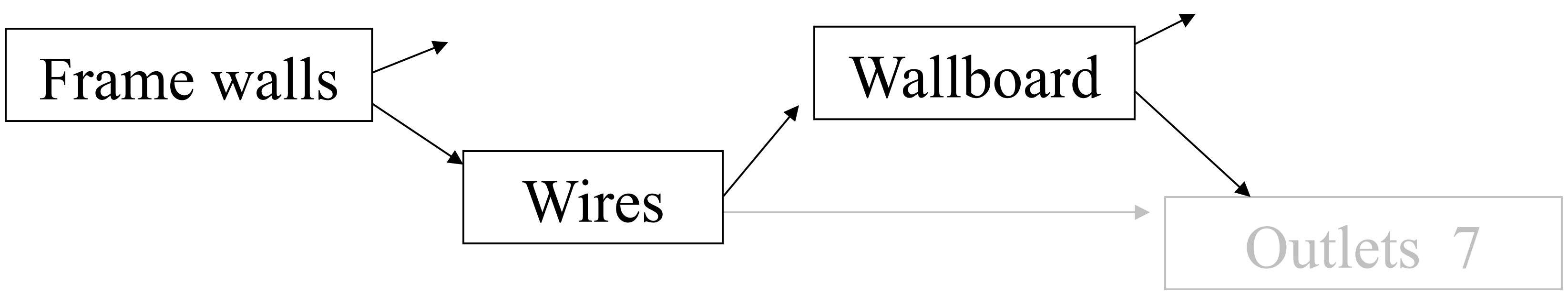
Topsort Example



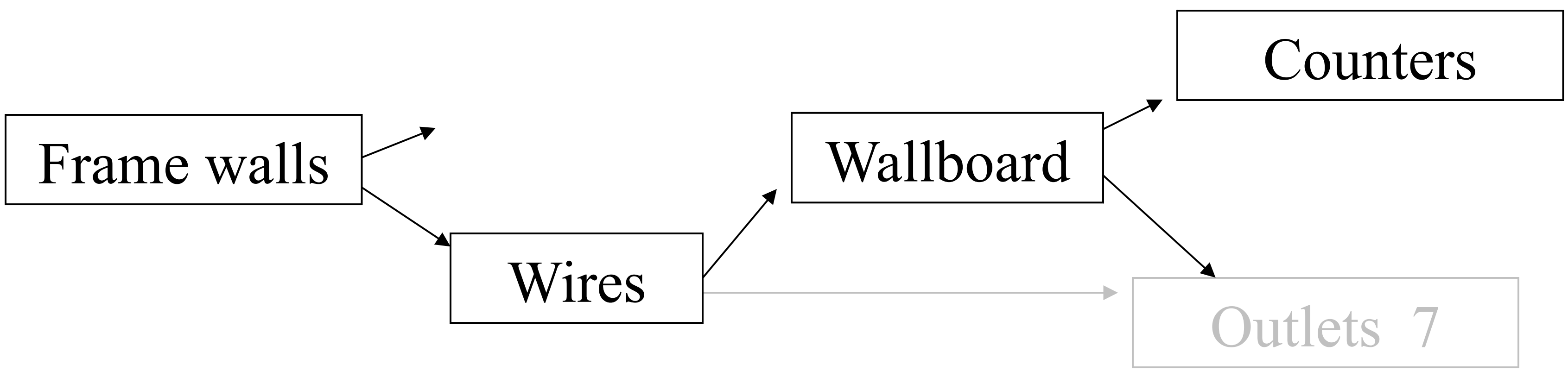
Topsort Example



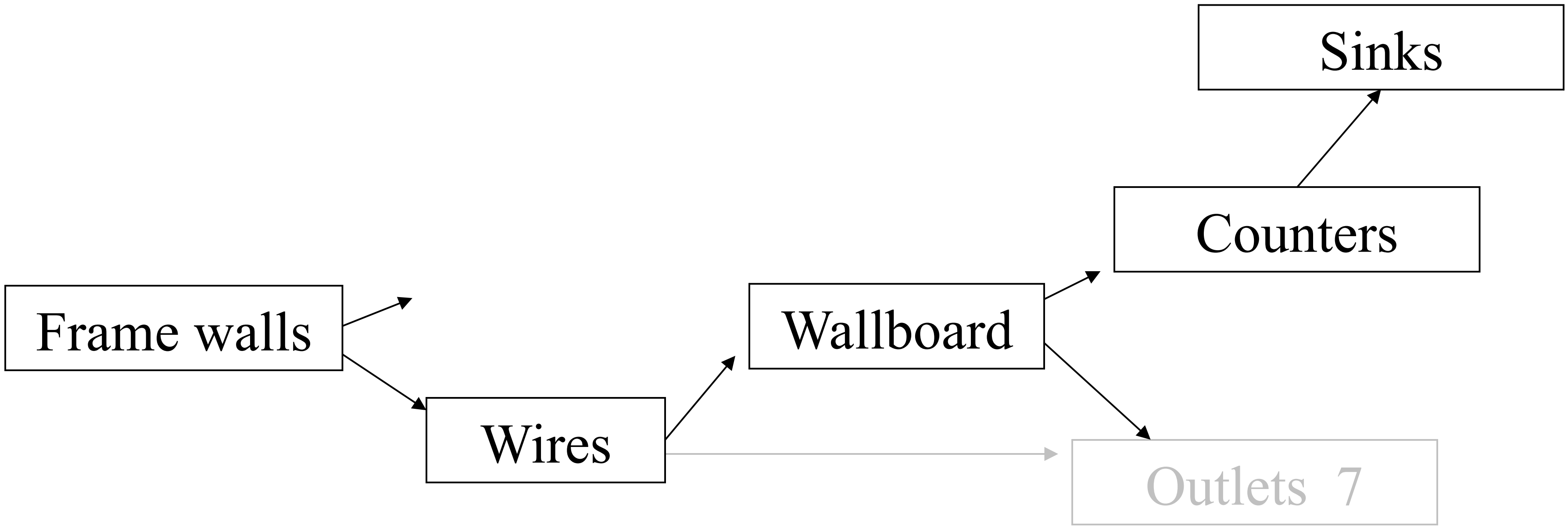
Topsort Example



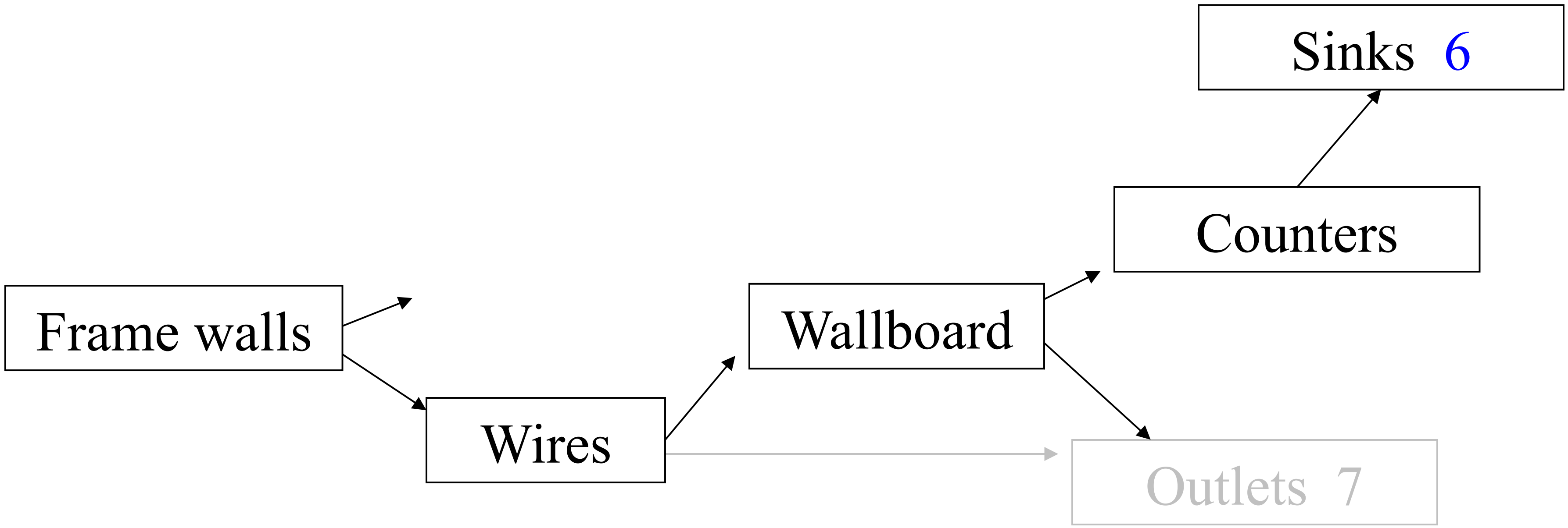
Topsort Example



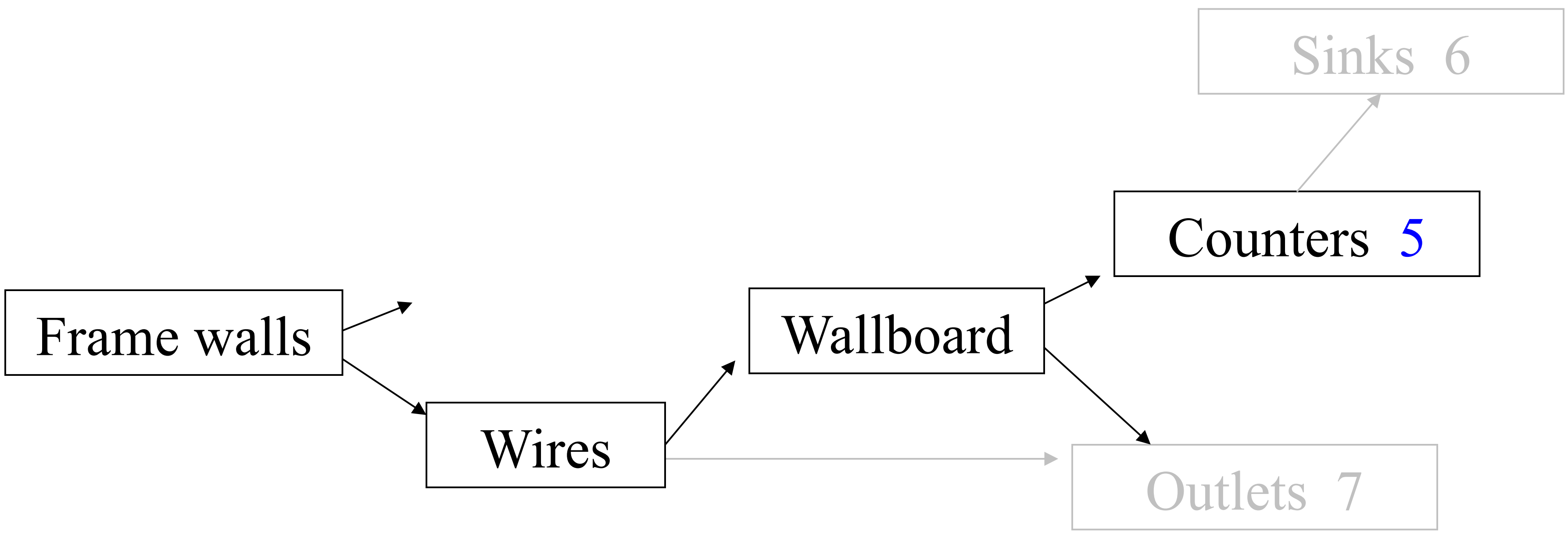
Topsort Example



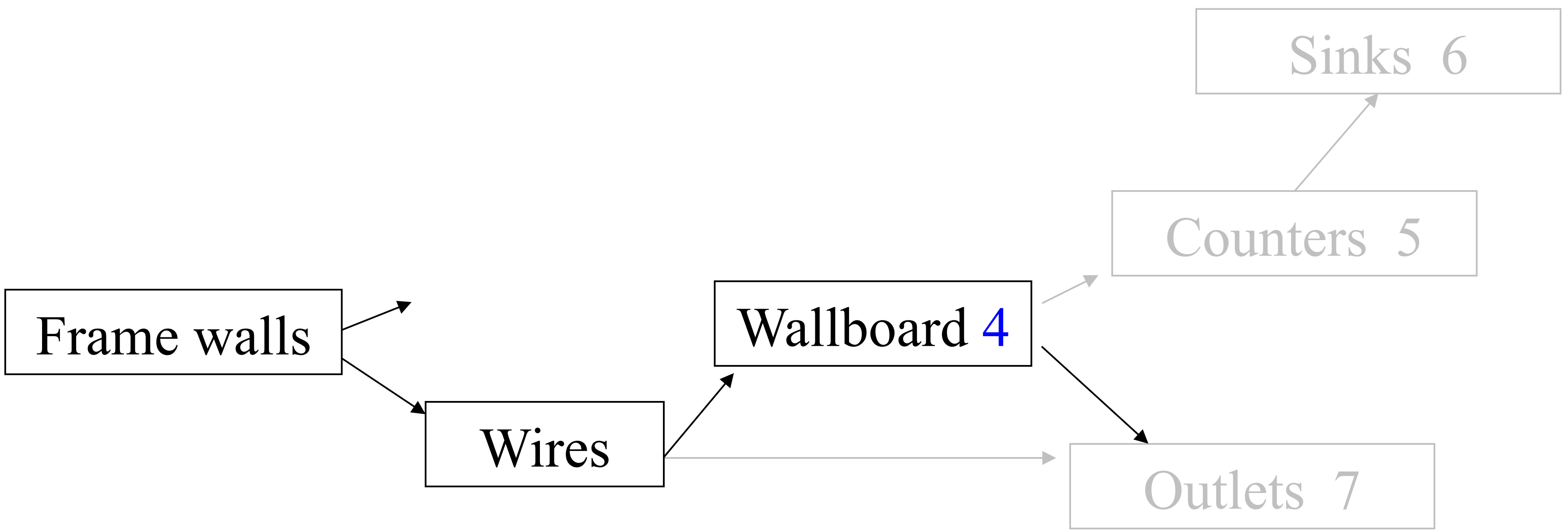
Topsort Example



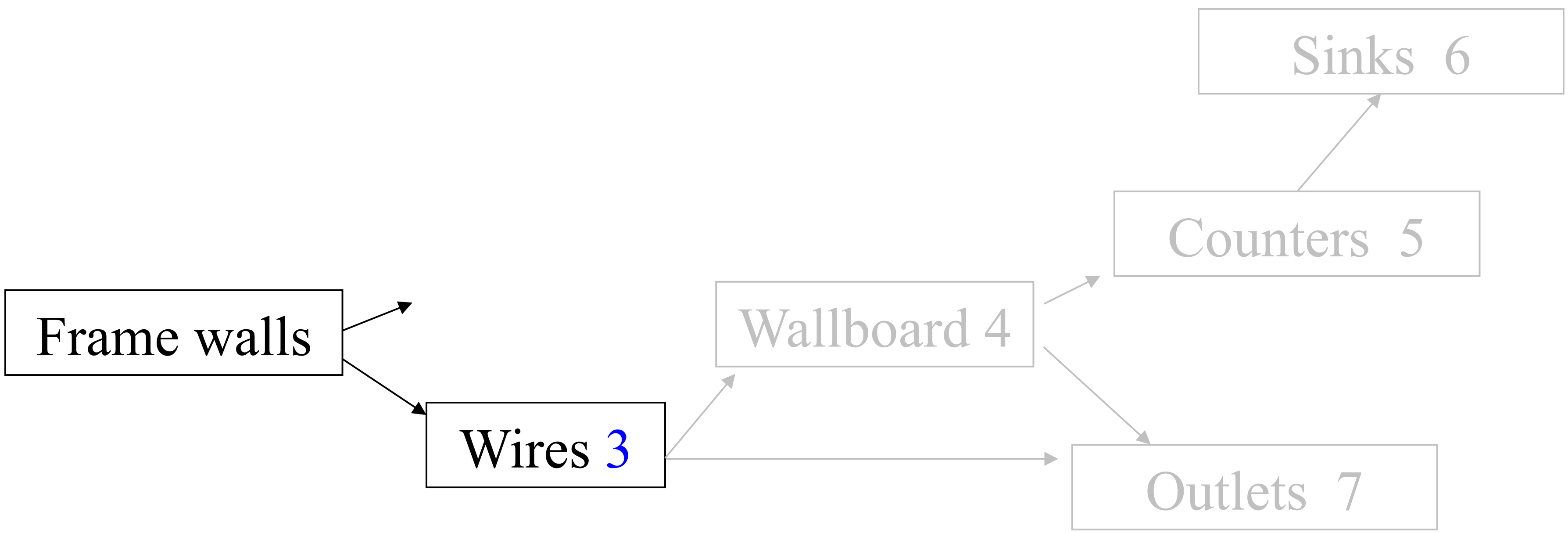
Topsort Example



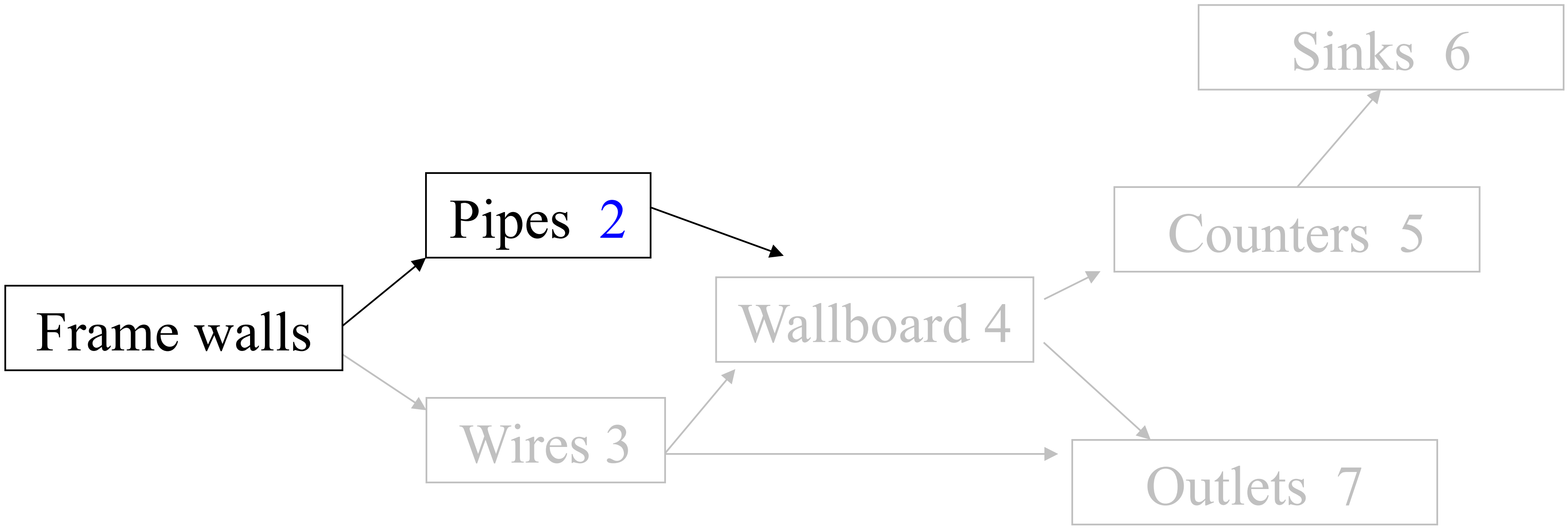
Topsort Example



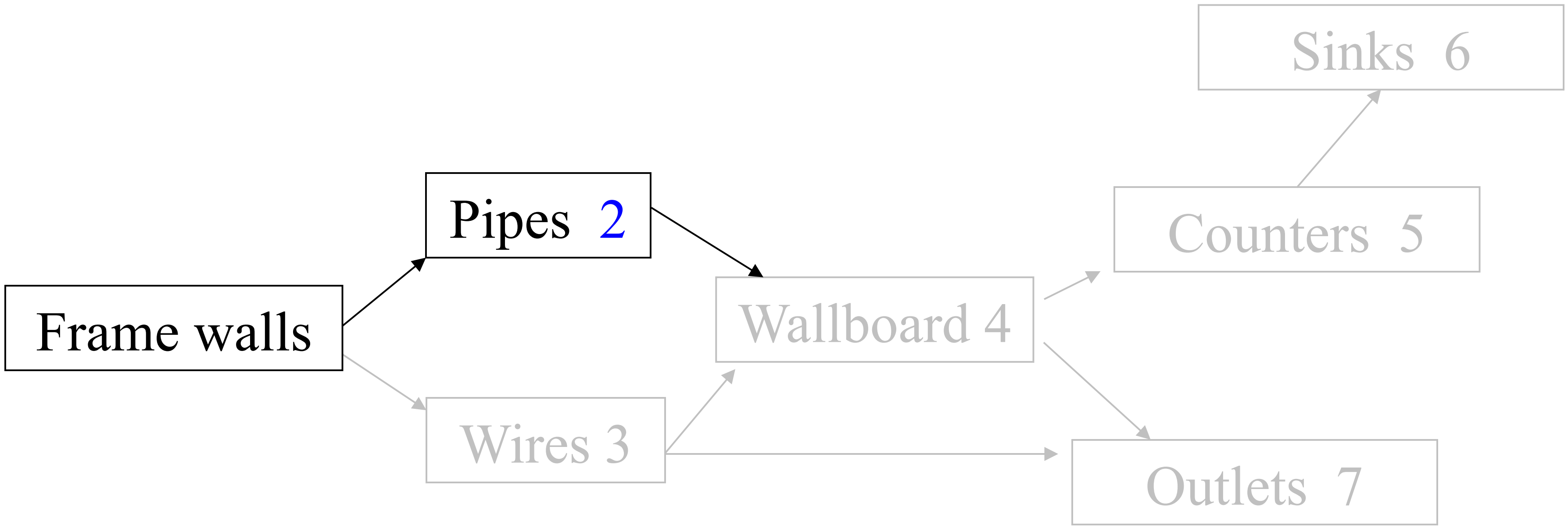
Topsort Example



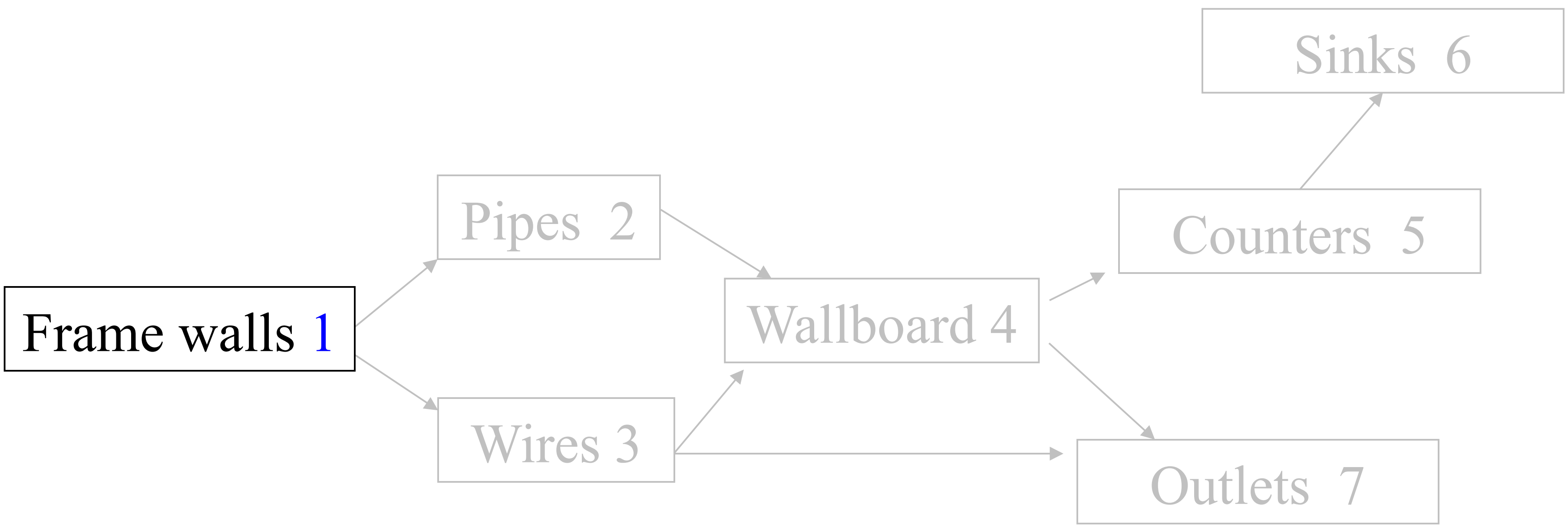
Topsort Example



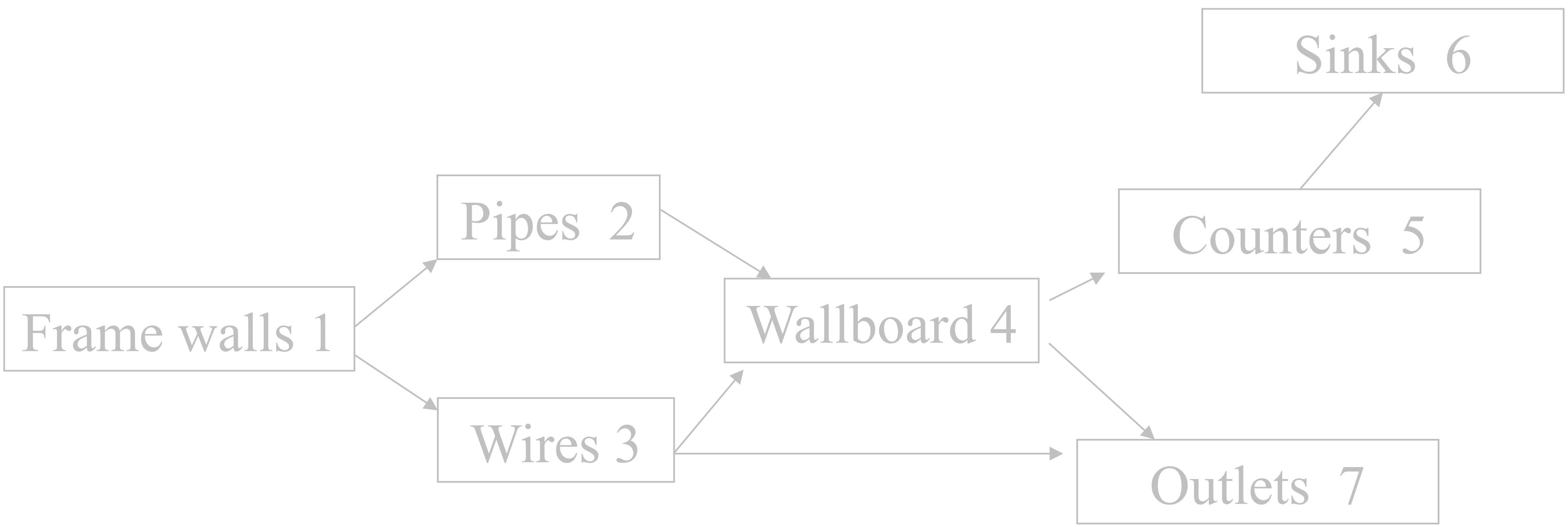
Topsort Example



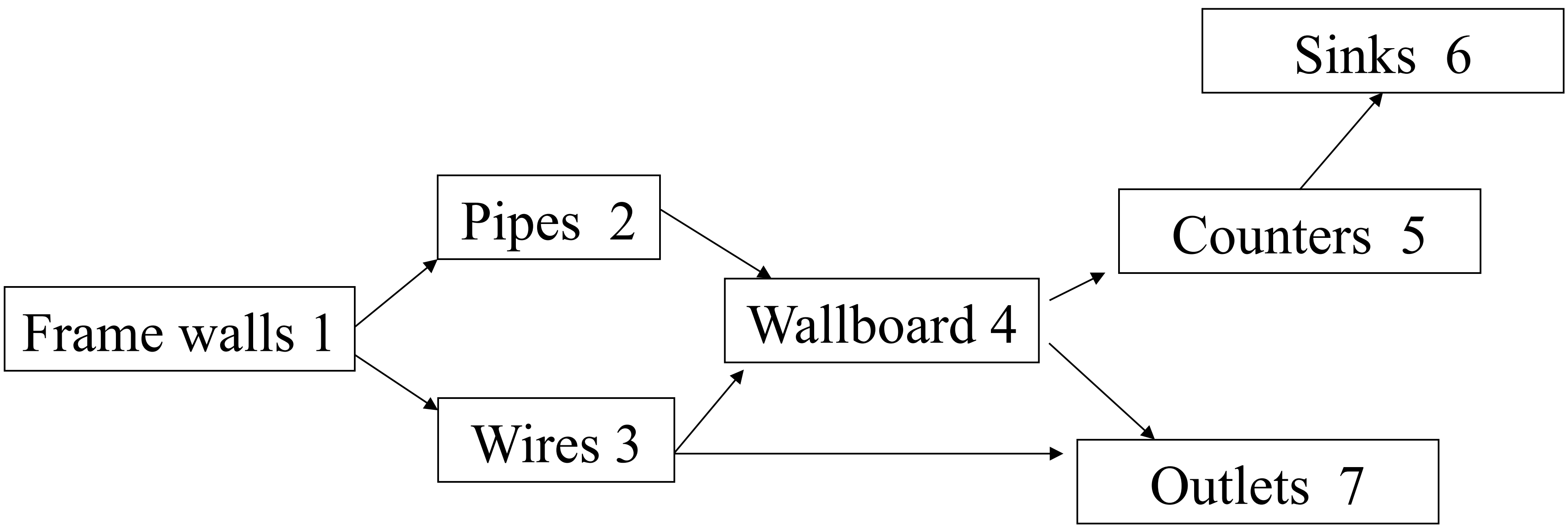
Topsort Example



Topsort Example



Topsort Example



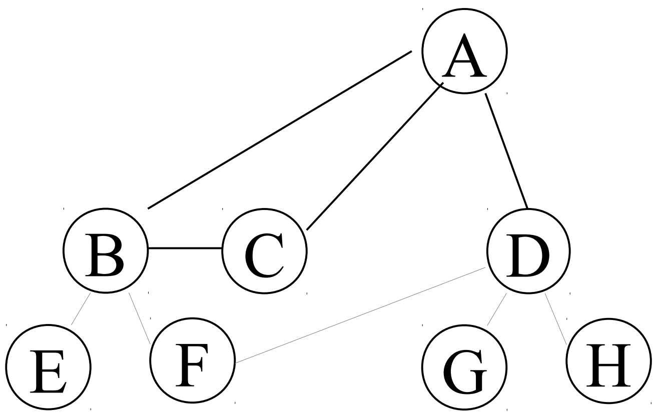
Topsort Cost

- **DFS + numbering**

  = O(n+e) + O(n) = O(n+e)

Review: Breadth First Search

- **Like breadth first search on tree**



- **Breadth First: A B C D E F G H**
- **Depth First: A B E F D G H C**

Breadth First Algorithm

- **bfsG(v):**

  visit and mark v

  enqueue v

  while not queue.empty( )

      dequeue into w

      for each neighbor n of w:

          if n not visited:

              visit and mark n

              enqueue n

Breadth First Cost

- **like depth-first:**
  - **visit every vertex**
  - **cross every edge**

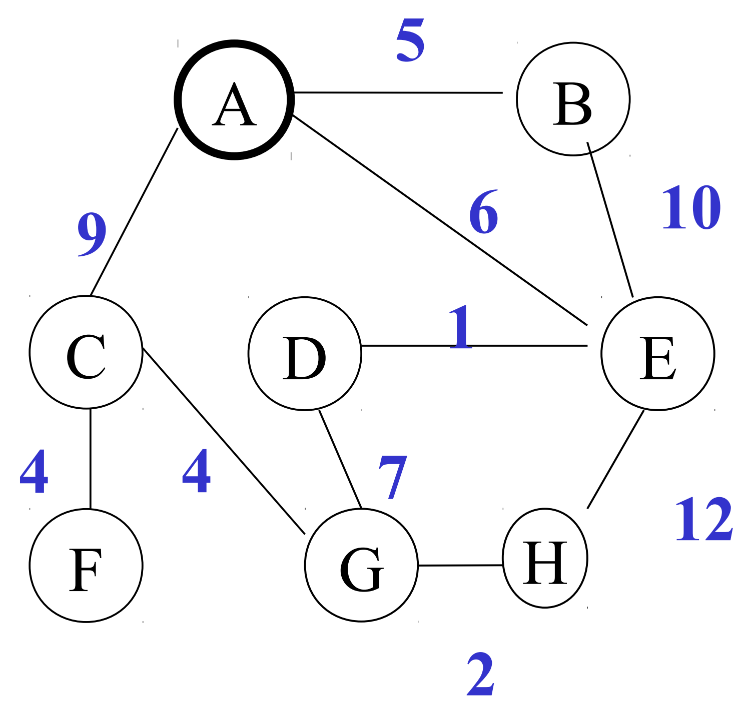
  O(n+e)

Review: Shortest Path

- **weighted graph**
  - **weights are all > 0**
- **“length” of a path = sum of weights of arcs on path**
- ~~**given start vertex, end vertex, find shortest path from start to end**~~
- **given start vertex, find shortest paths from start to each other vertex**

Shortest Paths

- How long is the path
  - AEH
  - AEDGH
- What is the shortest path from A to H?

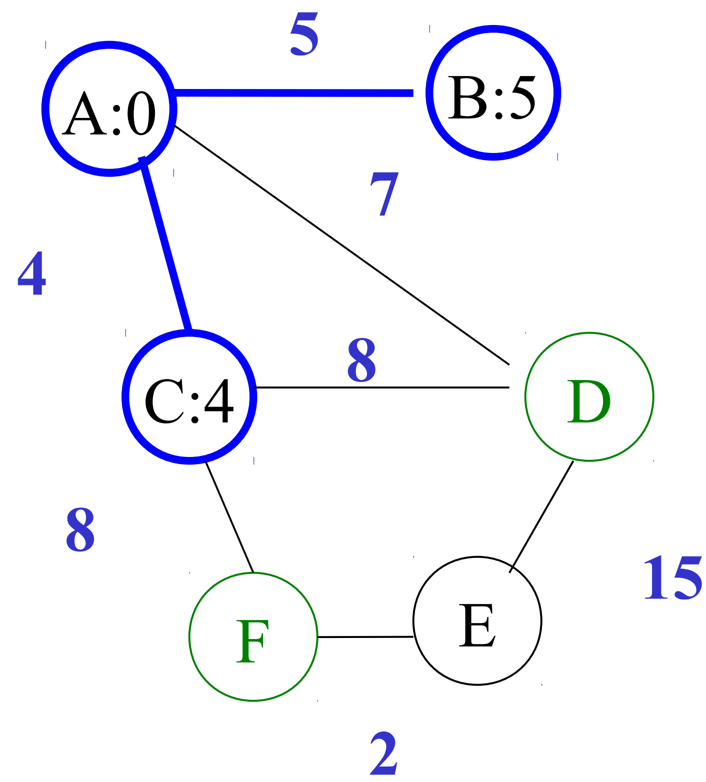


Dijkstra's Algorithm

- **Tree: nodes for which you know the shortest path from start, and the arcs on those paths**
- **Fringe: nodes that are not in the tree yet but have a neighbor in the tree**

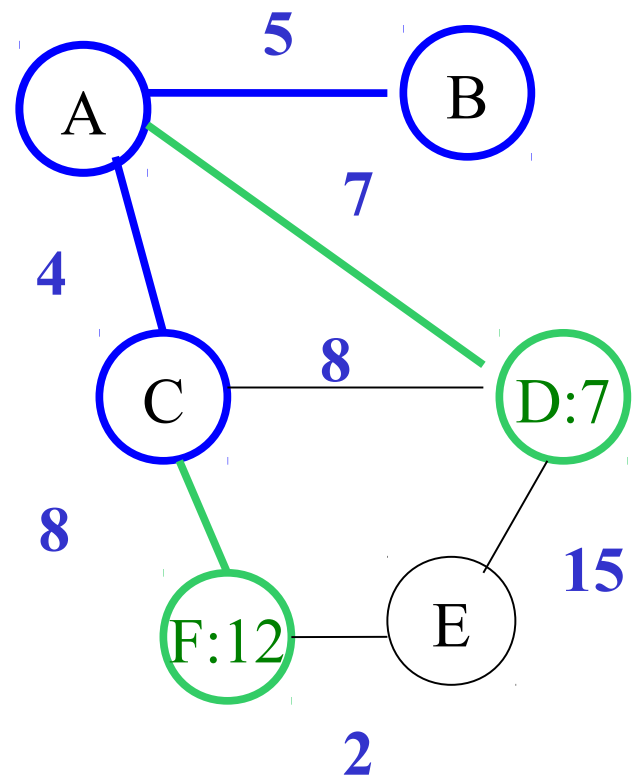
Dijkstra's Algorithm

- Vertices in the **tree** have
  - a link: last step on the shortest path to vertex from start
  - a distance: the length of that whole path



Dijkstra's Algorithm

- Vertices in the fringe have
  - a link: an arc from the tree if > 1 of these, use the arc that gives the shortest path back to start
  - a distance: the length of the path using link



Algorithm:

Put start vertex in the tree, update neighbors

While there are any vertices in fringe

Let v be vertex in fringe with

smallest distance-from-start.

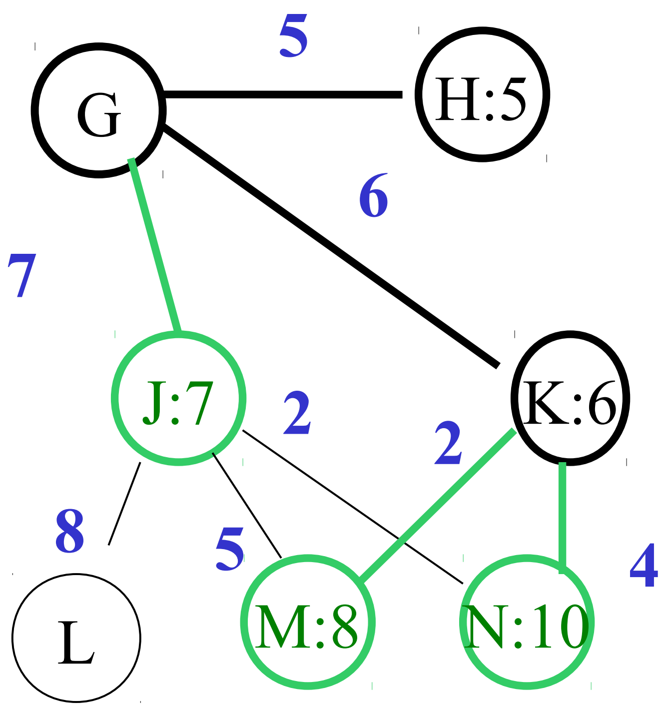
Put v in the tree.

Update neighbors

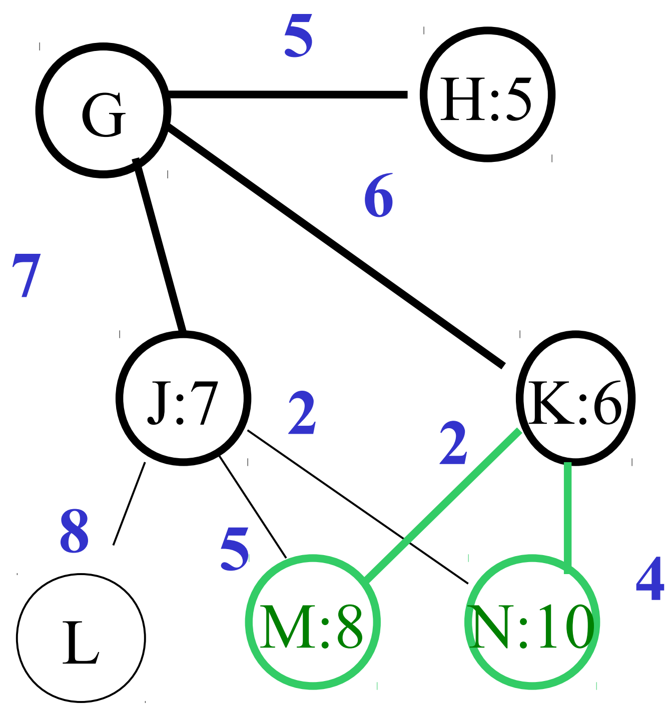
Update neighbors

- **Neighbor of v that is not in tree and not in fringe gets added to fringe with link v**
- **Neighbor of v that is in the fringe gets checked: would changing link to be v result in a smaller distance? If so, change link and distance**
- **Neighbor of v that is in tree is not changed**

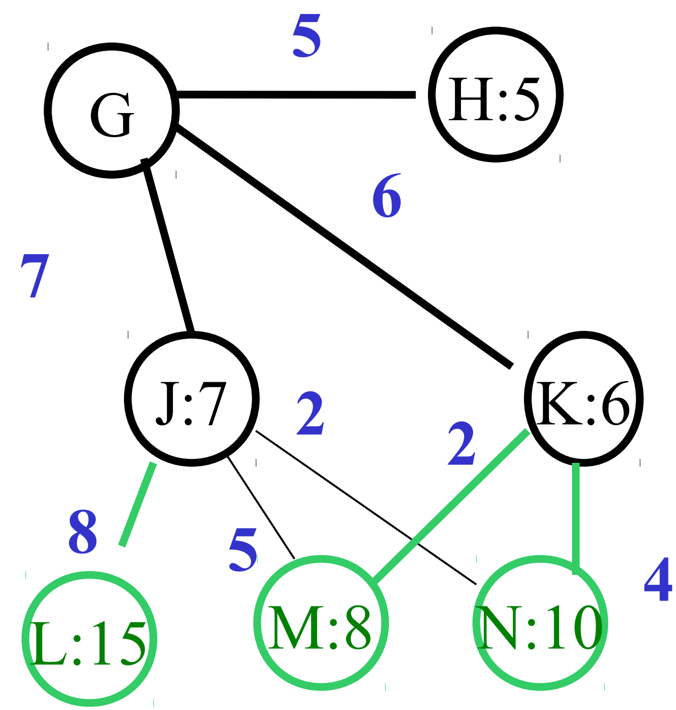
J → tree, Update neighbors



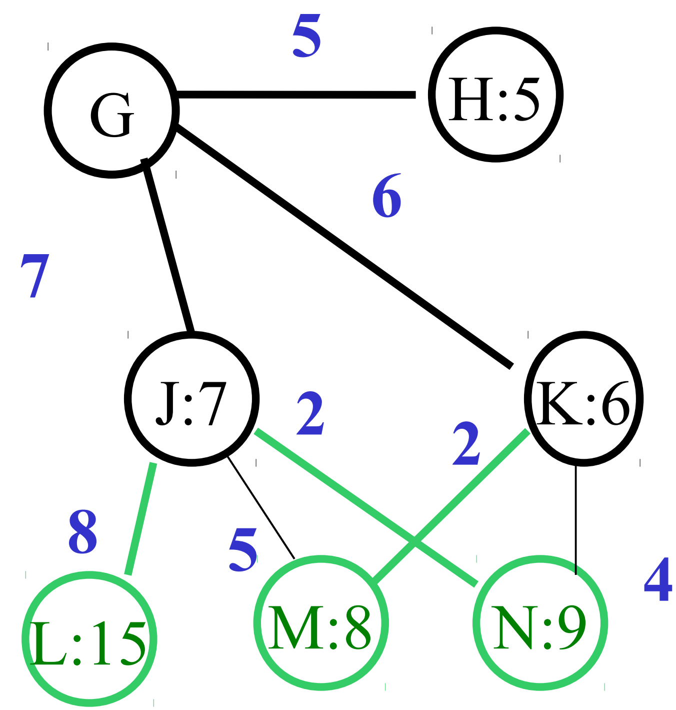
J → tree



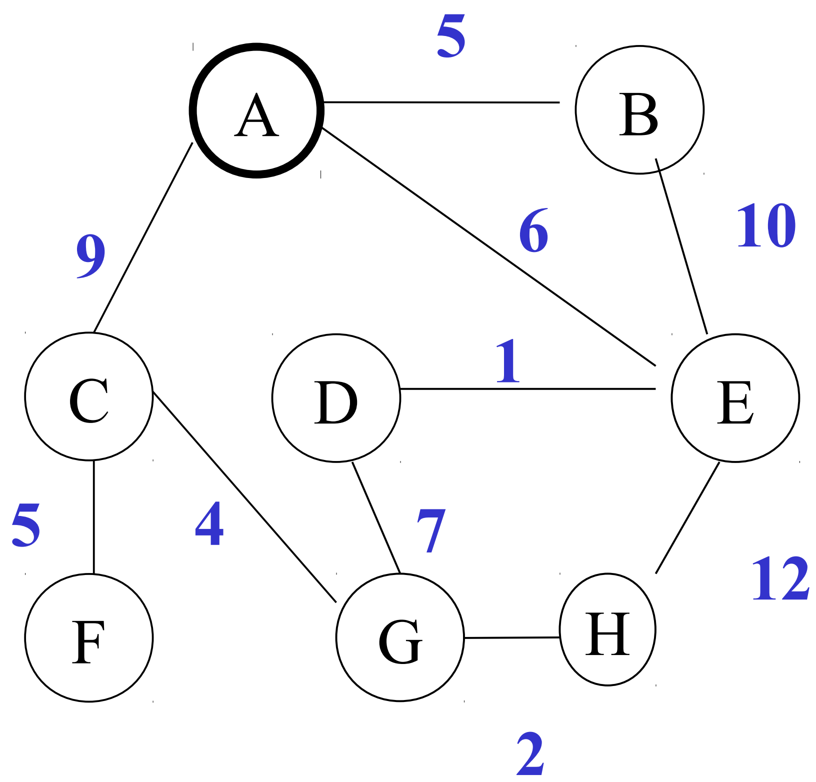
Update neighbors: neither tree nor fringe → fringe



Update neighbors: Fringe: check links



Example



Example

Tree	to B	to C	to D	to E	to F	to G	to H
A	<u>5:A</u>	9:A	∞	6:A	∞	∞	∞

Example

Tree	to B	to C	to D	to E	to F	to G	to H
A	<u>5:A</u>	9:A	∞	6:A	∞	∞	∞
B: 5A		9:A	∞	<u>6:A</u>	∞	∞	∞

Example

Tree	to B	to C	to D	to E	to F	to G	to H
A	5:A	9:A	∞	6:A	∞	∞	∞
B: 5A		9:A	∞	6:A	∞	∞	∞
E: 6A	`	9:A	7:E		∞	∞	18:E

Example

Tree	to B	to C	to D	to E	to F	to G	to H
A	<u>5:A</u>	9:A	∞	6:A	∞	∞	∞
B: 5A		9:A	∞	<u>6:A</u>	∞	∞	∞
E: 6A		9:A	<u>7:E</u>		∞	∞	18:E
D:7E		<u>9:A</u>			∞	14:D	18:E

Example

Tree	to B	to C	to D	to E	to F	to G	to H
A	<u>5:A</u>	9:A	∞	6:A	∞	∞	∞
B: 5A		9:A	∞	<u>6:A</u>	∞	∞	∞
E: 6A		9:A	<u>7:E</u>		∞	∞	18:E
D:7E		<u>9:A</u>			∞	14:D	18:E
C:9A					14:C	<u>13:C</u>	18:E

Example

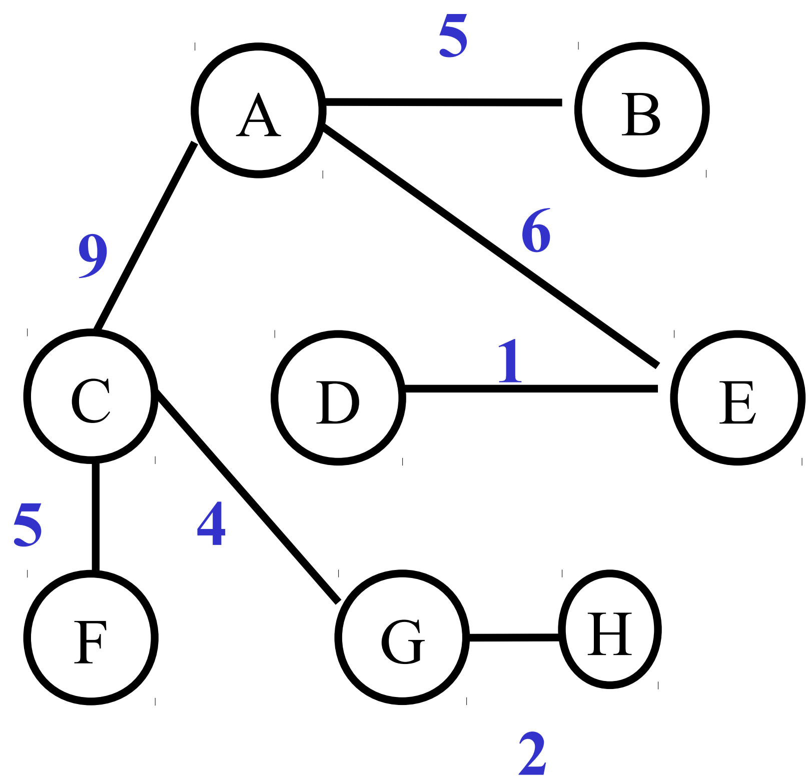
Tree	to B	to C	to D	to E	to F	to G	to H
A	<u>5:A</u>	9:A	∞	6:A	∞	∞	∞
B: 5A		9:A	∞	<u>6:A</u>	∞	∞	∞
E: 6A		9:A	<u>7:E</u>		∞	∞	18:E
D:7E		<u>9:A</u>			∞	14:D	18:E
C:9A					14:C	<u>13:C</u>	18:E
G:13C					<u>14:C</u>		15:G

Example

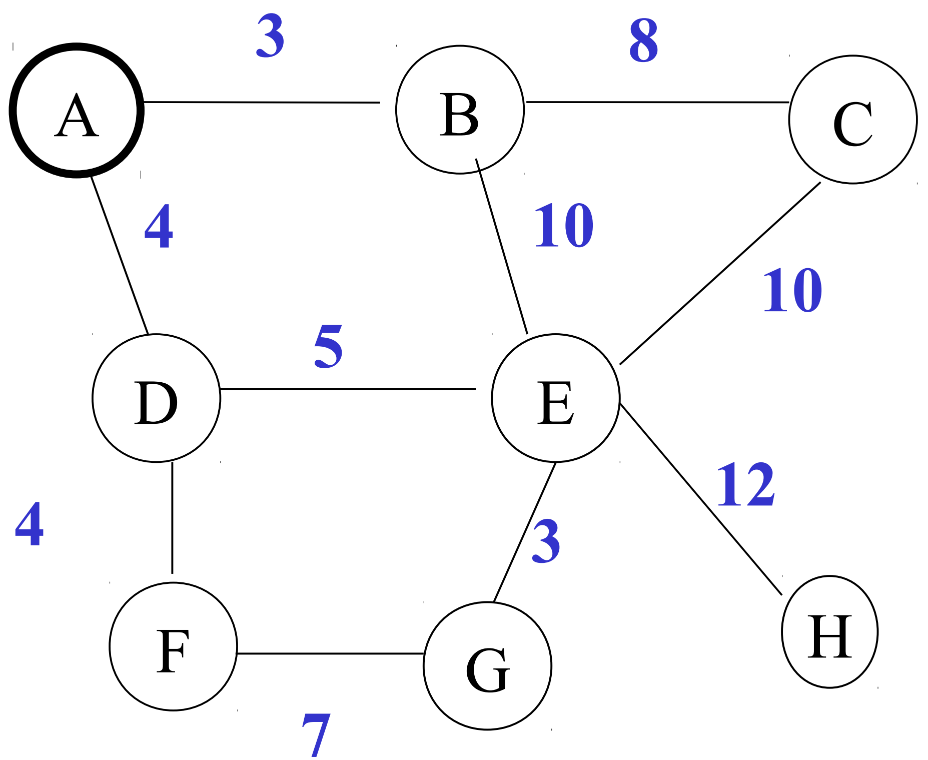
Tree	to B	to C	to D	to E	to F	to G	to H
A	<u>5:A</u>	9:A	∞	6:A	∞	∞	∞
B: 5A		9:A	∞	<u>6:A</u>	∞	∞	∞
E: 6A		9:A	<u>7:E</u>		∞	∞	18:E
D:7E		<u>9:A</u>			∞	14:D	18:E
C:9A					14:C	<u>13:C</u>	18:E
G:13C					<u>14:C</u>		15:G
F: 14C							<u>15:G</u>

Example

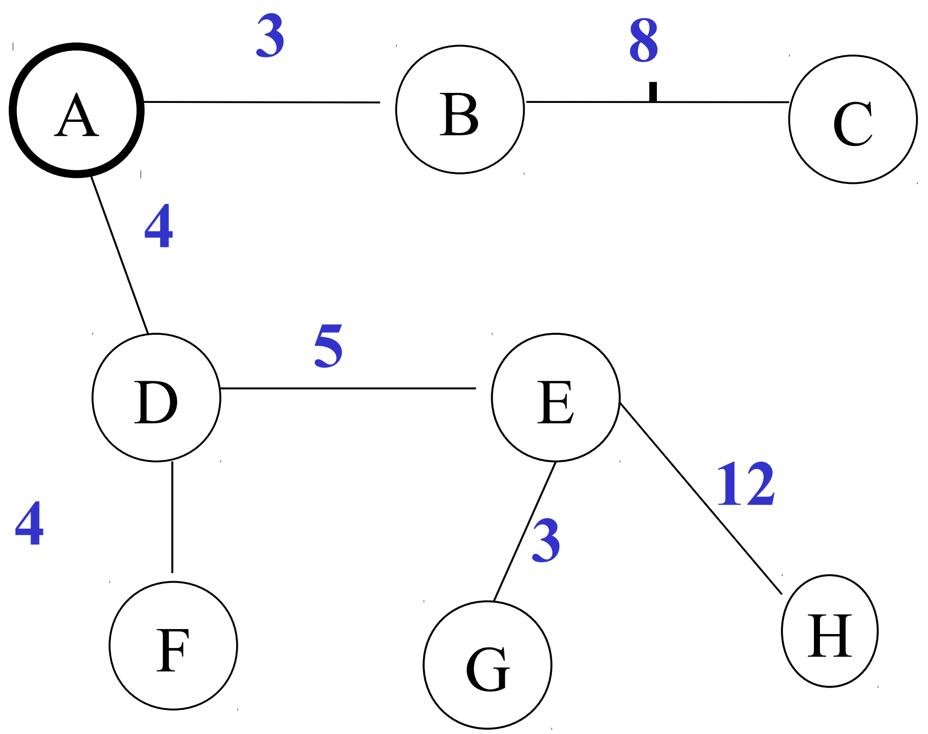
Tree	to B	to C	to D	to E	to F	to G	to H
A	<u>5:A</u>	9:A	∞	6:A	∞	∞	∞
B: 5A		9:A	∞	<u>6:A</u>	∞	∞	∞
E: 6A		9:A	<u>7:E</u>		∞	∞	18:E
D:7E		<u>9:A</u>			∞	14:D	18:E
C:9A					14:C	<u>13:C</u>	18:E
G:13C					<u>14:C</u>		15:G
F: 14C							<u>15:G</u>
H:15G							


5
A
B
6
9
C
D
1
E
5
4
F
G
H
2

Another Example



Result



New: Cost of Dijkstra's Algorithm

What are the operations to consider?

- **Picking the min-distance vertices from the fringe**
- **Adding vertices to the Tree**
- **Updating neighbors**
  - **Adding vertices to the Fringe**
  - **Updating links when needed**

Data Structures

- **For graph:**
  - adjacency matrix
  - adjacency list
- **For fringe:**
  - unordered linked list
  - ordered linked list
  - min heap

plus tree/fringe/neither marked on node

Matrix as Adjacency list, Fringe as Unordered linked list

- **Picking the min-distance vertices from the fringe**
  - **Worst case fringe is all vertices not in tree**

    $(n-1) + (n-2) + \ldots + 1 = O(n^2)$
- **Adding vertices to the Tree**

  $O(n + e)$
- **Updating neighbors**
  - **Adding vertices to the Fringe: $O(n)$**
  - **Checking and Updating links: $O(n+e)$**
- **Total: $O(n^2) + O(n) + O(n) + O(n+e) = O(n^2)$**

Matrix as Adjacency list, Fringe as min-heap

- Picking the min-distance vertices from the fringe
  - Worst case fringe is all vertices not in tree

    $\log(n-1) + \log(n-2) + \ldots + \log(1) = O(n \log n)$
- Adding vertices to the Tree

  $O(n + e)$
- Updating neighbors
  - Adding vertices to the Fringe: $O(n \log n)$
  - Checking and Updating links: $O(n + e \log n)$
- Total:

  $O(n \log n) + O(n \log n) + O(n) + O(n \log n) + O(n + e \log n)$
  $= O((n+e) \log n)$

New: Quicksort1

- **Quicksort:**
  - **Partition**
    - **Split data into two groups, all in one group < any in other group**
  - **sort groups separately**
    - **use quicksort recursively**
  - **append**
    - **if partition & sort are in-place there is nothing to do here**

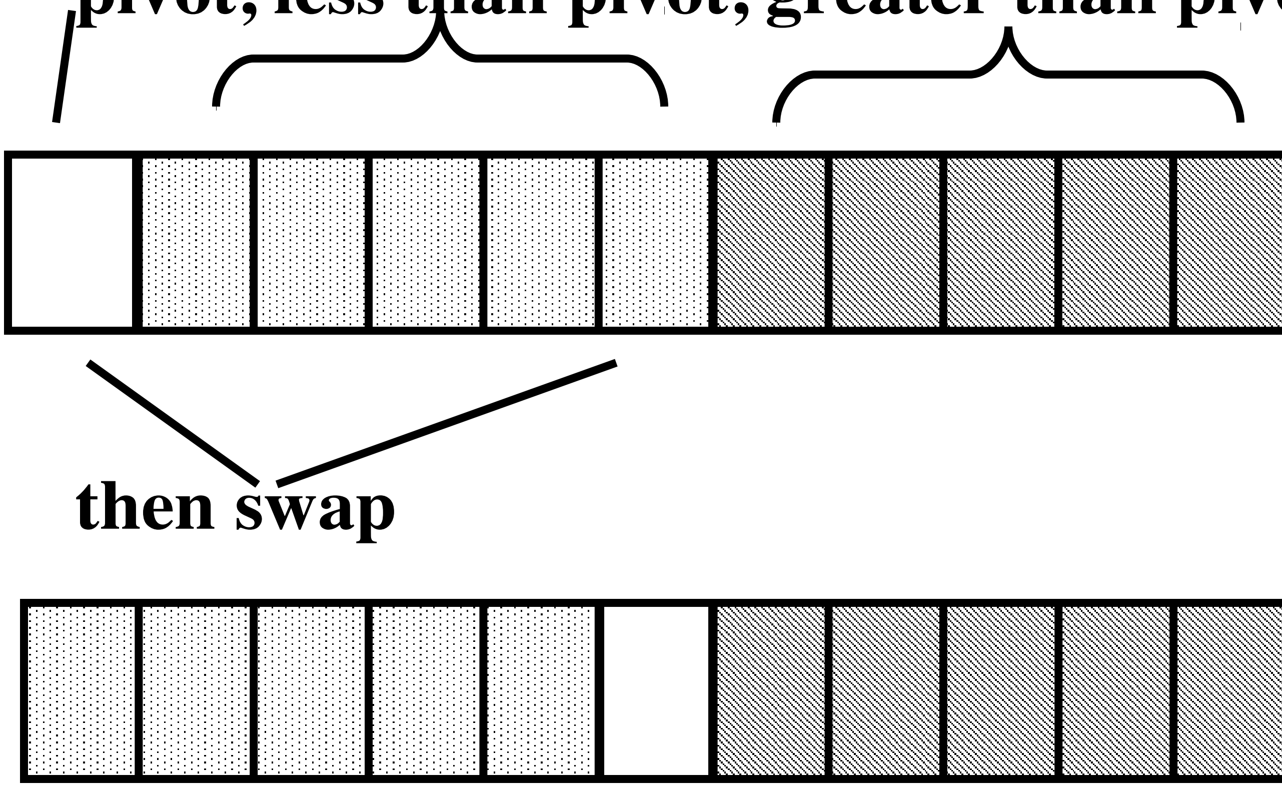
Partition

- Choose a “pivot” value from data
  - ideal would be median => equal size lists
  - but takes too long to find median
  - simplest: pivot = first
    - but in order -> worst case
  - safer: median of 1st, last, middle

Partition

- **Trick: first partition like this**

pivot, less than pivot, greater than pivot



Partition

- Use 2 pointers: left and right
  - move left from low+1 up until A[left] > pivot
  - move right from high down until A[right]<pivot
  - Swap numbers in A[left] and A[right]
  - Repeat until left>=right

Partition

1 12 5 11 3 7 6 8 15 9 10 13 2 4 14

8 12 5 11 3 7 6 1 15 9 10 13 2 4 14

8 4 5 11 3 7 6 1 15 9 10 13 2 12 14

8 4 5 2 3 7 6 1 15 9 10 13 11 12 14

1 4 5 2 3 7 6 8 15 9 10 13 11 12 14

Quicksort

- **How sort regions left & right of pivot?**
  - **Quicksort! (unless < 3 numbers in region)**
    - **Actually, insertion sort faster for small regions**
      - **size<10 or so**

Quick Sort

Unsorted:	2	5	1	8	3	9	4
pivot=4							
Partition:	2	1	3	4	5	9	8
Sort Groups:	1	2	3	4	5	8	9
Result:	1	2	3	4	5	8	9

Quick Sort

1 12 5 11 3 7 6 8 15 9 10 13 2 4 14

1 5 3 7 6 2 4   **8**   12 11 15 9 10 13 14

1 2 3   **4**   5 7 6   **8**   9 11 10   **12**   15 13 14

1 **2** 3 **4** 5 **6** 7 **8** 9 **10** 11 **12** 13 **14** 15

Complexity

- Partition takes O(n) time where n is the number of numbers to partition
- Best case: assume partition always into equal halves
  - Suppose 15 numbers in array
  - partition 0 - 14   15 compares
  - partition 0-6, 8-14   7+7=14 compares

    …   … always O(n) compares
- Each level divides partition size by 2, stop at size 1
  - log n levels
- Total: O(n log n)

Complexity

- **Worst case: always divide into 0 and all-but pivot**
  - **15 -> 14 -> 13 -> … 1: O(n) levels, total O(n^2)**
- **Average case: O(n logn) like best**

Code

- See http://www.cs.ubc.ca/~harrison/Java/QSortAlgorithm.java.html