CSCI 136
Data Structures & Advanced Programming

Lecture 32

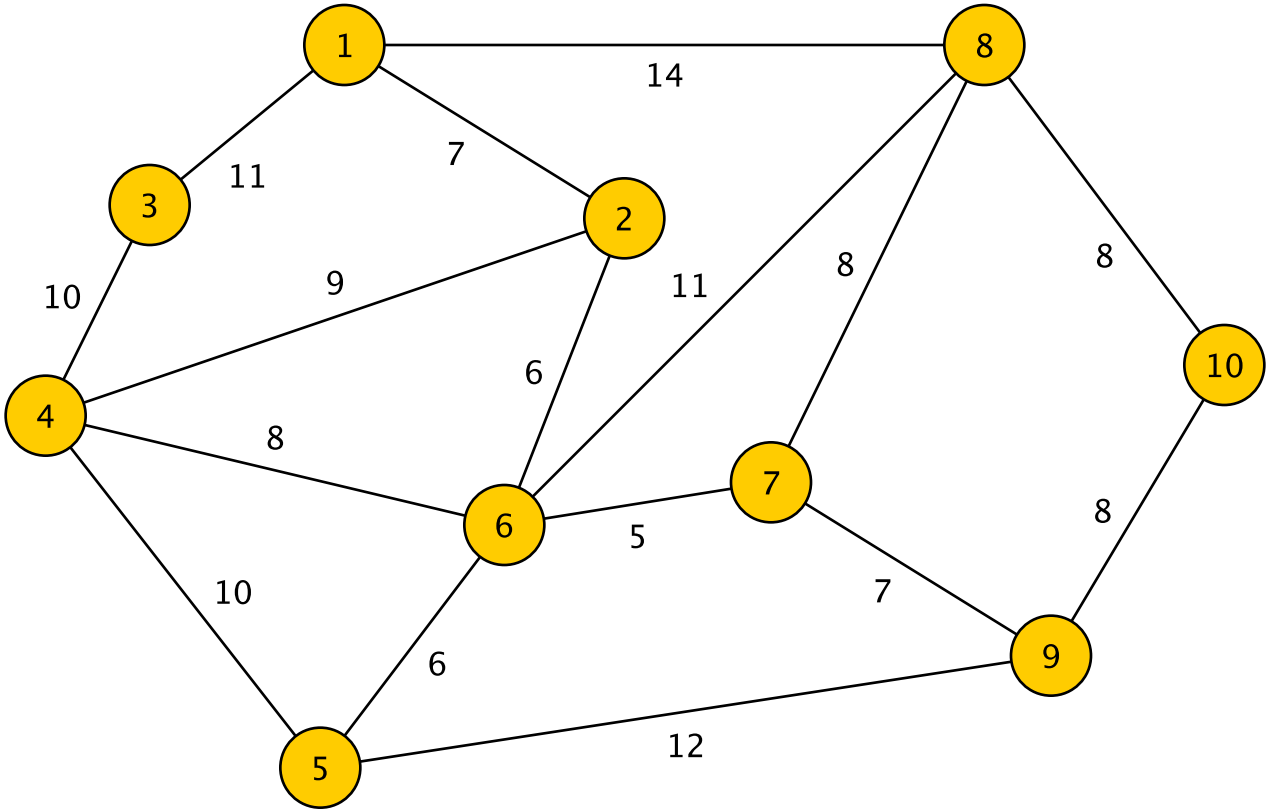
Fall 2018

Instructors: Bill and Sam

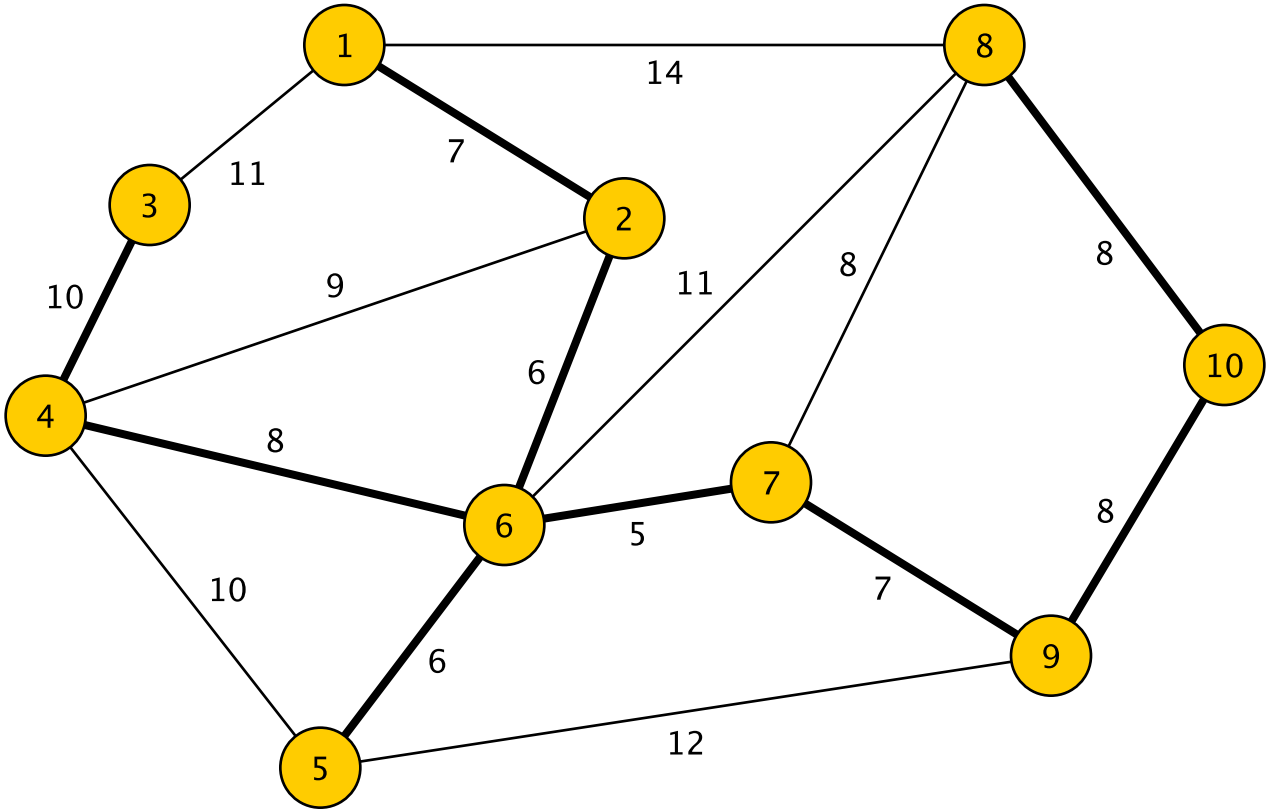
Today's Outline

- More on Prim's Algorithm
- More Core Algorithms: Directed Graphs
  - Dijkstra's Algorithm

Minimum-Cost Spanning Trees



Minimum-Cost Spanning Trees



Recall: An Amazing Fact

Thm: (Prim 1957) The greedy tree-growing algorithm always finds a minimum-cost spanning tree for any connected graph.

Contrast this with the greedy exam scheduling algorithm, which does *not* always find a minimum coloring

Implementing Prim's Algorithm

- We'll "build" the MCST by marking its edges as "visited" in G
- If a vertex is reached we mark it as visited
- How should we represent the list of possible new edges?
  - What operations are important to this list?
    - Add edges
    - Remove cheapest edge
- When we remove an edge from the list, check to ensure it has one end in each of V_1 and V_2

ComparableEdge Class

- Values in a PriorityQueue need to implement Comparable
- We wrap edges of the PQ in a class called ComparableEdge
  - It requires the label used by graph edges to be of a Comparable type

Prim : Space Complexity

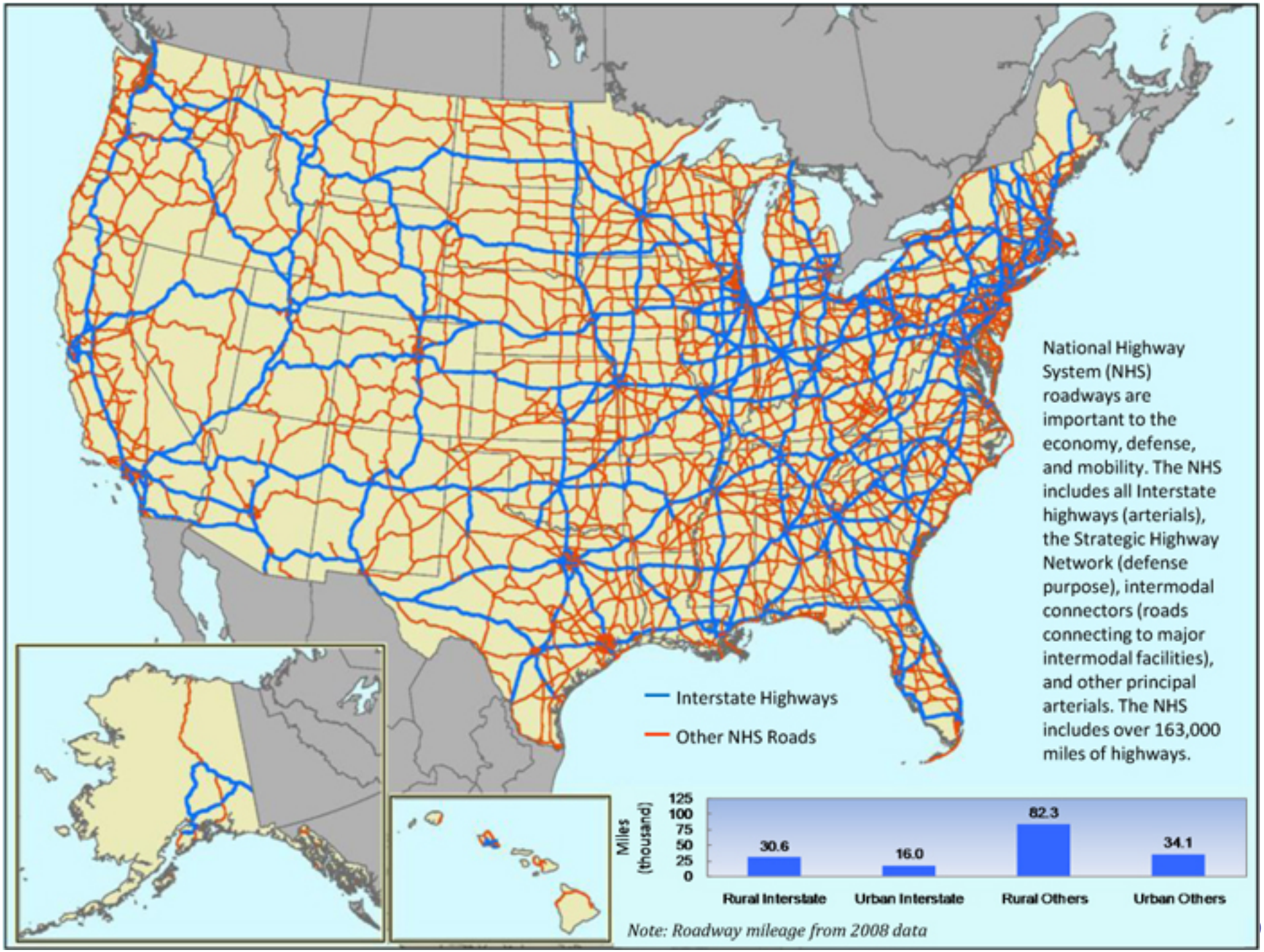
- Graph: O(|V| + |E|)
  - Each vertex and edge uses a constant amount of space
- Priority Queue O(|E|)
  - Each edge takes up constant amount of space
- Every other object (including the neighbor iterator) uses a constant amount of space
- Result: O(|V| + |E|)
  - Optimal in Big-O sense!

Prim : Time Complexity

Assume Map ops are O(1) time (not quite true!)

For each iteration of do ... while loop

- Add neighbors to queue: O(log |E|) each
  - Iterator operations are O(1) [Why?]
  - Adding an edge to the queue is O(log |E|)
- Find next edge: O(# edges checked * log |E|)
  - Removing an edge from queue is O(log |E|) time
  - All other operations are O(1) time
- O(E log E) time


Interstate Highways
Other NHS Roads
National Highway System (NHS) roadways are important to the economy, defense, and mobility. The NHS includes all Interstate highways (arterials), the Strategic Highway Network (defense purpose), intermodal connectors (roads connecting to major intermodal facilities), and other principal arterials. The NHS includes over 163,000 miles of highways.
Miles (thousand)
125
100
75
50
25
0
30.6
16.0
82.3
34.1
Rural Interstate
Urban Interstate
Rural Others
Urban Others
Note: Roadway mileage from 2008 data

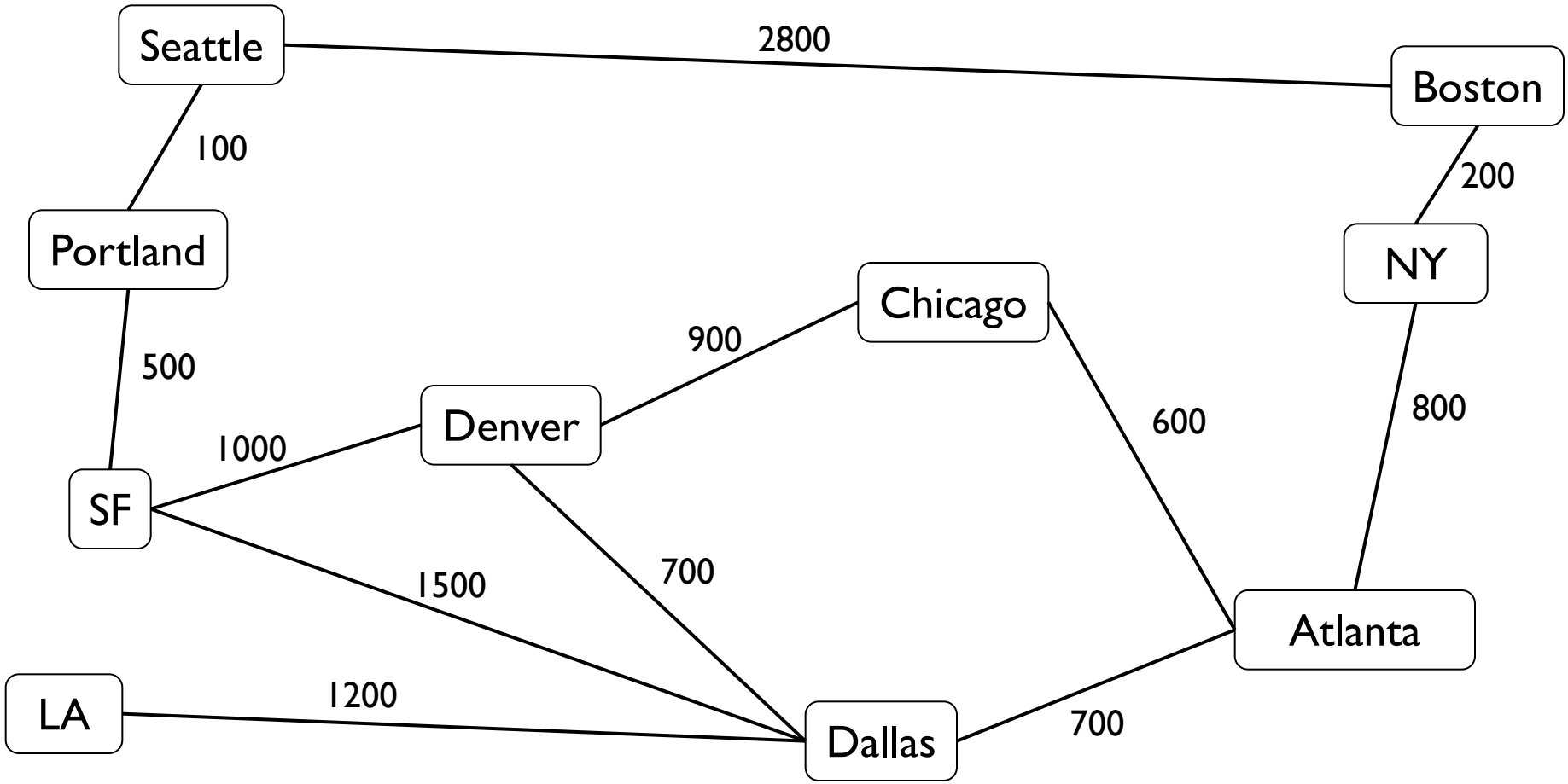
Single Source Shortest Paths

The Problem: Given a graph G and a starting vertex v, find, for *each* vertex u != v reachable from v, a shortest path from v to u.

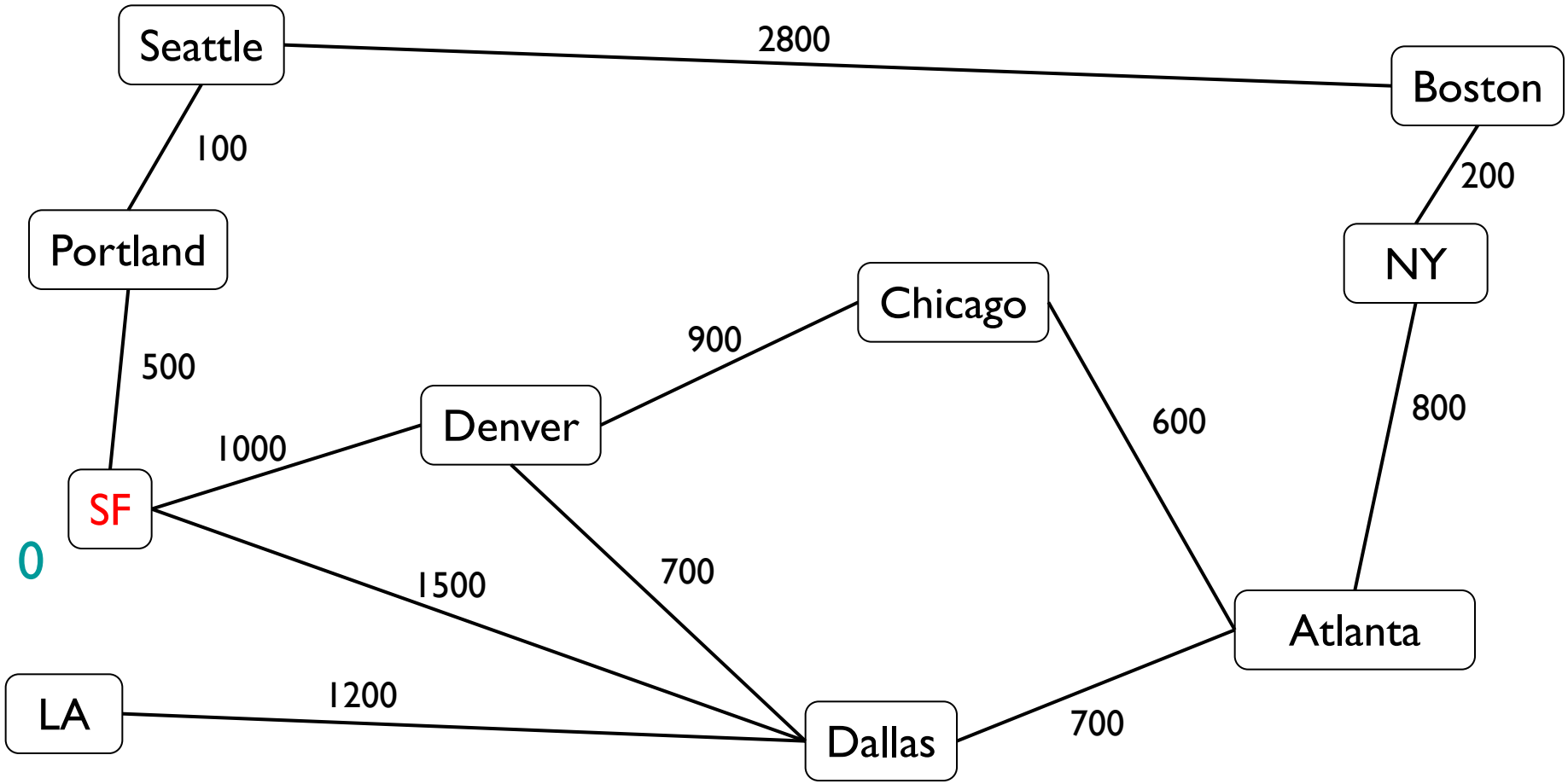
- The Single Source Shortest Paths Problem
- Arises in many contexts, including network communications
- Uses edge weights (but we'll call them "lengths"): assume they are non-negative numbers
- Could be a directed or undirected graph

Dijkstra's Algorithm

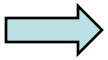
```
Dijkstra(G, s)  // l(e) is the length of edge e
let T←({s}, ∅)  and PQ be an empty priority queue
for each neighbor v of s, add edge (s,v) to PQ with priority l(e)
while T doesn't have all vertices of G and PQ is non-empty
    repeat
        e ← PQ.removeMin()   // skip edges with both ends in T
    until  PQ is empty or e=(u,v) for u∈T, v ∉ T
    if  e=(u,v) for u∈T, v ∉ T
        add e (and v) to T
        for each neighbor w of v
            add edge (v,w) to PQ with weight/key d(s,v) + l(v,w)
```

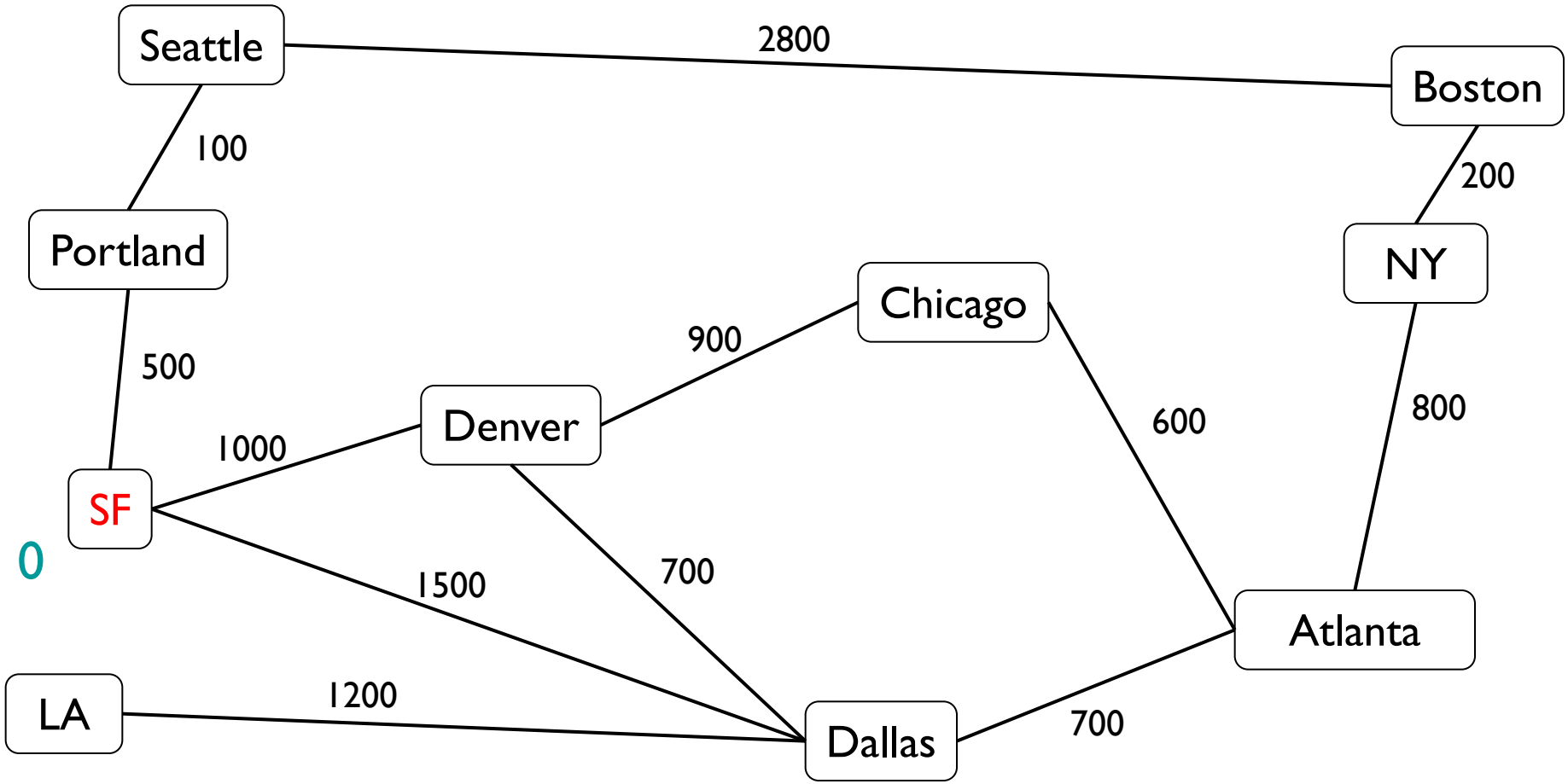


Dijkstra's Algorithm



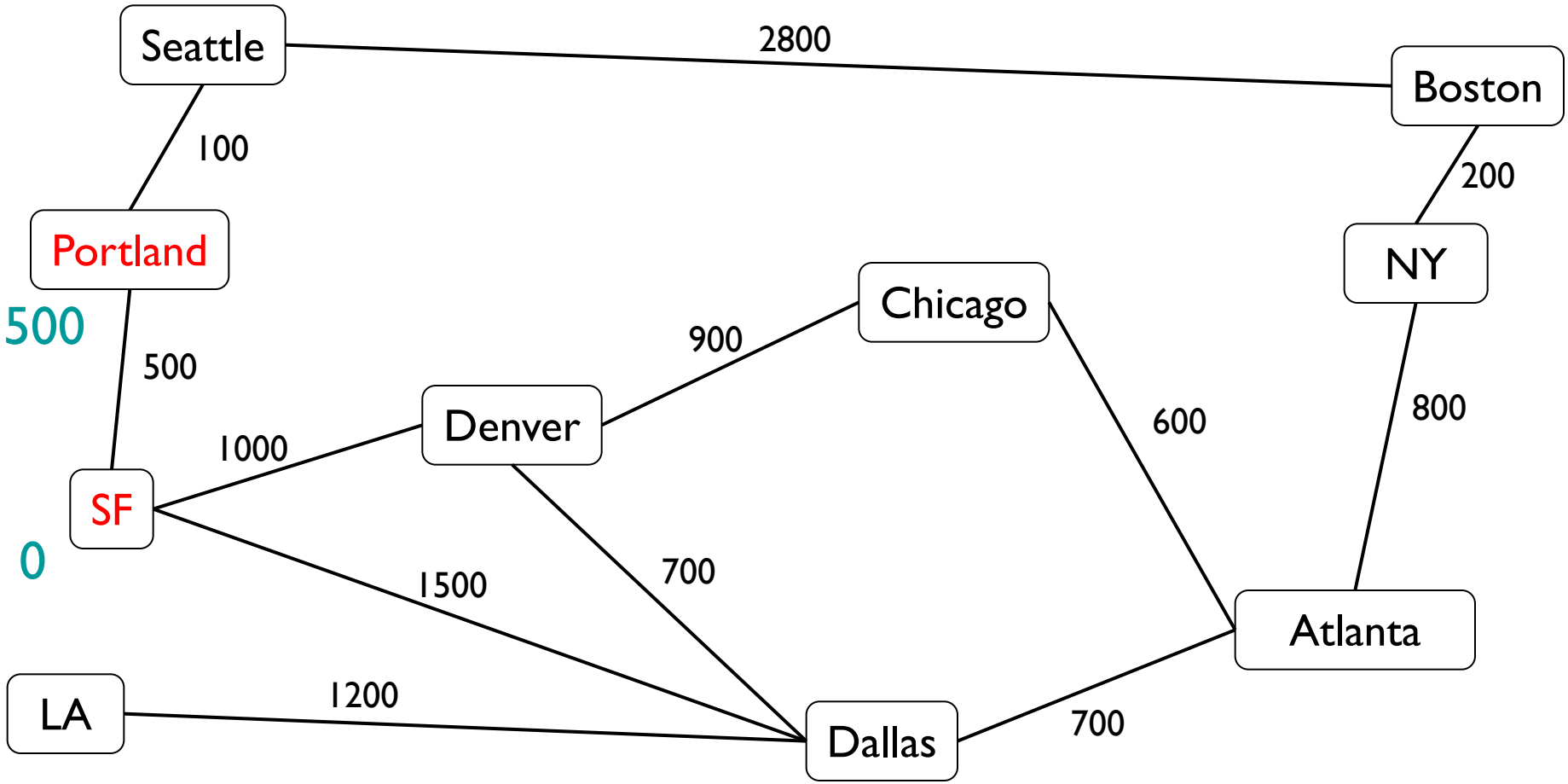
Priority Queue





Priority Queue

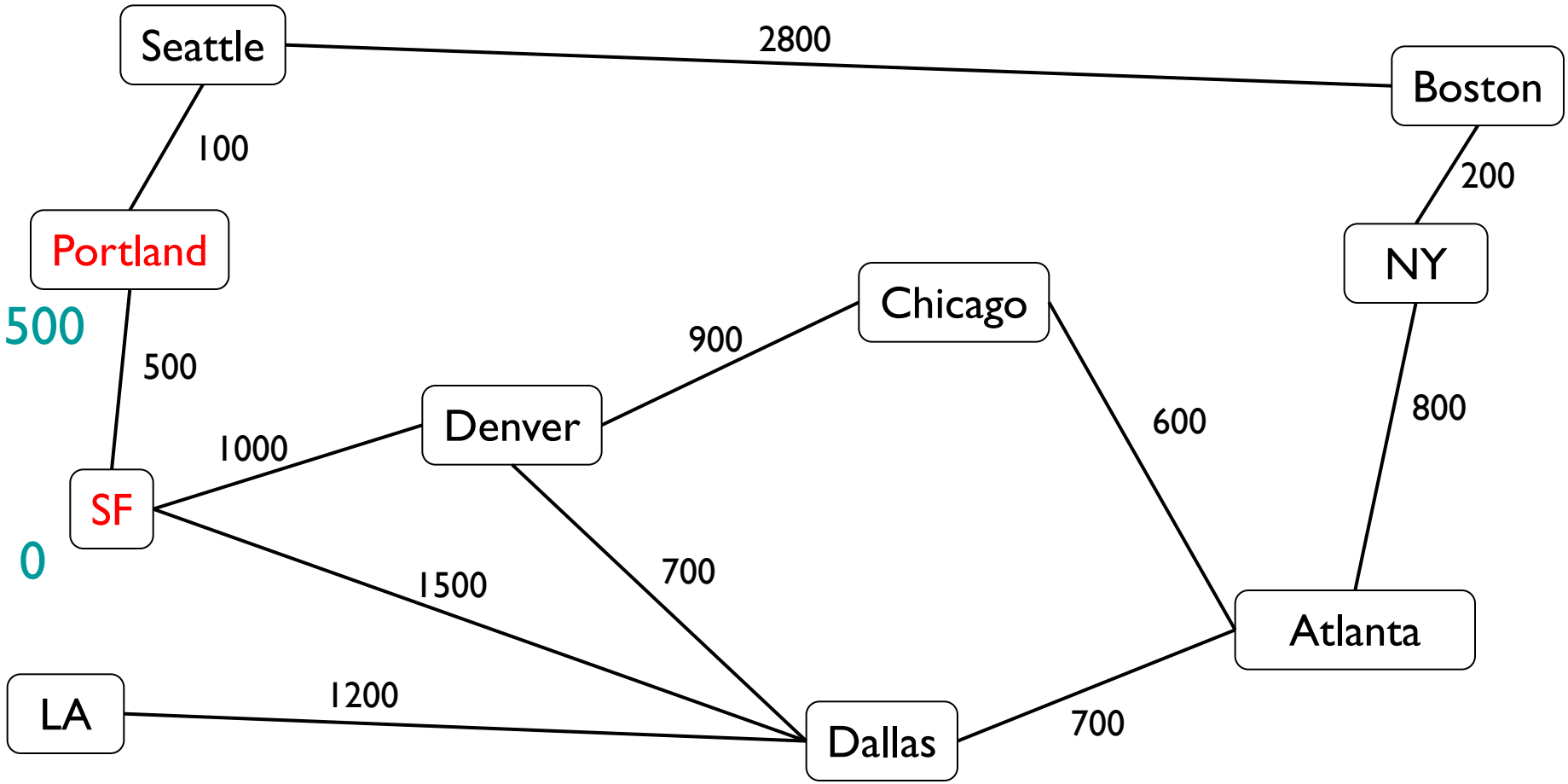
SF->Port;	SF->Den;	SF->Dal
500	1000	1500



Current: 500 SF->Port (need to add Port's neighbors to PQ)

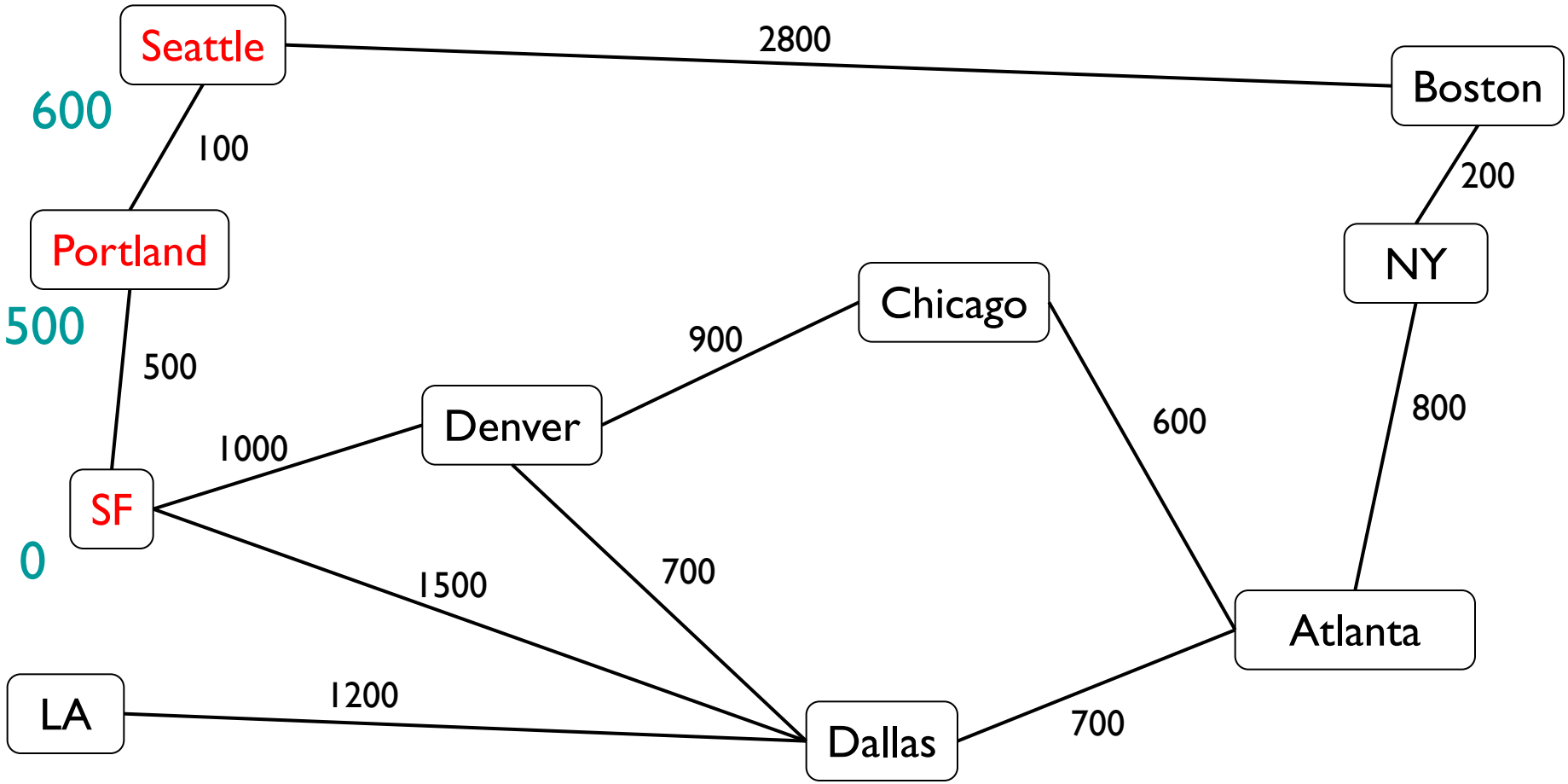
SF->Den; 1000

SF->Dal 1500



Current: 500 SF->Port

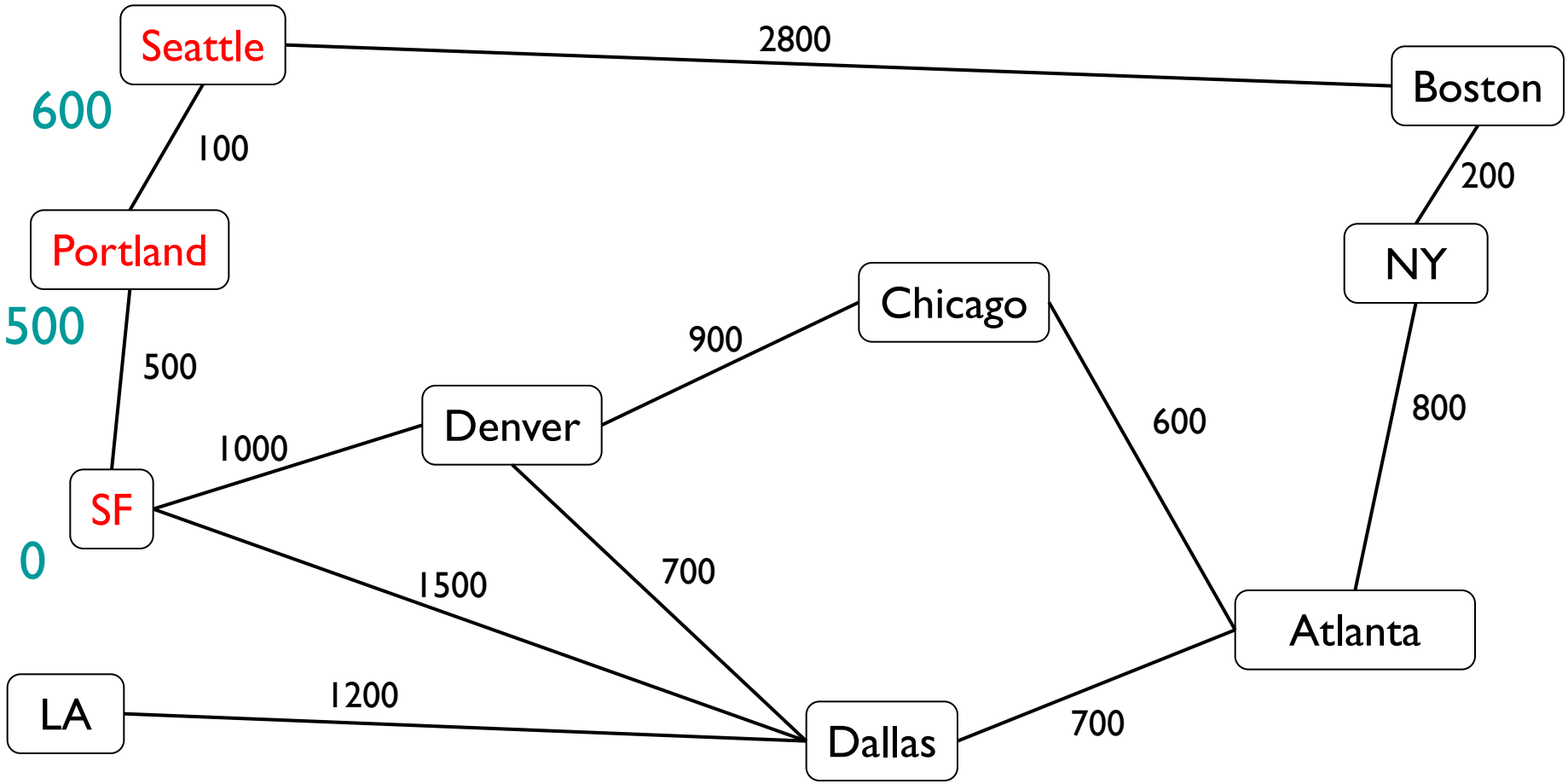
SF->Port->Sea;	SF->Den;	SF->Dal
600	1000	1500



Current: 600 SF->Port->Sea

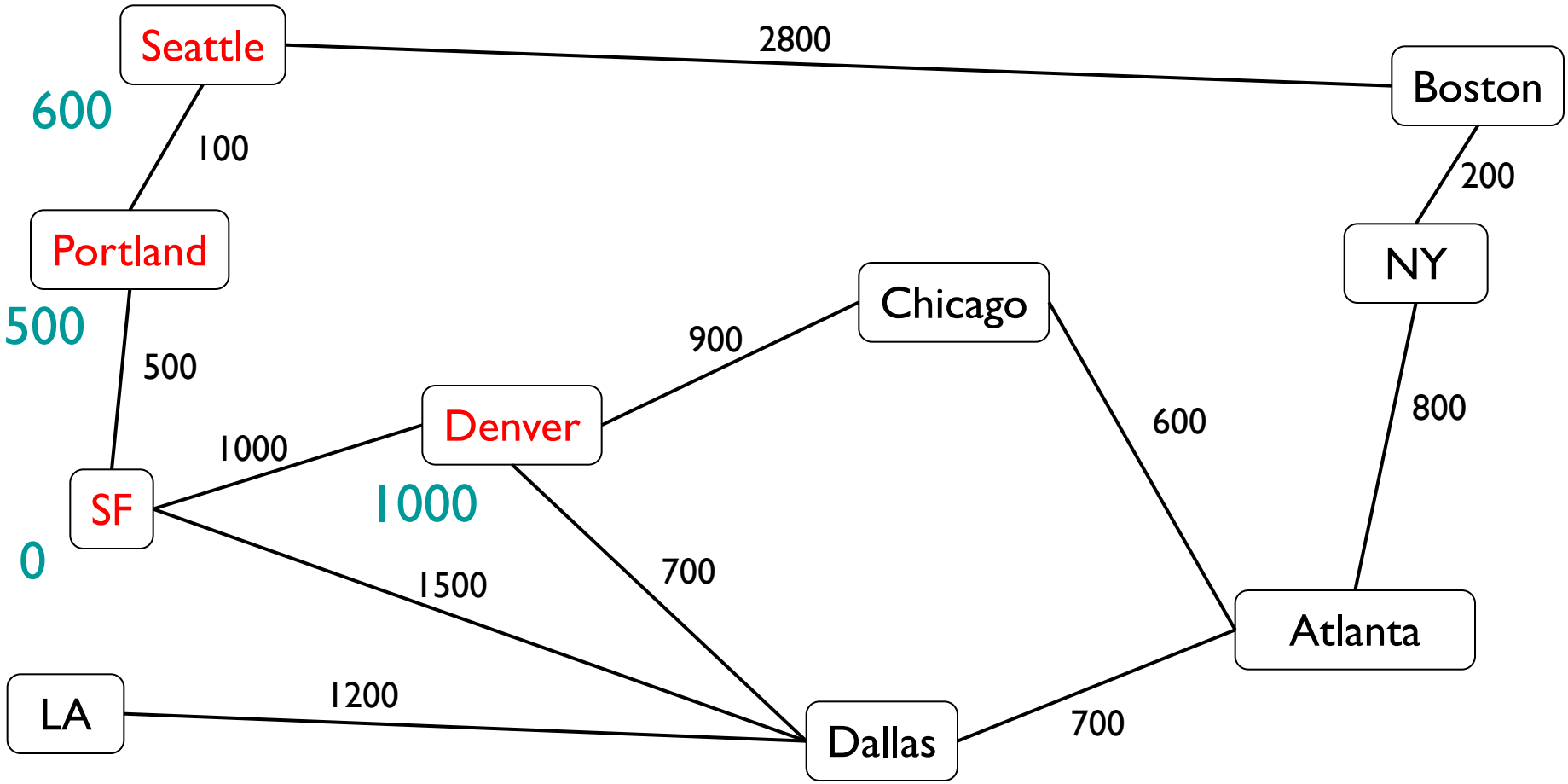
SF->Den; 1000

SF->Dal 1500



Current: 600 SF->Port->Sea

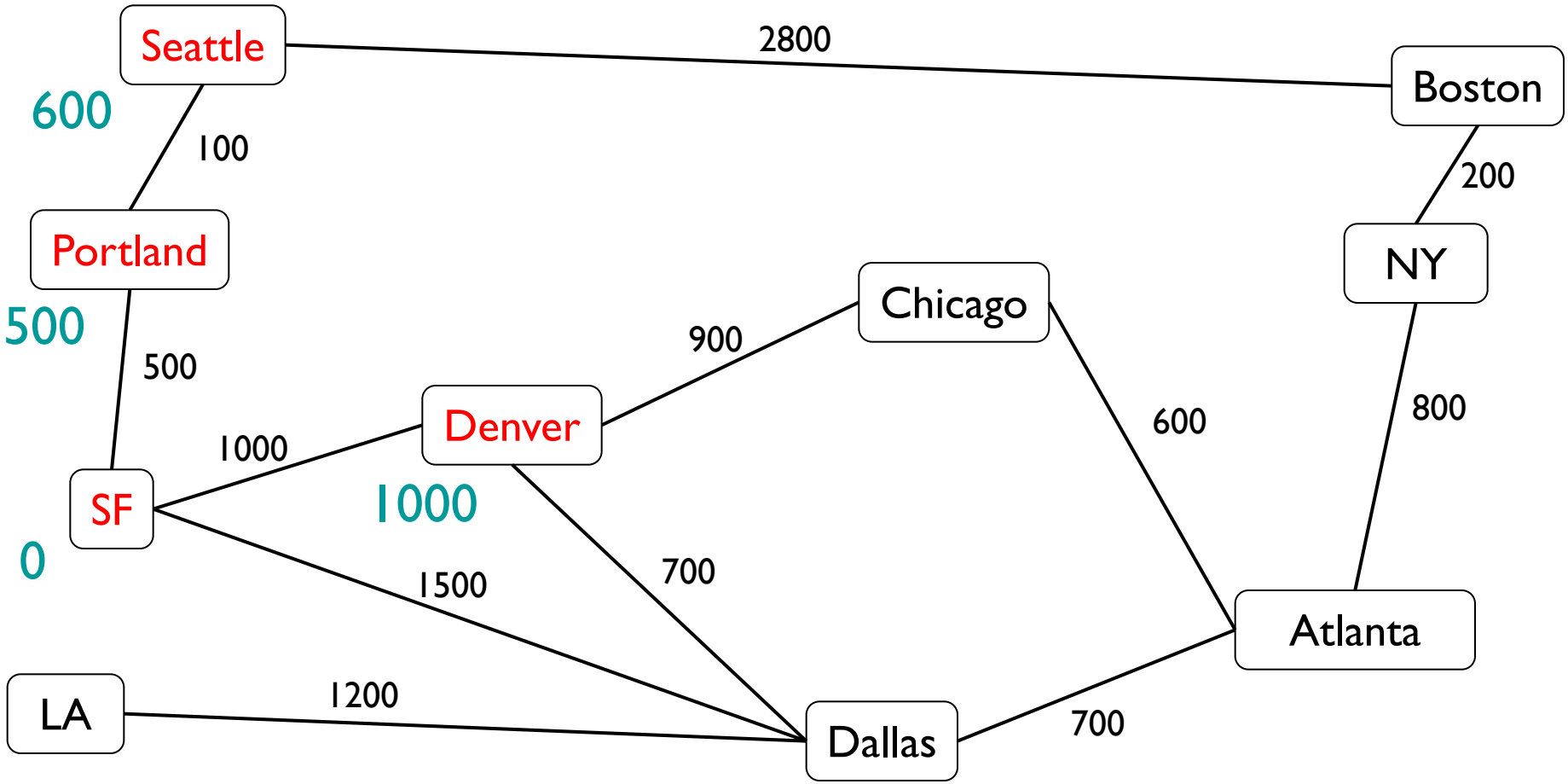
SF->Den;	SF->Dal;	SF->Port->Sea->Bos
1000	1500	3400



Current: 1000 SF->Den

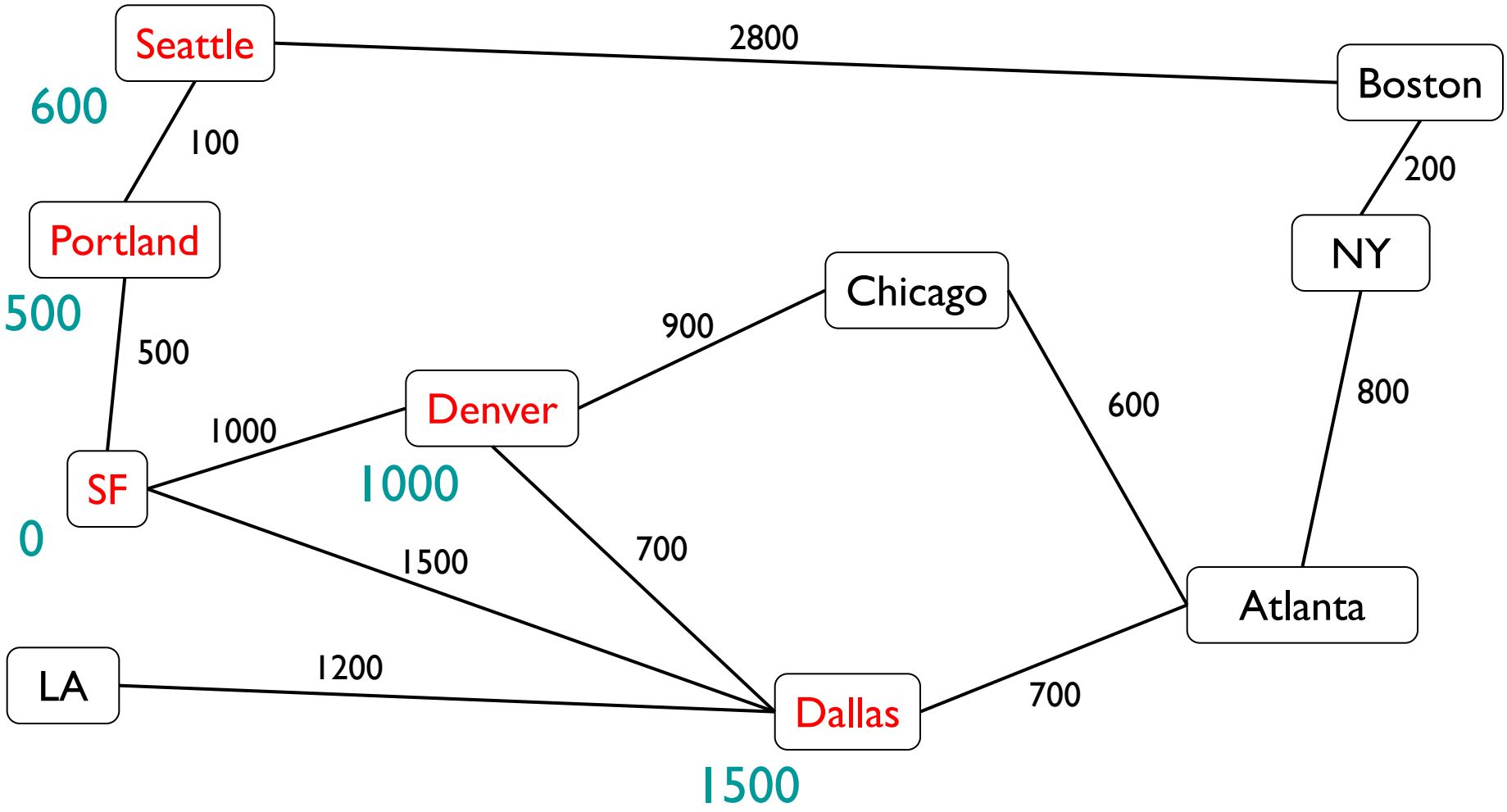
SF->Dal; 1500

SF->Port->Sea->Bos 3400



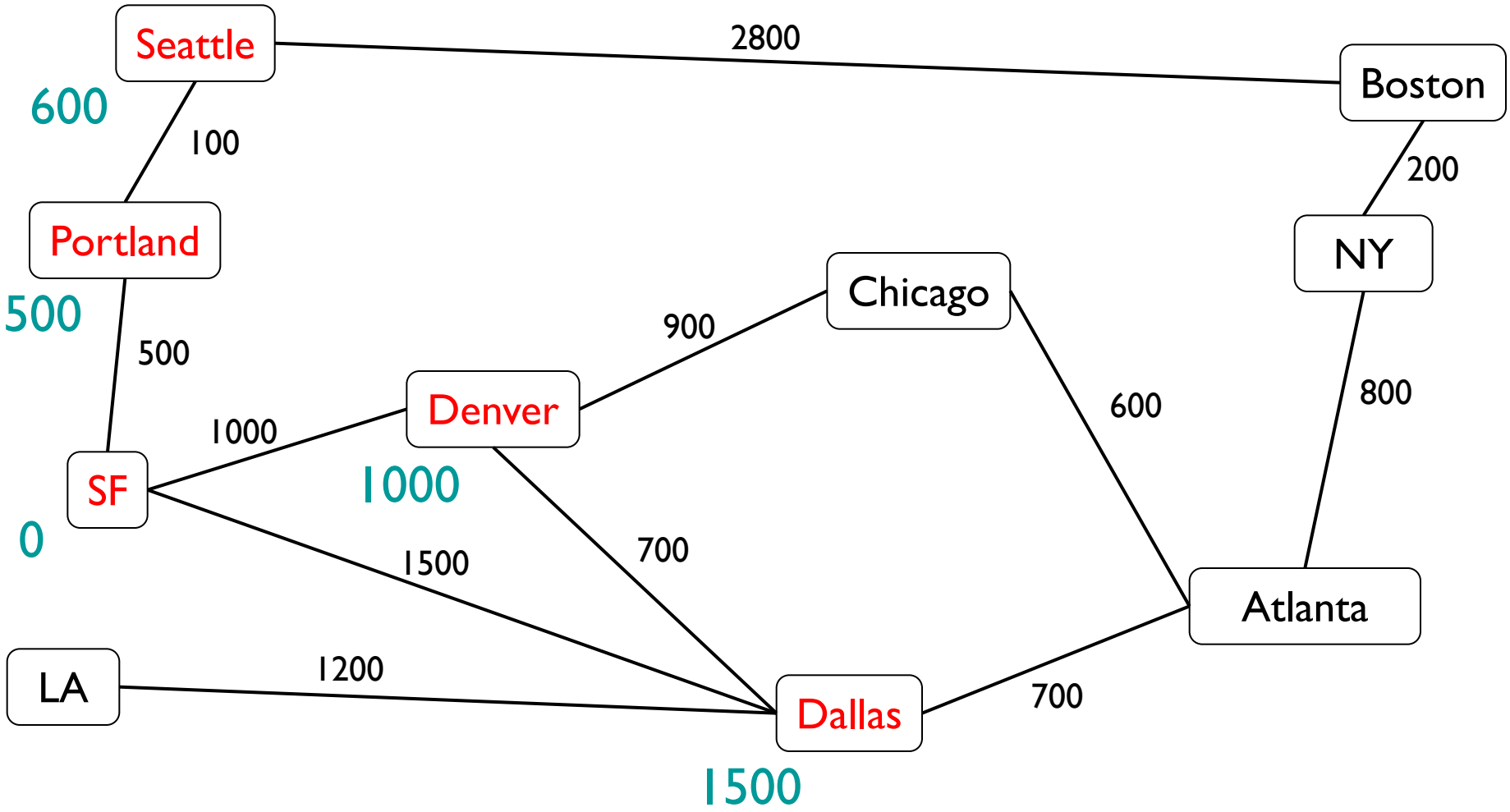
Current: 1000 SF->Den

SF->Dal; 1500	SF->Den->Dal; 1700	SF->Den->Chi; 1900	SF->Port->Sea->Bos 3400



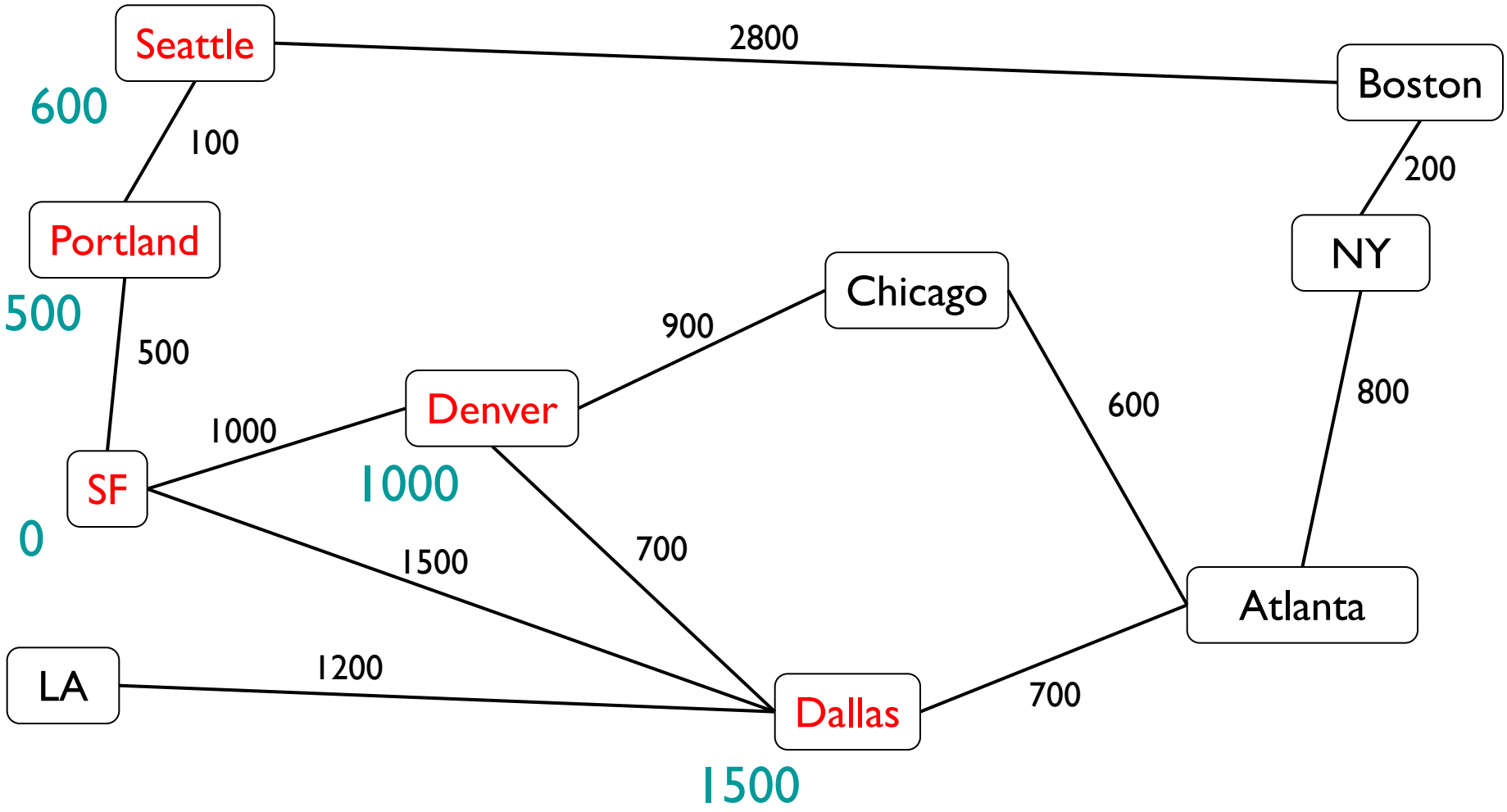
Current: 1500 SF->Dal

SF->Den->Dal; 1700	SF->Den->Chi; 1900	SF->Port->Sea->Bos 3400



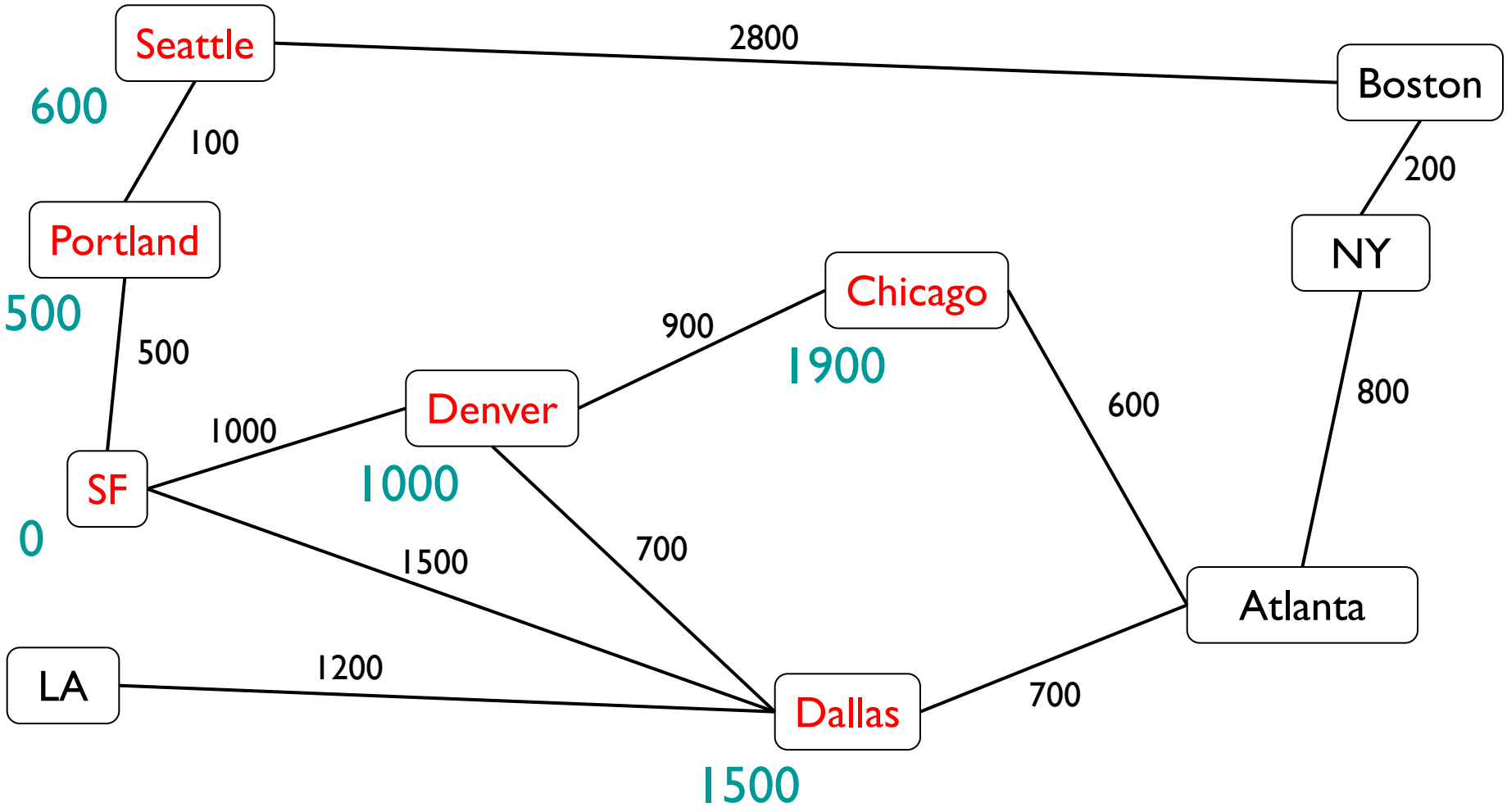
Current: 1500 SF->Dal

SF->Den->Dal; 1700	SF->Den->Chi; 1900	SF->Dal->Atl; 2200	SF->Dal->LA; 2700	SF->Port->Sea->Bos 3400



Current: 1700 SF->Den->Dal (we already have Dallas!)

SF->Den->Chi; 1900	SF->Dal->Atl; 2200	SF->Dal->LA; 2700	SF->Port->Sea->Bos 3400

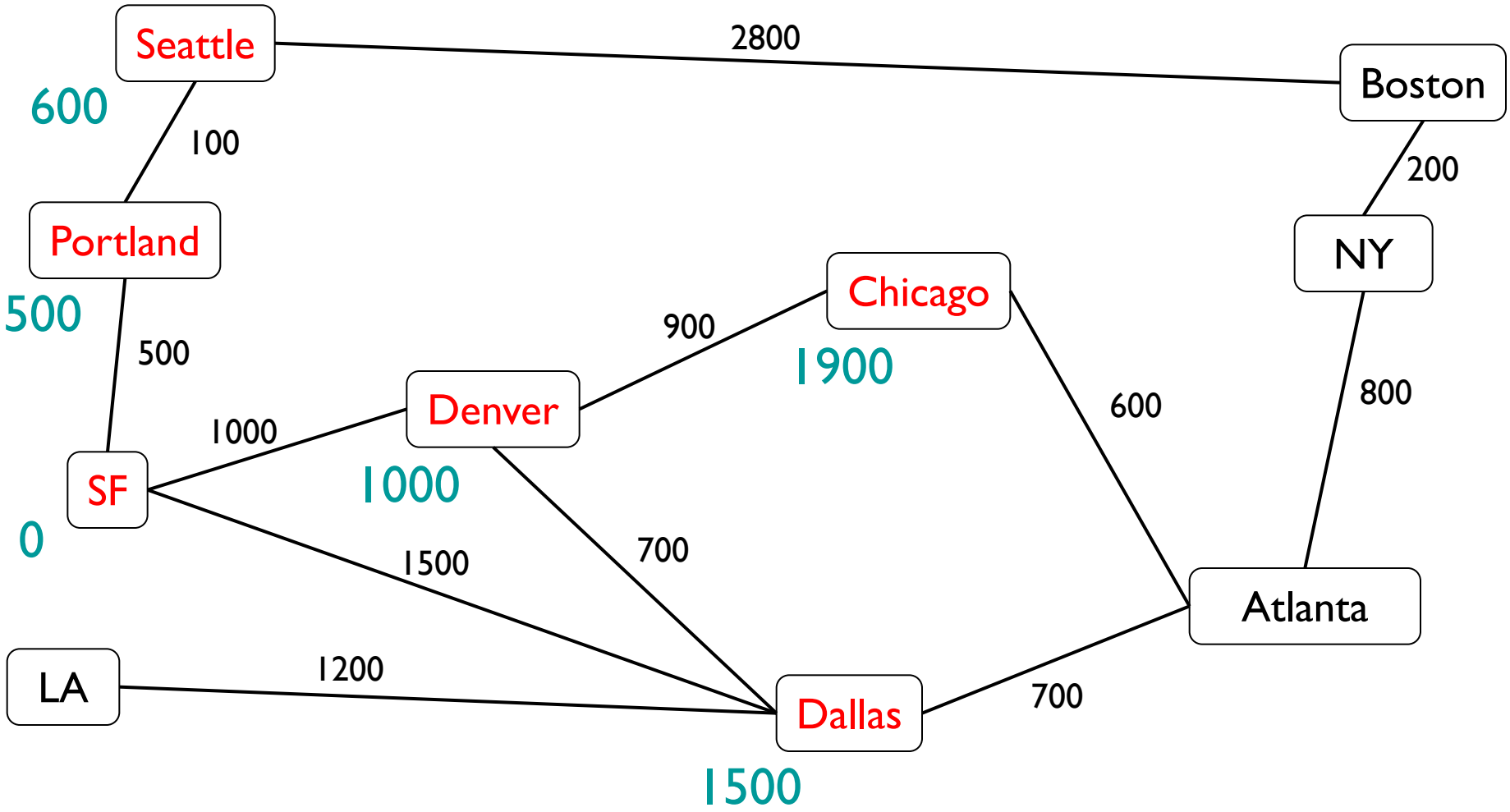


Current: 1900 SF->Den->Chi

SF->Dal->Atl; 2200

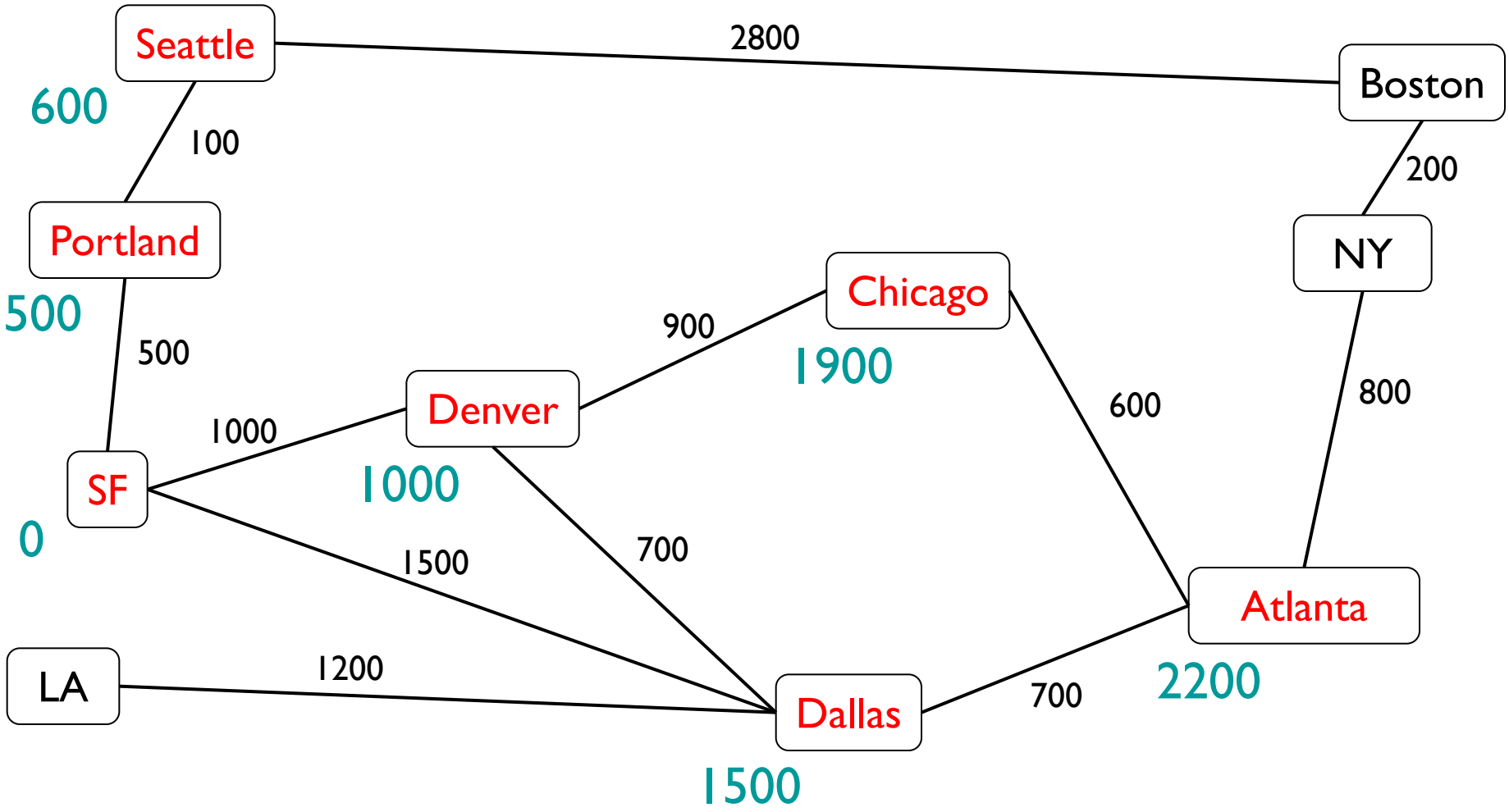
SF->Dal->LA; 2700 3400

SF->Port->Sea->Bos



Current: 1900 SF->Den->Chi

SF->Dal->Atl; 2200	SF->Den->Chi->Atl; 2500	SF->Dal->LA; 2700	SF->Port->Sea->Bos 3400

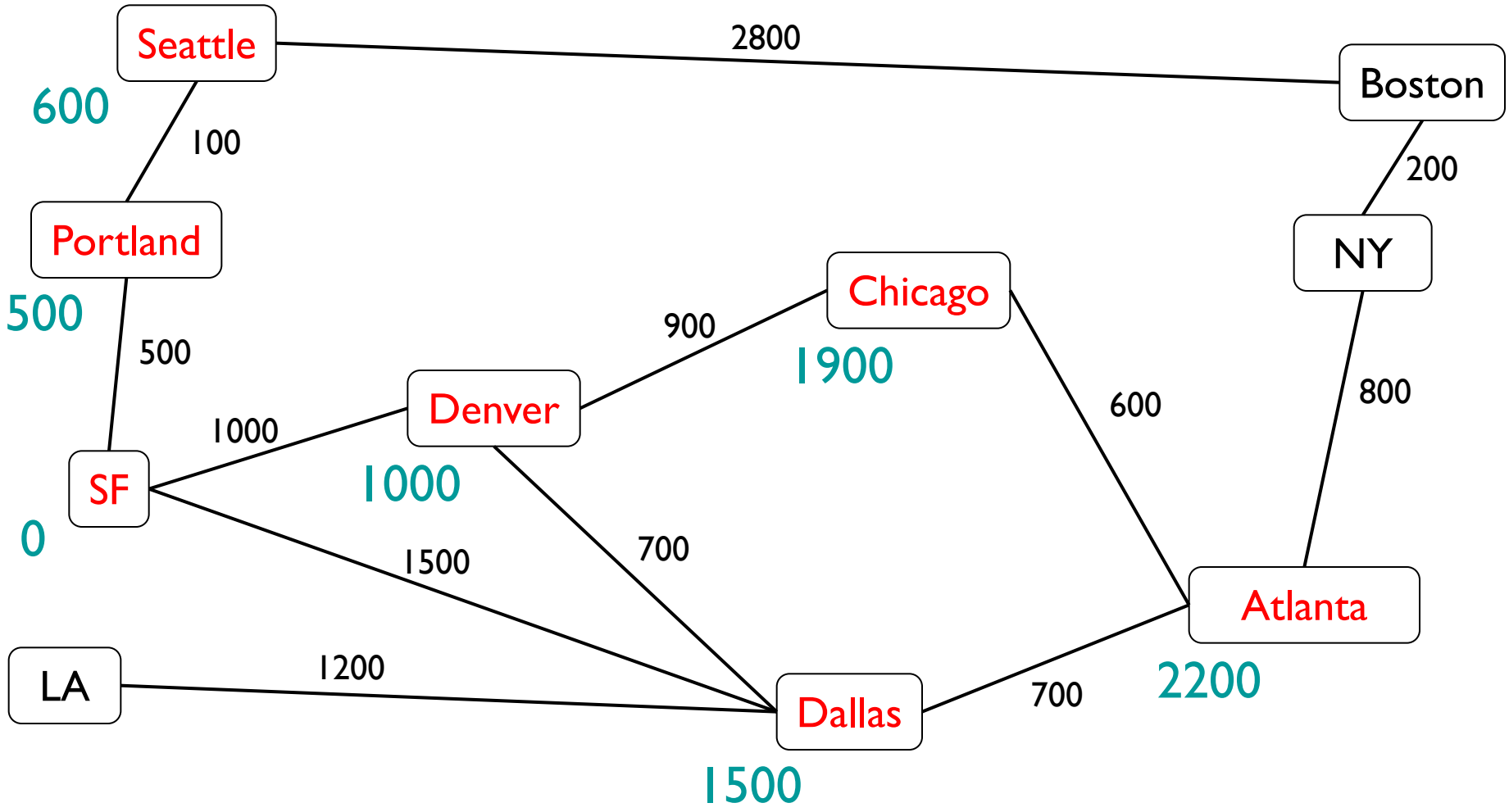


Current: 2200 SF->Dal->Atl

SF->Den->Chi->Atl; 2500

SF->Dal->LA; 2700

SF->Port->Sea->Bos 3400



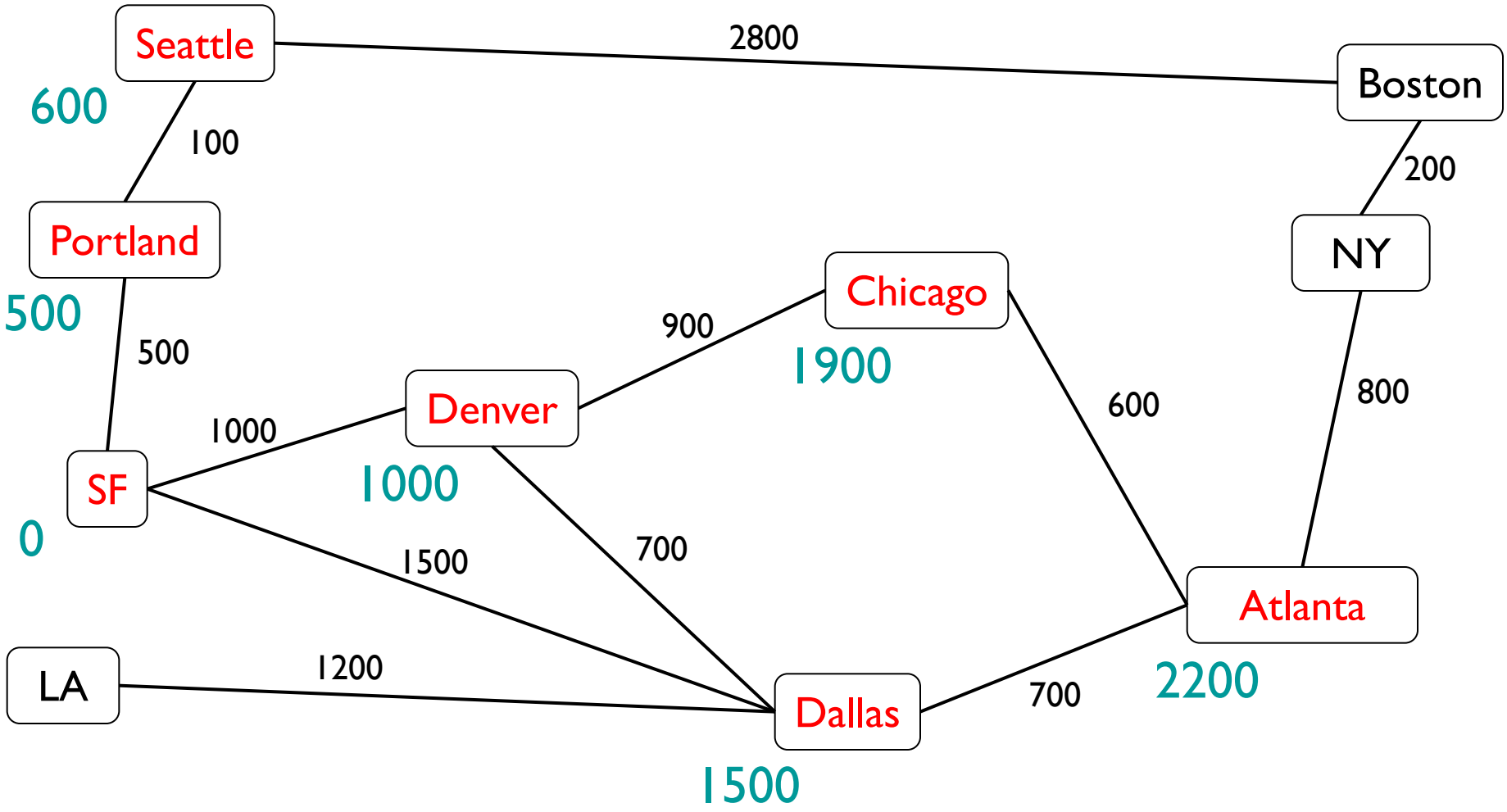
Current: 2200 SF->Dal->Atl

SF->Den->Chi->Atl; 2500

SF->Dal->LA; 2700

SF->Dal->Atl->NY; 3000

SF->Port->Sea->Bos 3400

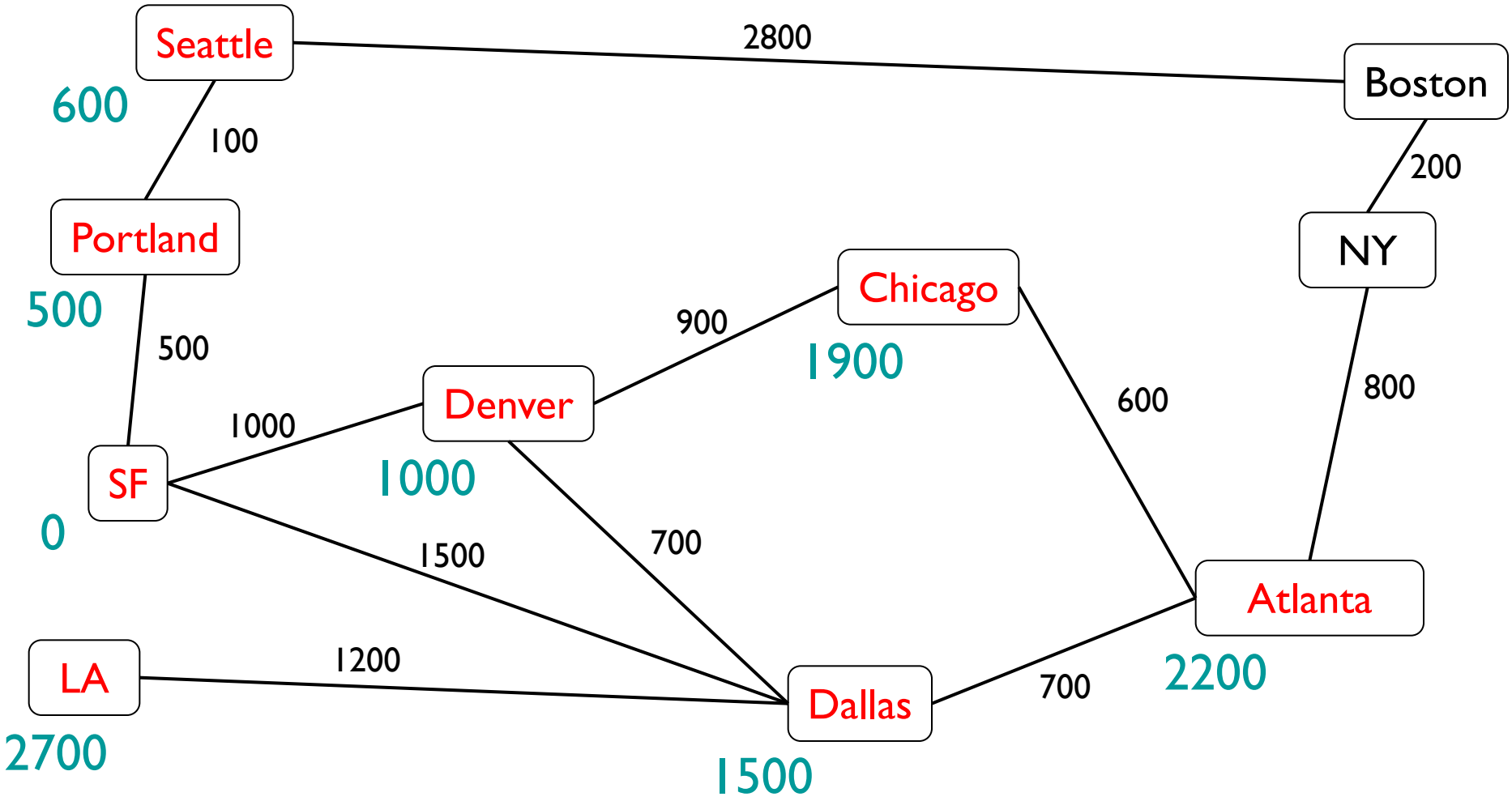


Current: 2500 SF->Den->Chi->Atl

SF->Dal->LA; 2700

SF->Dal->Atl->NY; 3000

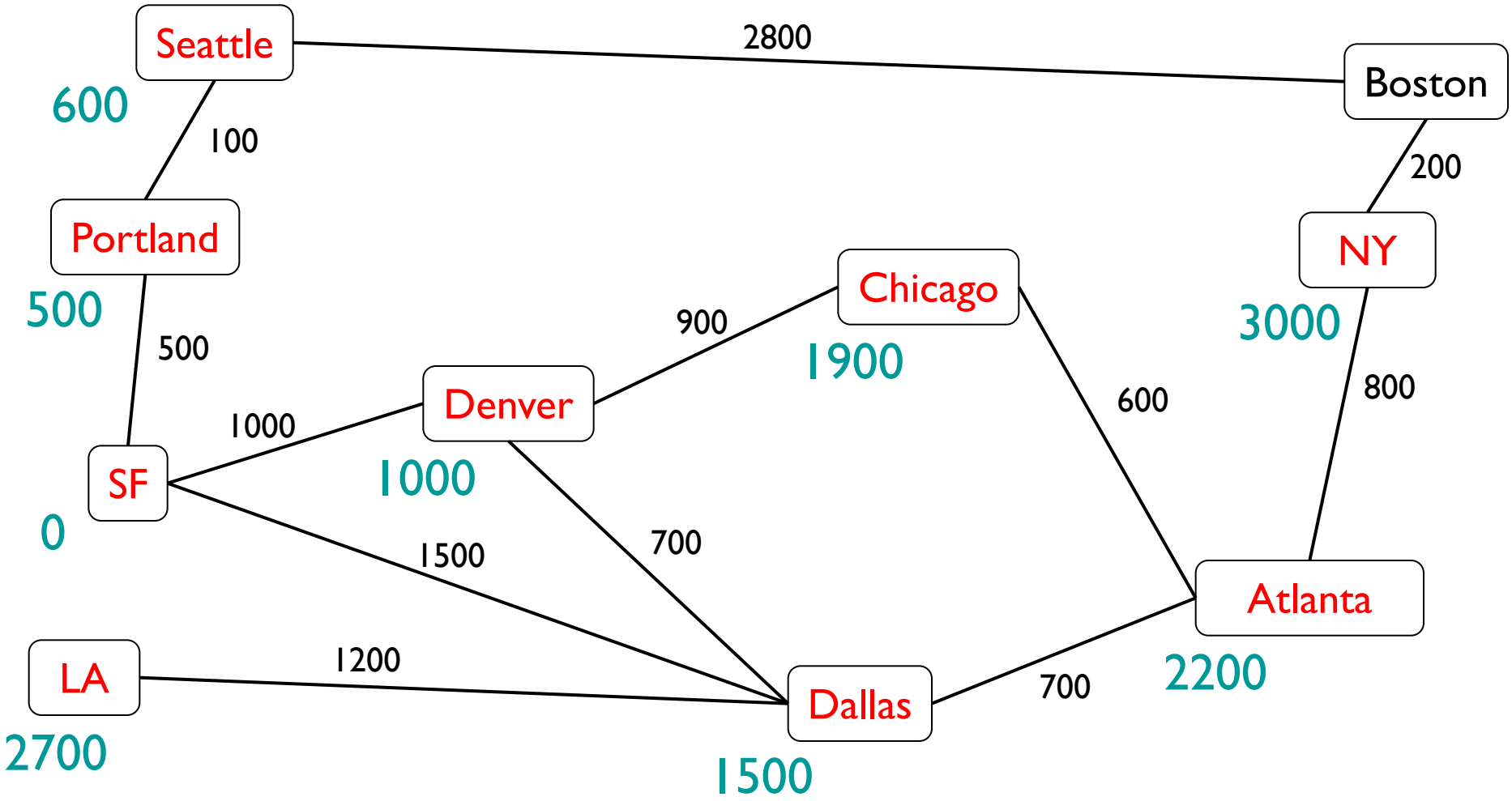
SF->Port->Sea->Bos 3400



Current: 2700 SF->Dal->LA

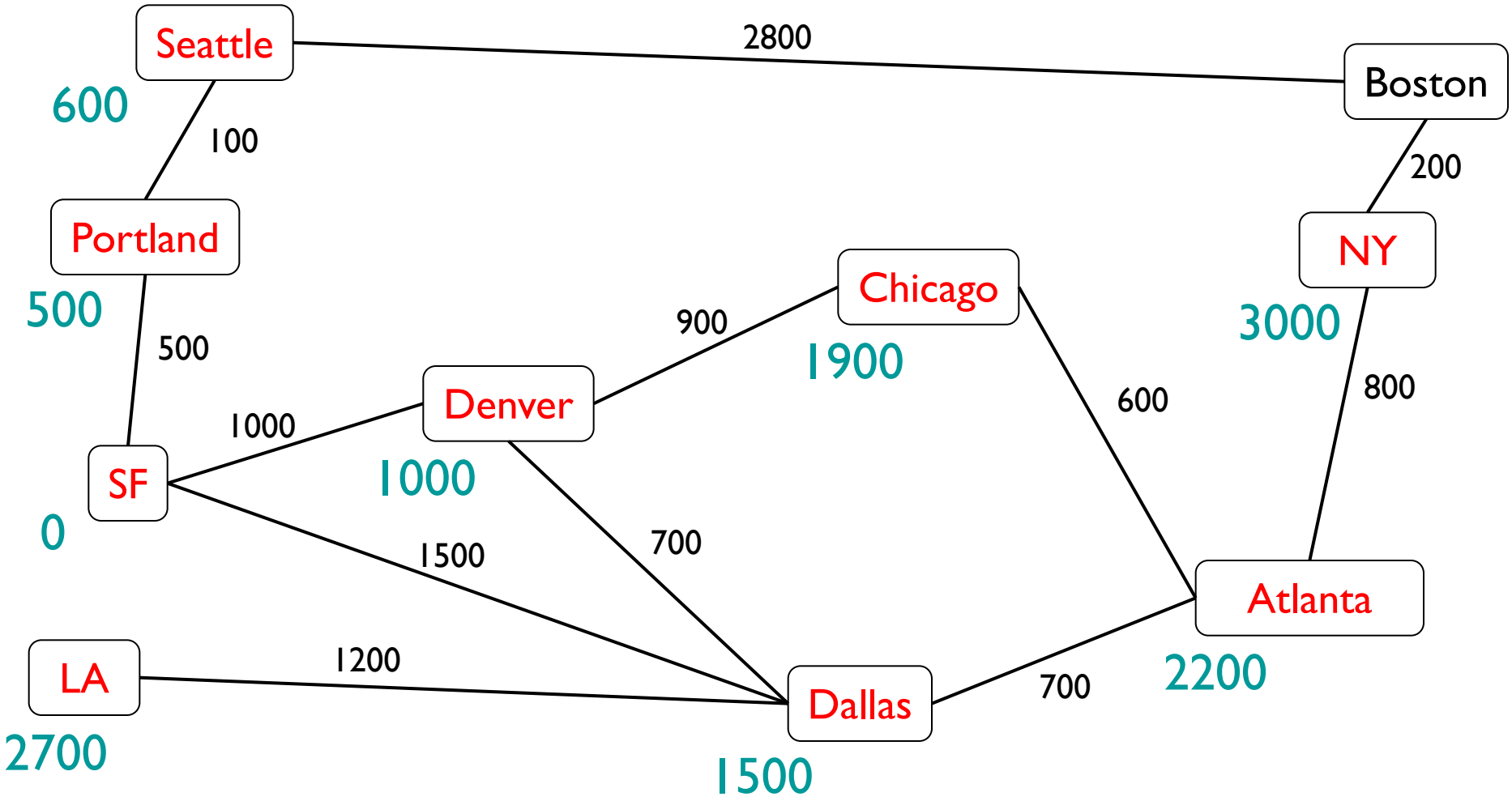
SF->Dal->Atl->NY;
3000

SF->Port->Sea->Bos
3400



Current: 3000 SF->Dal->Atl->NY

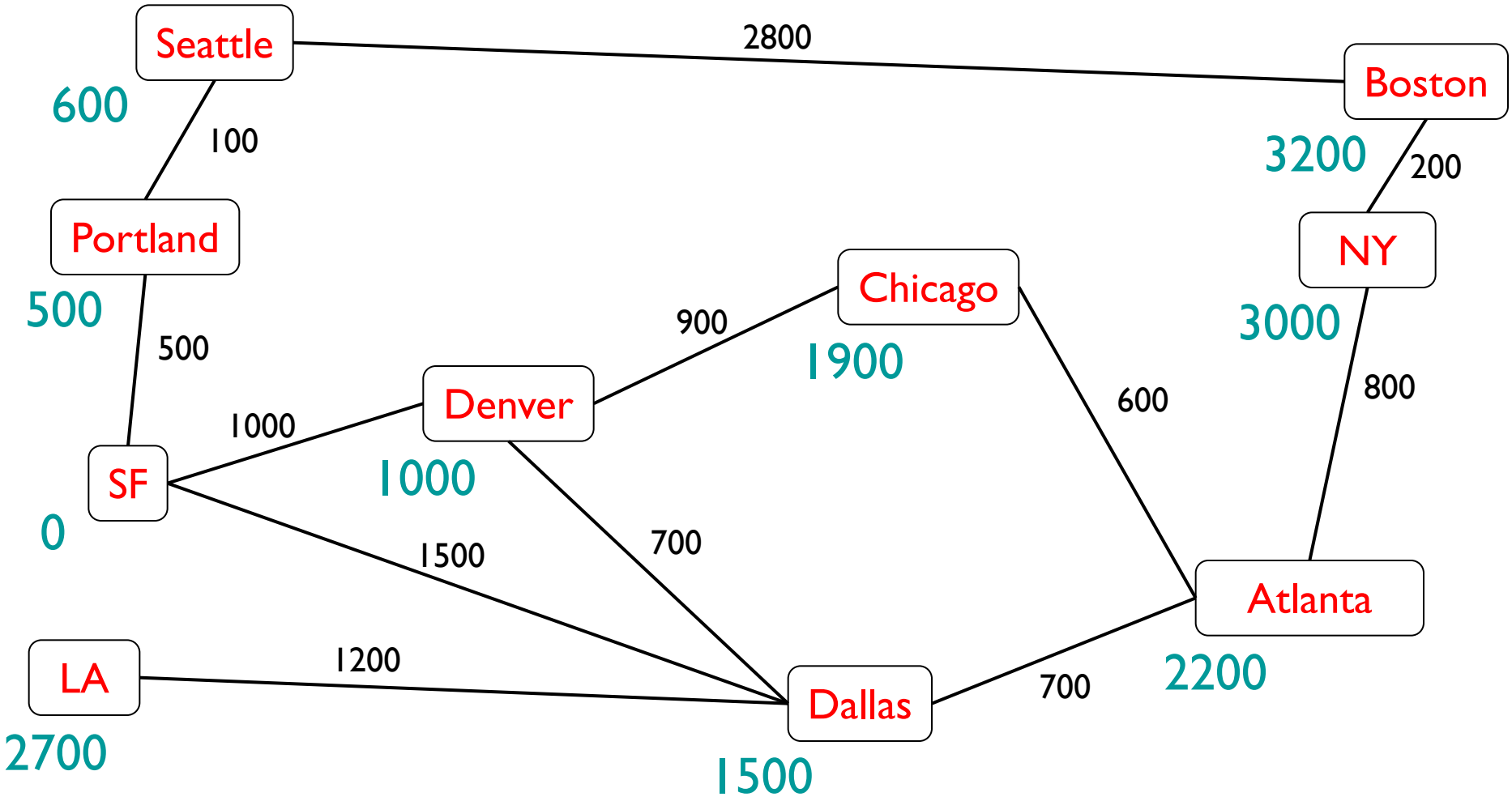
SF->Port->Sea->Bos
3400



Current: 3000 SF->Dal->Atl->NY

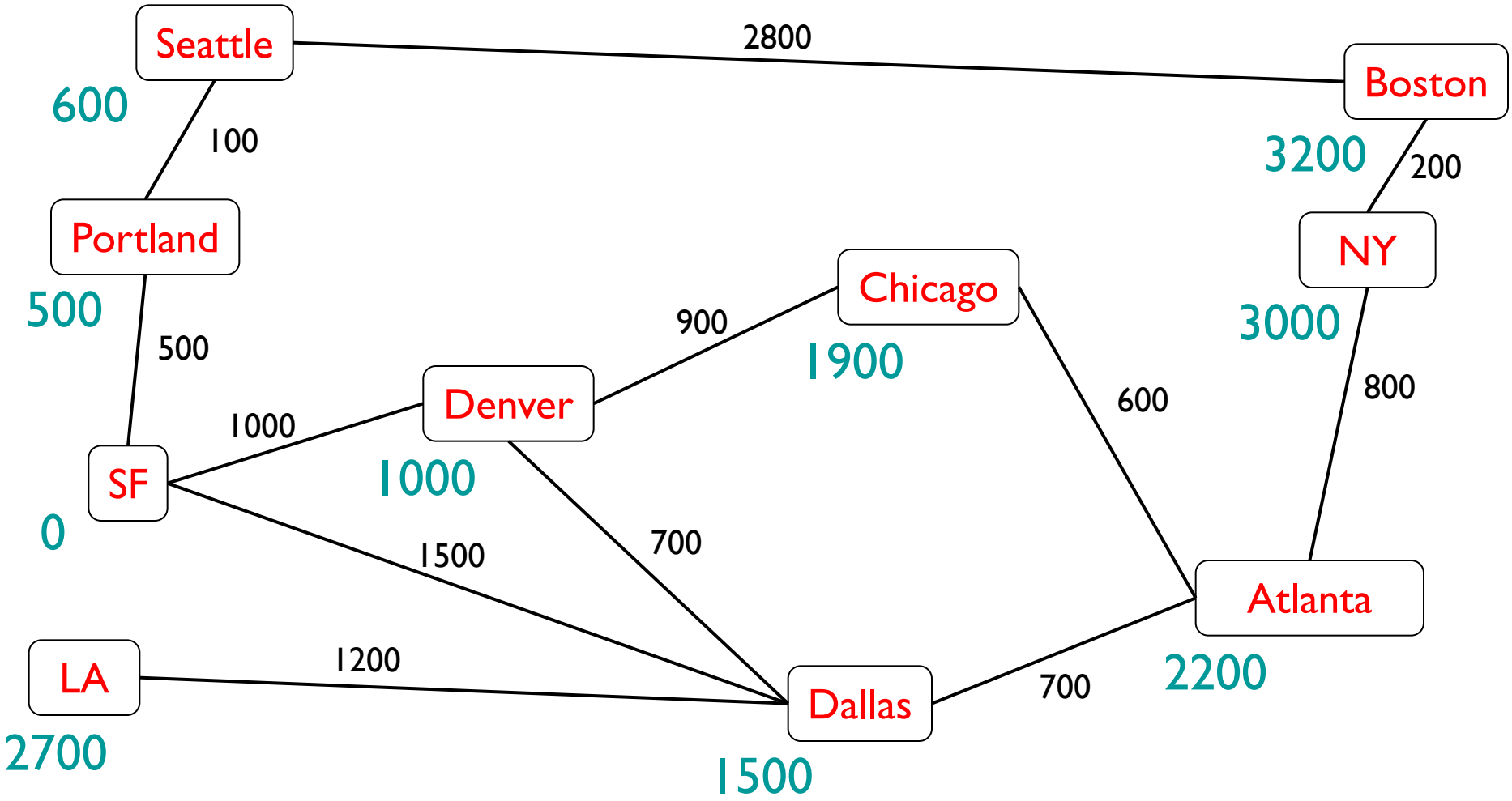
SF->Dal->Atl->NY->Bos; 3200

SF->Port->Sea->Bos 3400

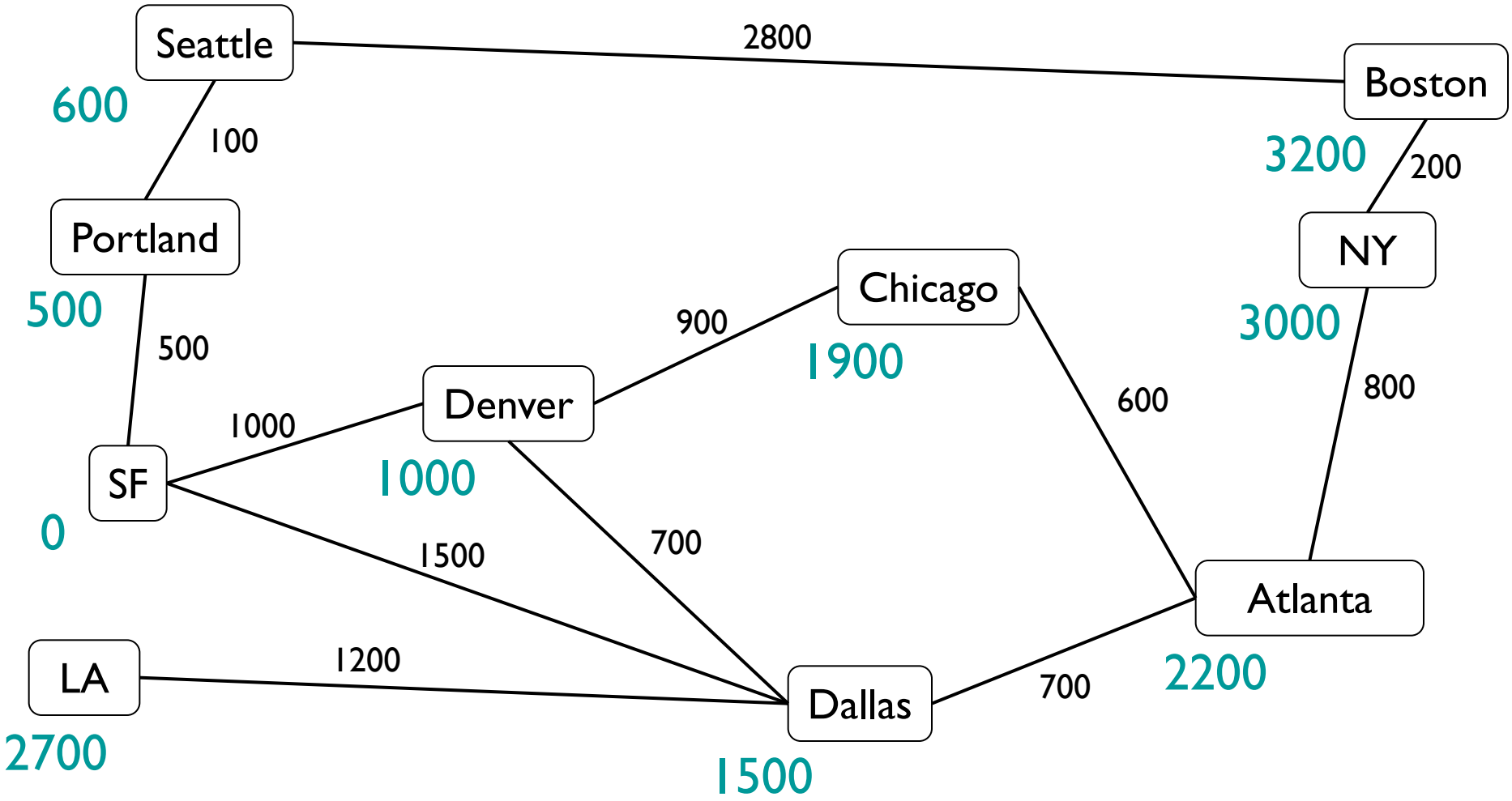


Current: 3200 SF->Dal->Atl->NY->Bos

SF->Port->Sea->Bos
3400



Current: 3400 SF->Port->Sea->Bos


Seattle
2800
Boston
600
100
3200
200
Portland
NY
500
Chicago
900
3000
500
1900
800
Denver
600
1000
1000
SF
0
700
1500
Atlanta
LA
1200
Dallas
700
2200
2700
1500

Current:

Dijkstra: What Do We Return?

- As we find a new vertex v to add to the tree of shortest paths, add edges e=(v,w) to a map.
- Precisely:
  - Use the PQ association(X,Y) edgeInfo where
    - X is d(s,v) + l(v,w)
    - Y is the edge e=(v,w)
  - Add the key/value pair (w, edgeInfo) to the map
- So the map entry with key w tells us the edge the best path used to get from the tree to w

Dijkstra: Space Complexity

- Graph: O(|V| + |E|)
  - Each vertex and edge uses a constant amount of space
- Priority Queue O(|E|)
  - Each edge takes up constant amount of space
- Result: O(|V| + |E|)
  - Optimal in Big-O sense!

Dijkstra : Time Complexity

Assume Map ops are O(1) time

Across *all* iterations of outer while loop

- Edges are added to and removed from the priority queue
  - But any edge is added (and removed) at most once!
  - Total PQ operation cost is O(|E| log |E|) time
  - All other operations take constant time
- Thus time complexity is O(|E| log |E|)

Improvements?

- Dijkstra's can be improved to O(|E| + |V| log |V|) using a *Fibonacci heap*
  - Can "decrease key" in O(1)

- Prim's can be improved to O(|E| α(|E|)) time
  - Where α is the incredibly-slowly-growing "inverse Ackermann" function

- It is not known if either of these can be improved further

Let's look at the code

After Thanksgiving

- Maps!!!
- Start reviewing for the final