Network Layer

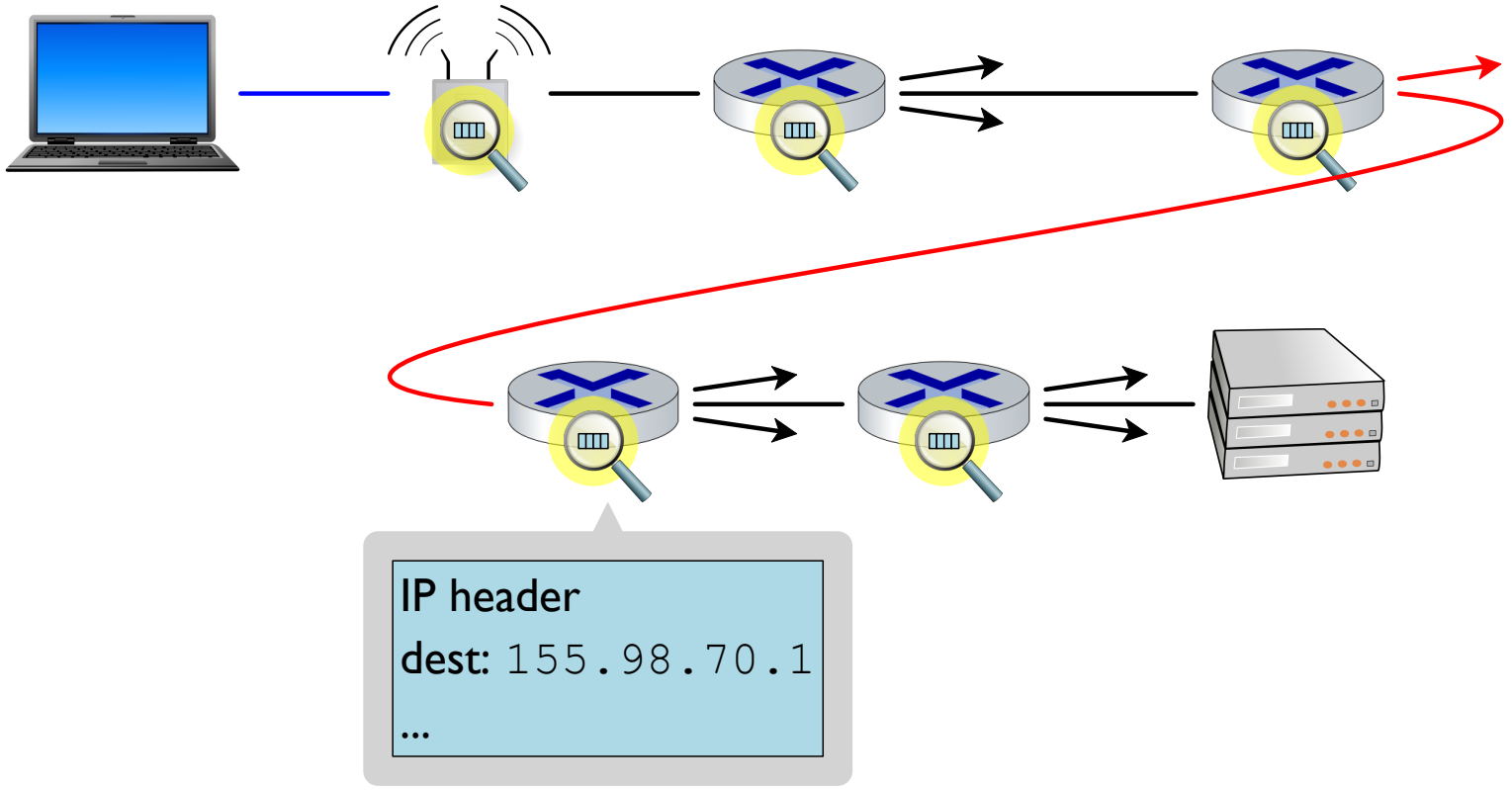
So far:

Data plane: packet content and per-router forwarding/flow rules

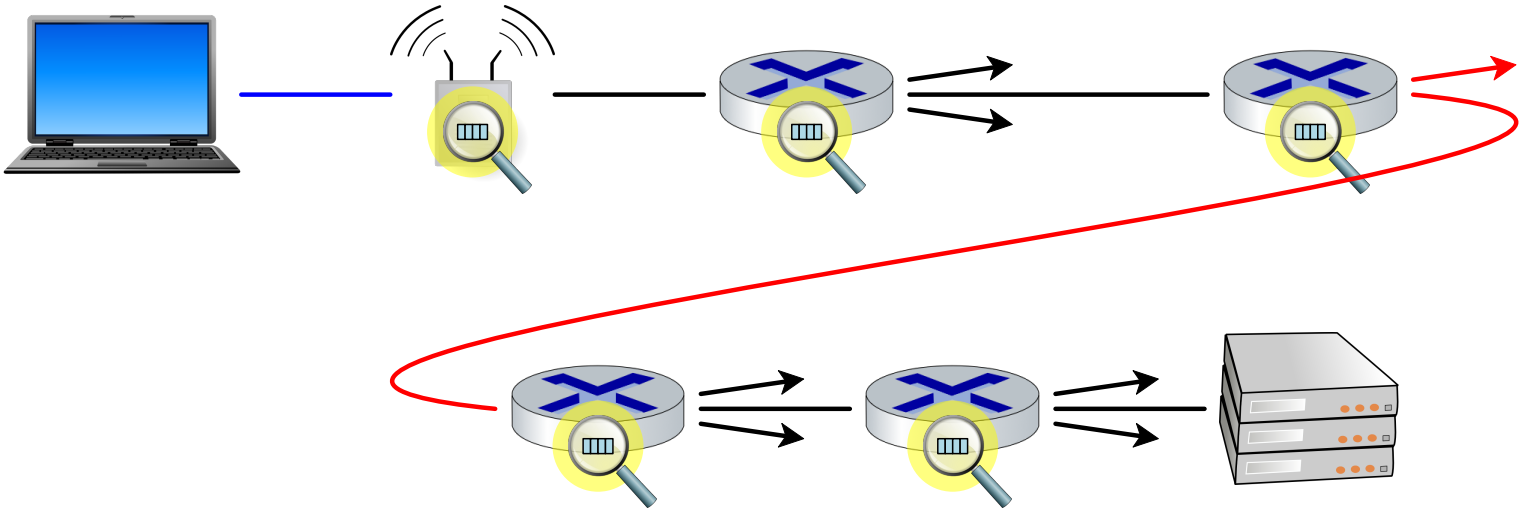
Today:

Control plane: how forwarding/flow rules get installed

Routing

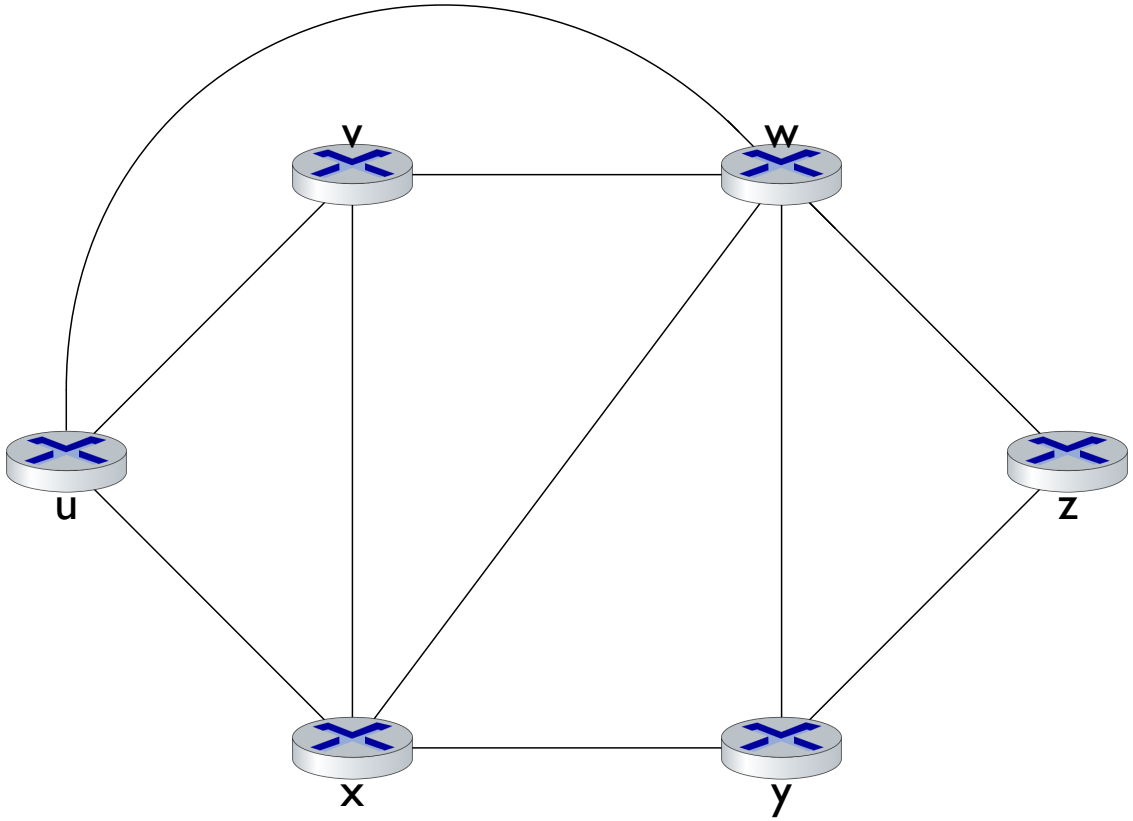


Routing

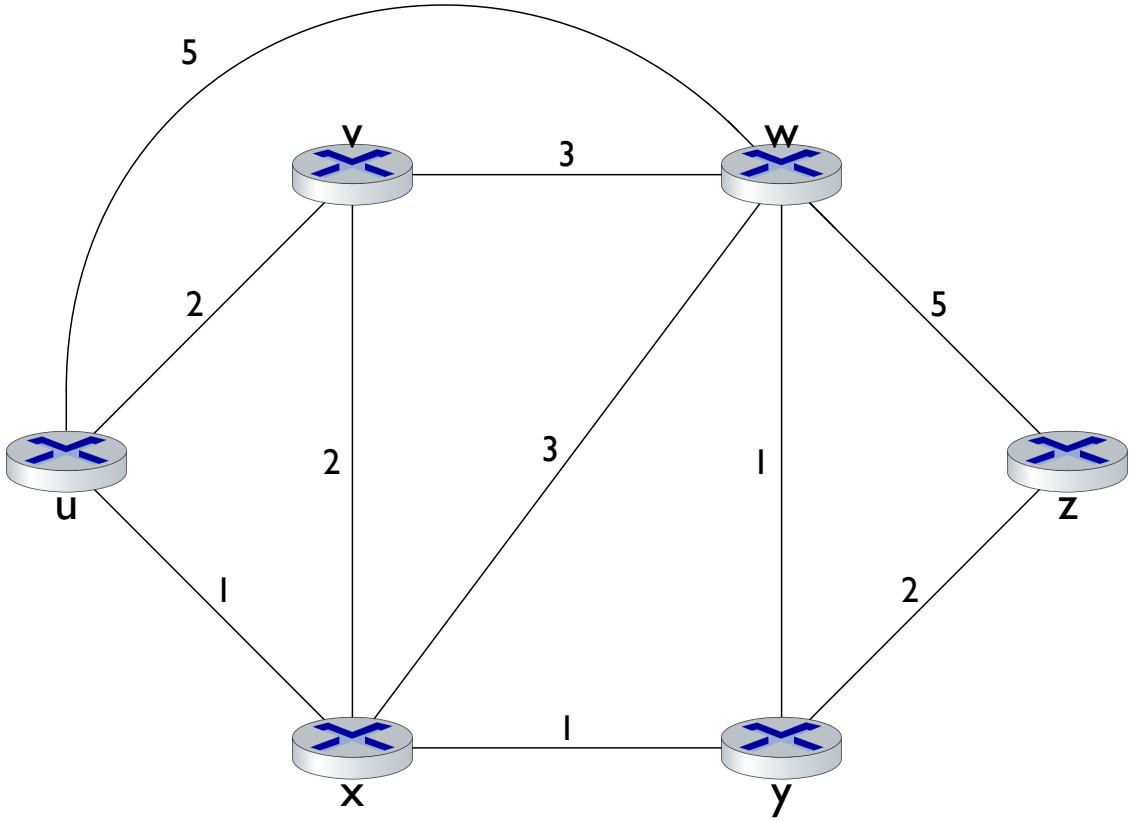


```
155.98.68.0/23  →  4
117.12.0.0/16   →  0
155.0.0.0/8     →  1
…
```

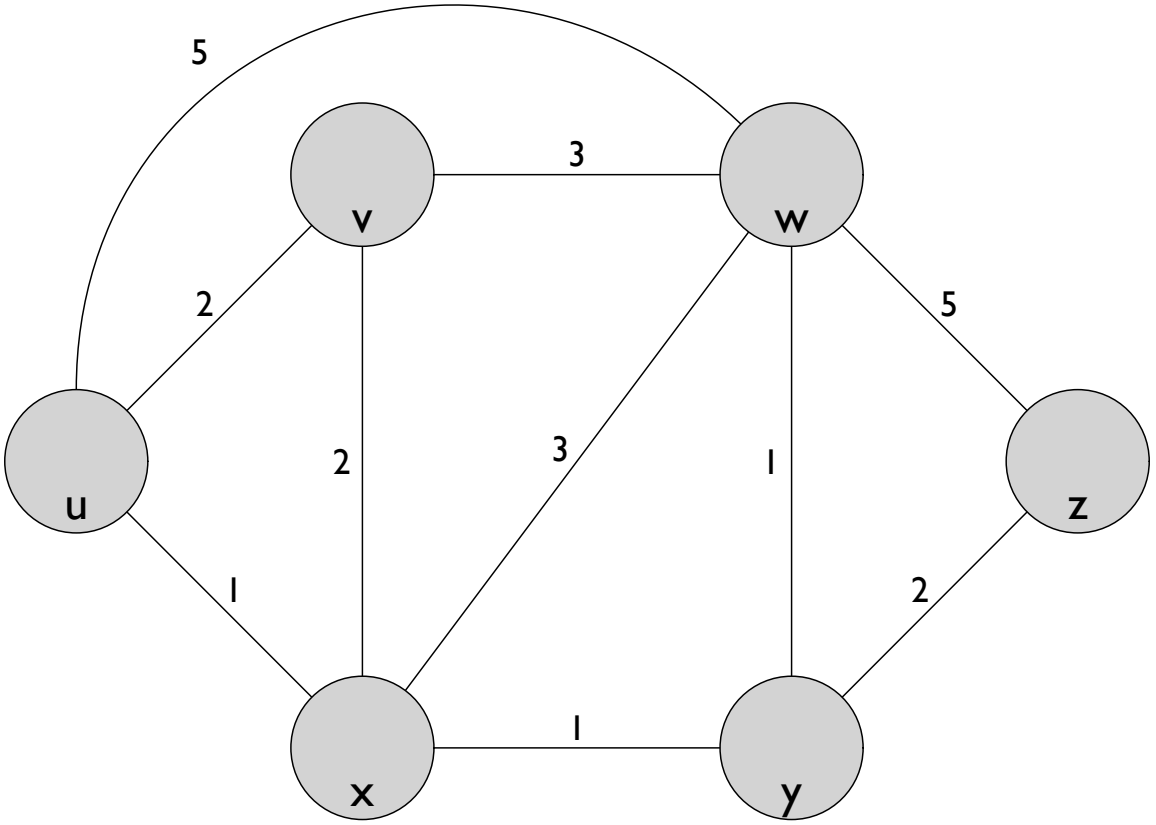
Finding Routes



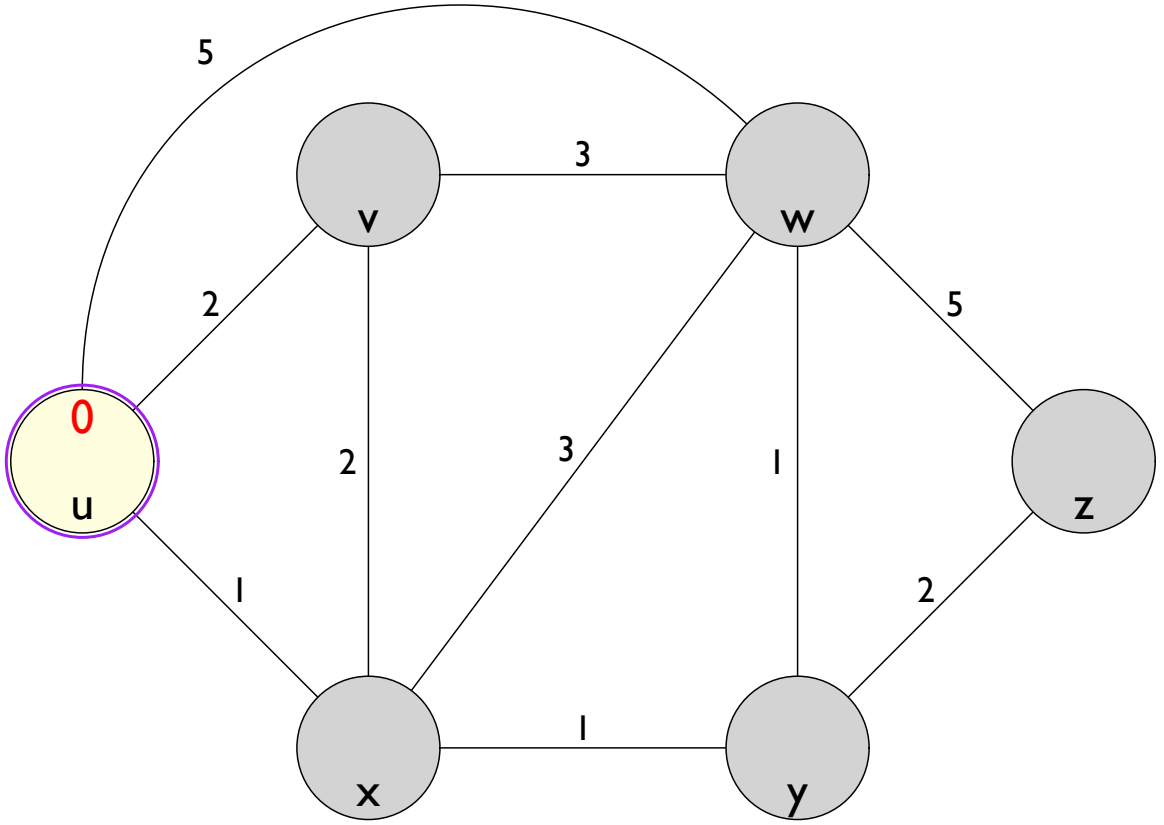
Finding Routes



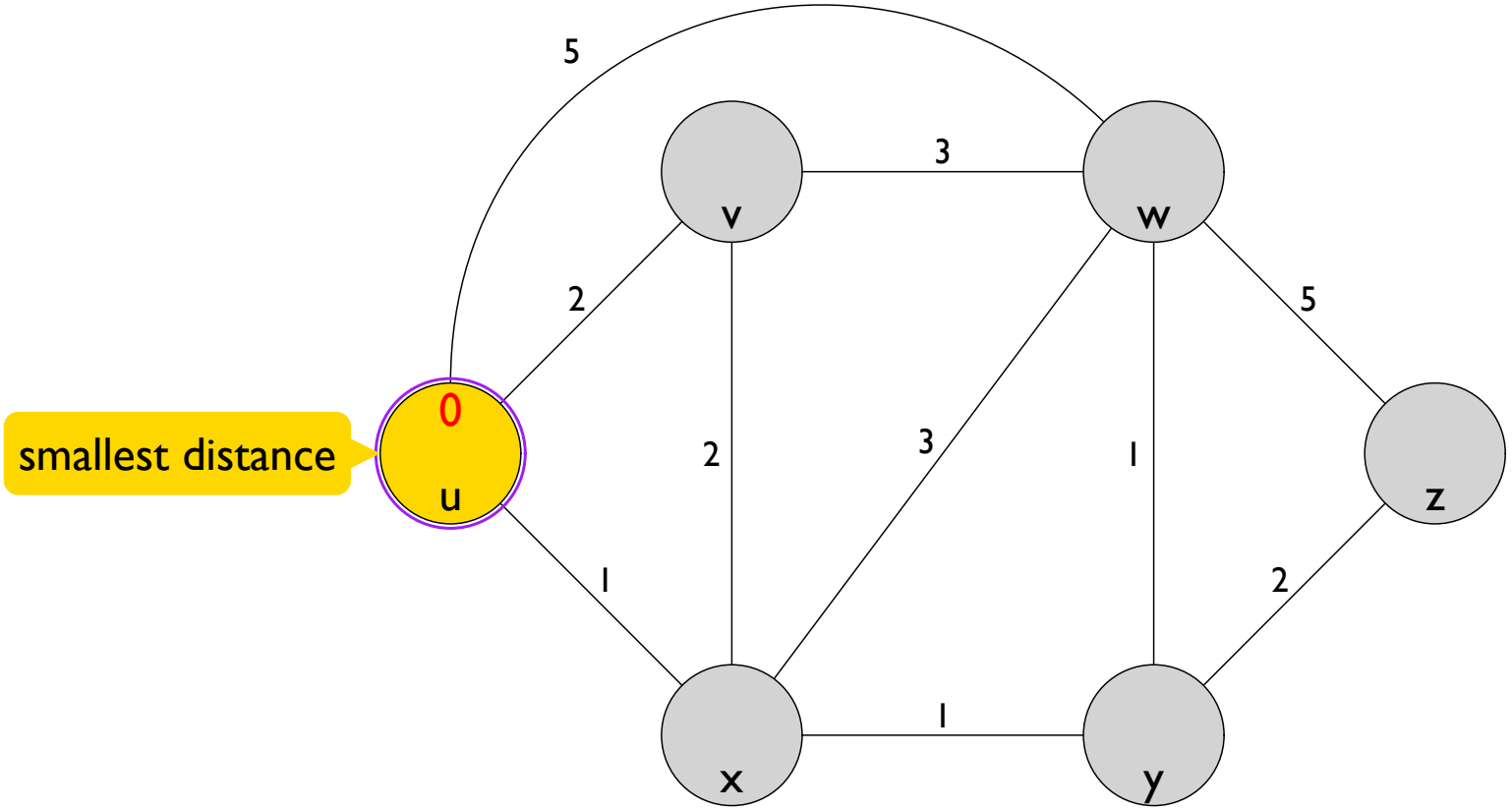
Finding Routes



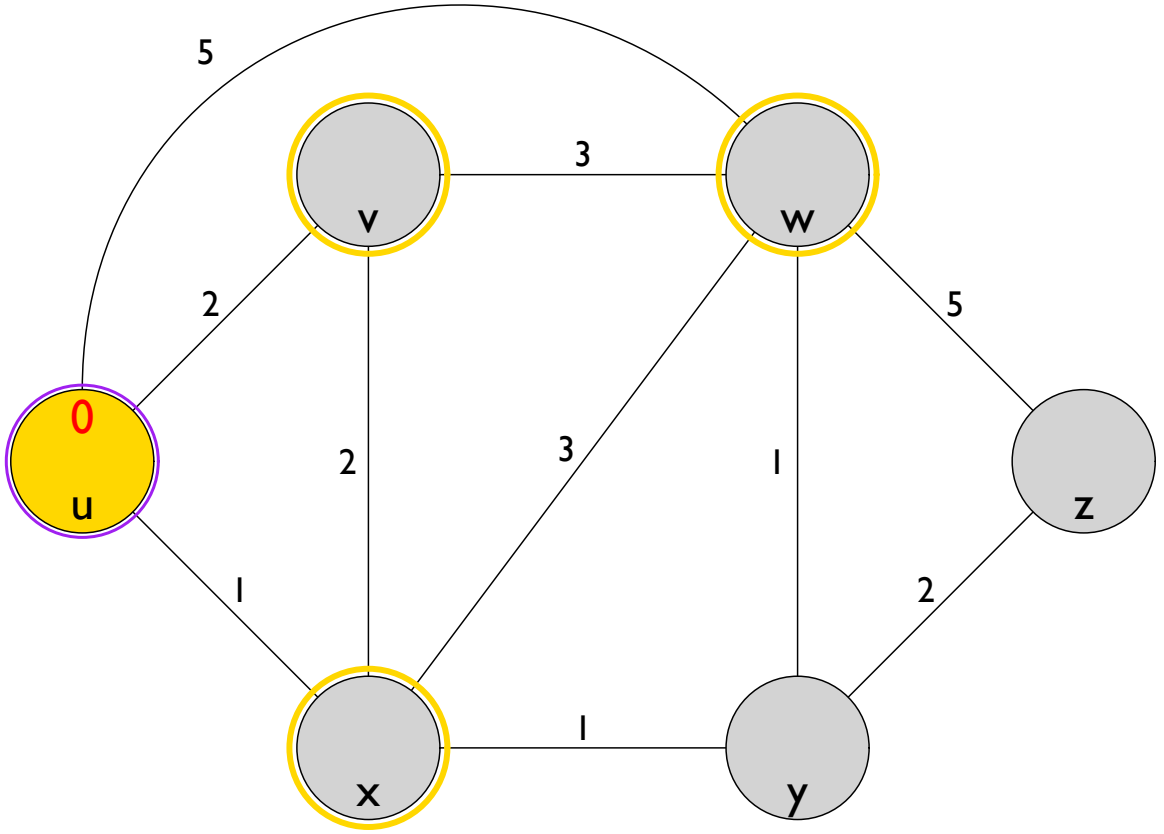
Finding Routes with Dijkstra's



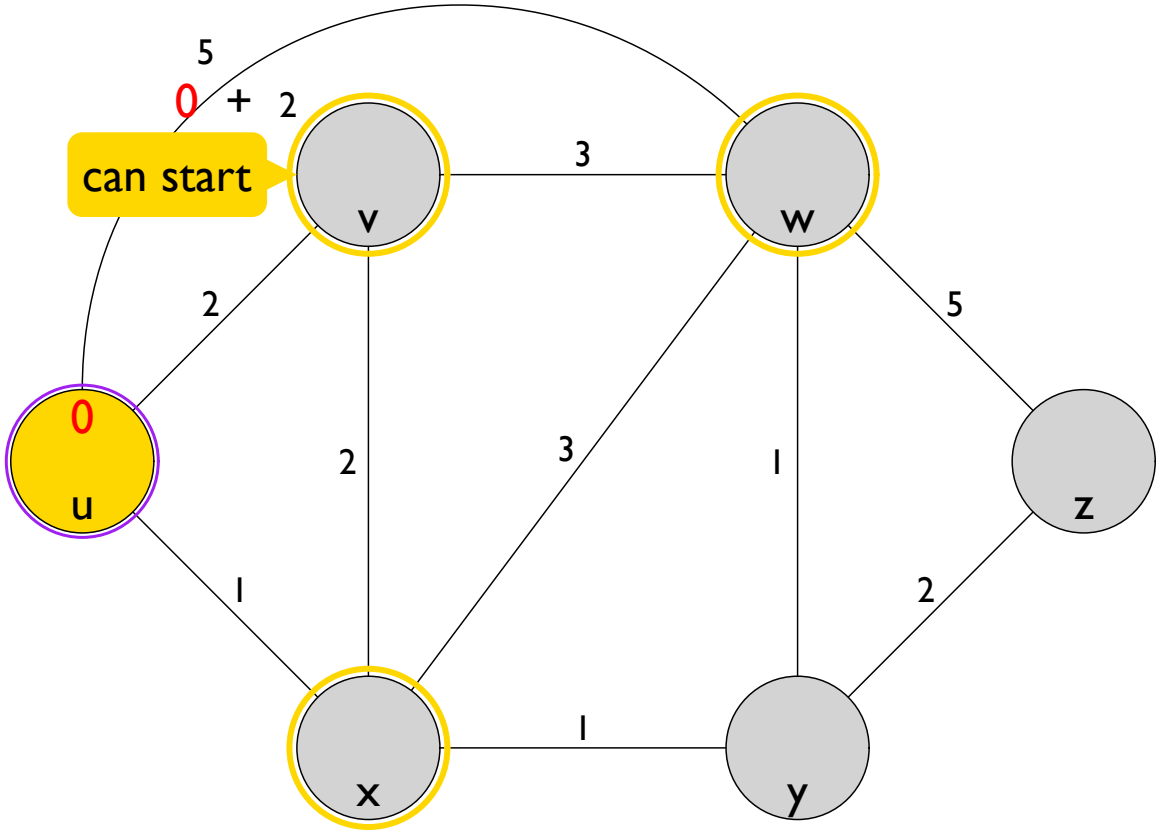
Finding Routes with Dijkstra's



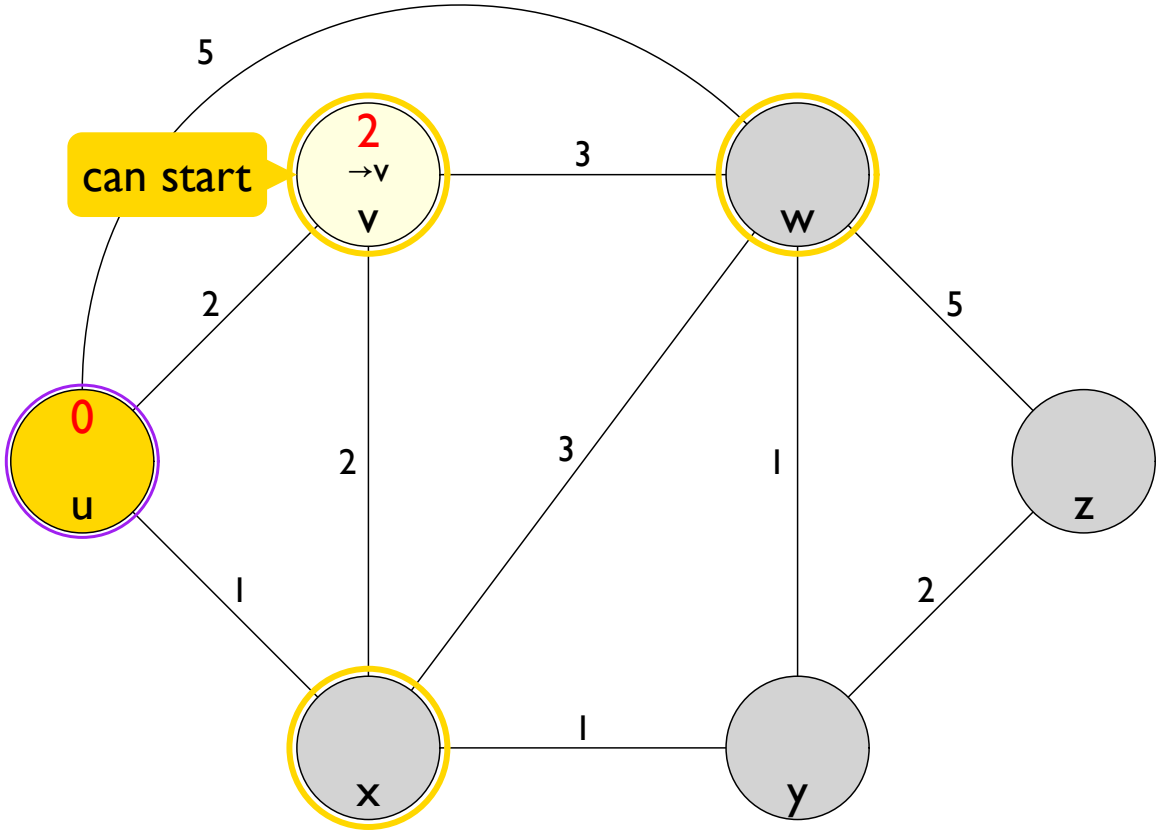
Finding Routes with Dijkstra's



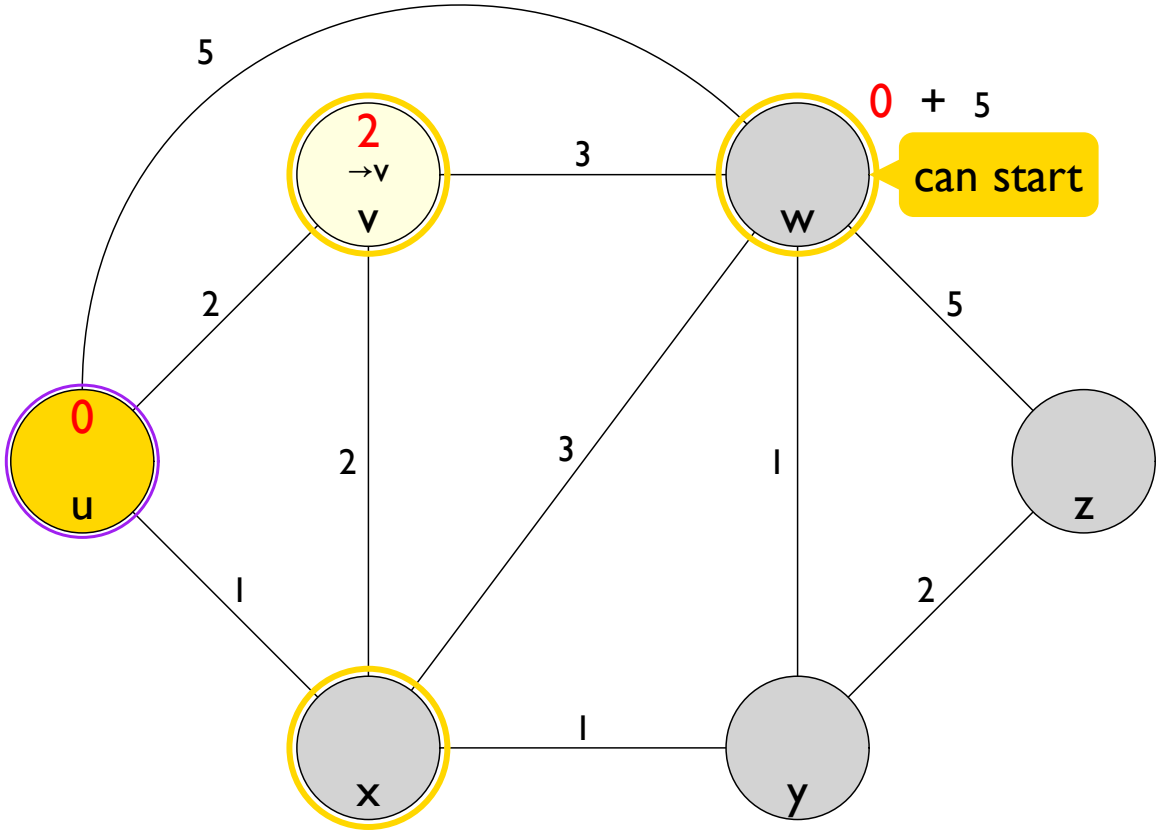
Finding Routes with Dijkstra's



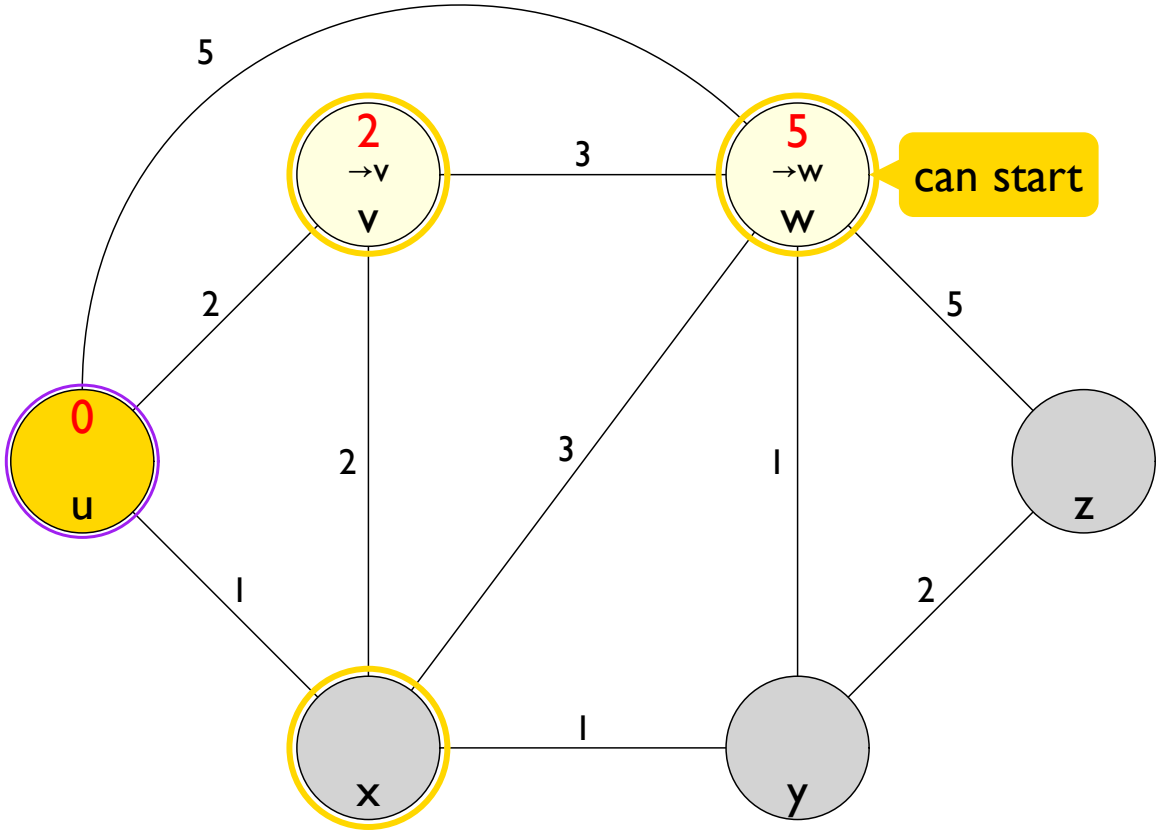
Finding Routes with Dijkstra's



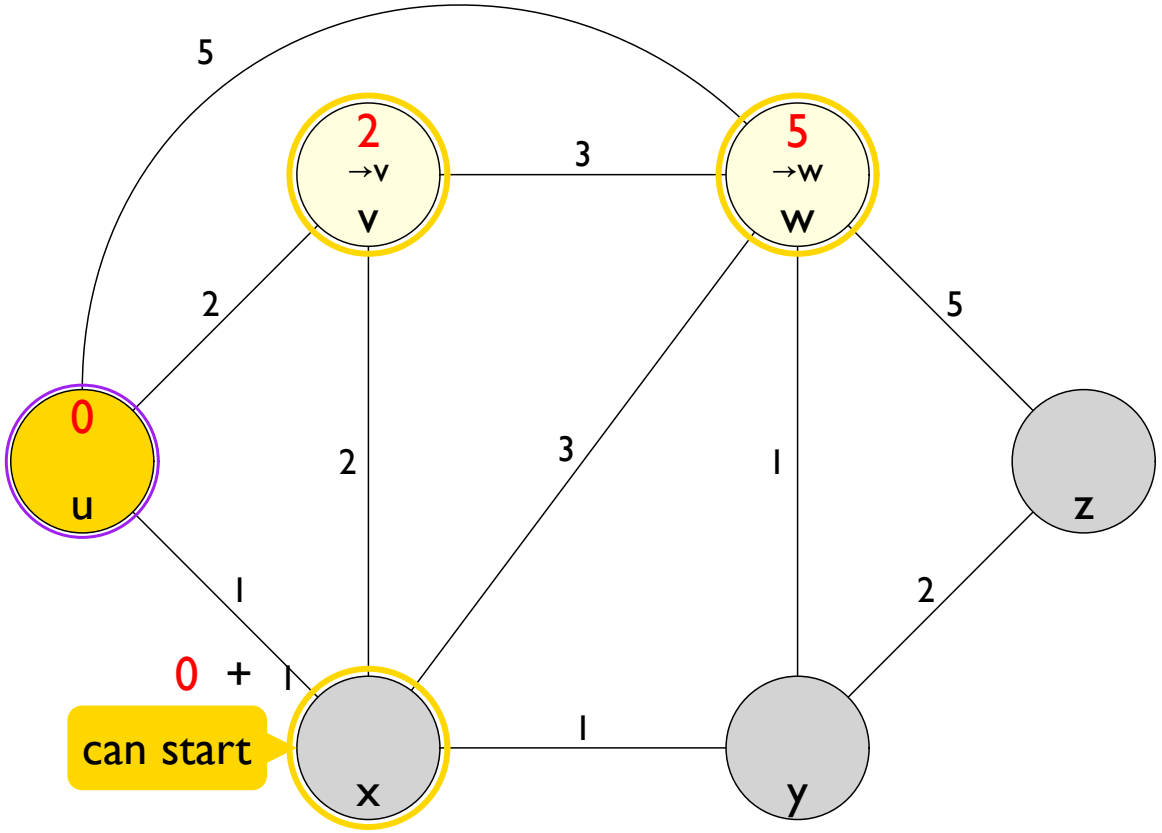
Finding Routes with Dijkstra's



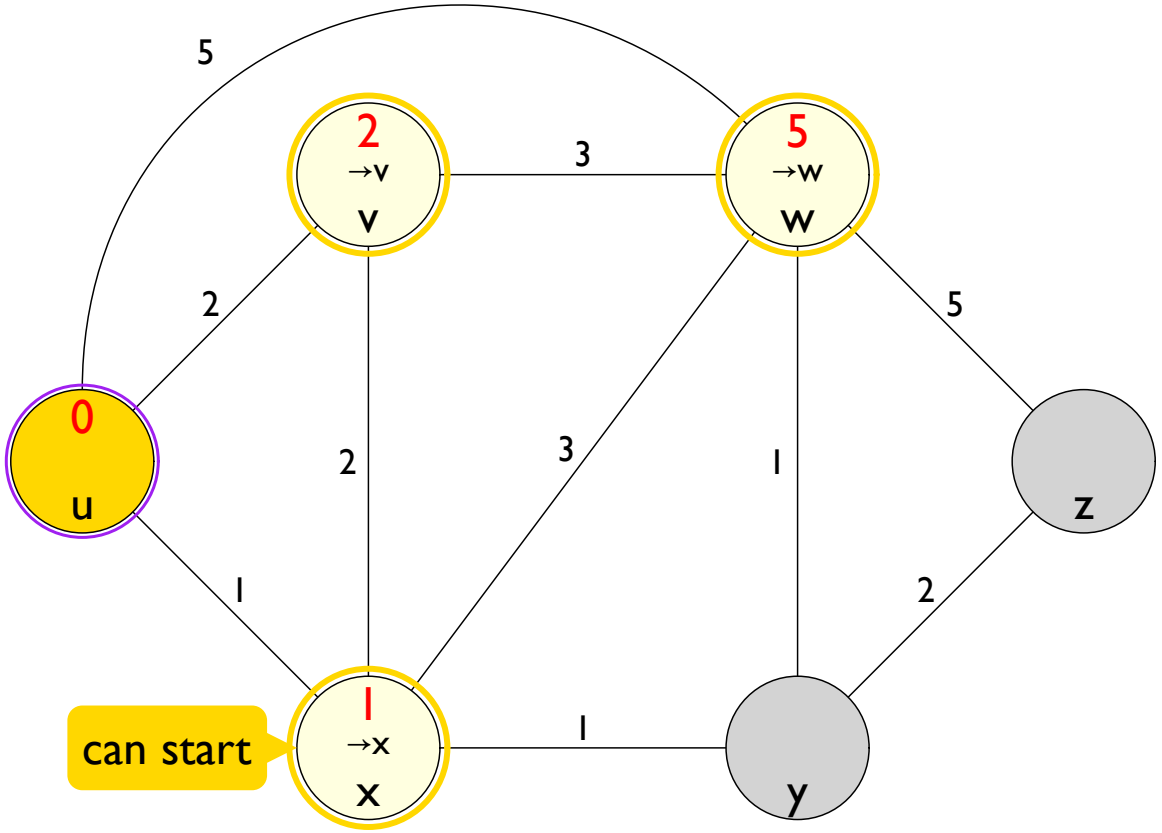
Finding Routes with Dijkstra's



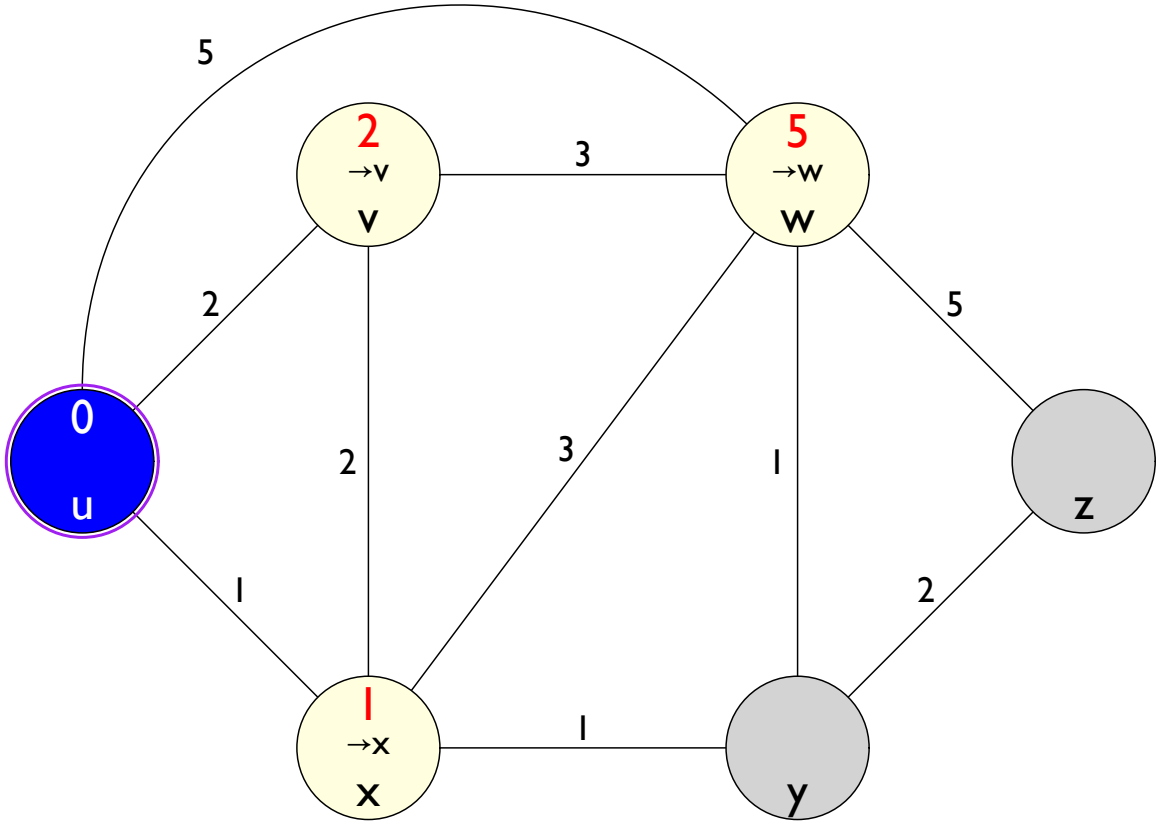
Finding Routes with Dijkstra's



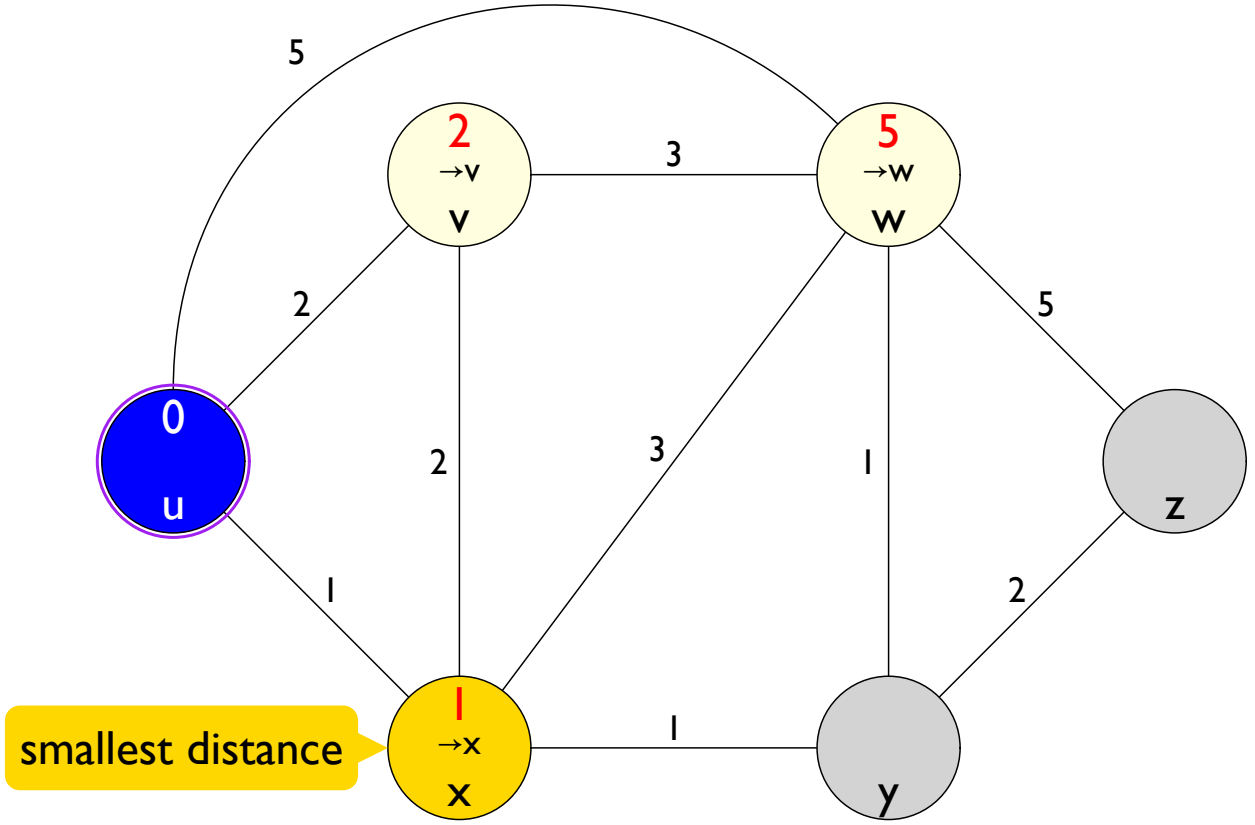
Finding Routes with Dijkstra's



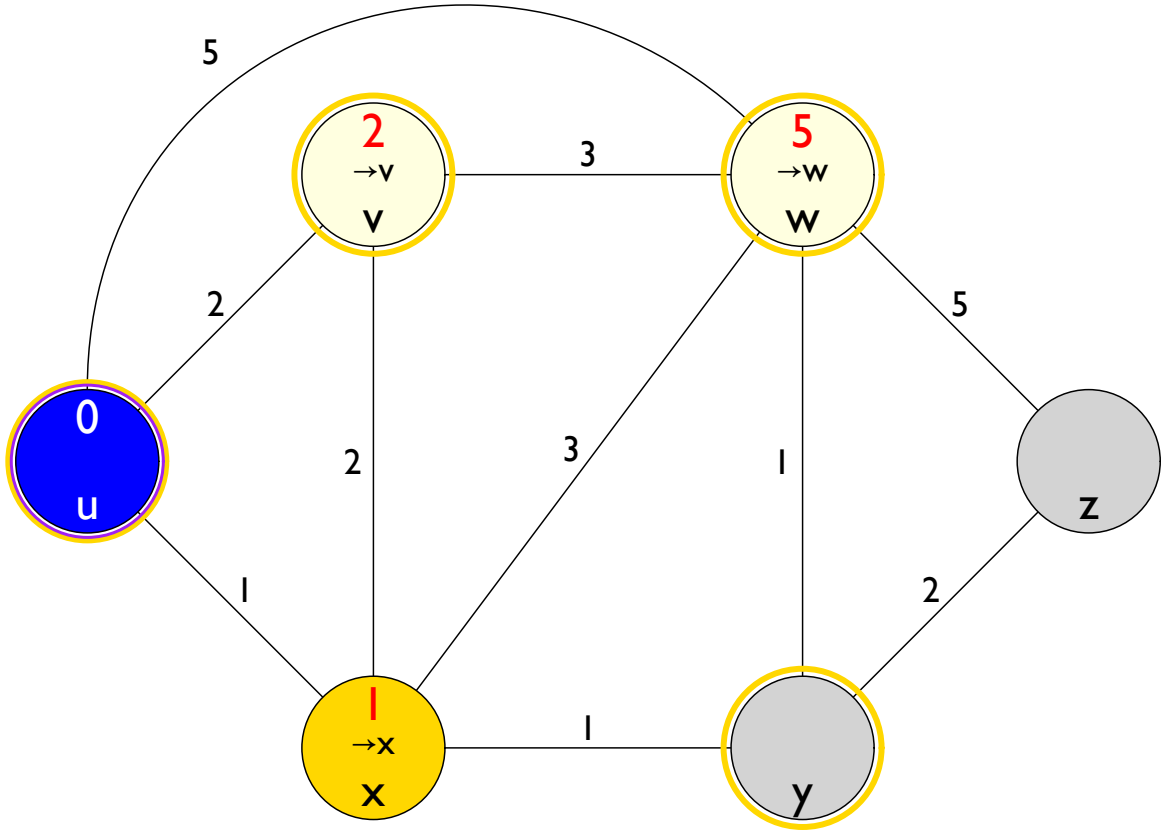
Finding Routes with Dijkstra's



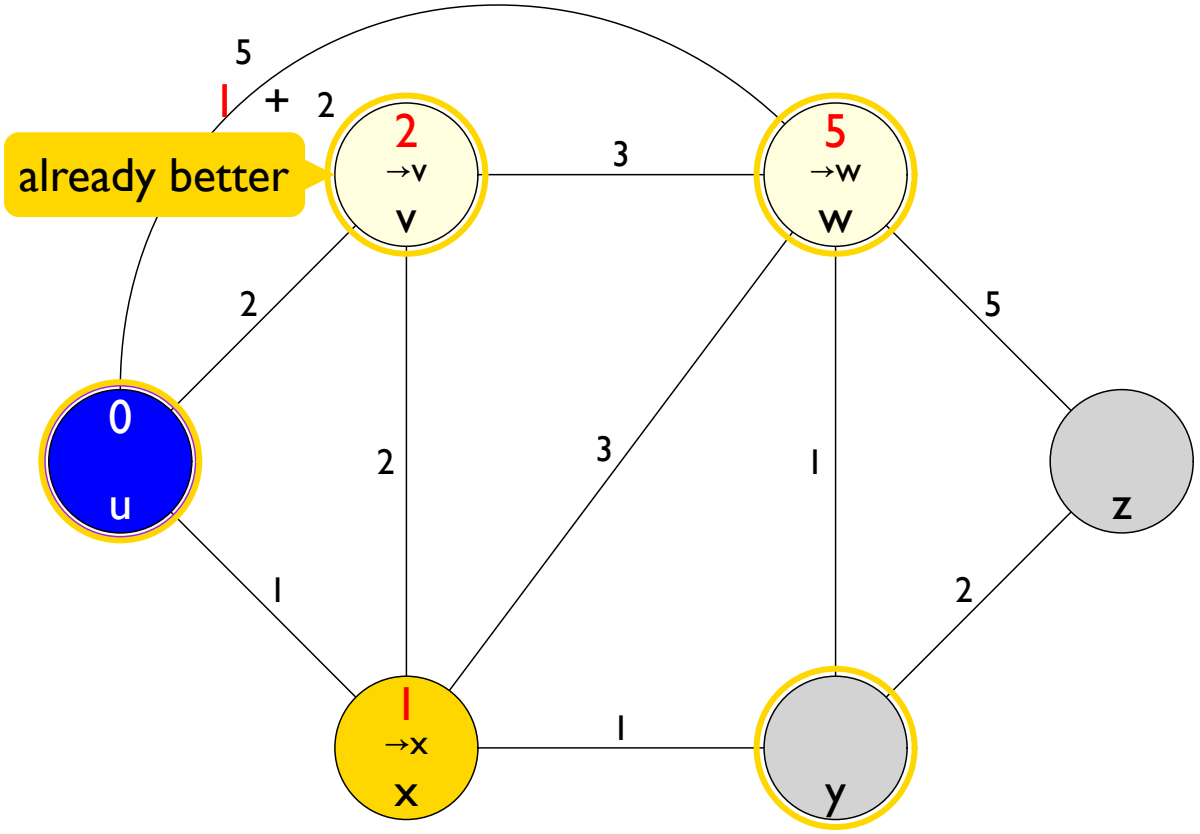
Finding Routes with Dijkstra's



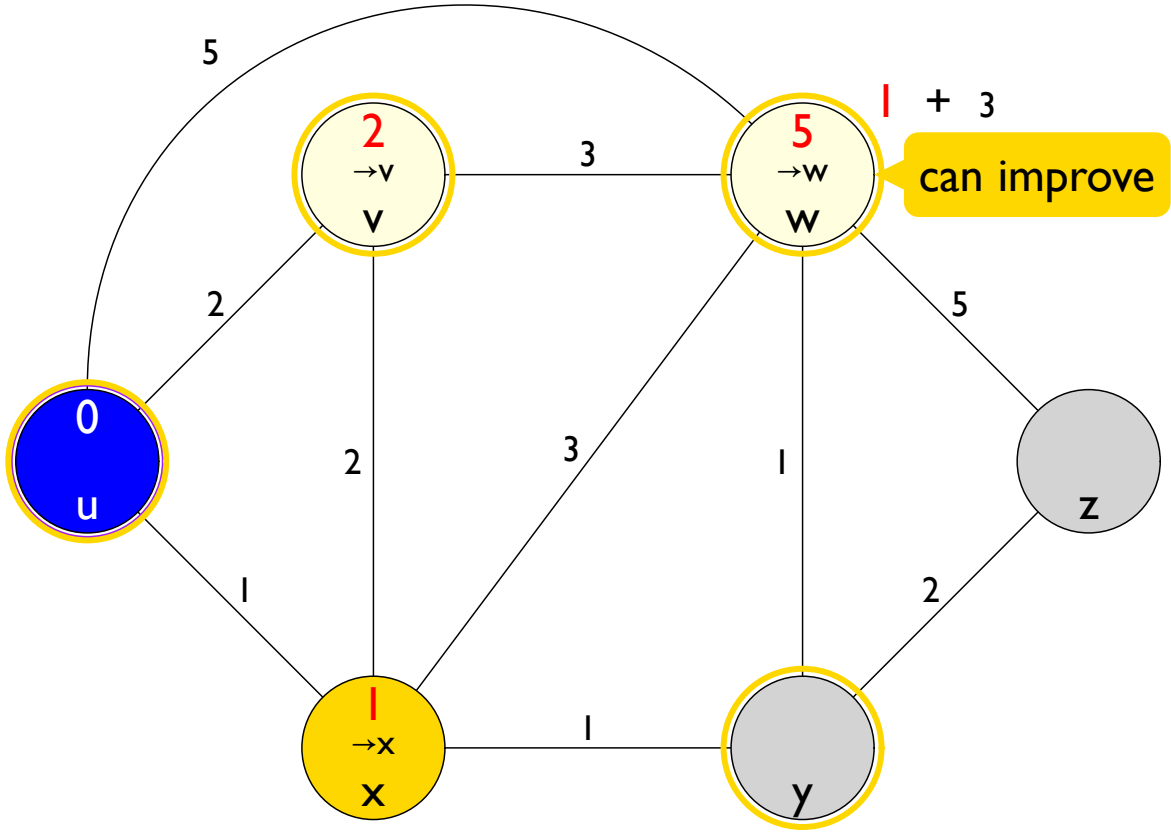
Finding Routes with Dijkstra's



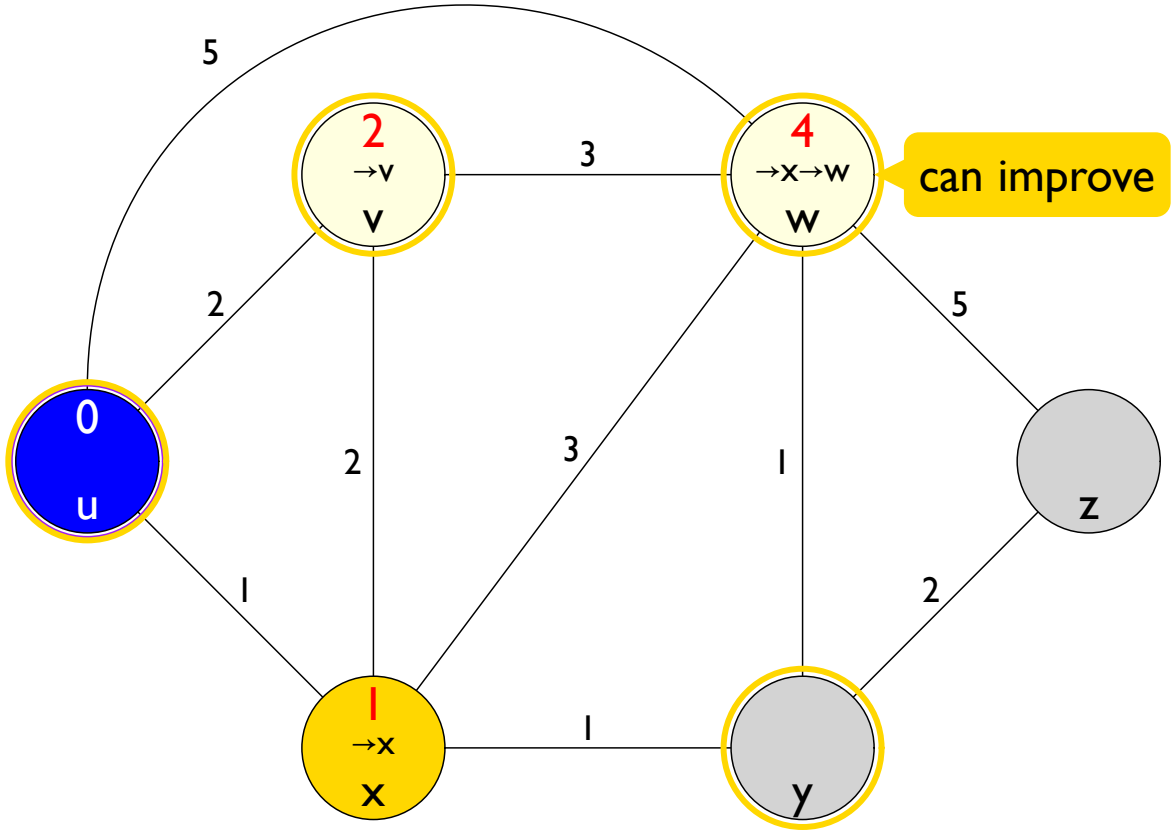
Finding Routes with Dijkstra's



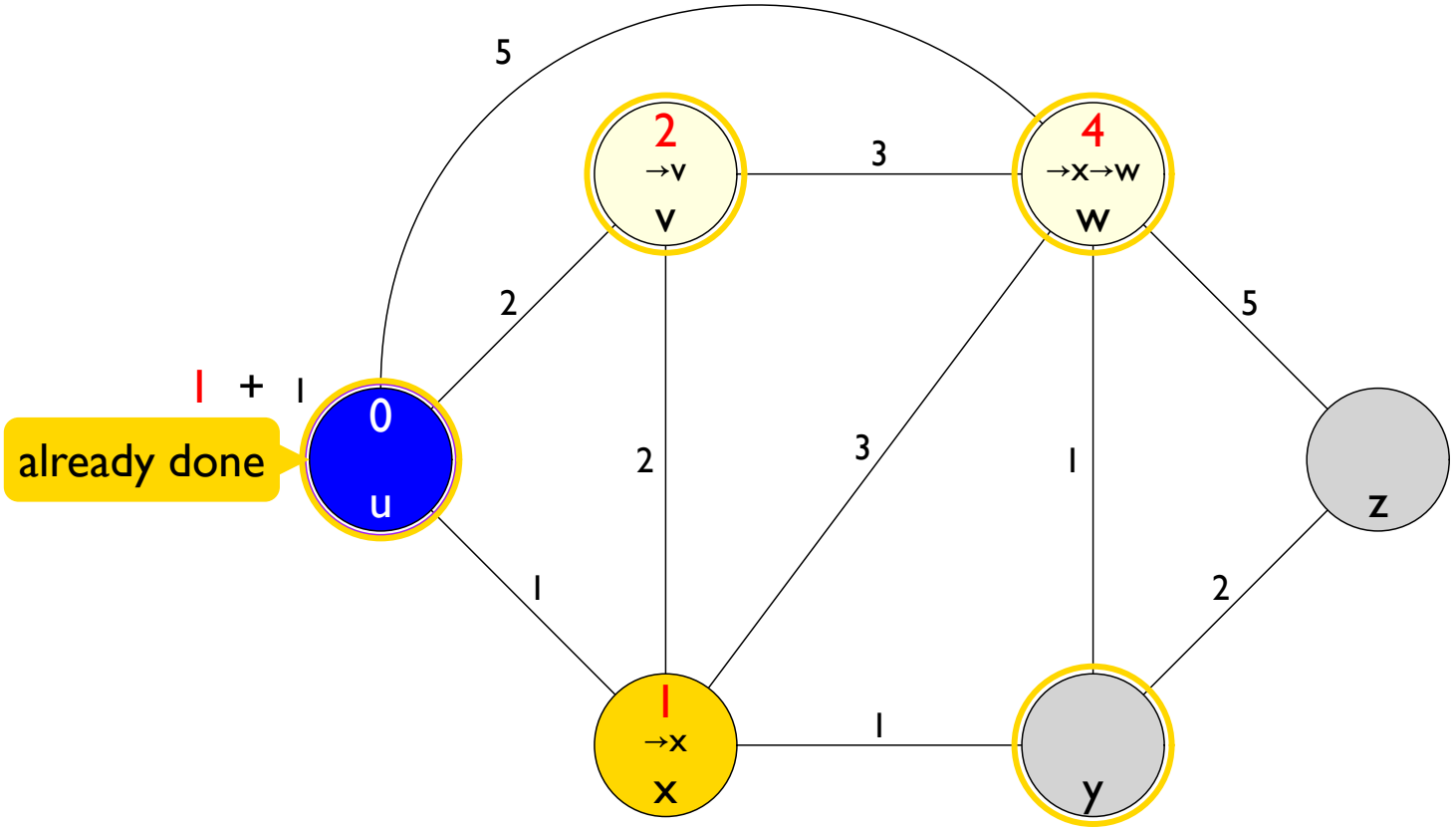
Finding Routes with Dijkstra's



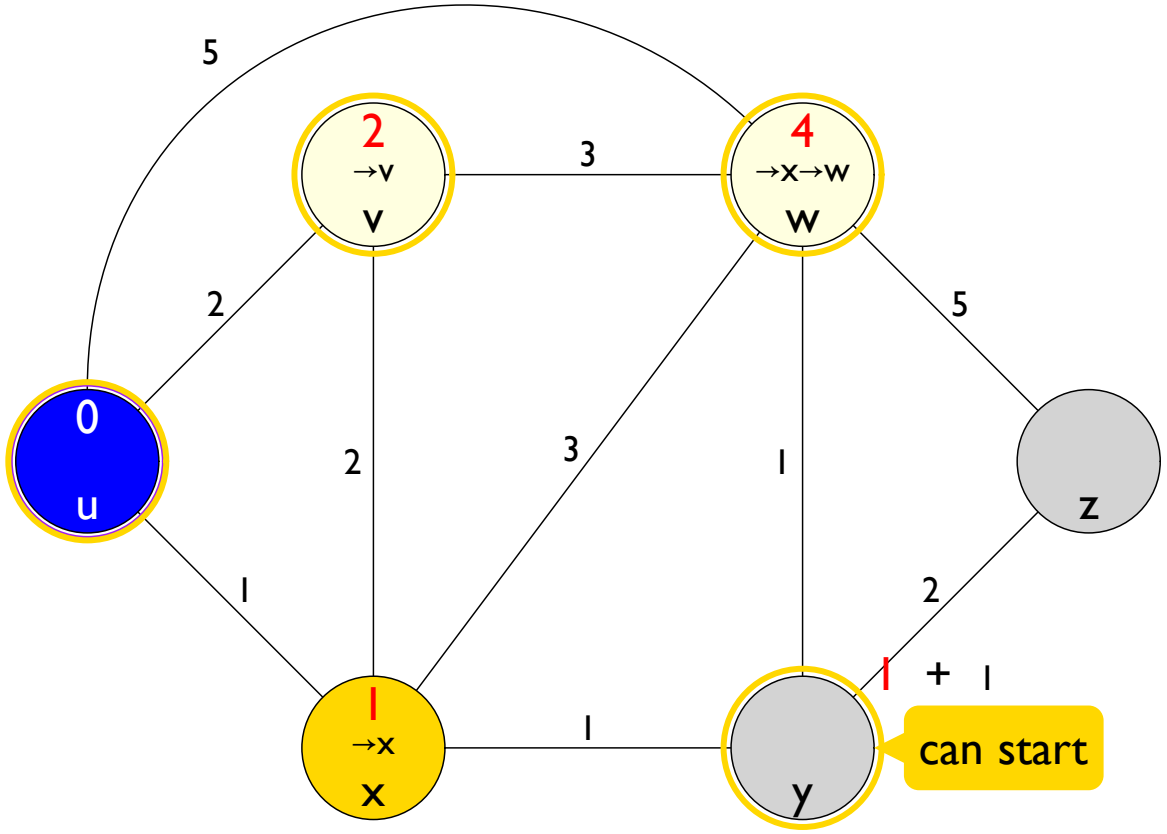
Finding Routes with Dijkstra's



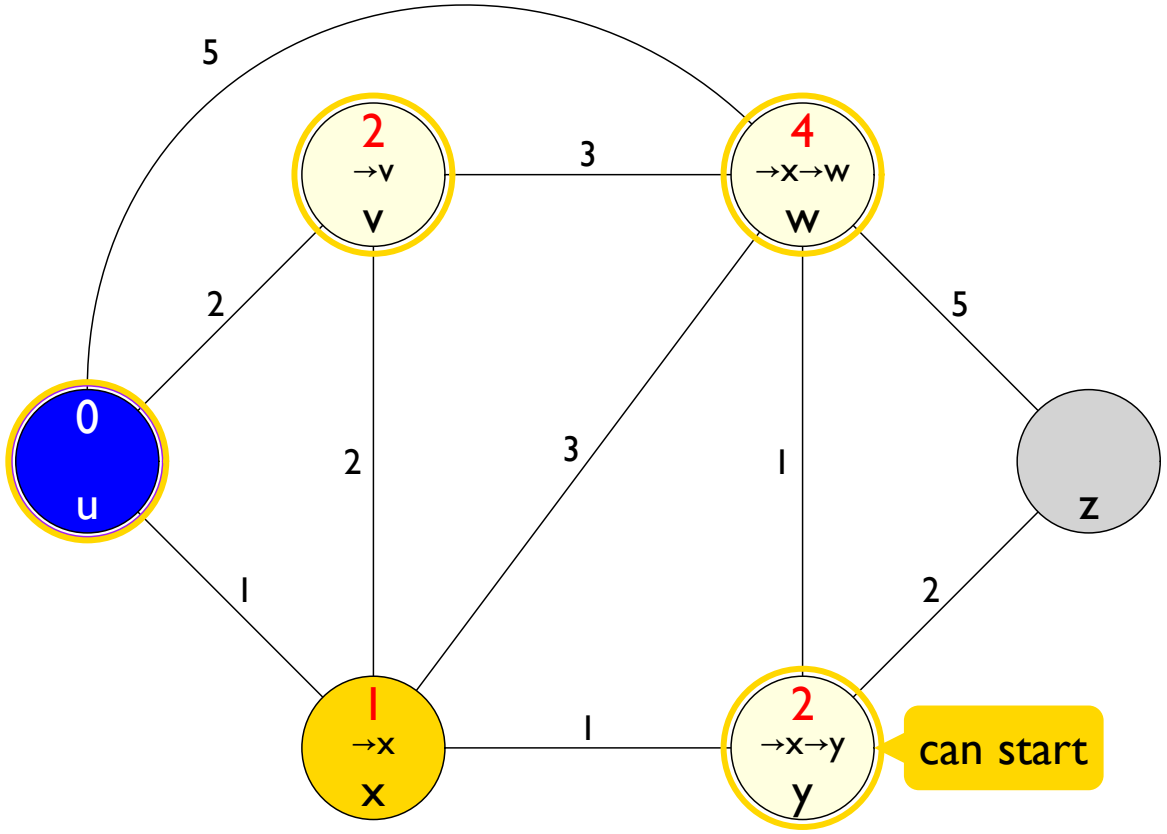
Finding Routes with Dijkstra's



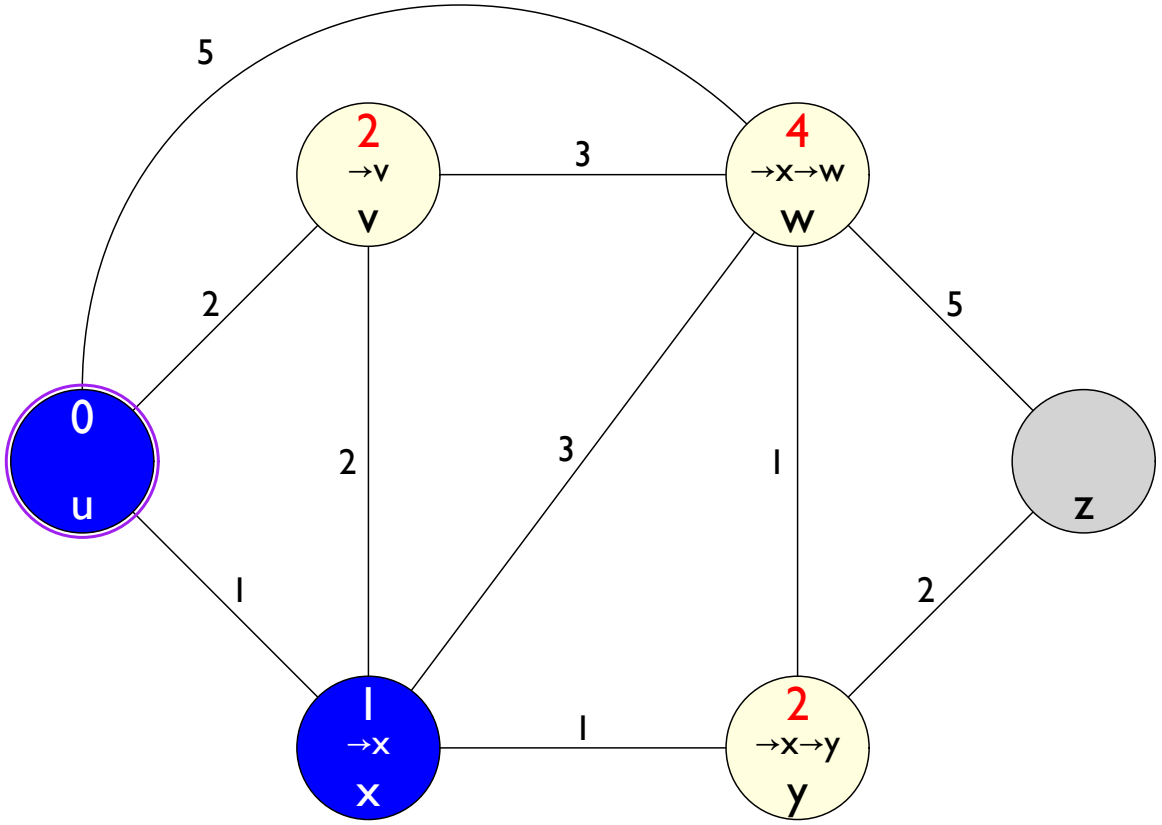
Finding Routes with Dijkstra's



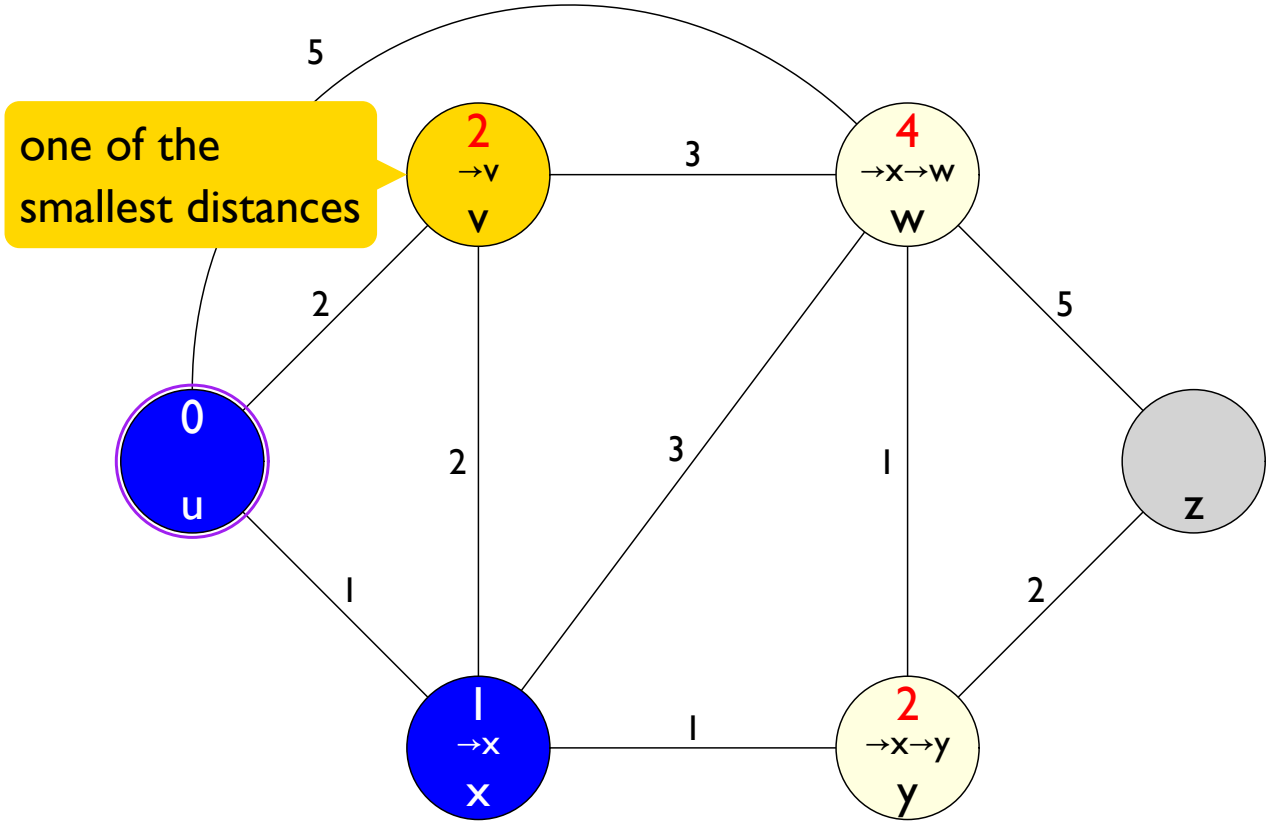
Finding Routes with Dijkstra's



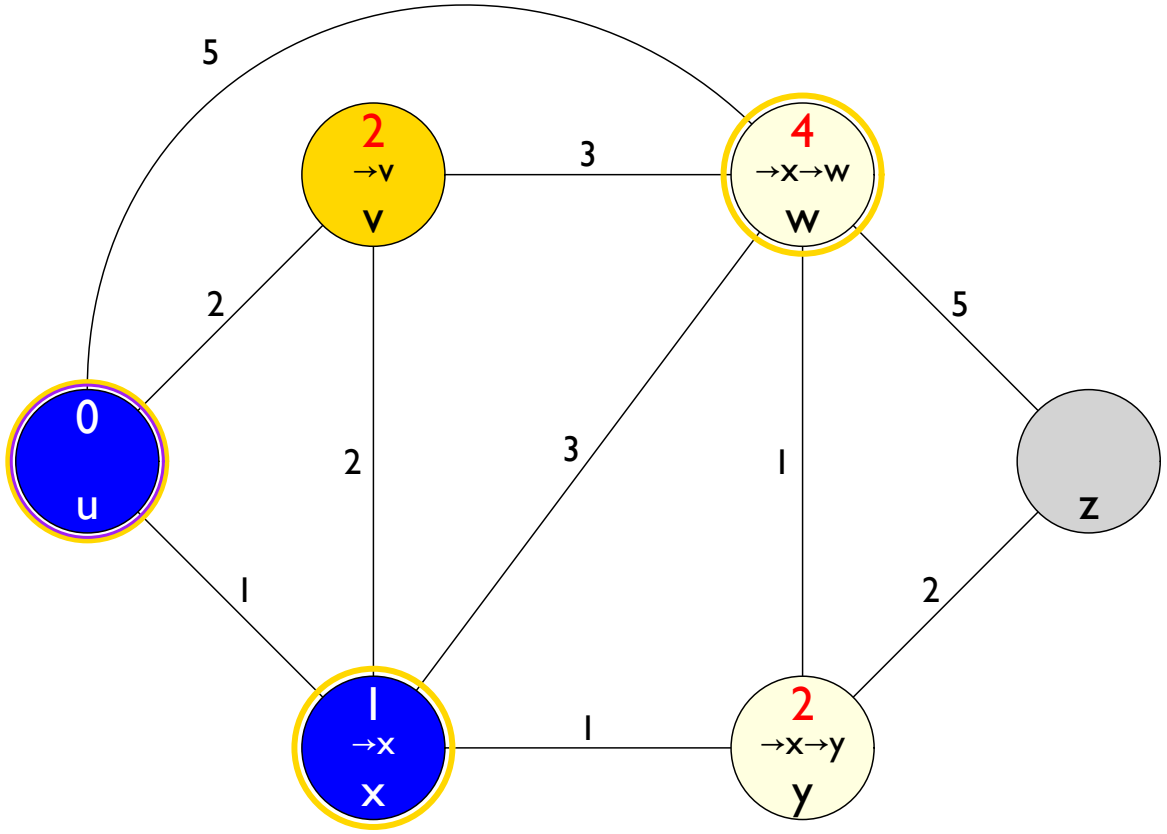
Finding Routes with Dijkstra's



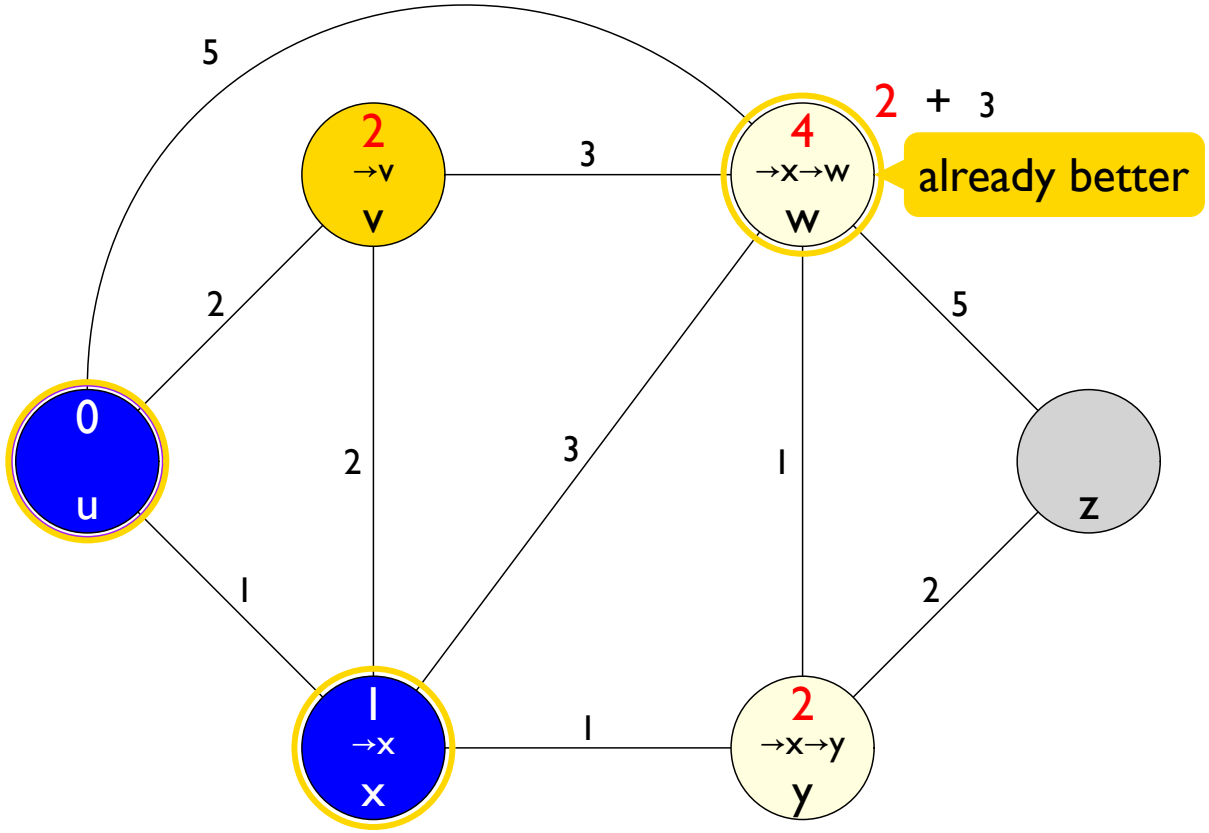
Finding Routes with Dijkstra's



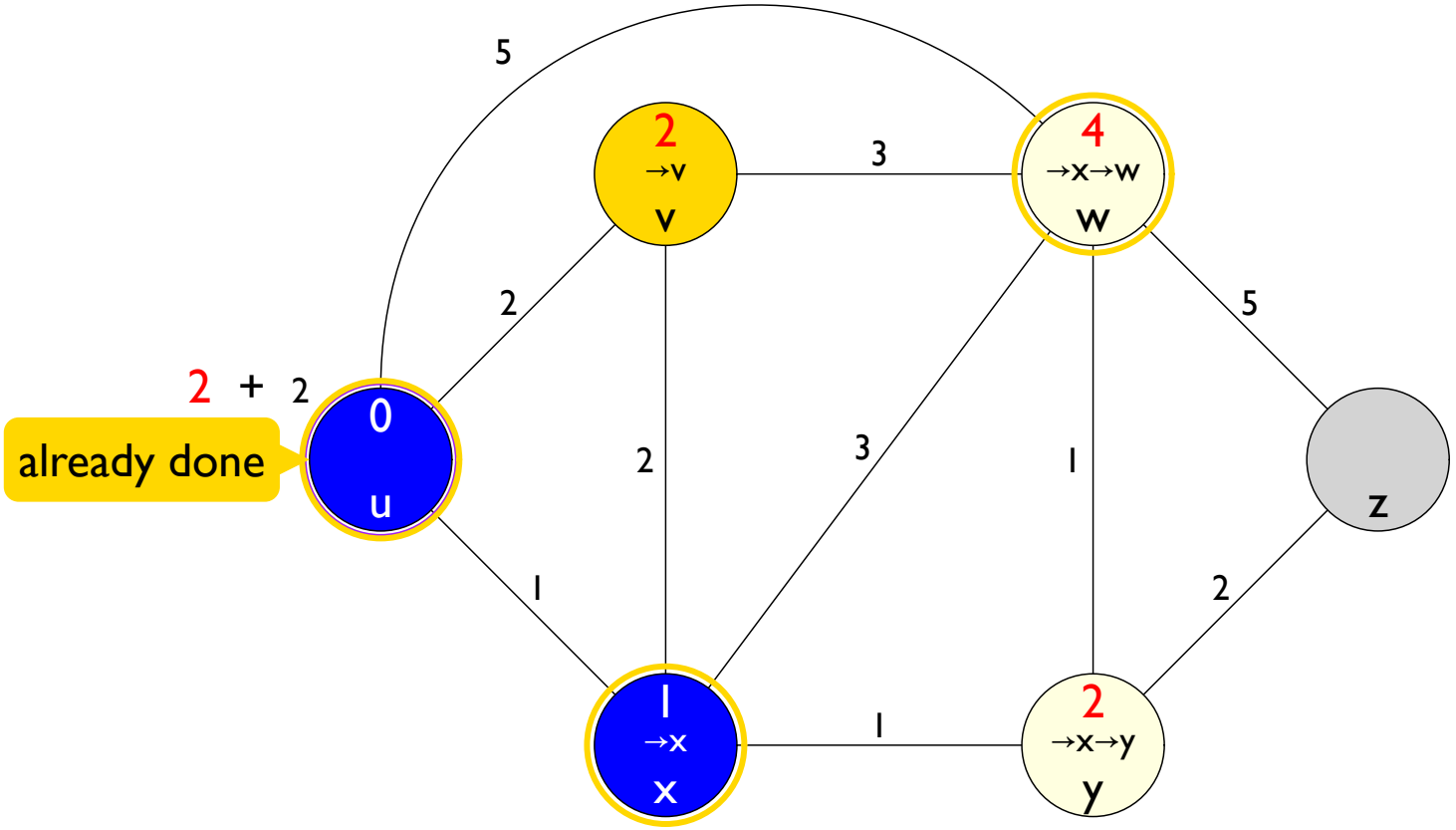
Finding Routes with Dijkstra's



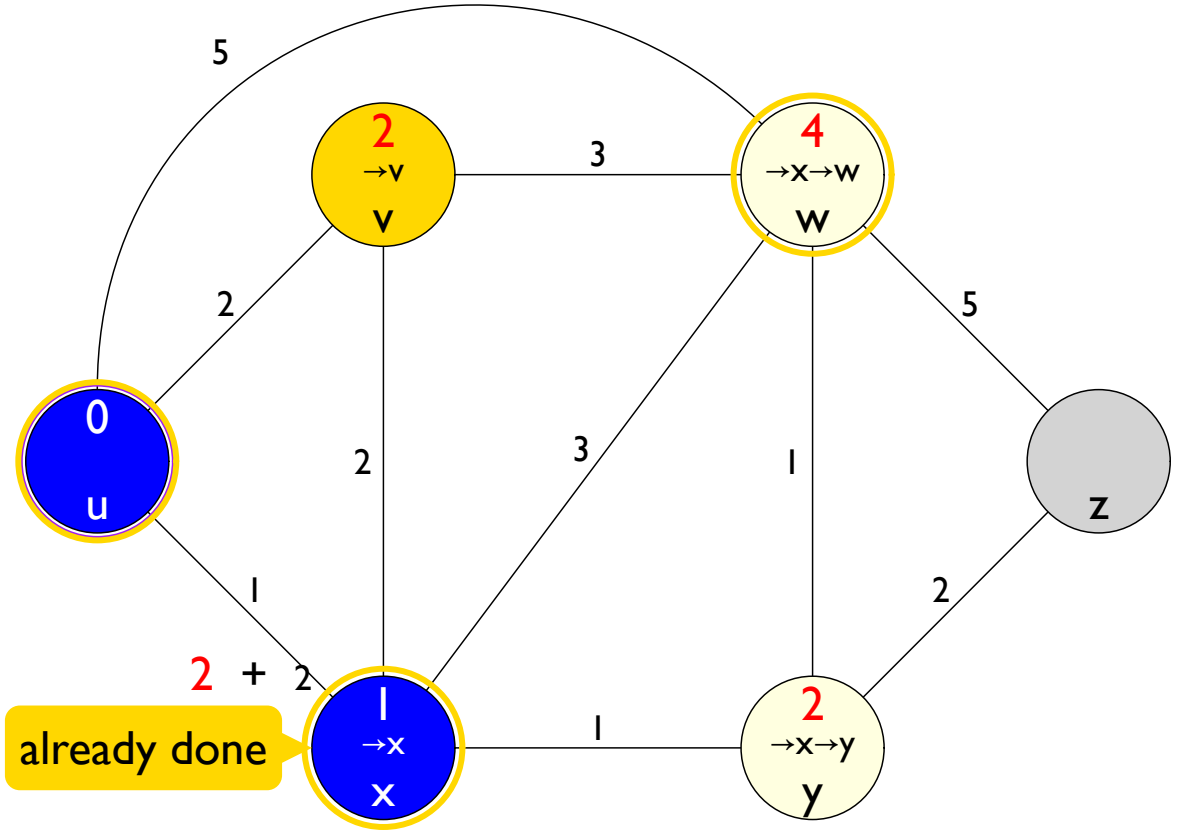
Finding Routes with Dijkstra's



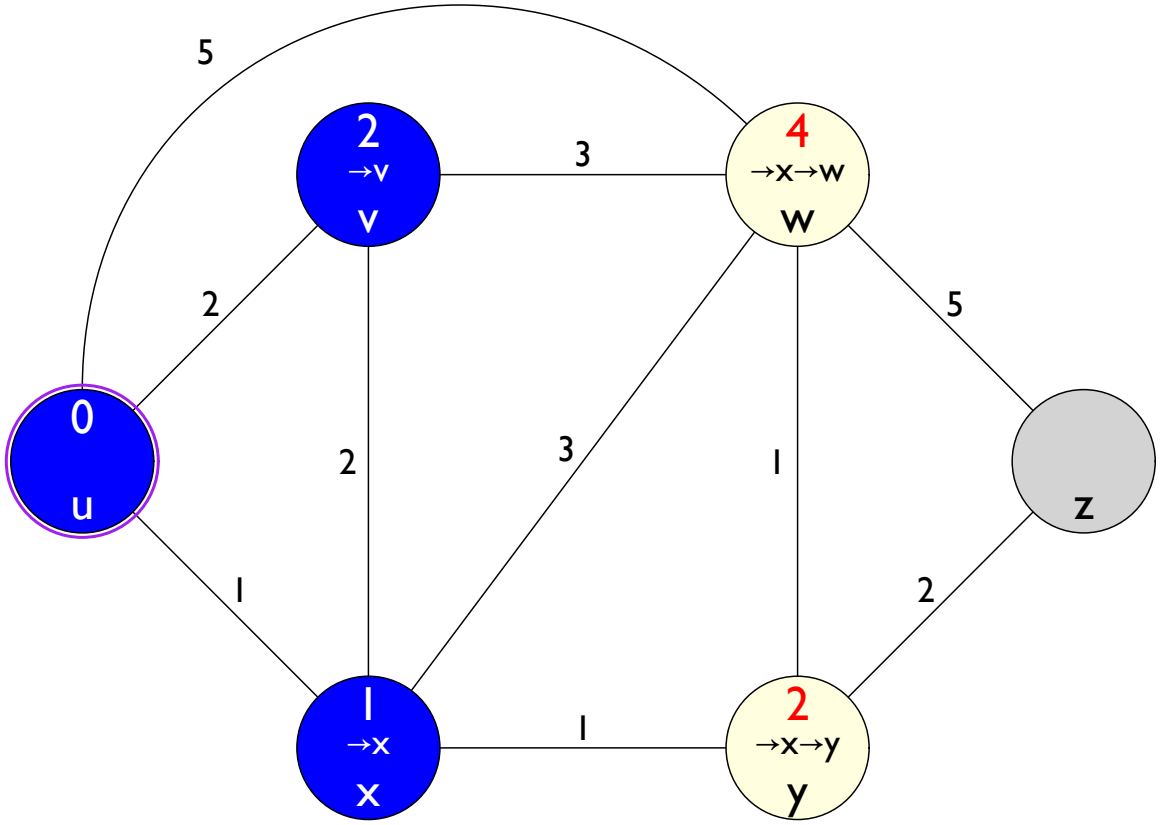
Finding Routes with Dijkstra's



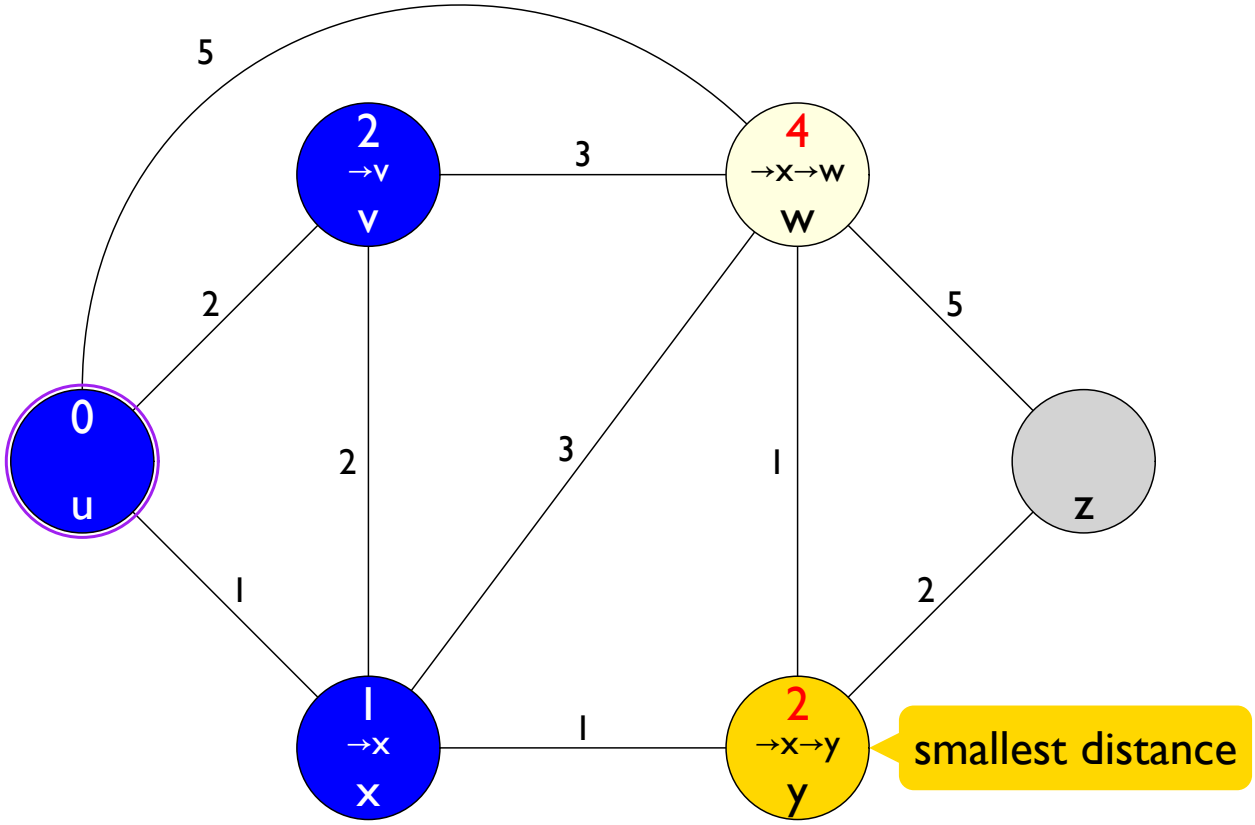
Finding Routes with Dijkstra's



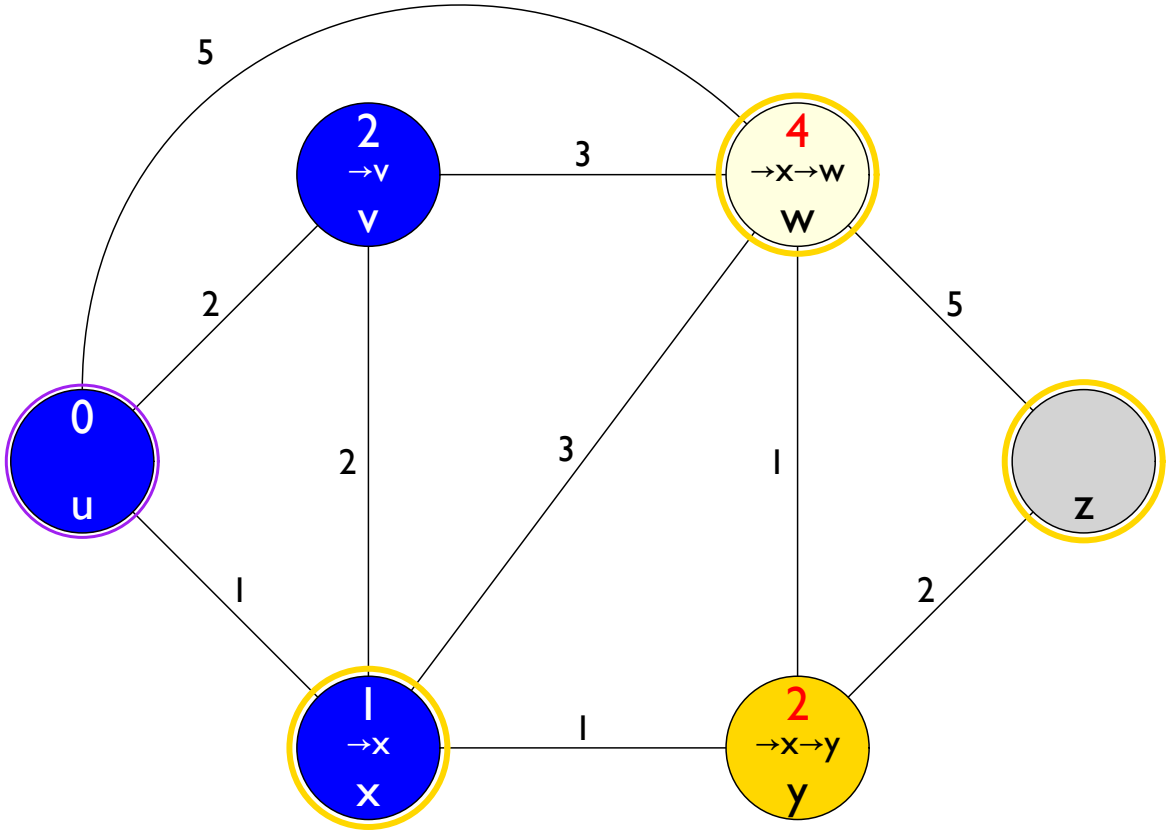
Finding Routes with Dijkstra's



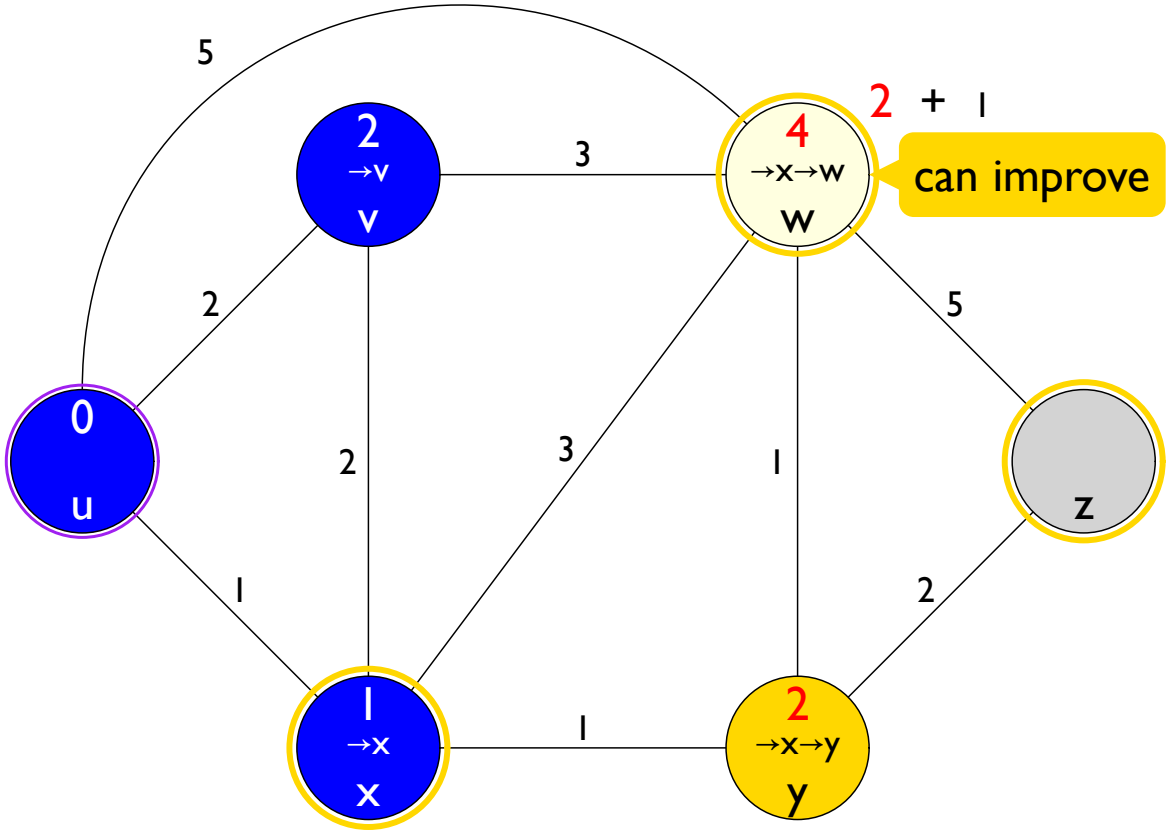
Finding Routes with Dijkstra's



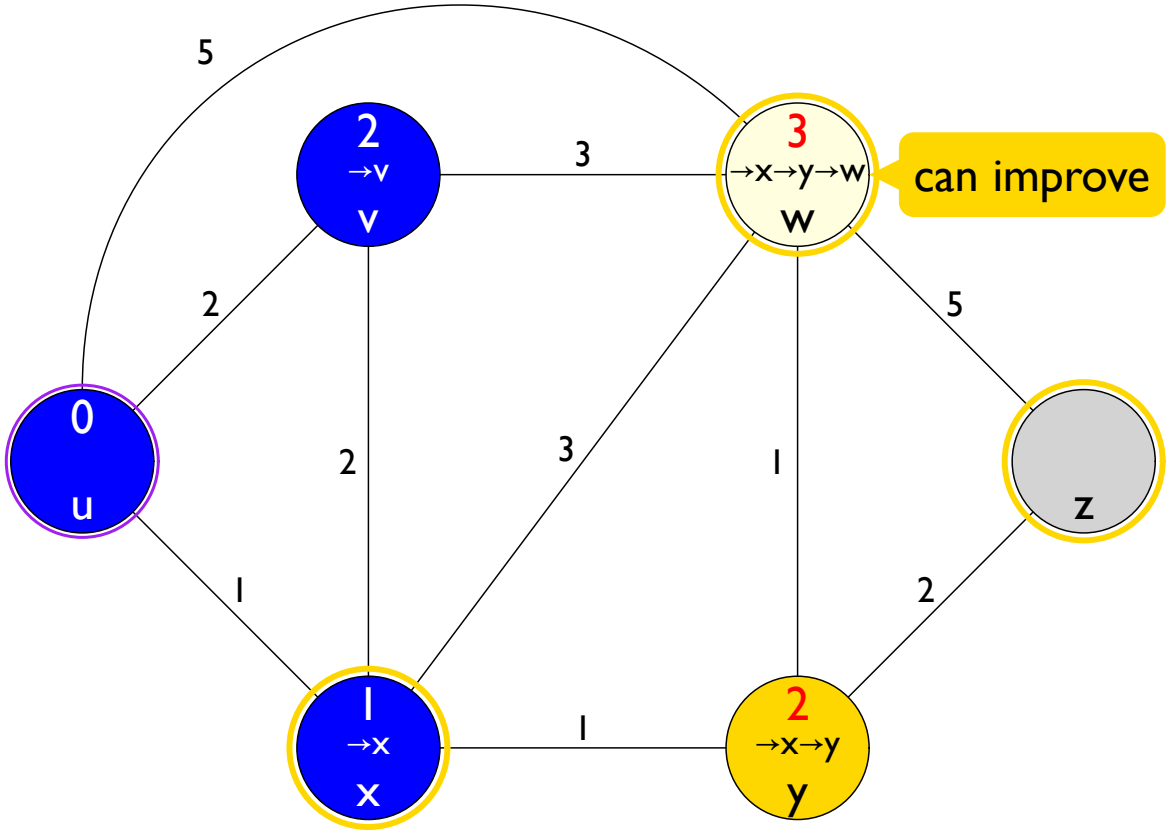
Finding Routes with Dijkstra's



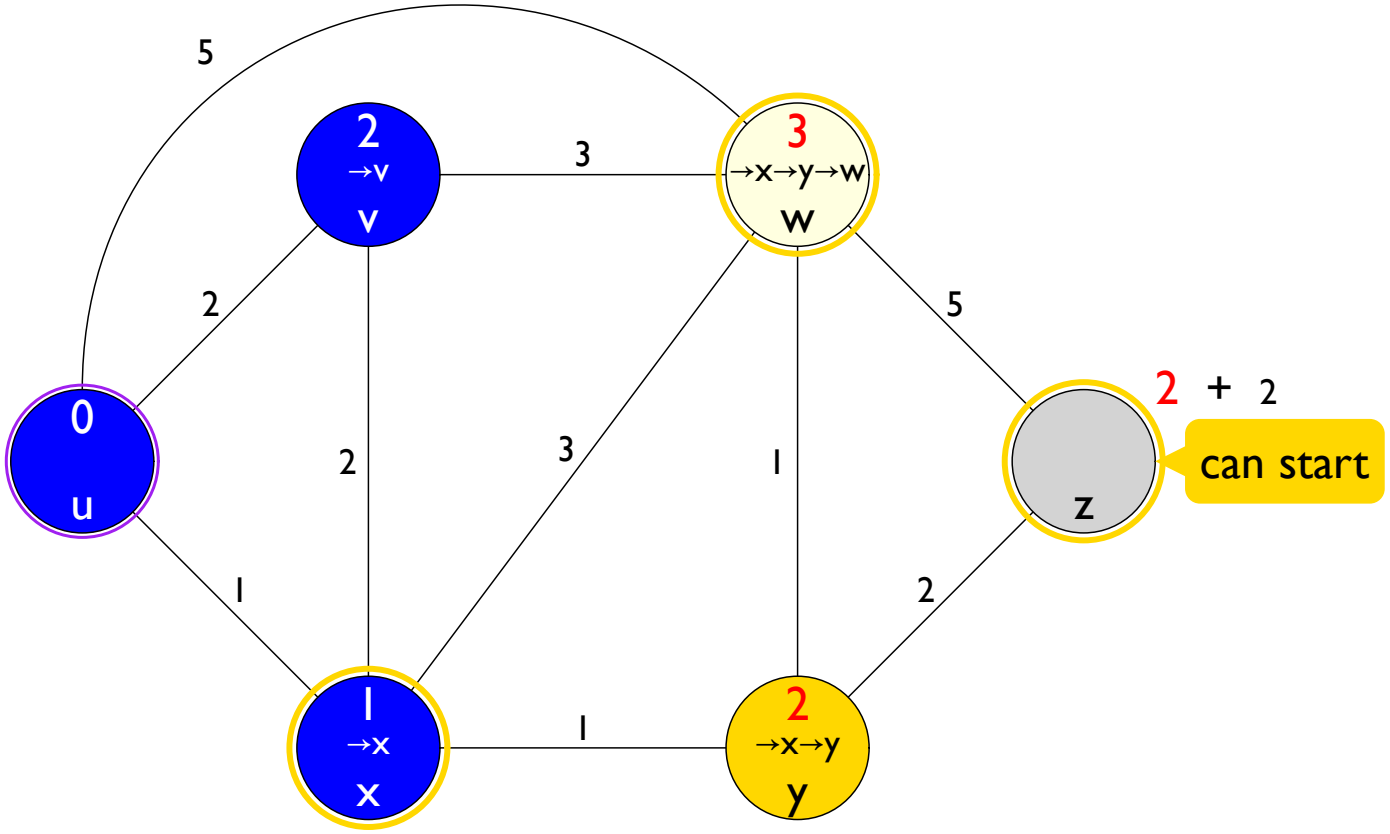
Finding Routes with Dijkstra's



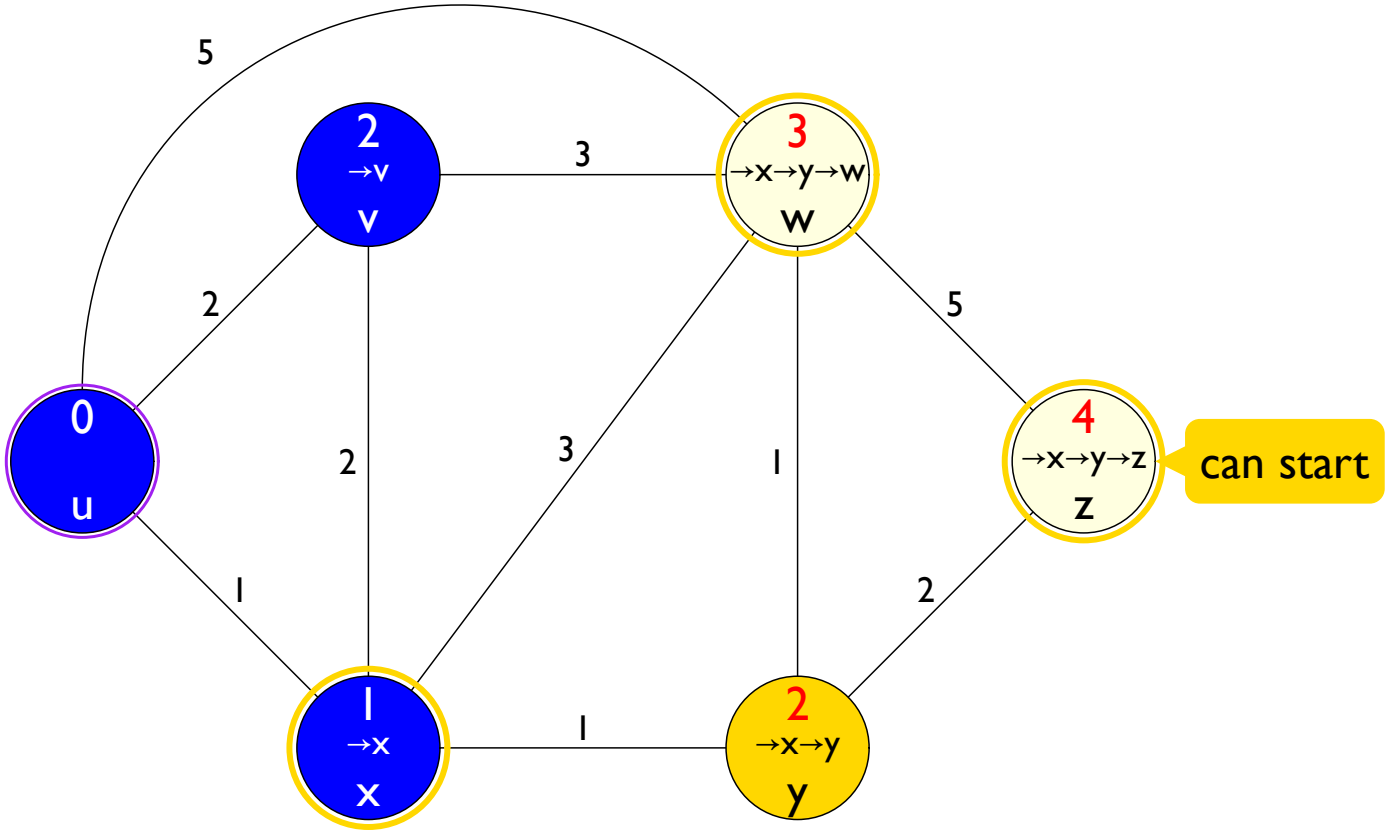
Finding Routes with Dijkstra's



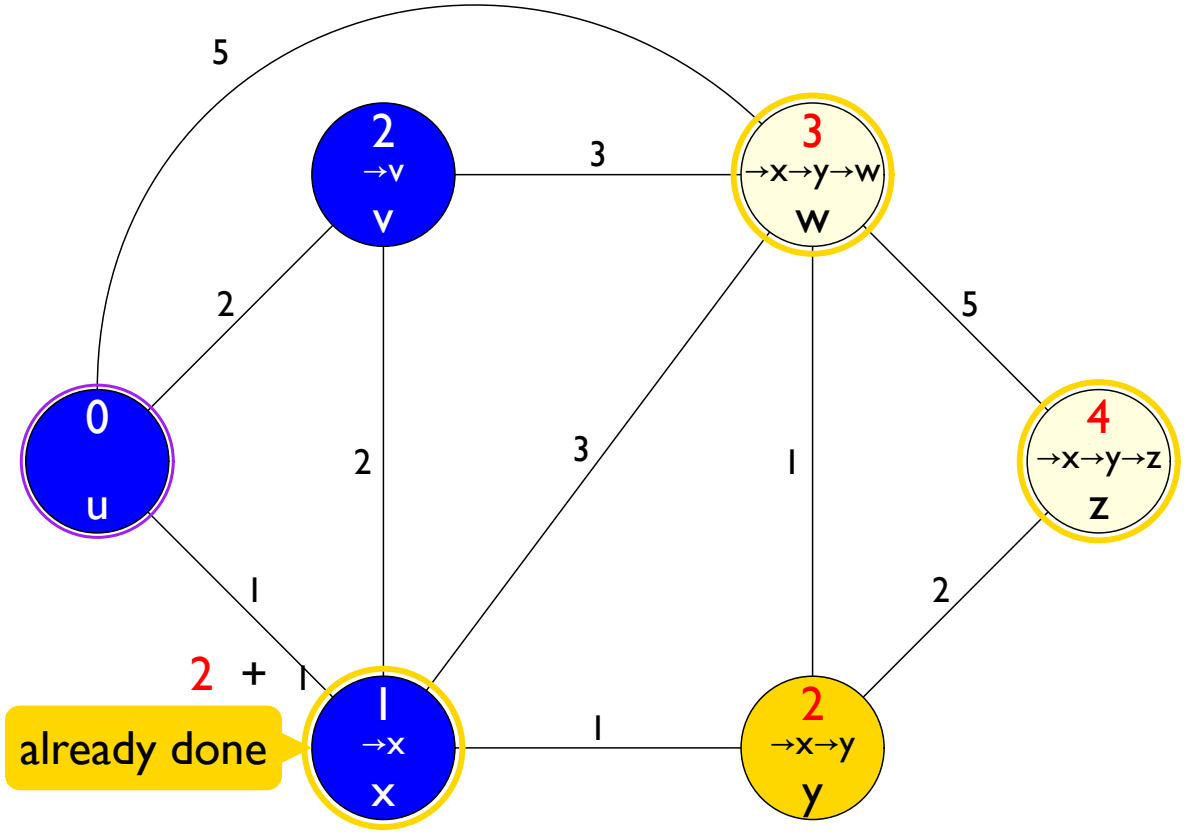
Finding Routes with Dijkstra's



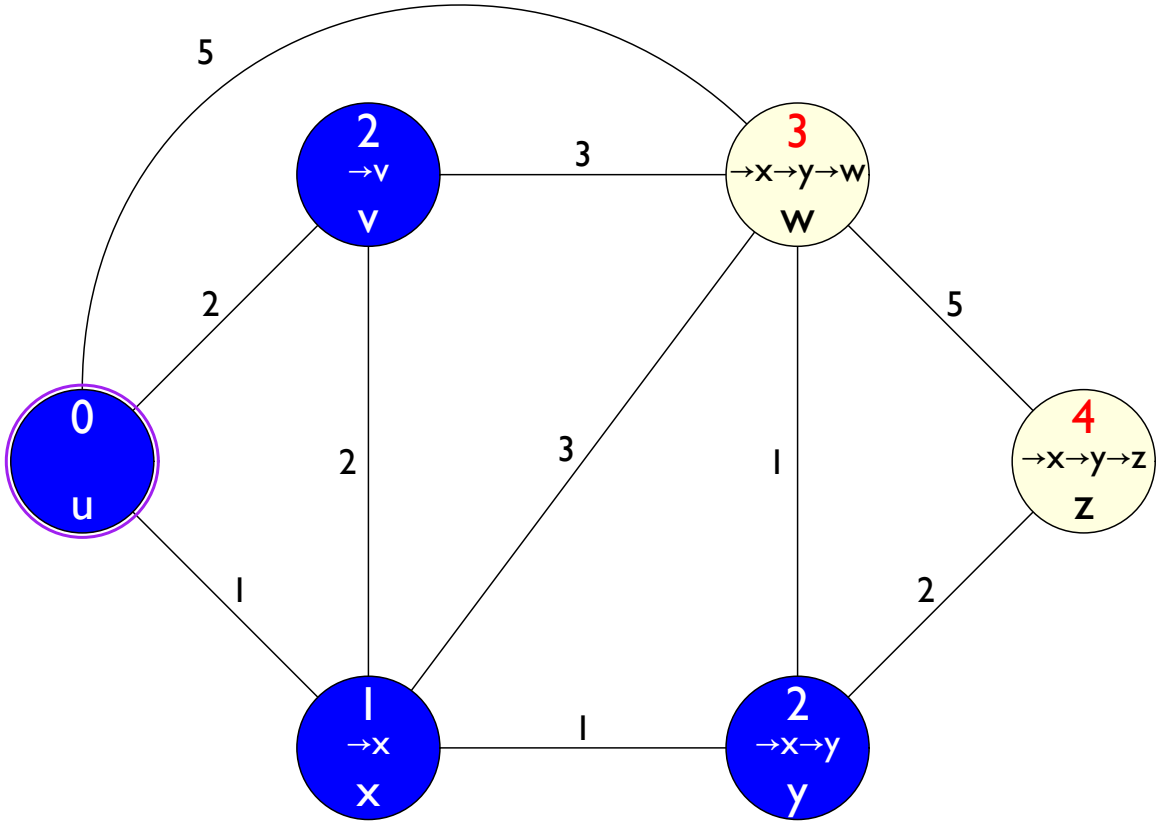
Finding Routes with Dijkstra's



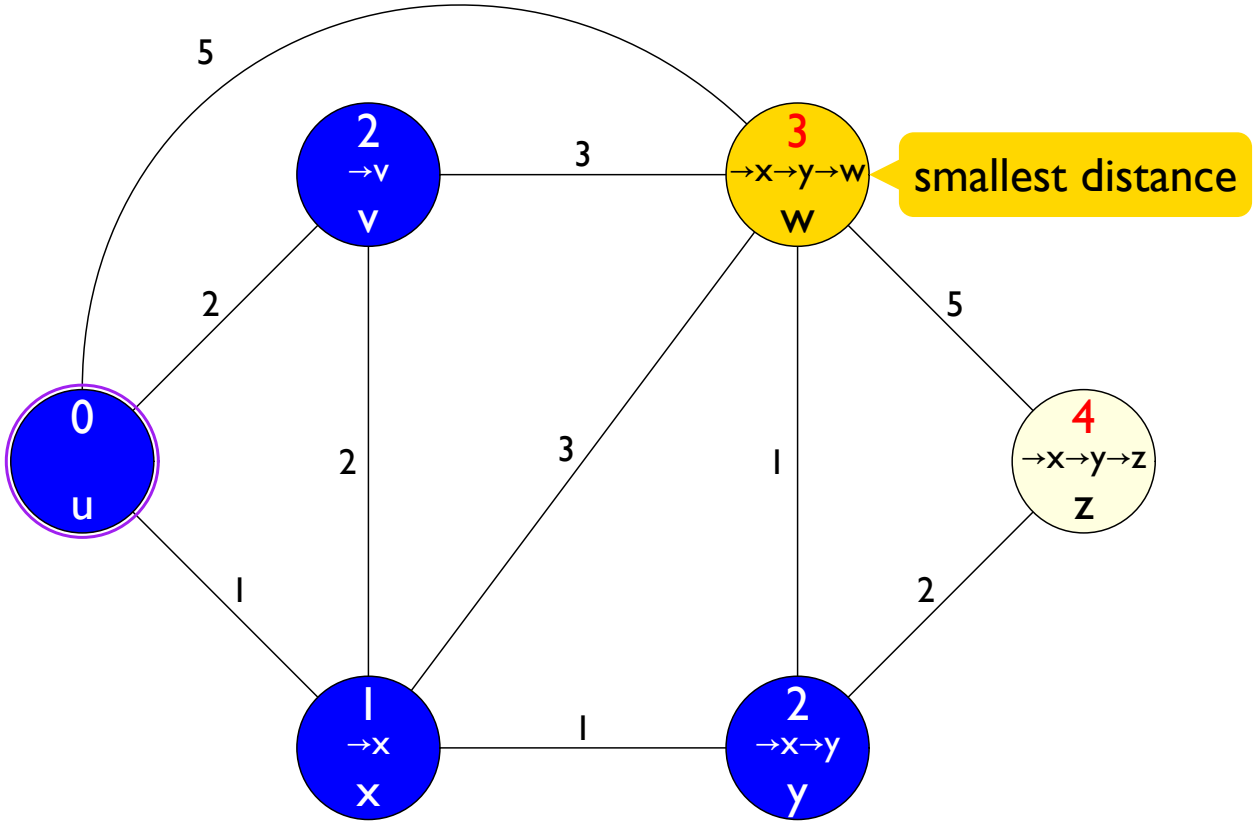
Finding Routes with Dijkstra's



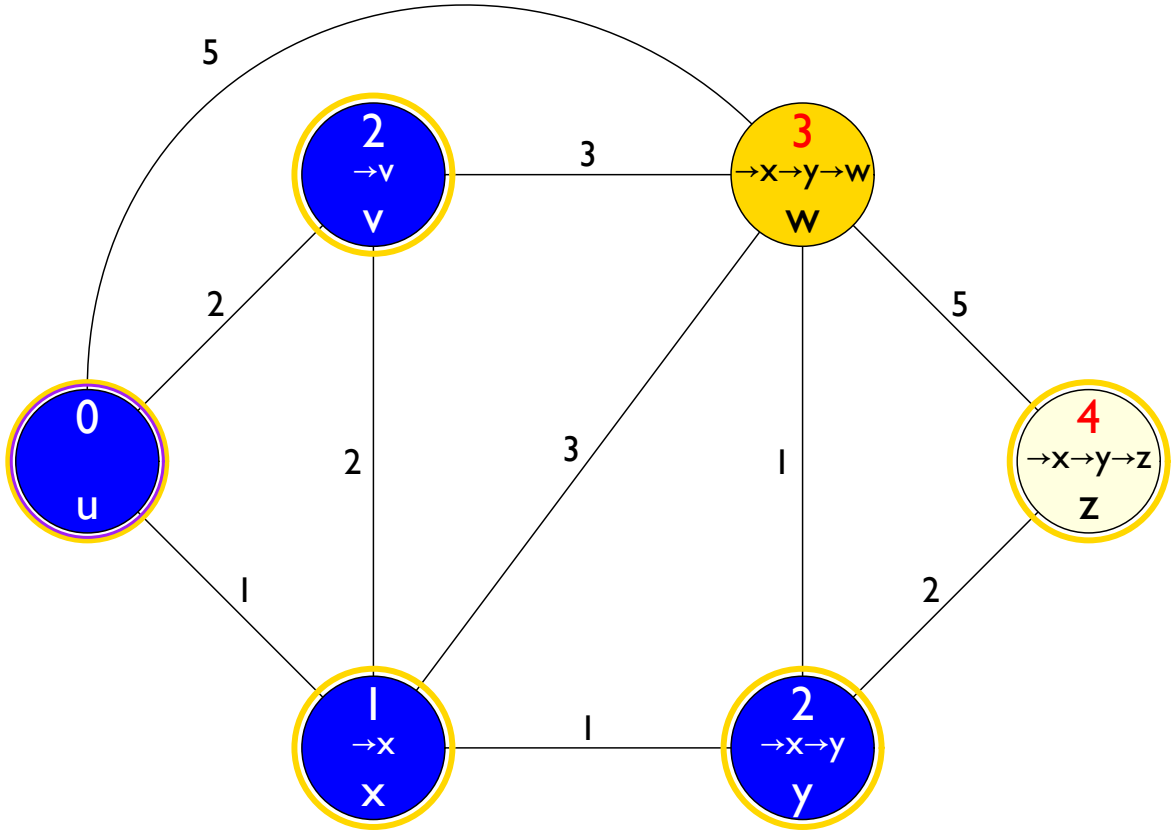
Finding Routes with Dijkstra's



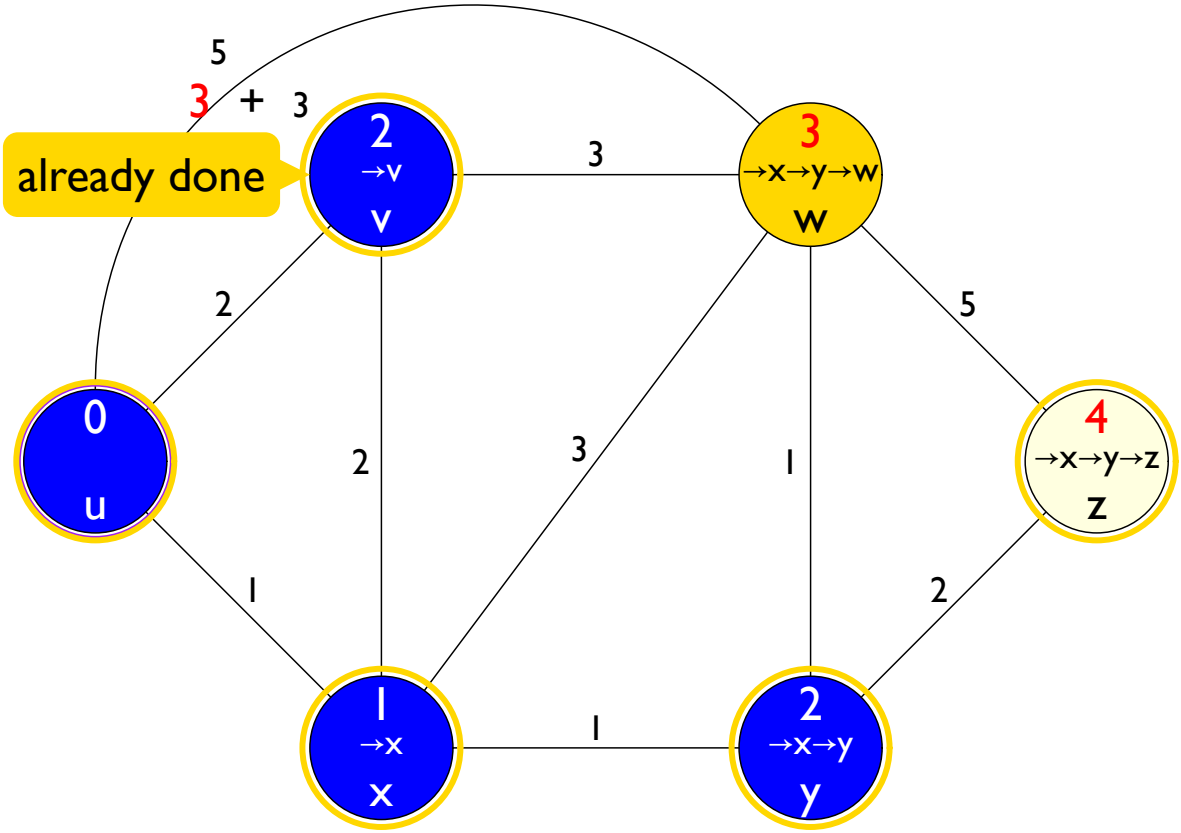
Finding Routes with Dijkstra's



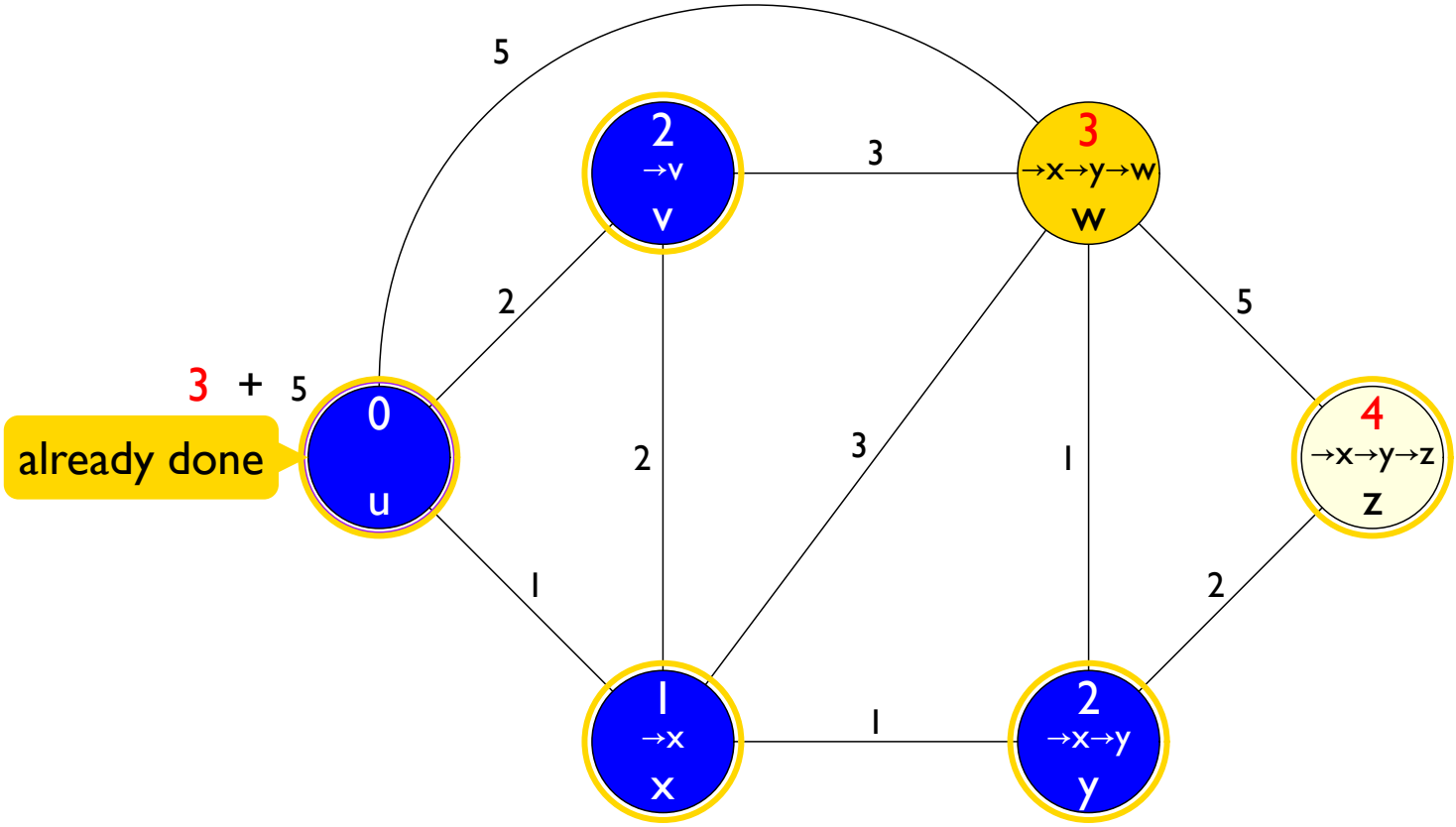
Finding Routes with Dijkstra's



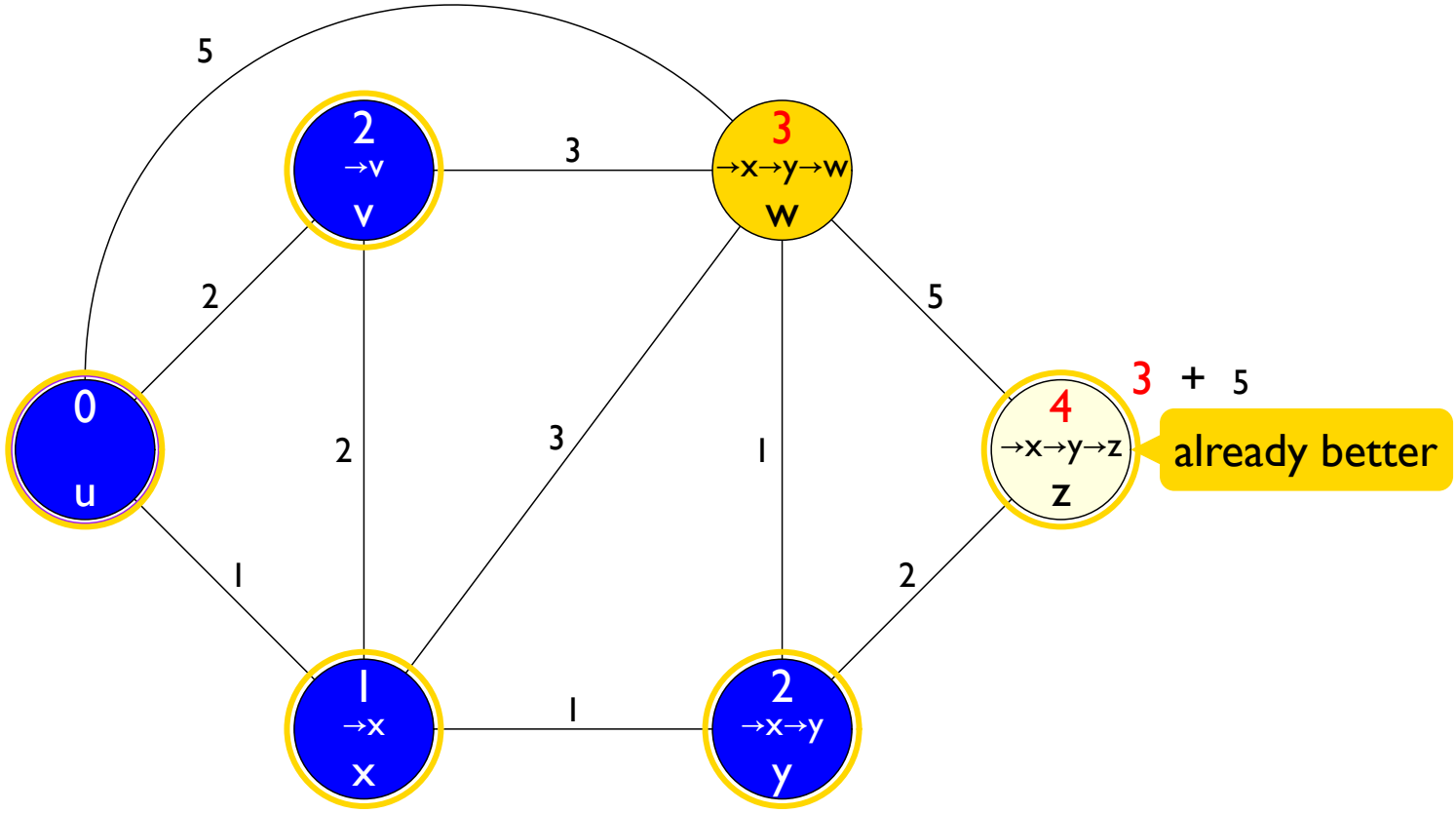
Finding Routes with Dijkstra's



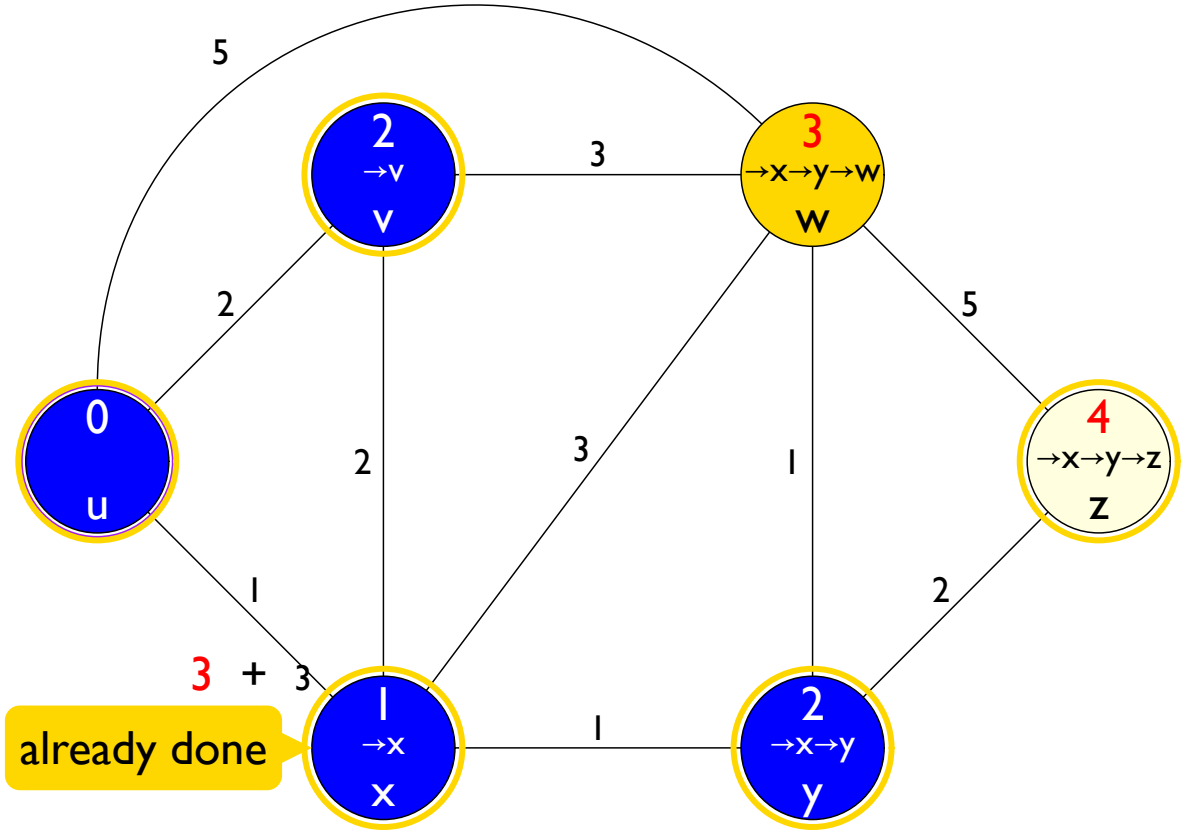
Finding Routes with Dijkstra's



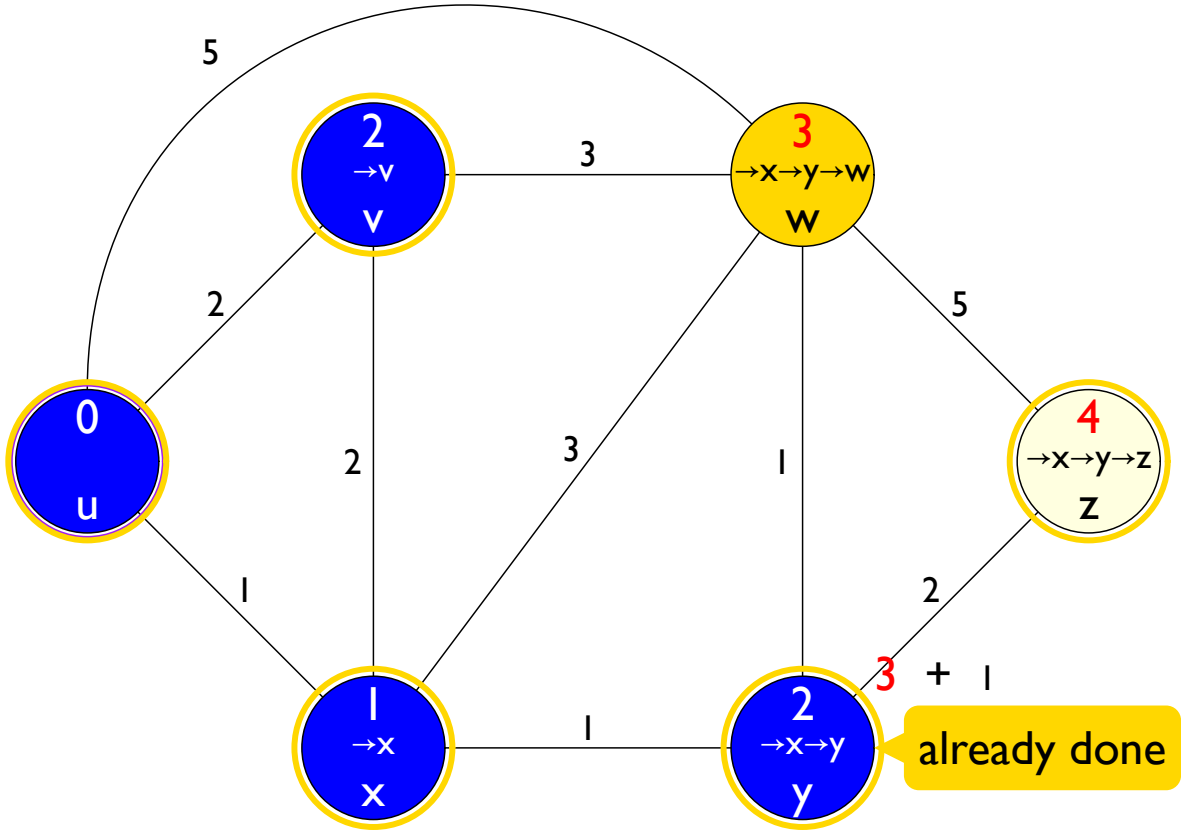
Finding Routes with Dijkstra's



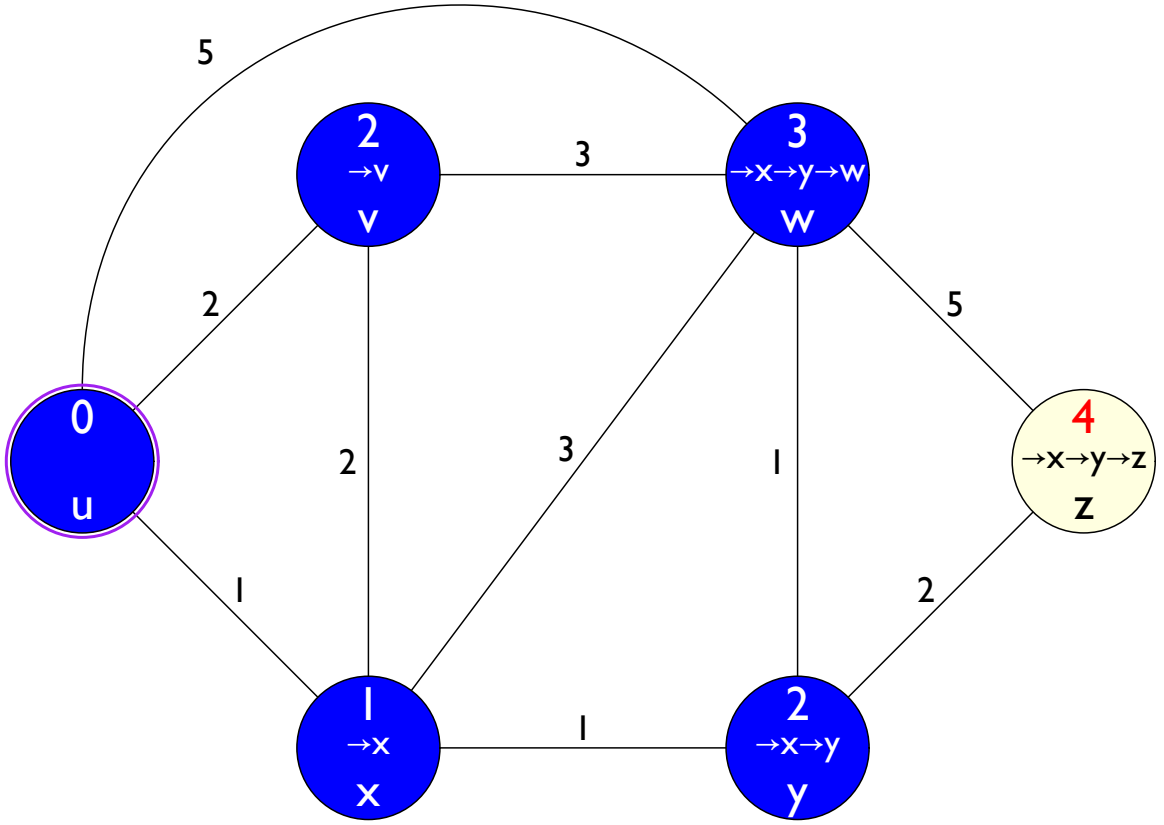
Finding Routes with Dijkstra's



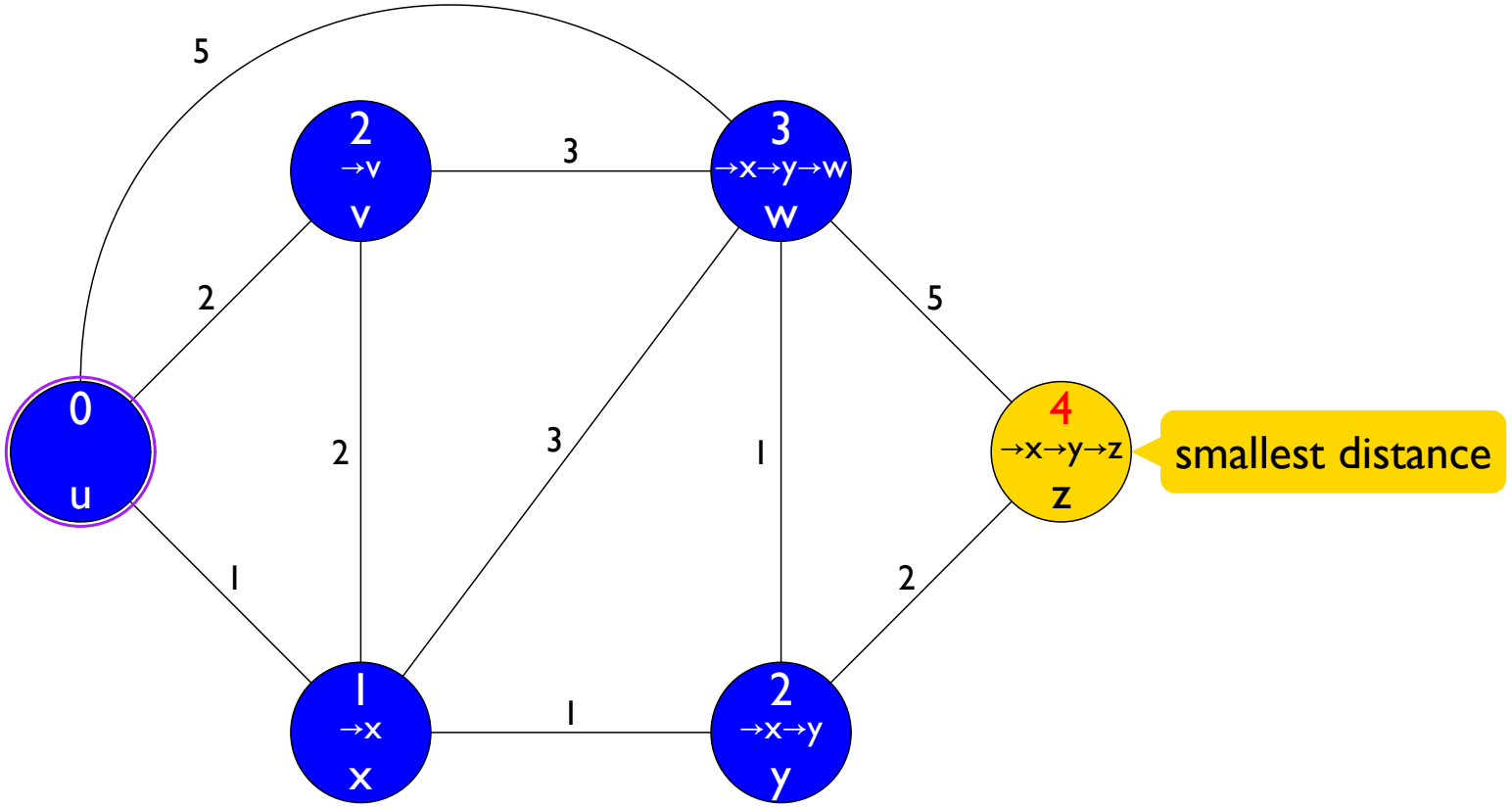
Finding Routes with Dijkstra's



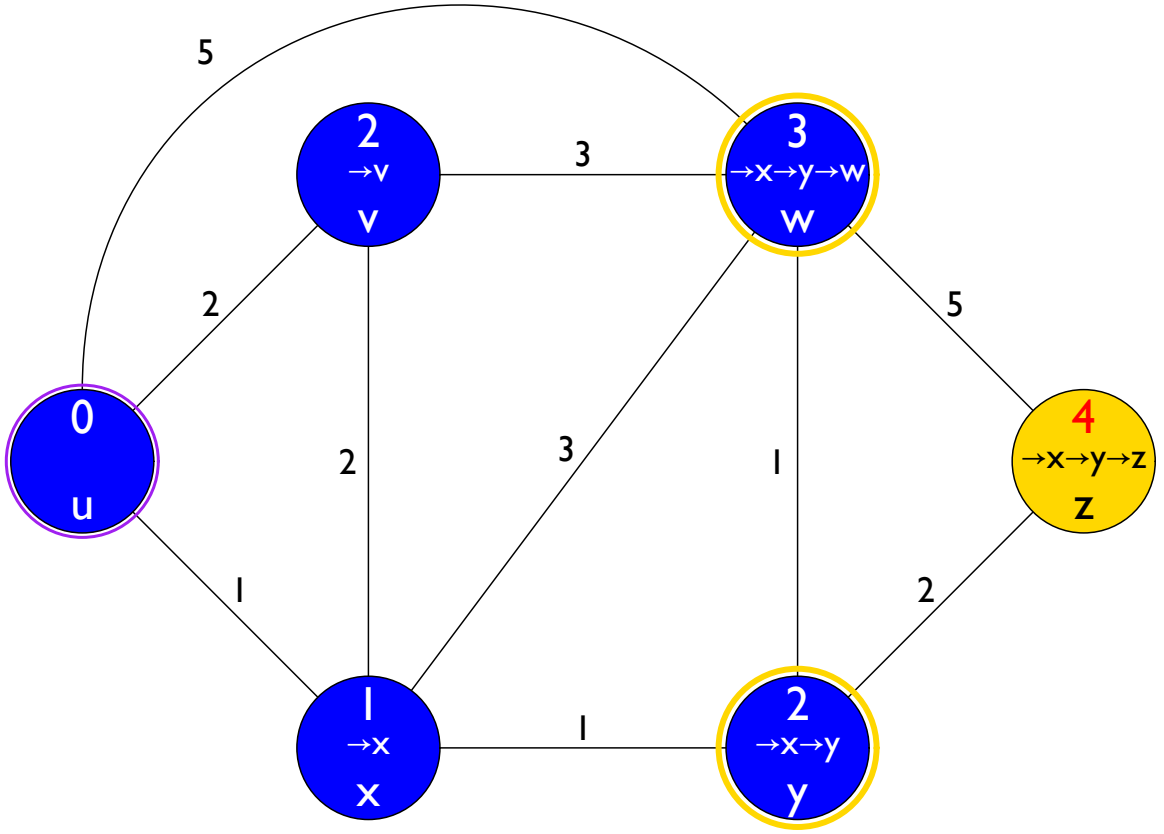
Finding Routes with Dijkstra's



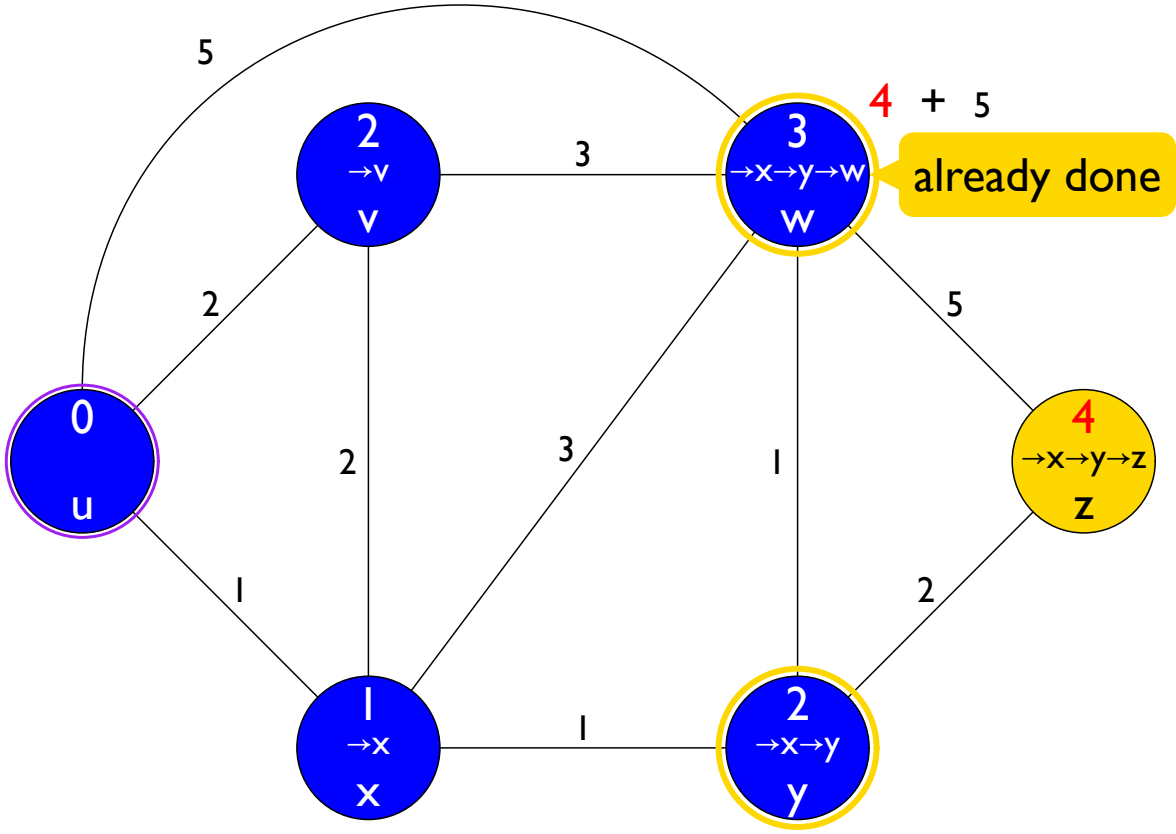
Finding Routes with Dijkstra's



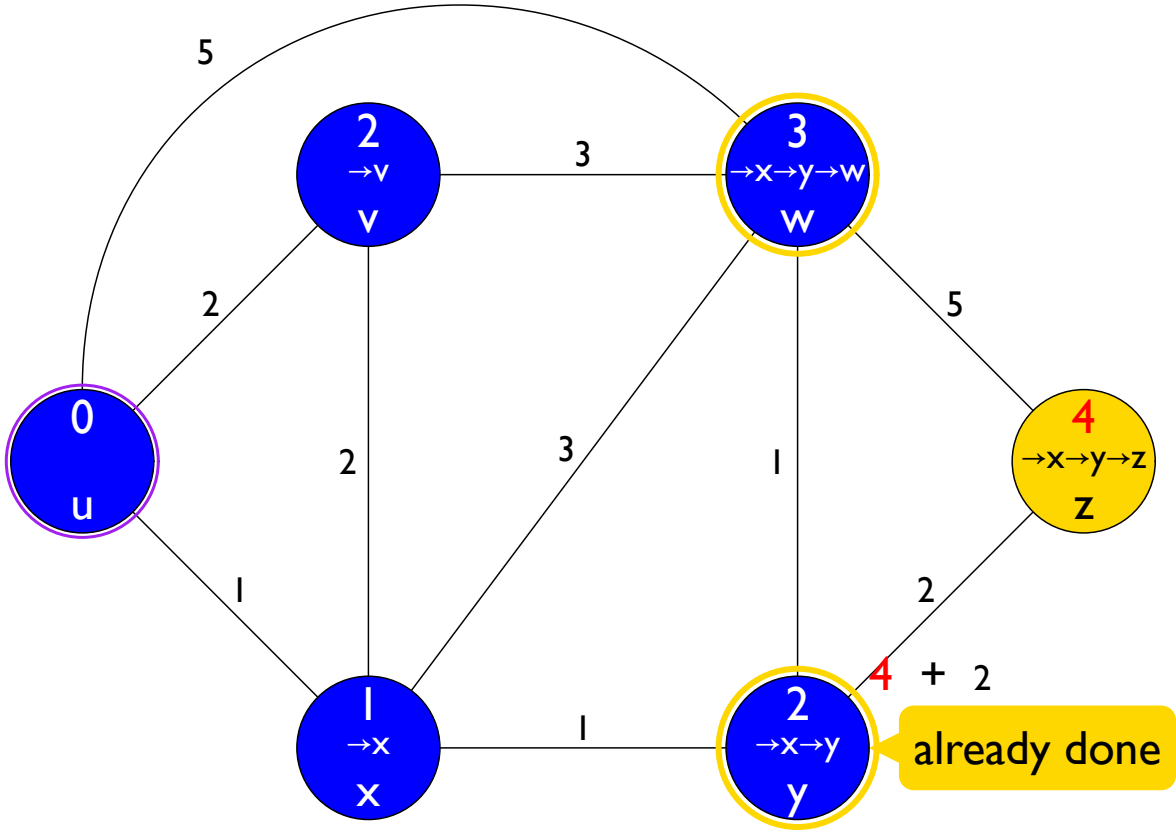
Finding Routes with Dijkstra's



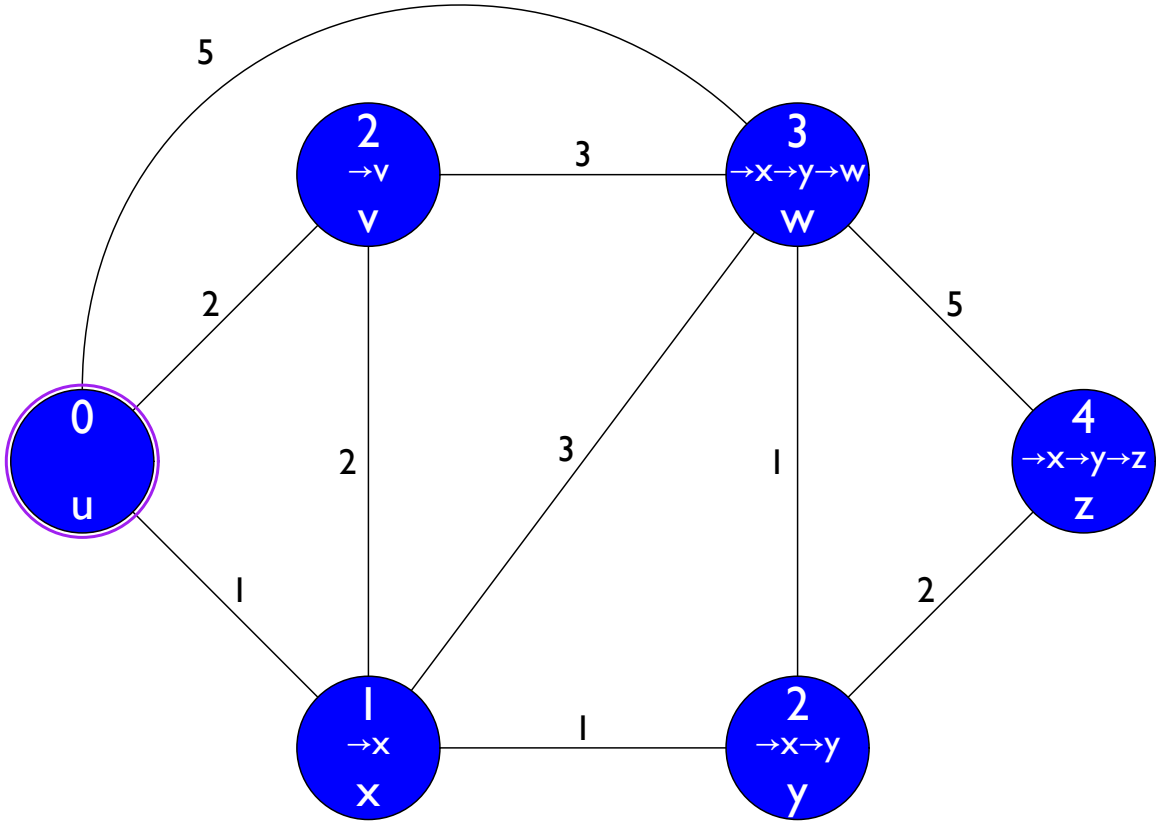
Finding Routes with Dijkstra's



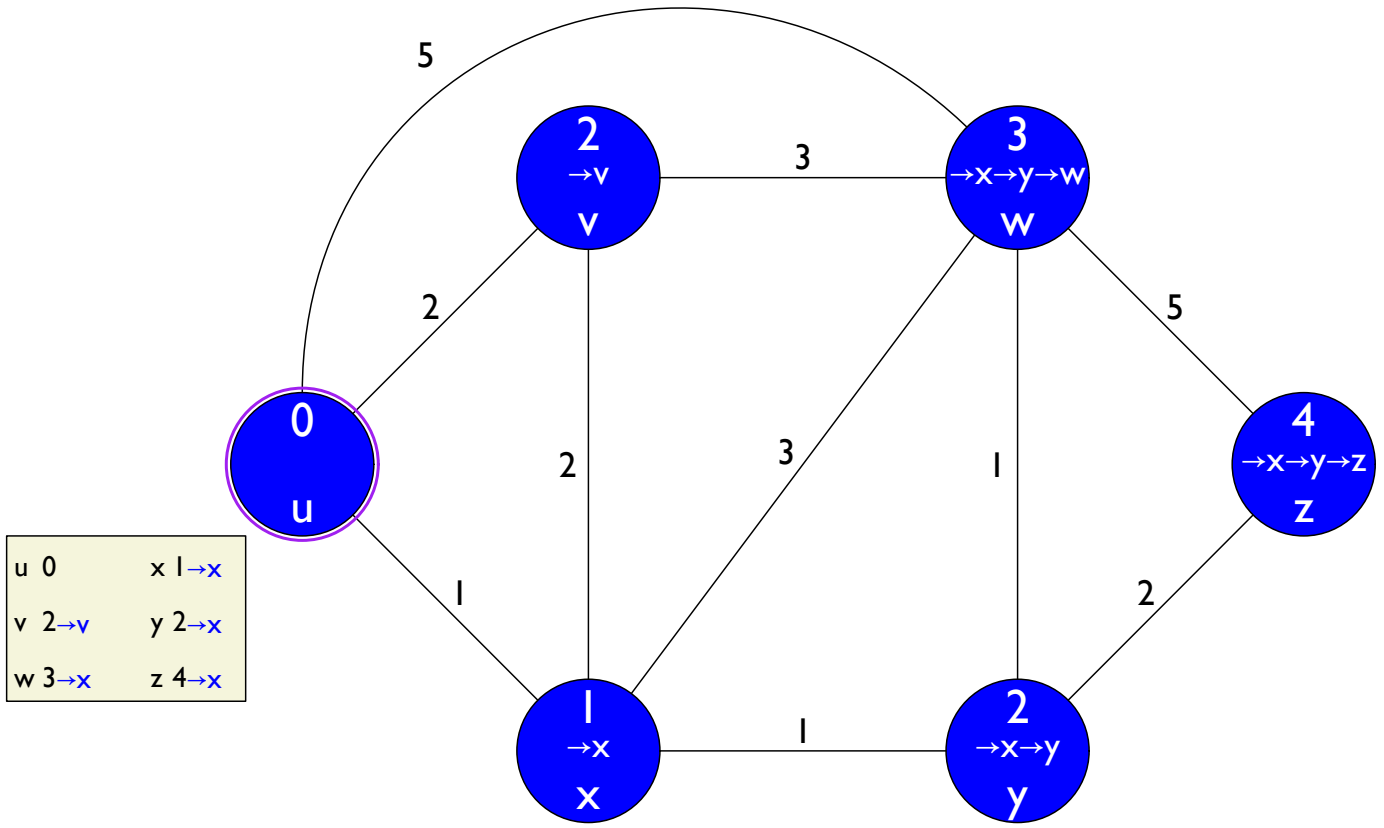
Finding Routes with Dijkstra's



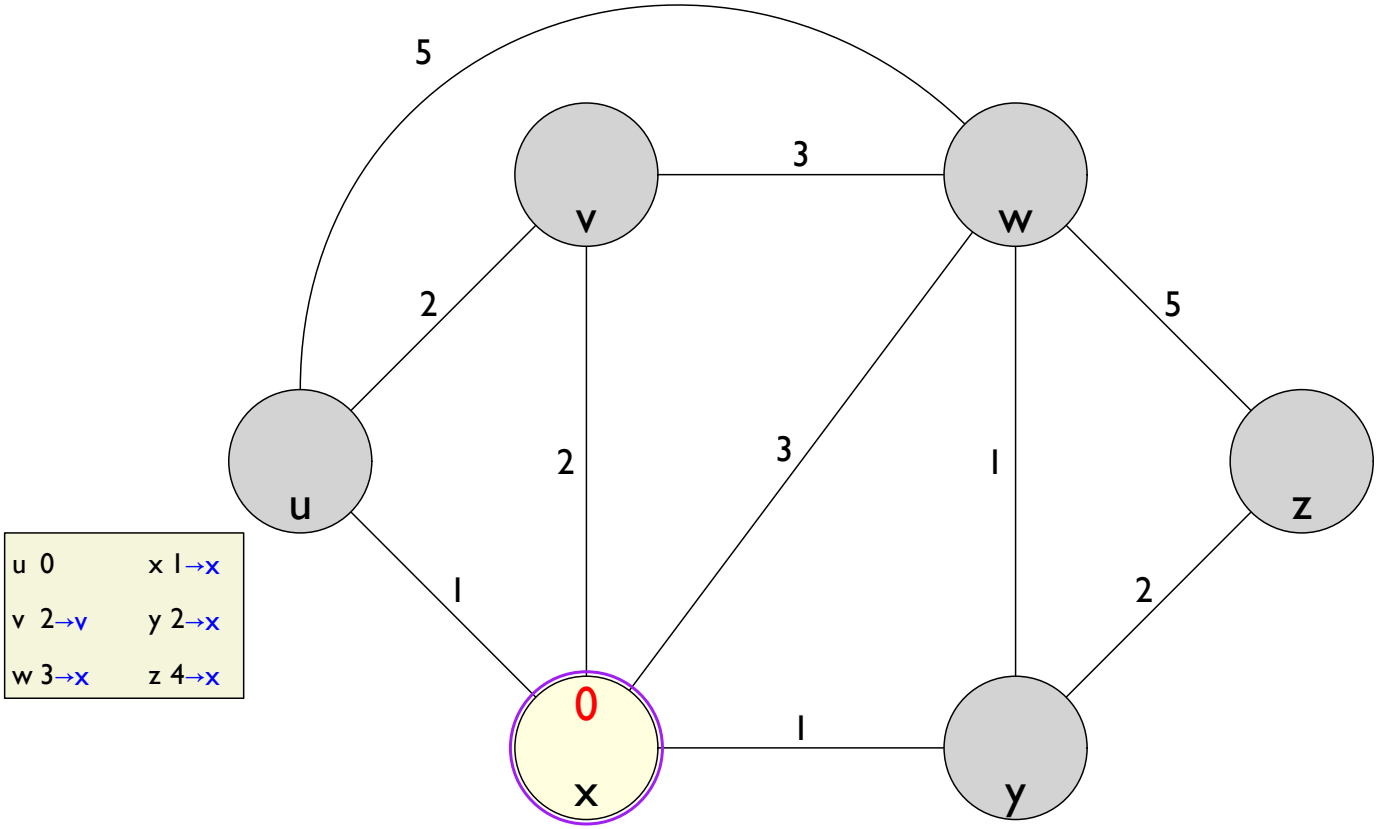
Finding Routes with Dijkstra's



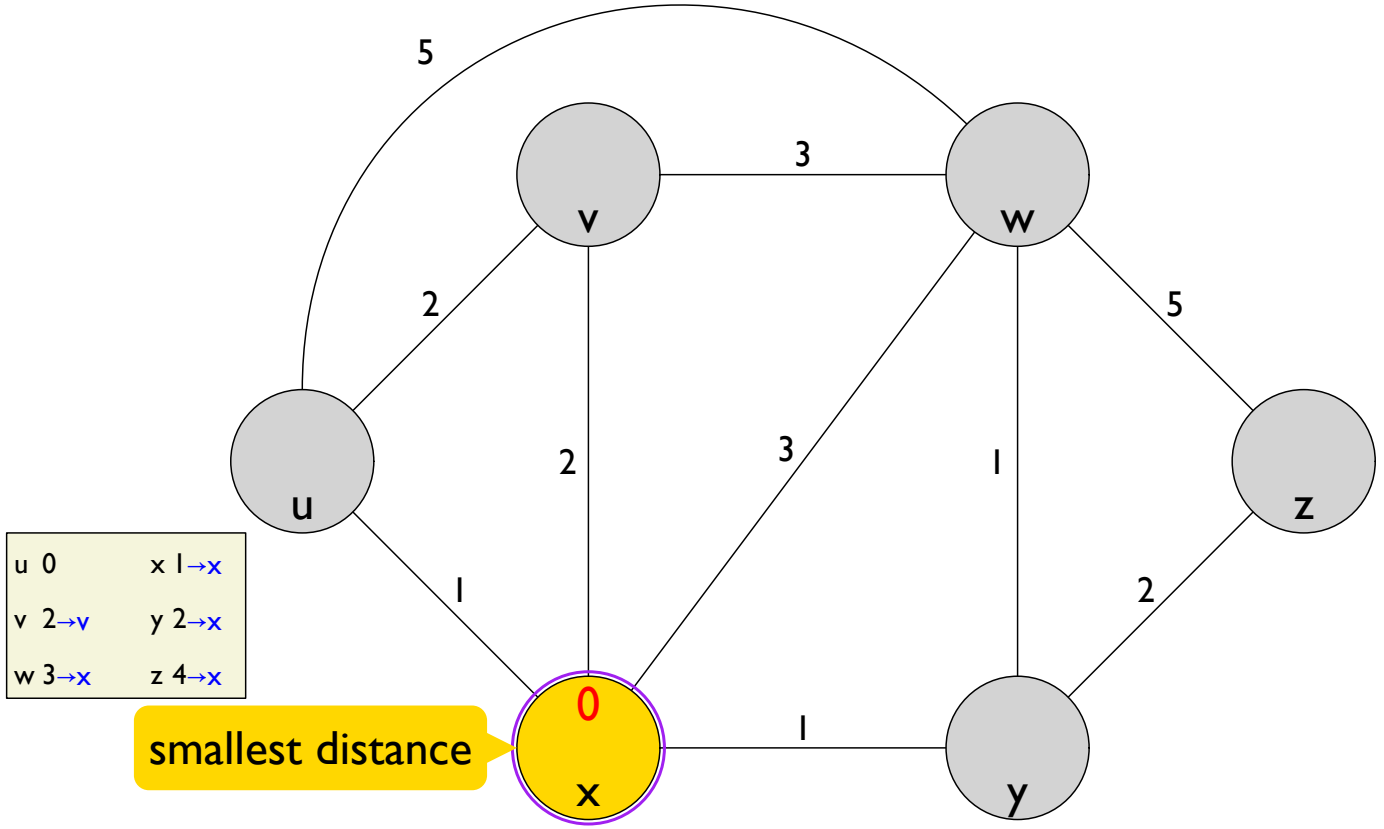
Finding Routes with Dijkstra's



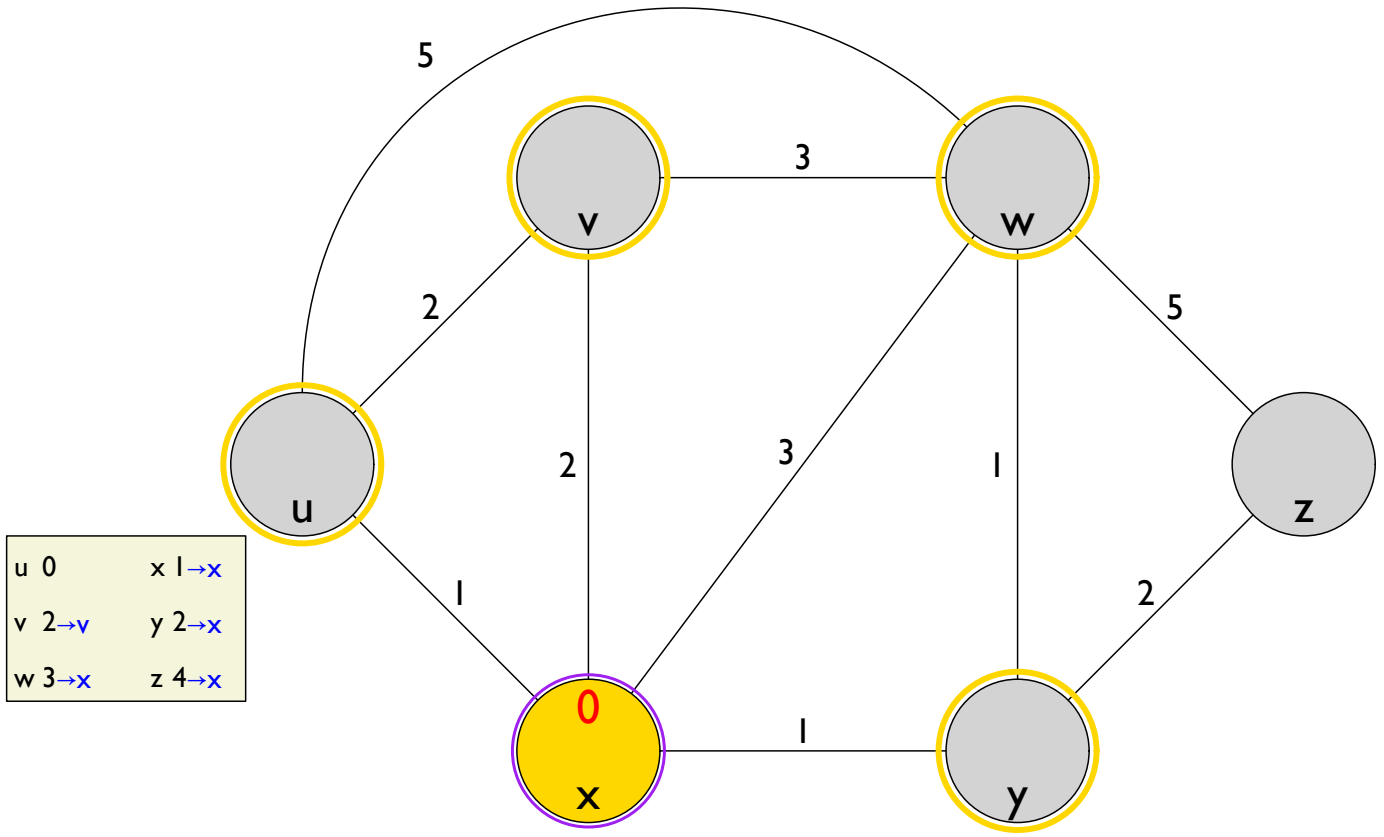
Finding Routes with Dijkstra's



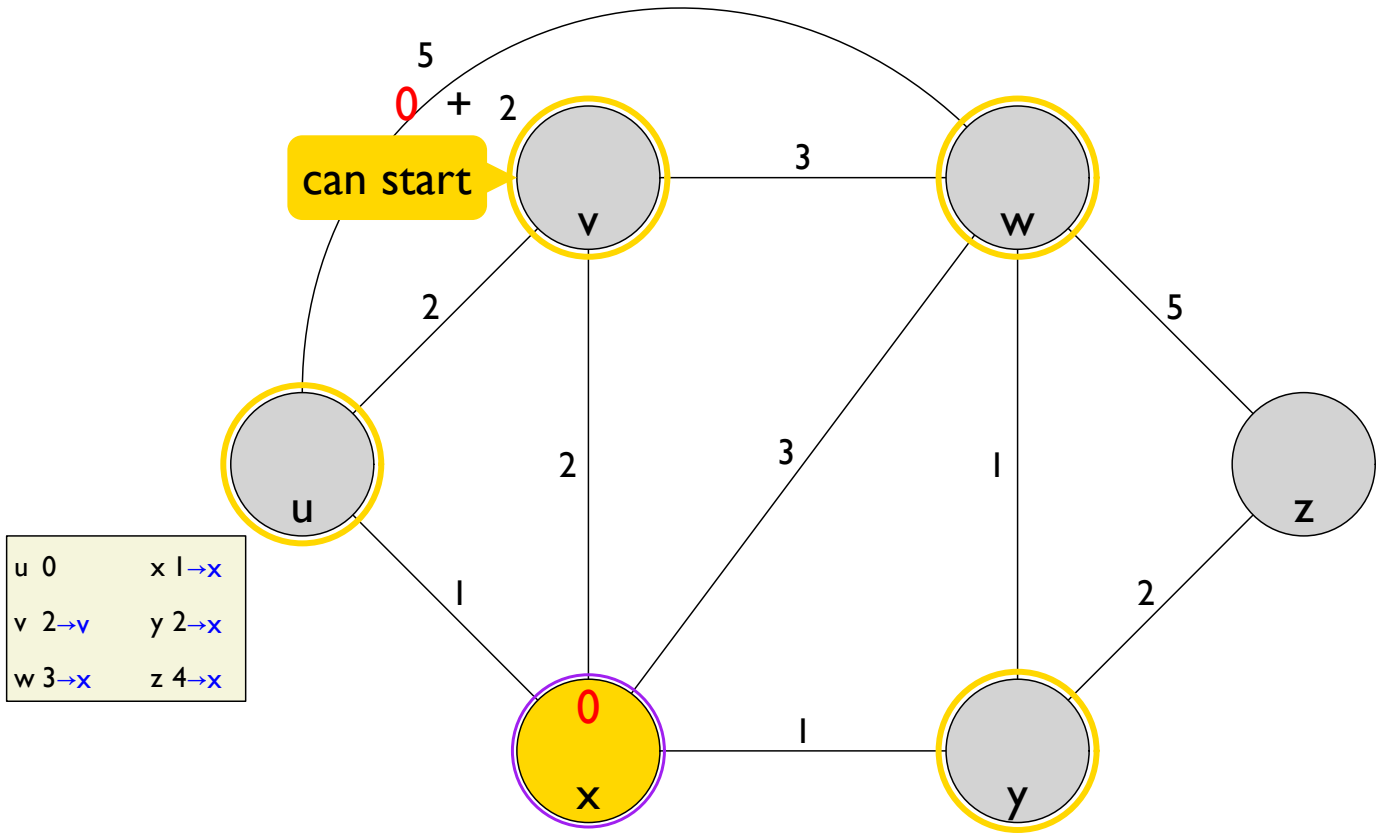
Finding Routes with Dijkstra's



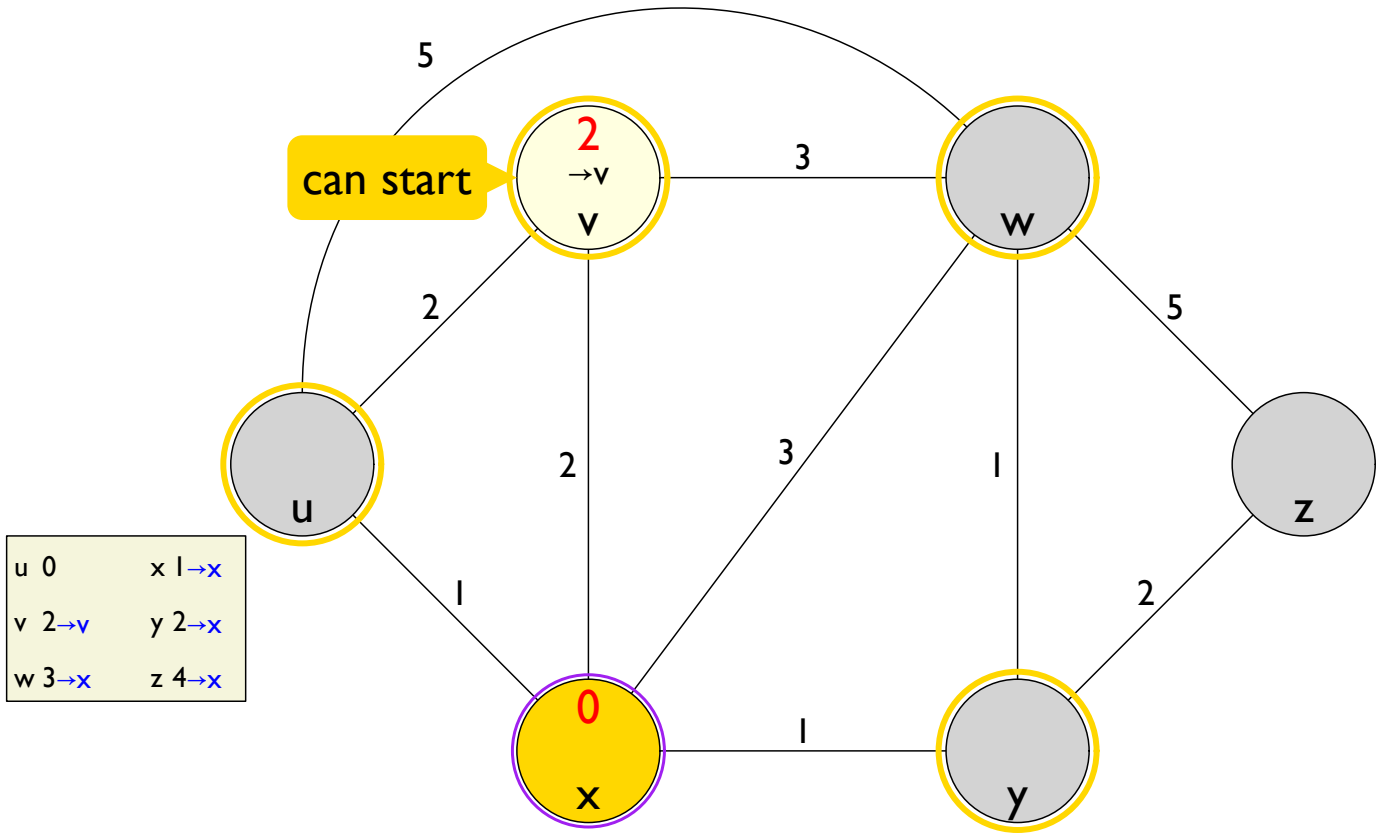
Finding Routes with Dijkstra's



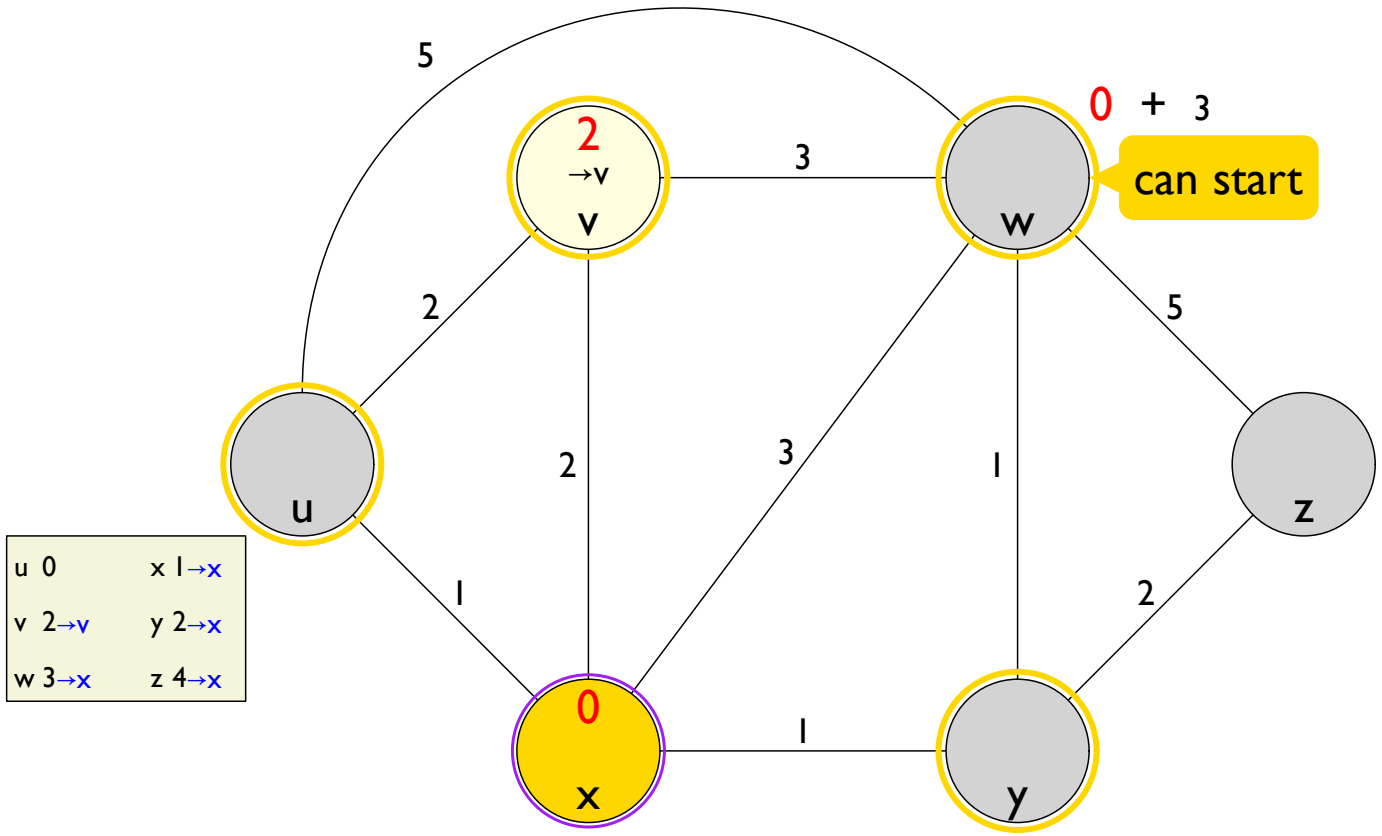
Finding Routes with Dijkstra's



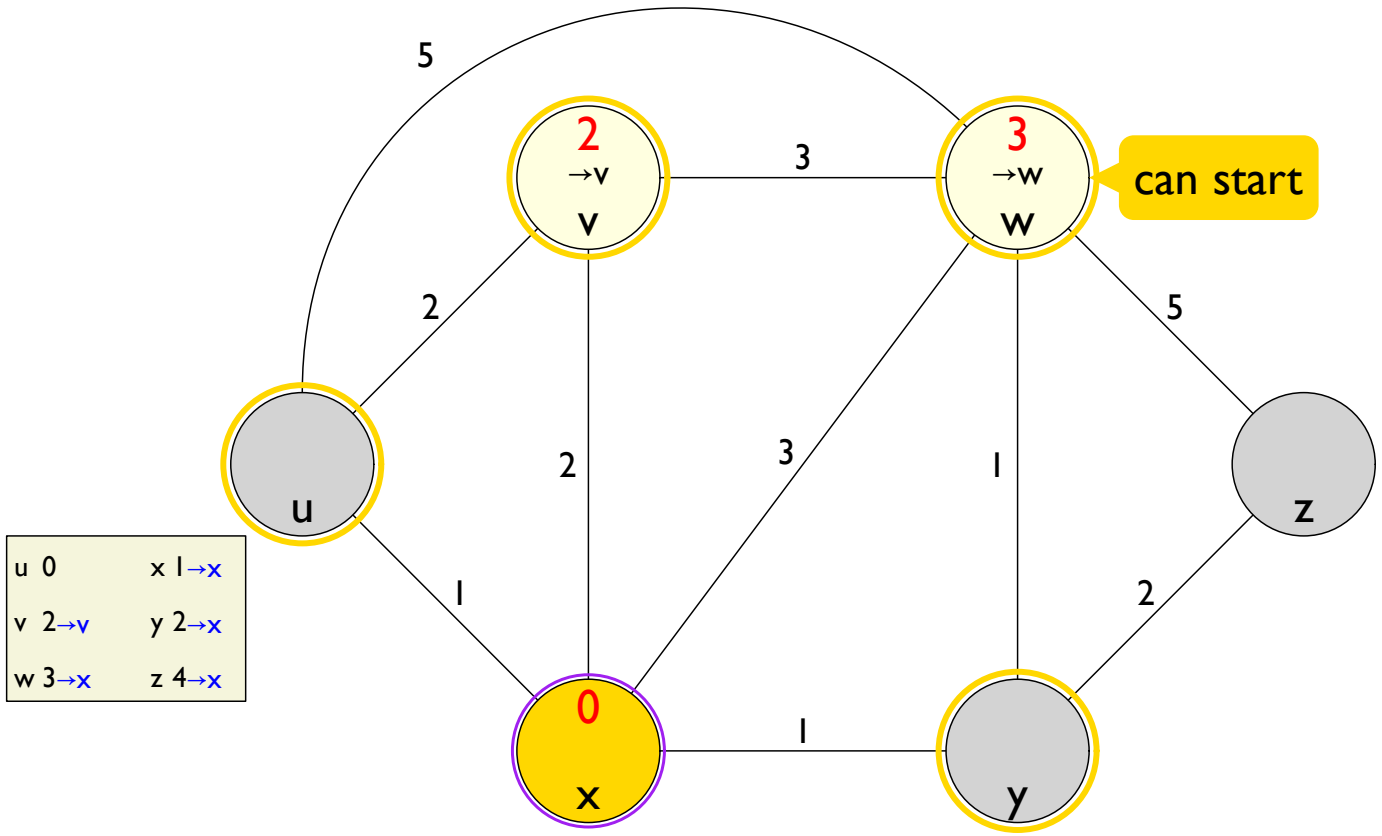
Finding Routes with Dijkstra's



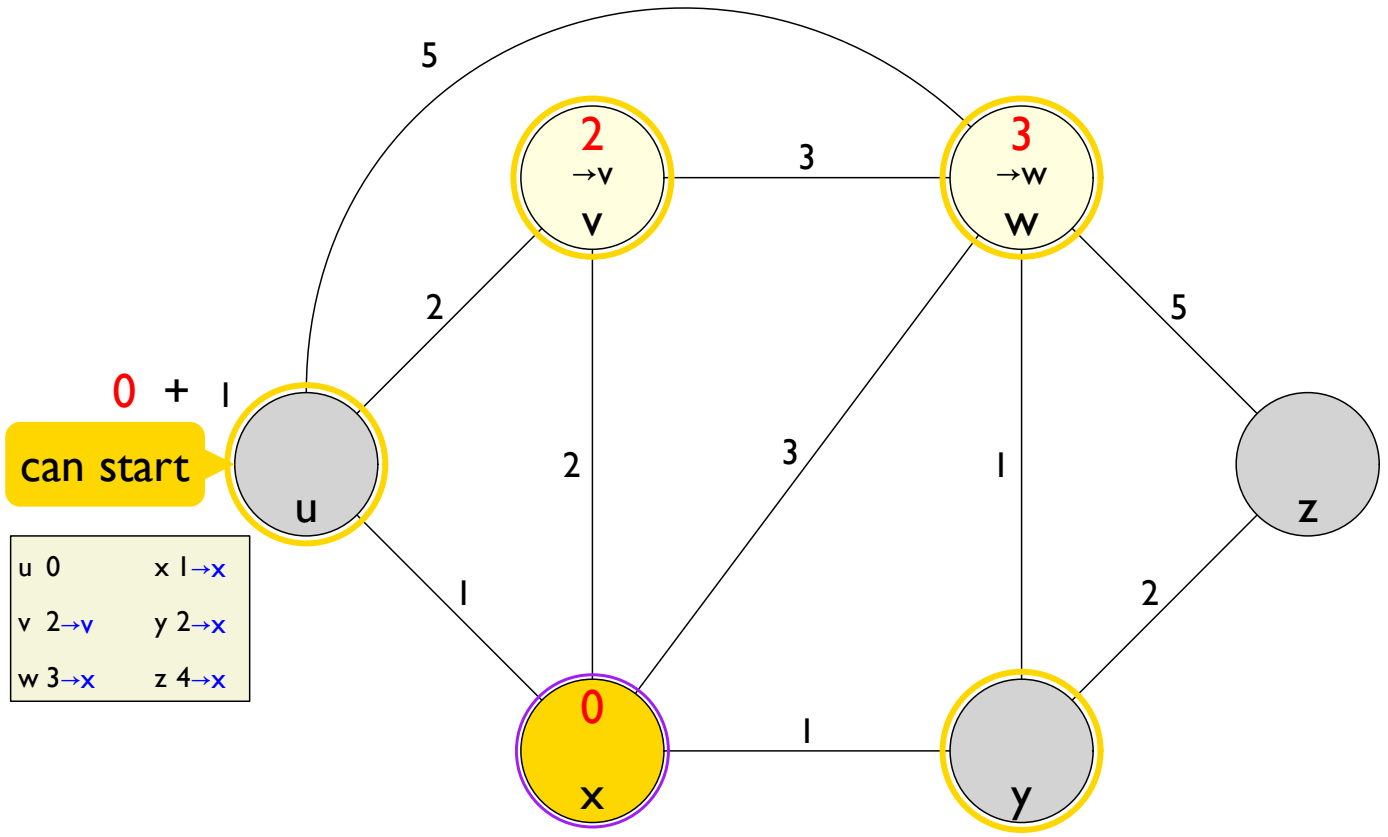
Finding Routes with Dijkstra's



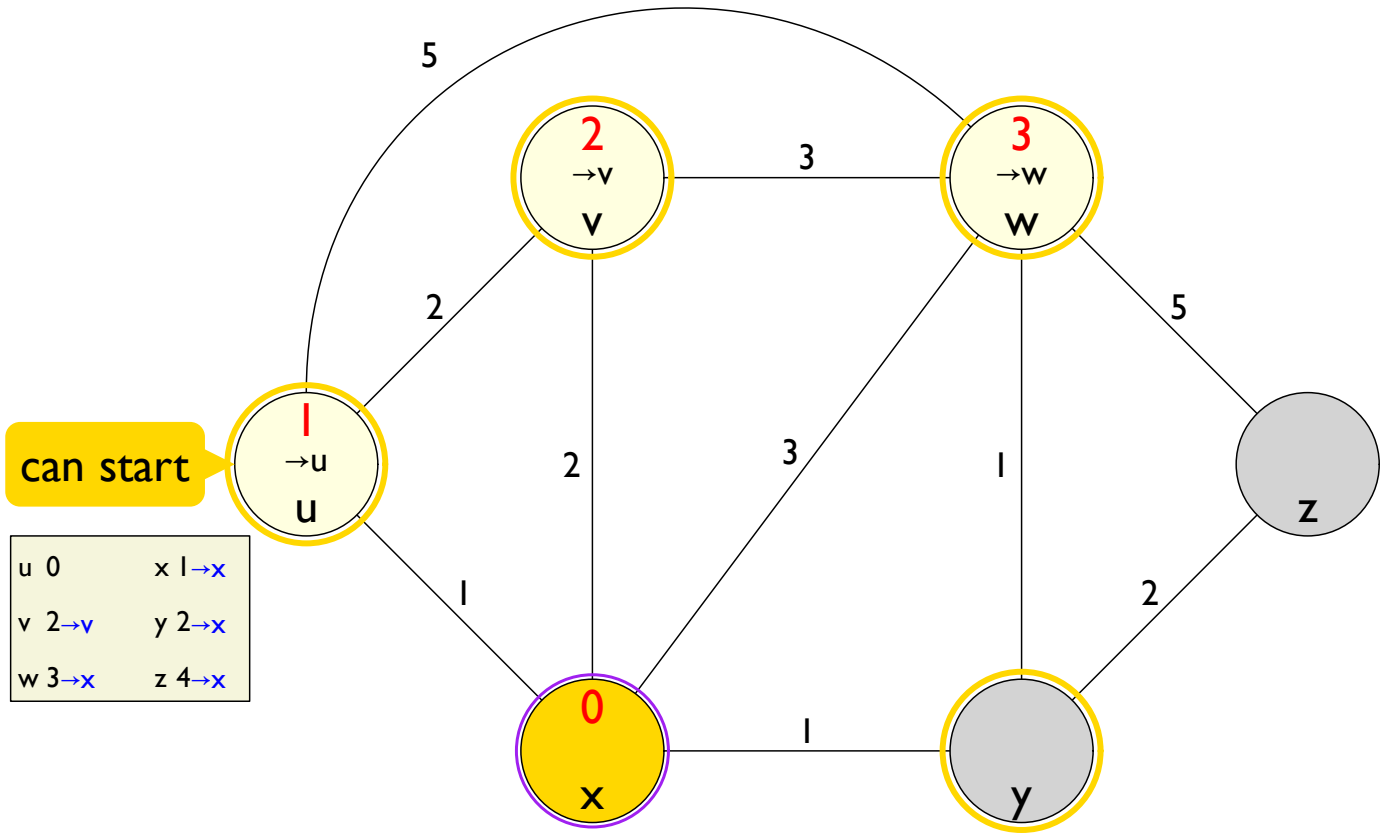
Finding Routes with Dijkstra's



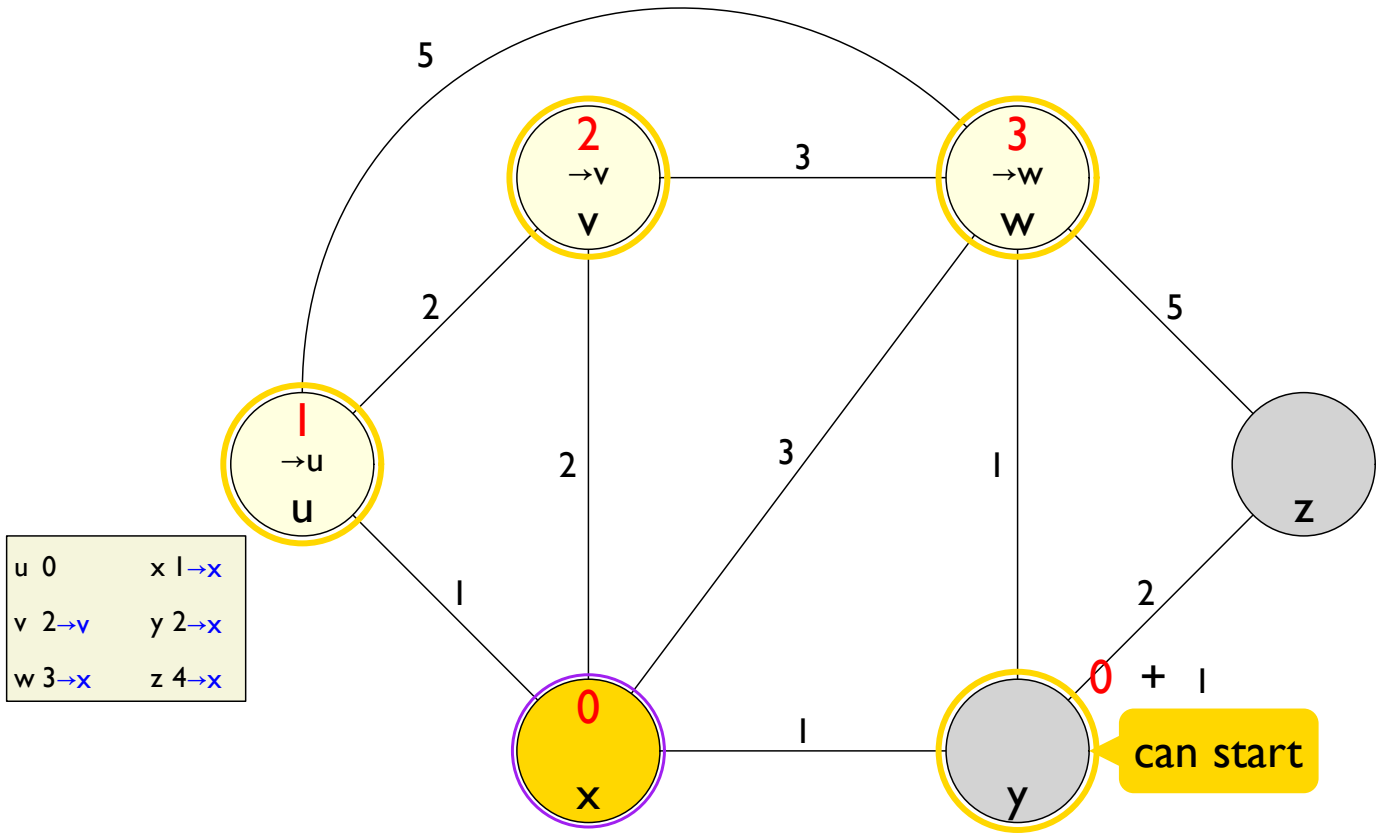
Finding Routes with Dijkstra's



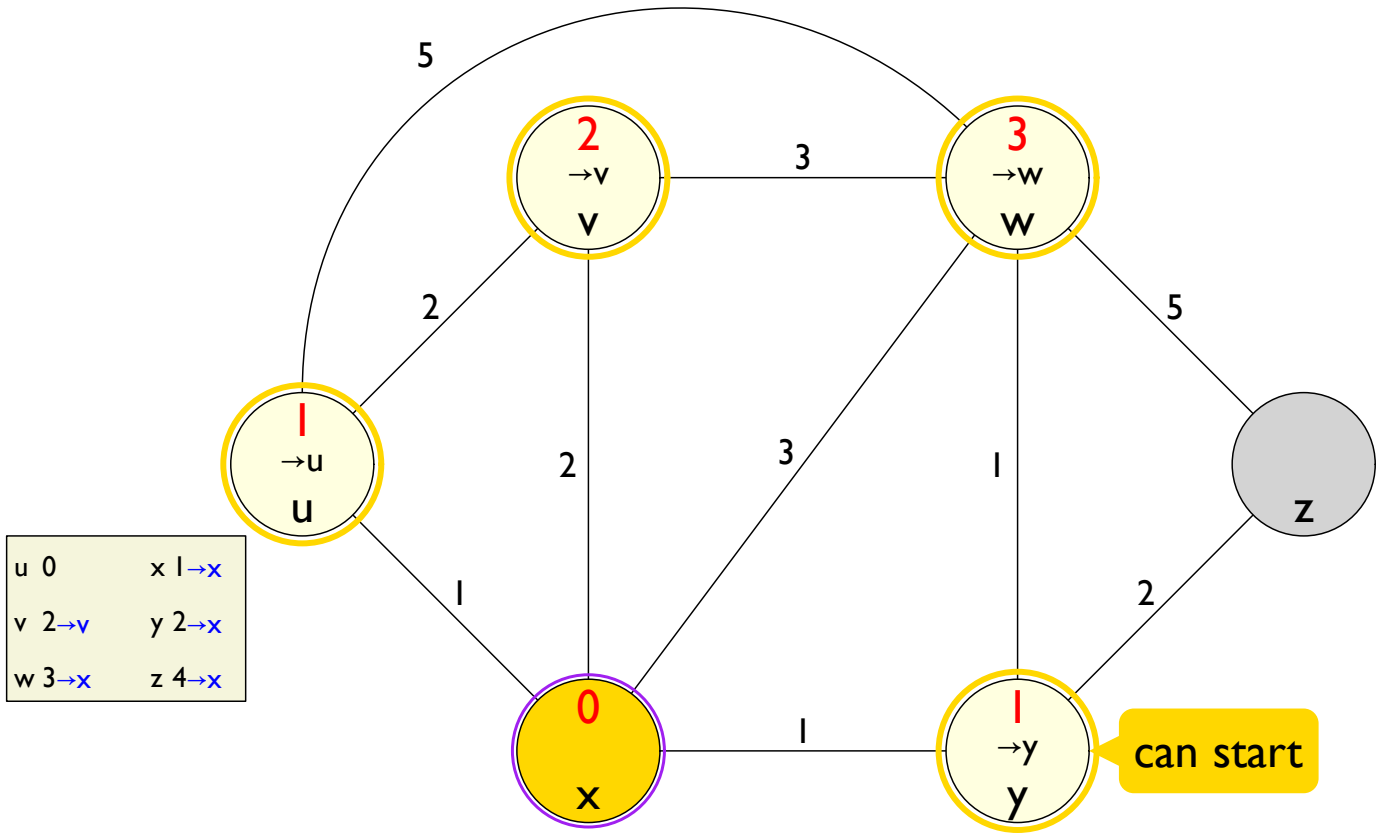
Finding Routes with Dijkstra's



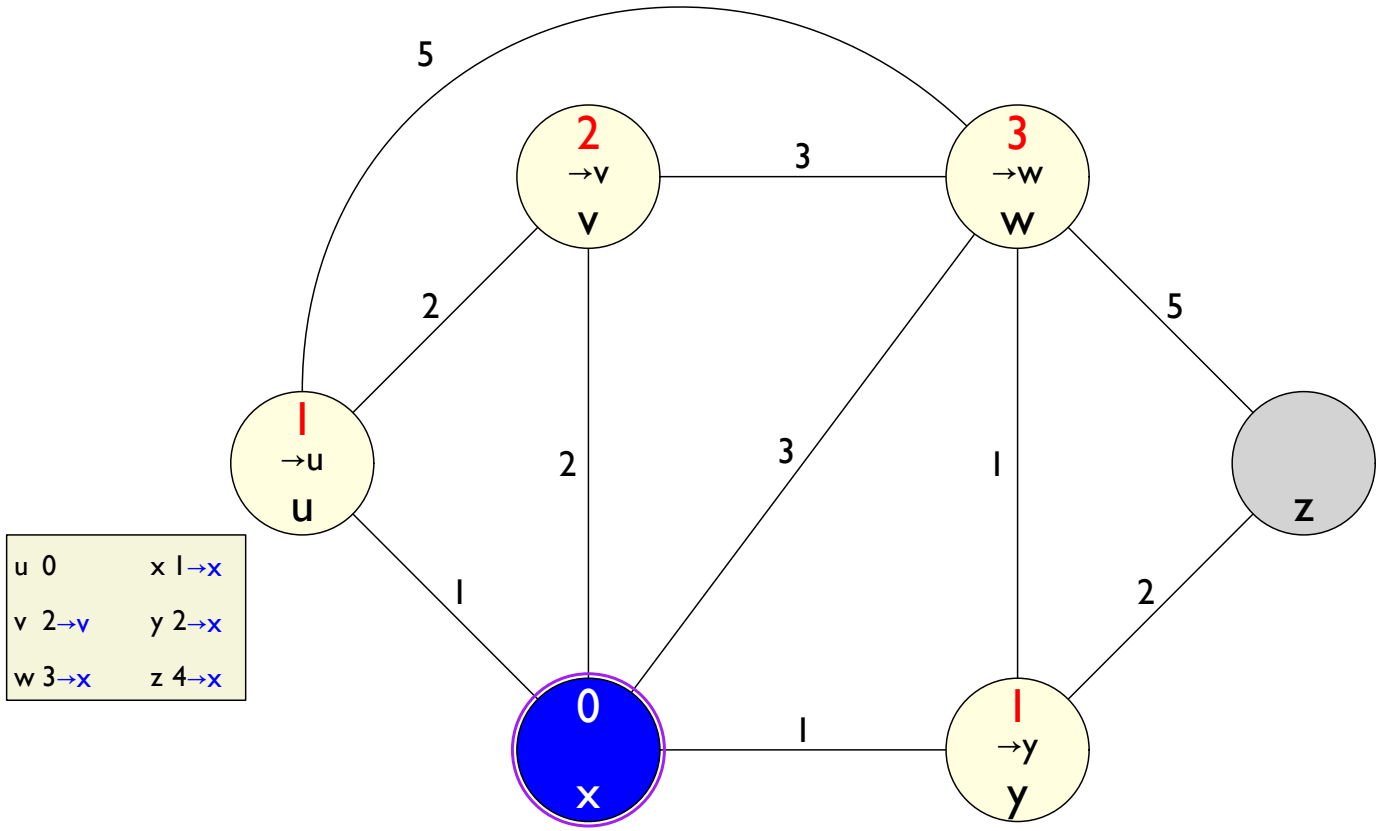
Finding Routes with Dijkstra's



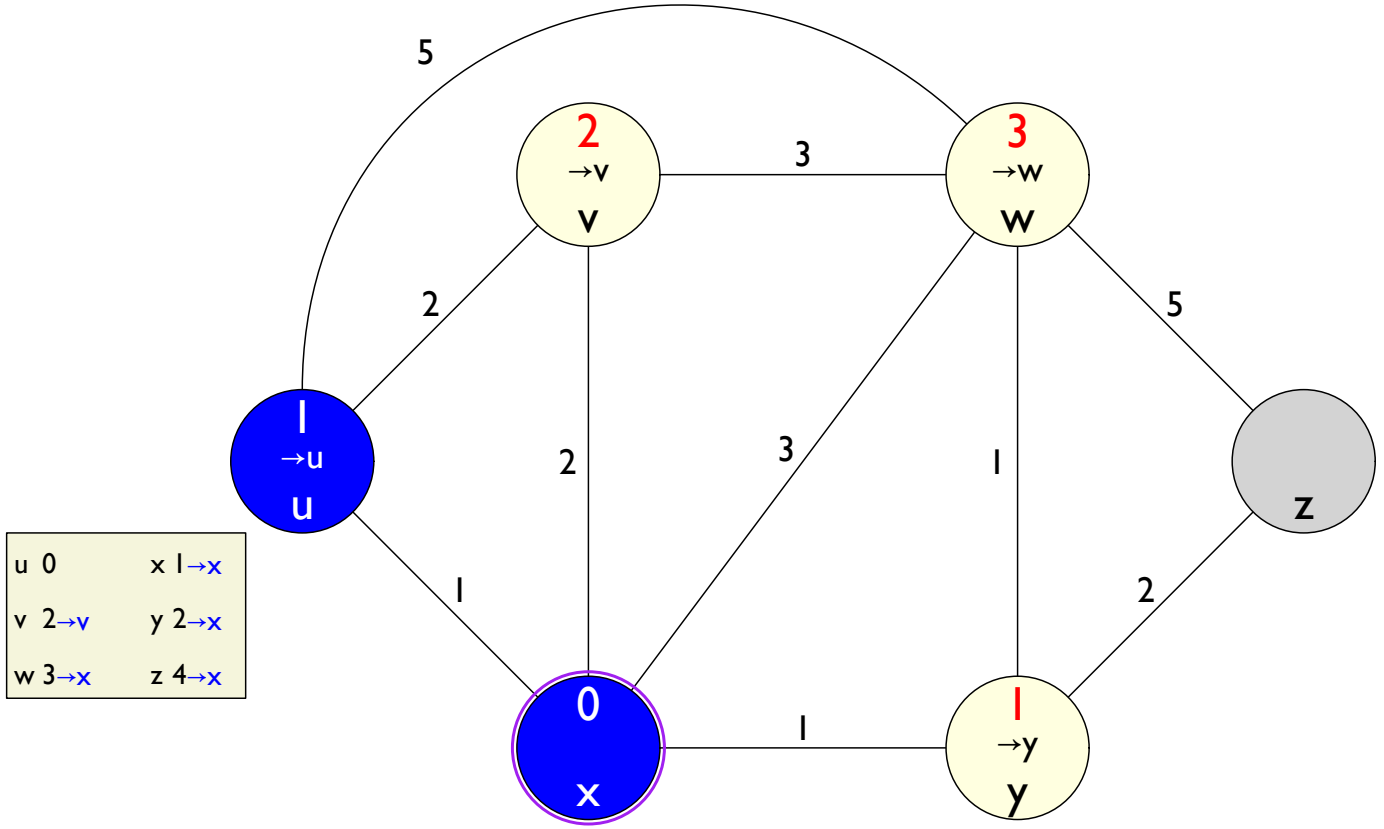
Finding Routes with Dijkstra's



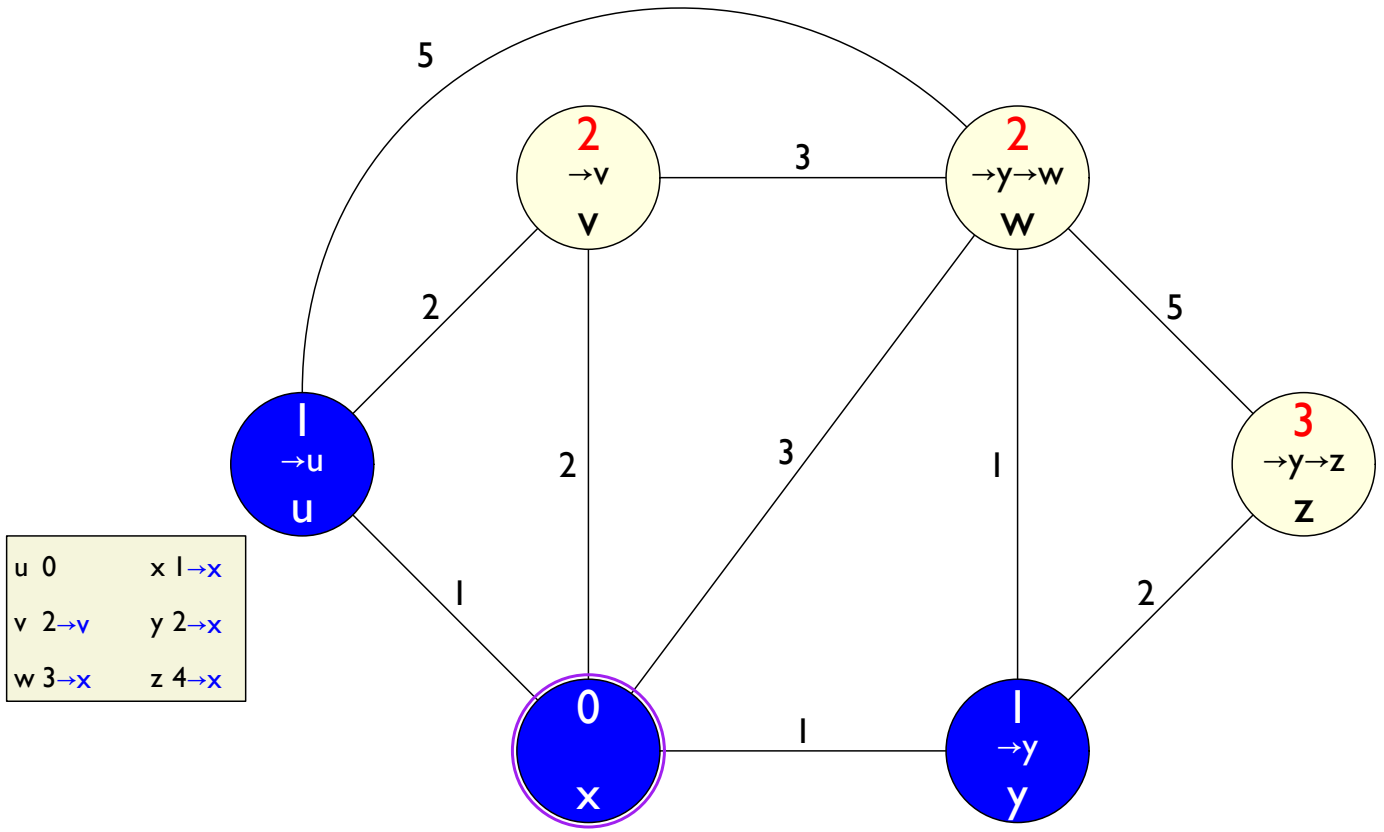
Finding Routes with Dijkstra's



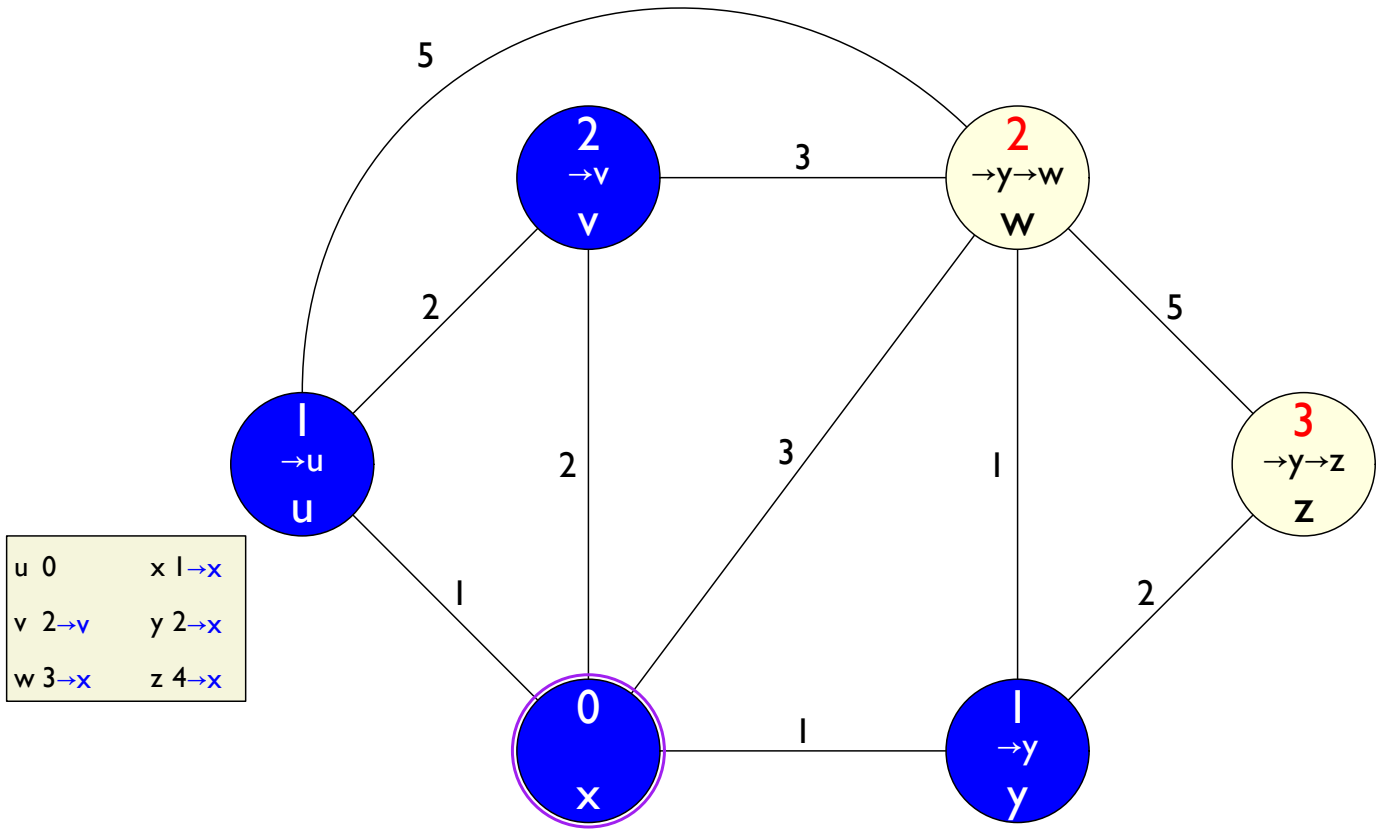
Finding Routes with Dijkstra's



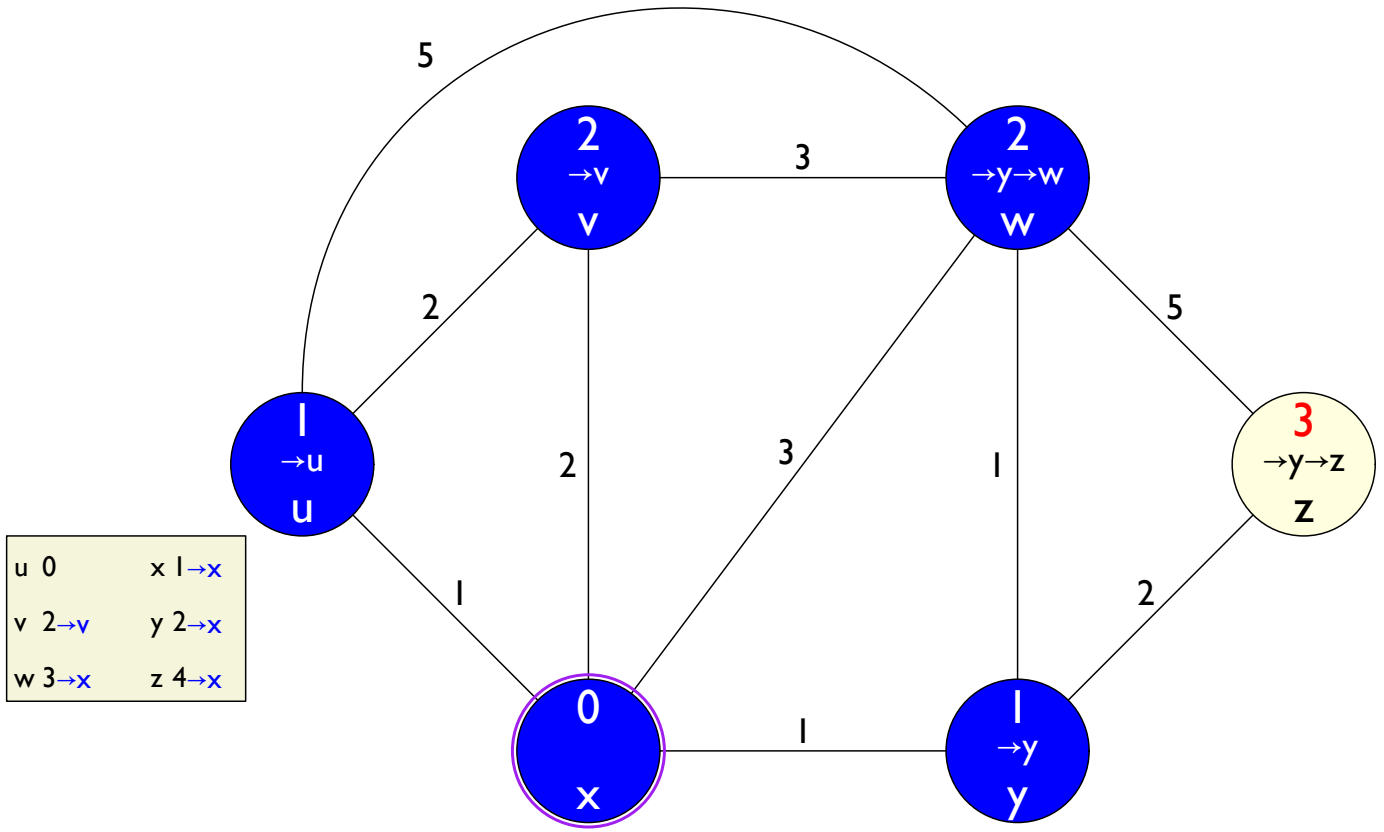
Finding Routes with Dijkstra's



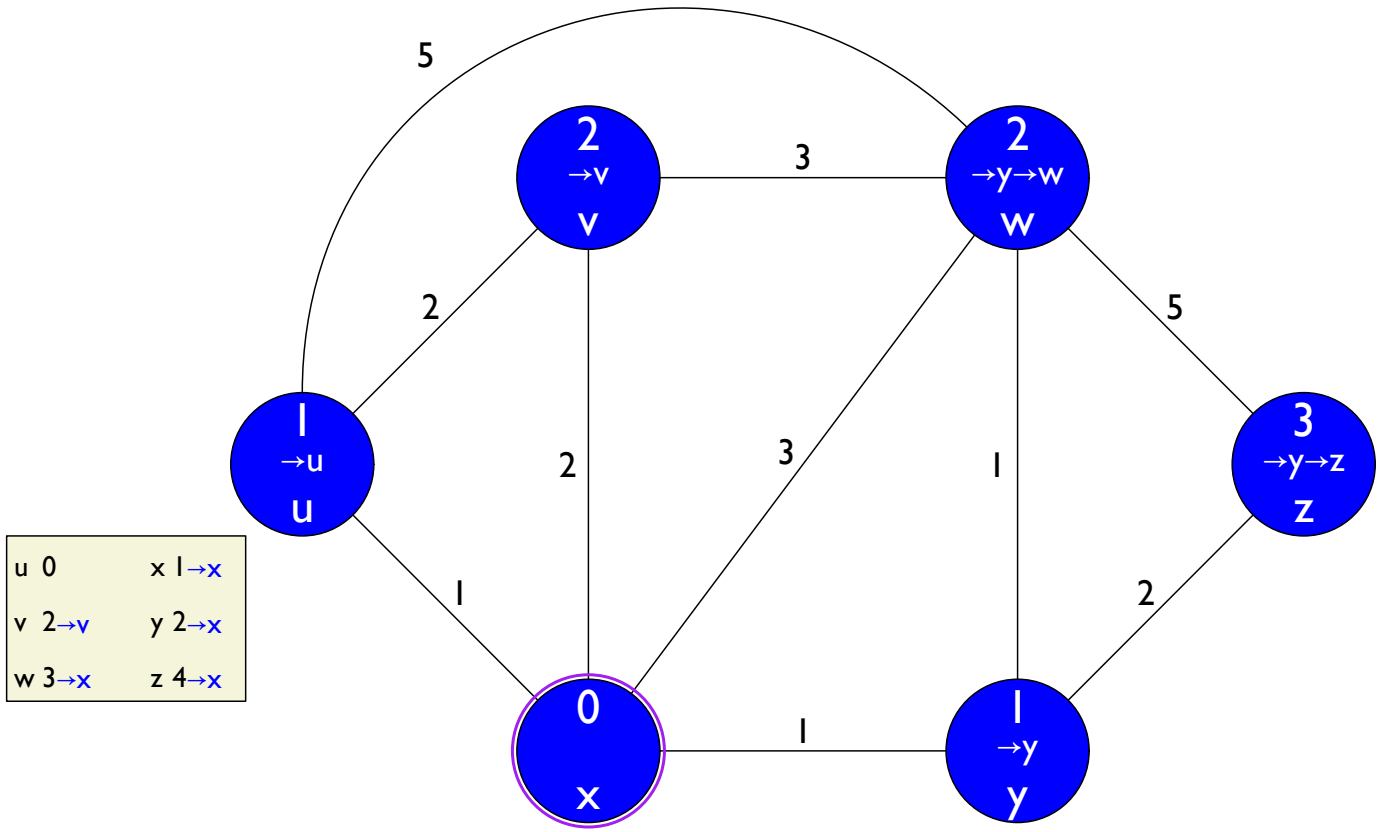
Finding Routes with Dijkstra's



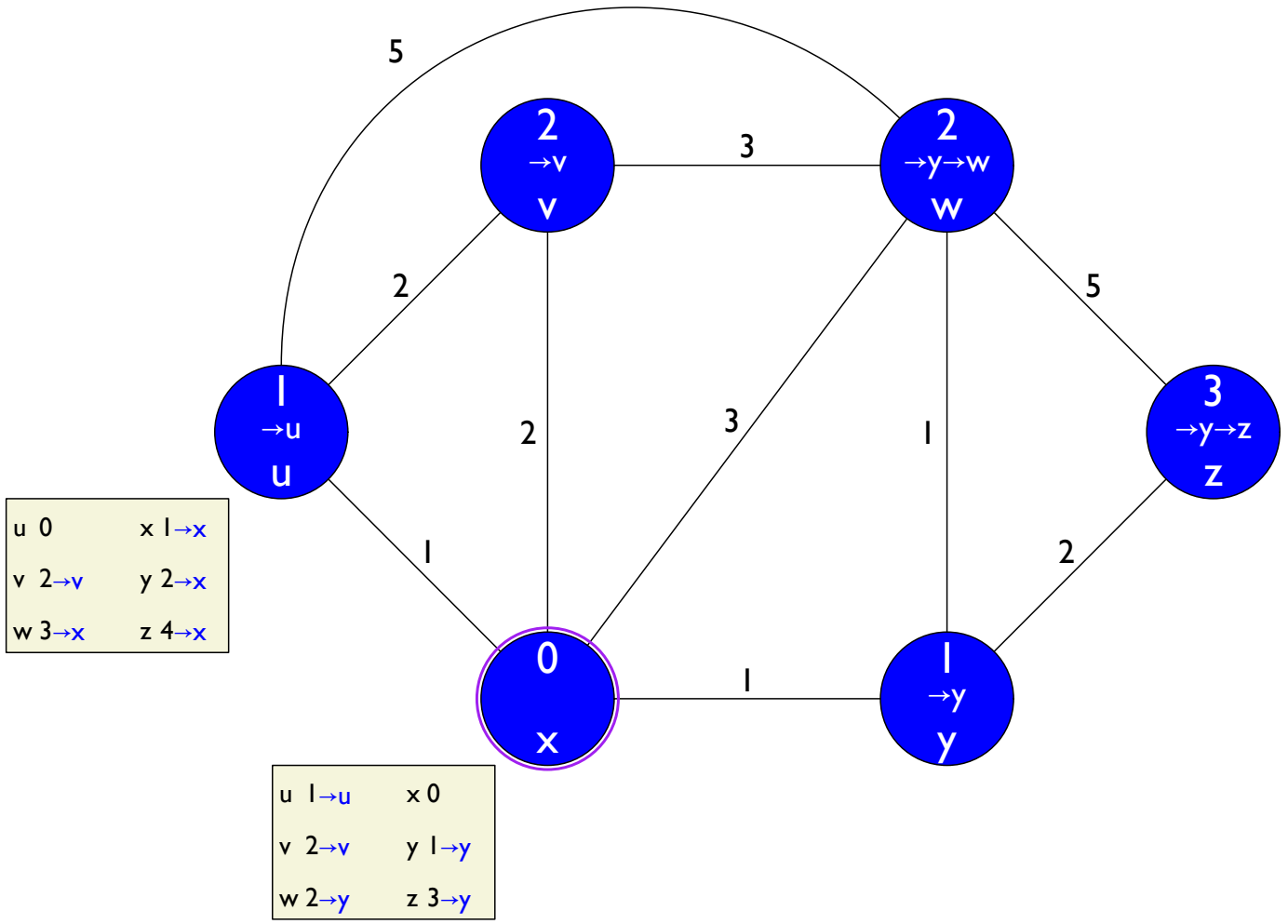
Finding Routes with Dijkstra's



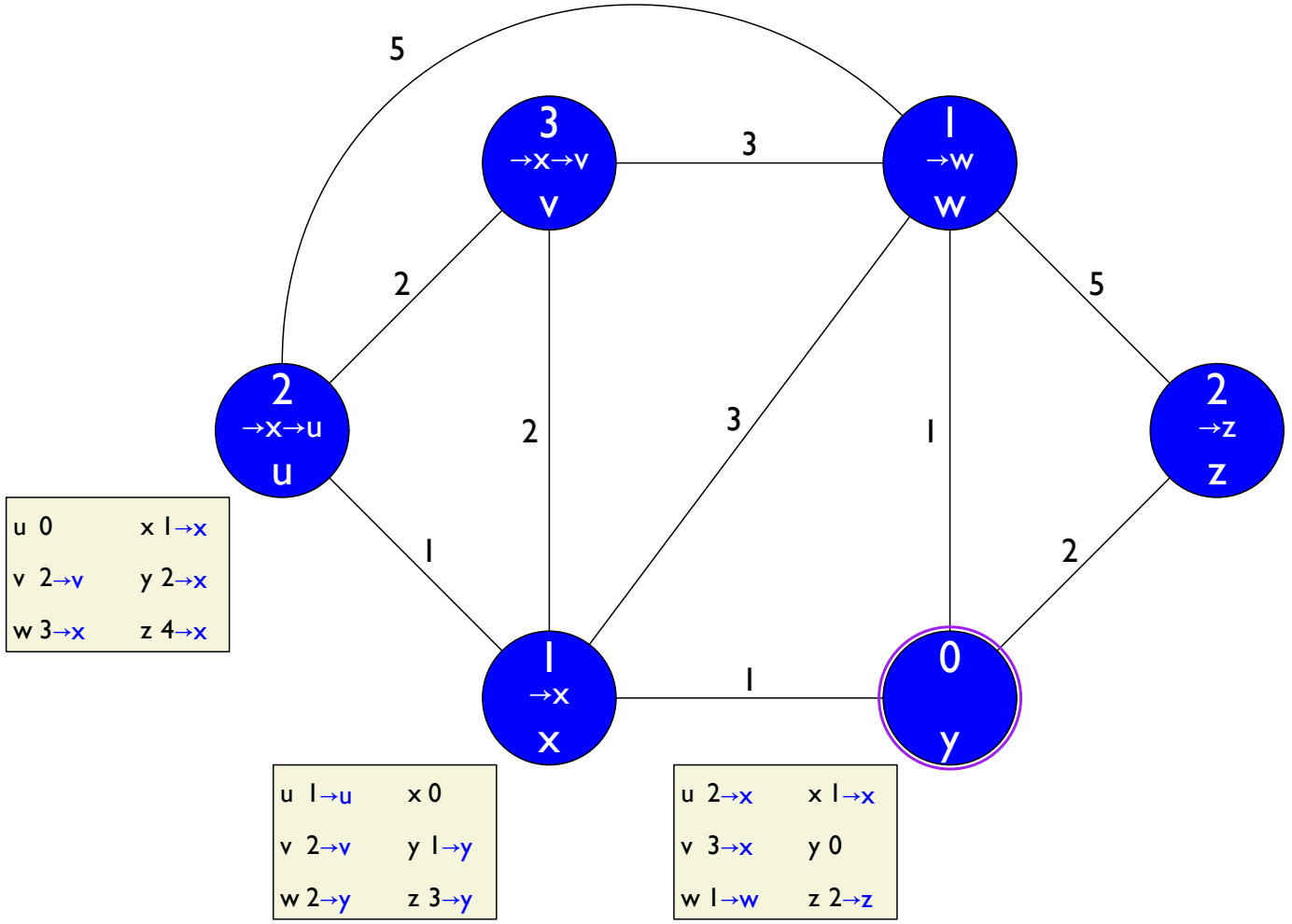
Finding Routes with Dijkstra's



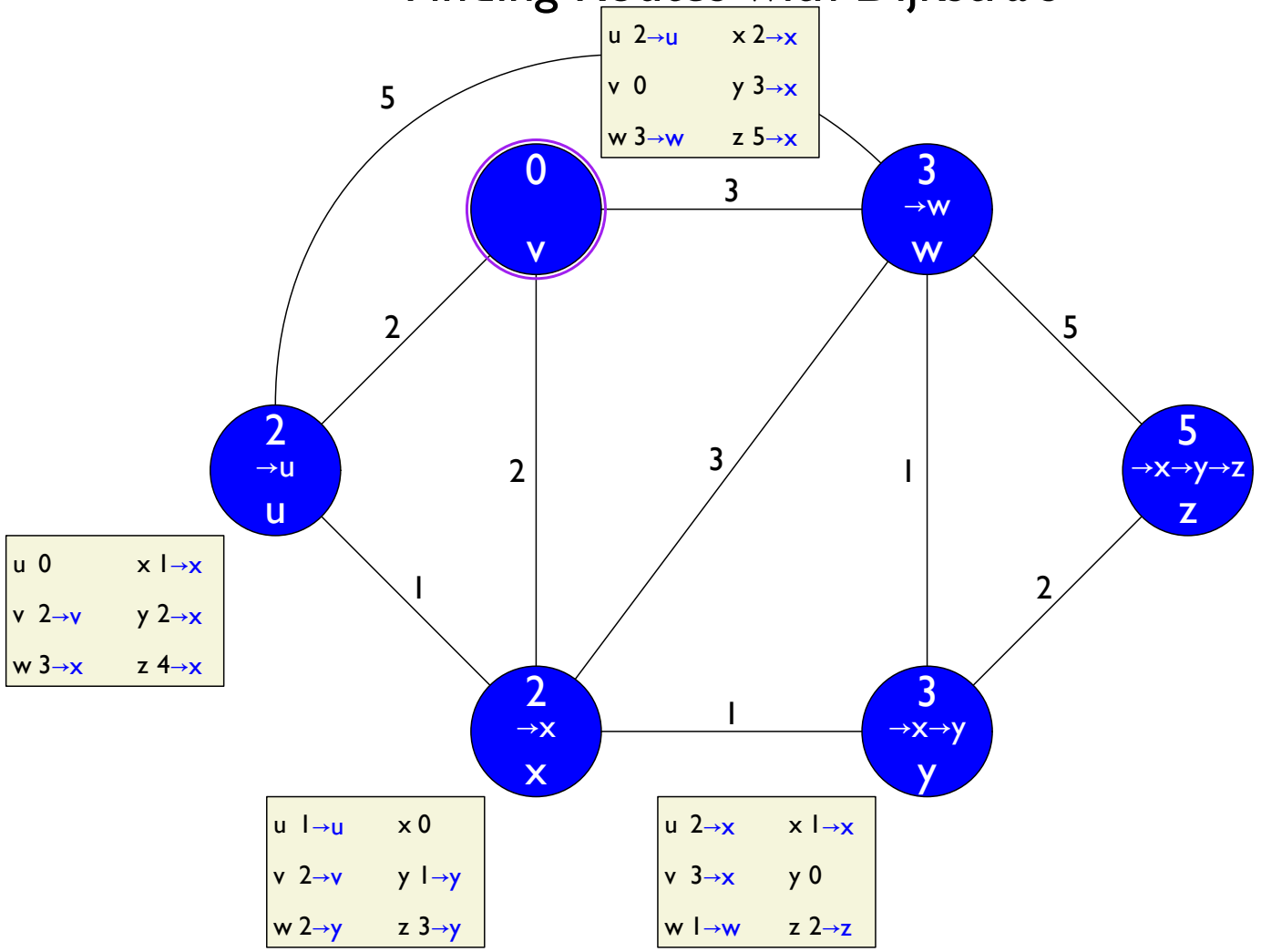
Finding Routes with Dijkstra's



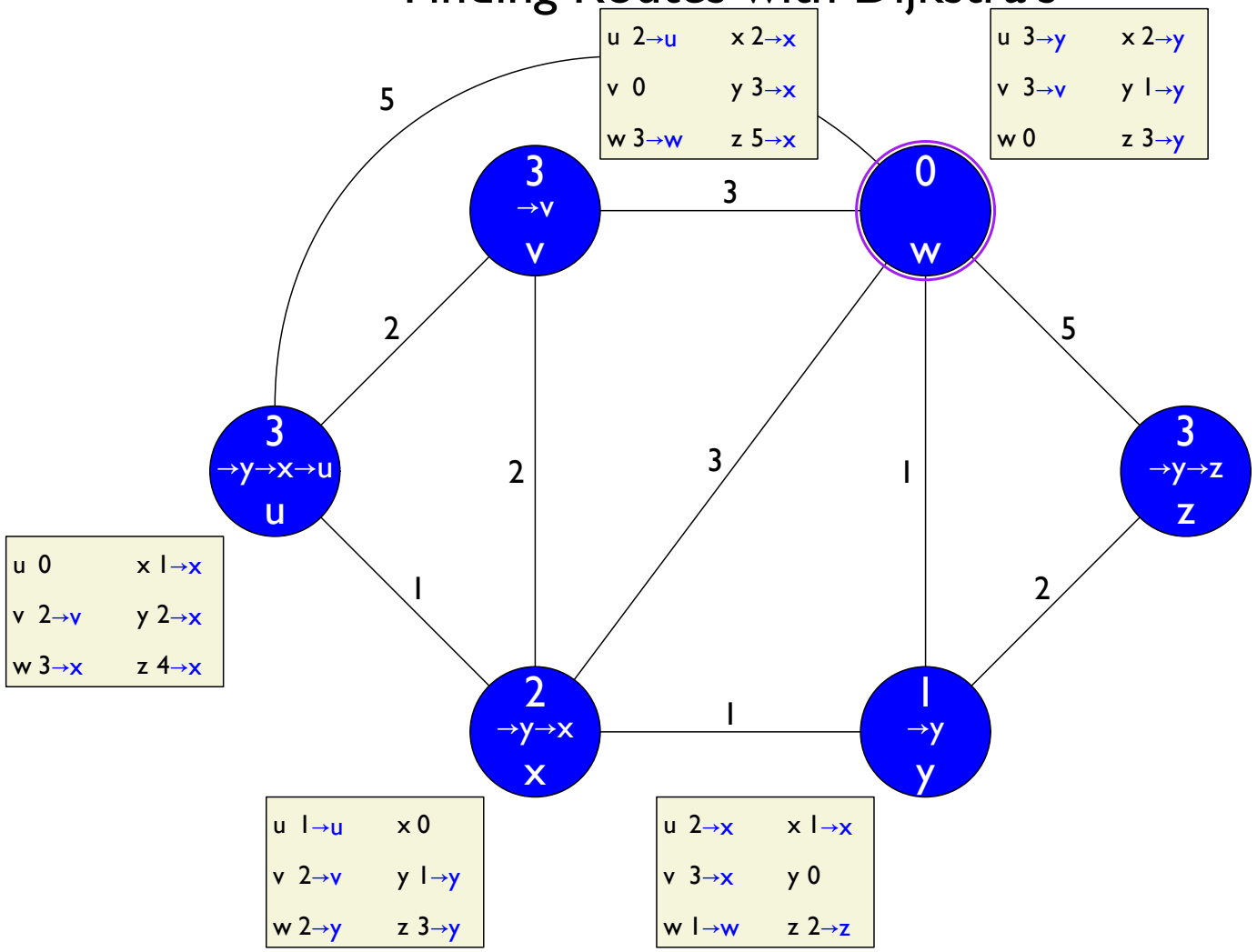
Finding Routes with Dijkstra's



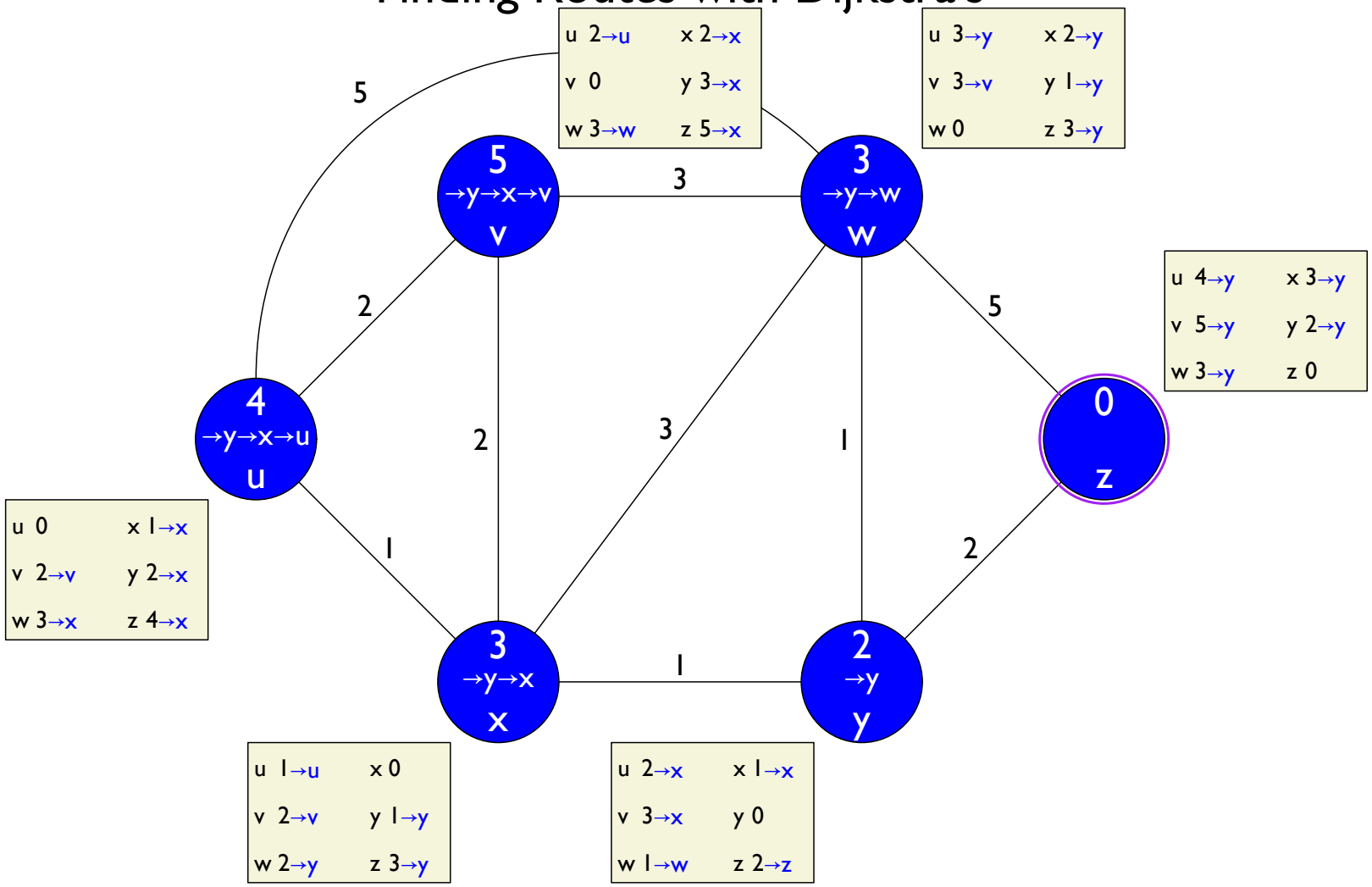
Finding Routes with Dijkstra's



Finding Routes with Dijkstra's



Finding Routes with Dijkstra's



Finding Routes with Dijkstra's

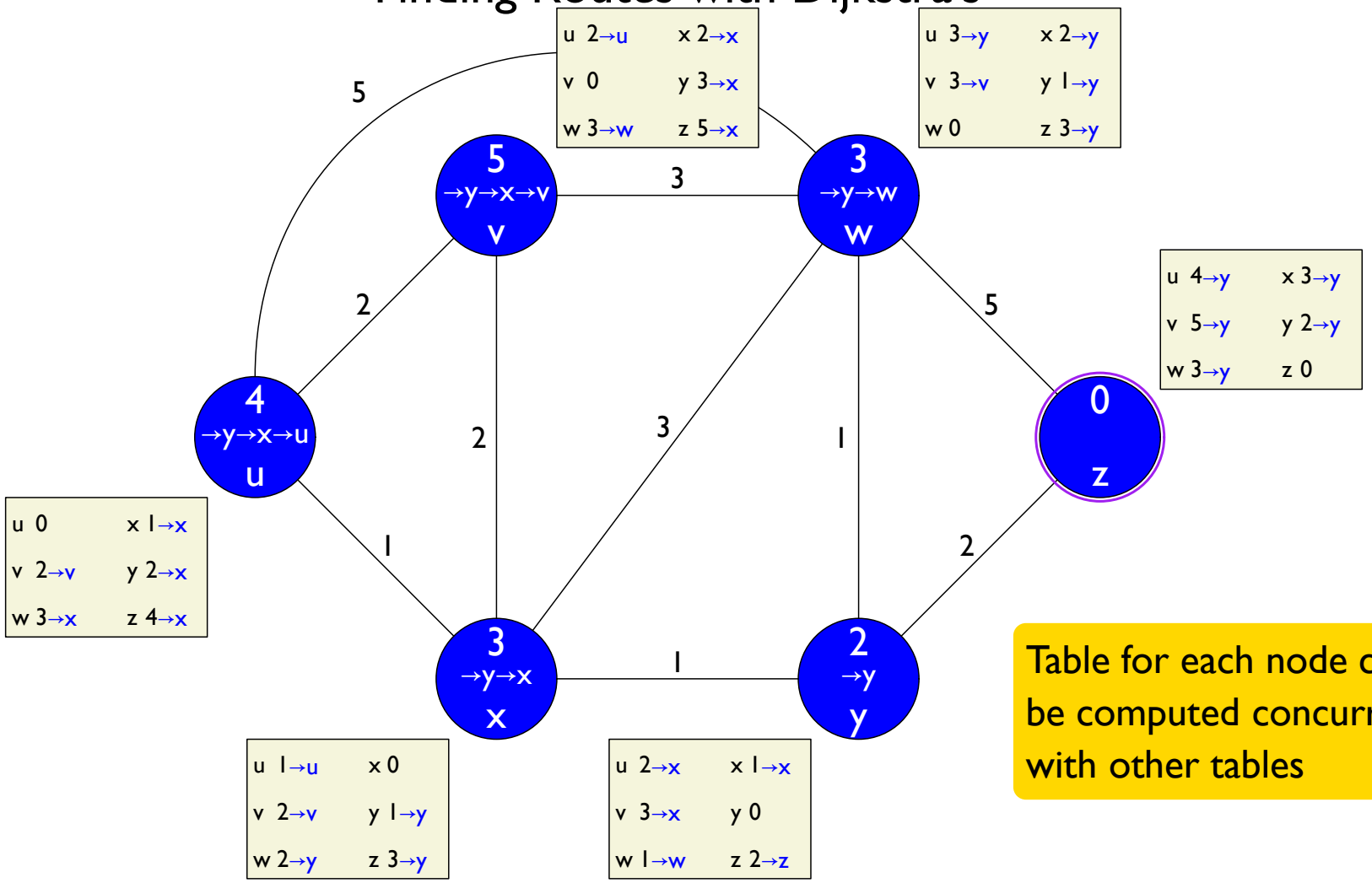
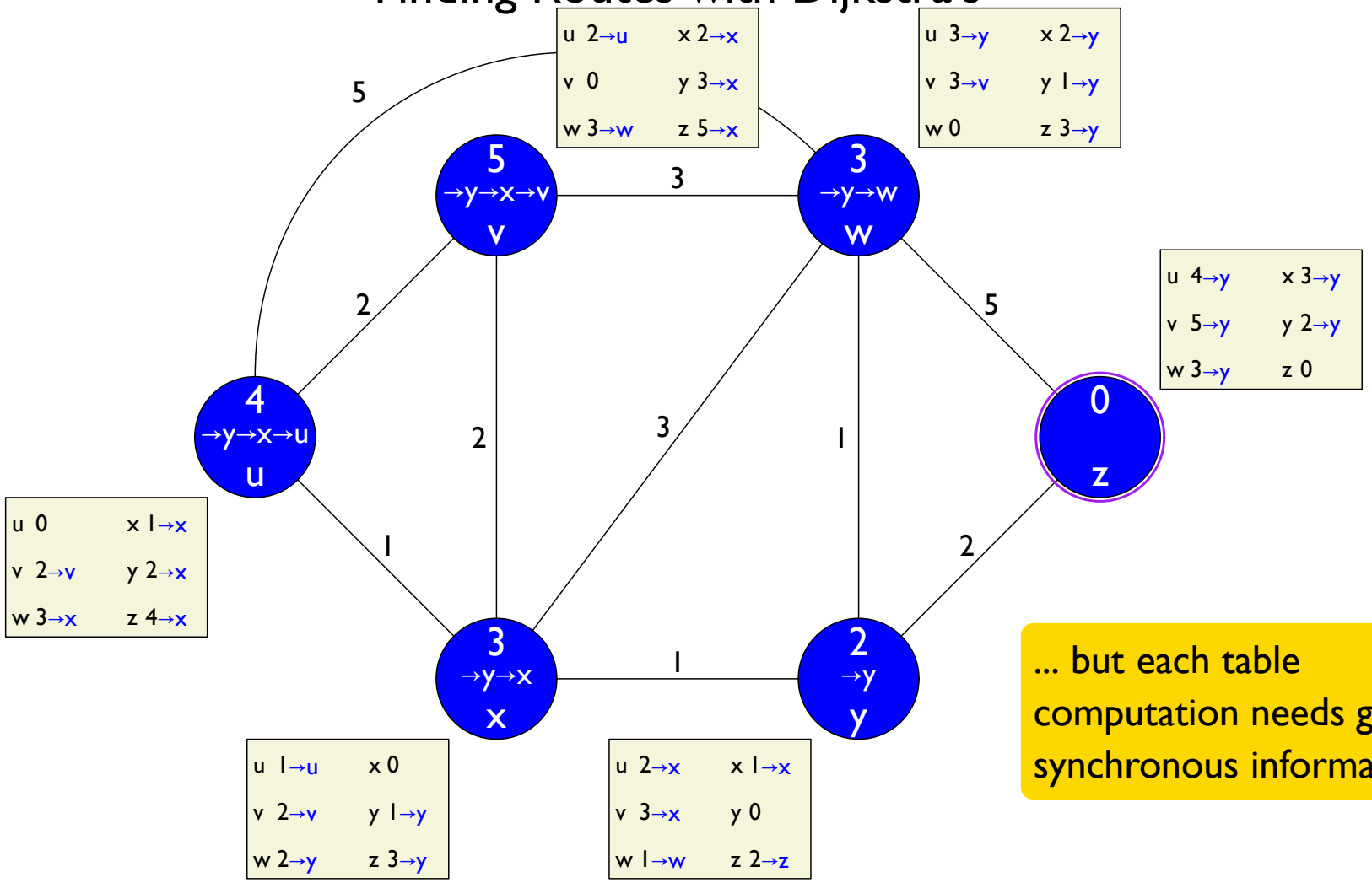


Table for each node can be computed concurrently with other tables

Finding Routes with Dijkstra's



... but each table computation needs global, synchronous information

Rationale for Dijkstra's

$d_x(y)$ is the shortest distance from x to y

$c(v, y)$ reports the distance from v to immediately connected y

$$d_x(y) \quad = \quad \min_v \{ d_x(v) + c(v, y) \}$$

Using $v \in$ set of nodes for which we've found $d_x(v)$:

- add smallest possible $d_x(y)$ among available y
- we've always picked smallest so far,
  and each next pick is same or longer,
  so we'll never find a smaller route later

Rationale for Bellman-Ford

$d_x(y)$ is the shortest distance from x to y

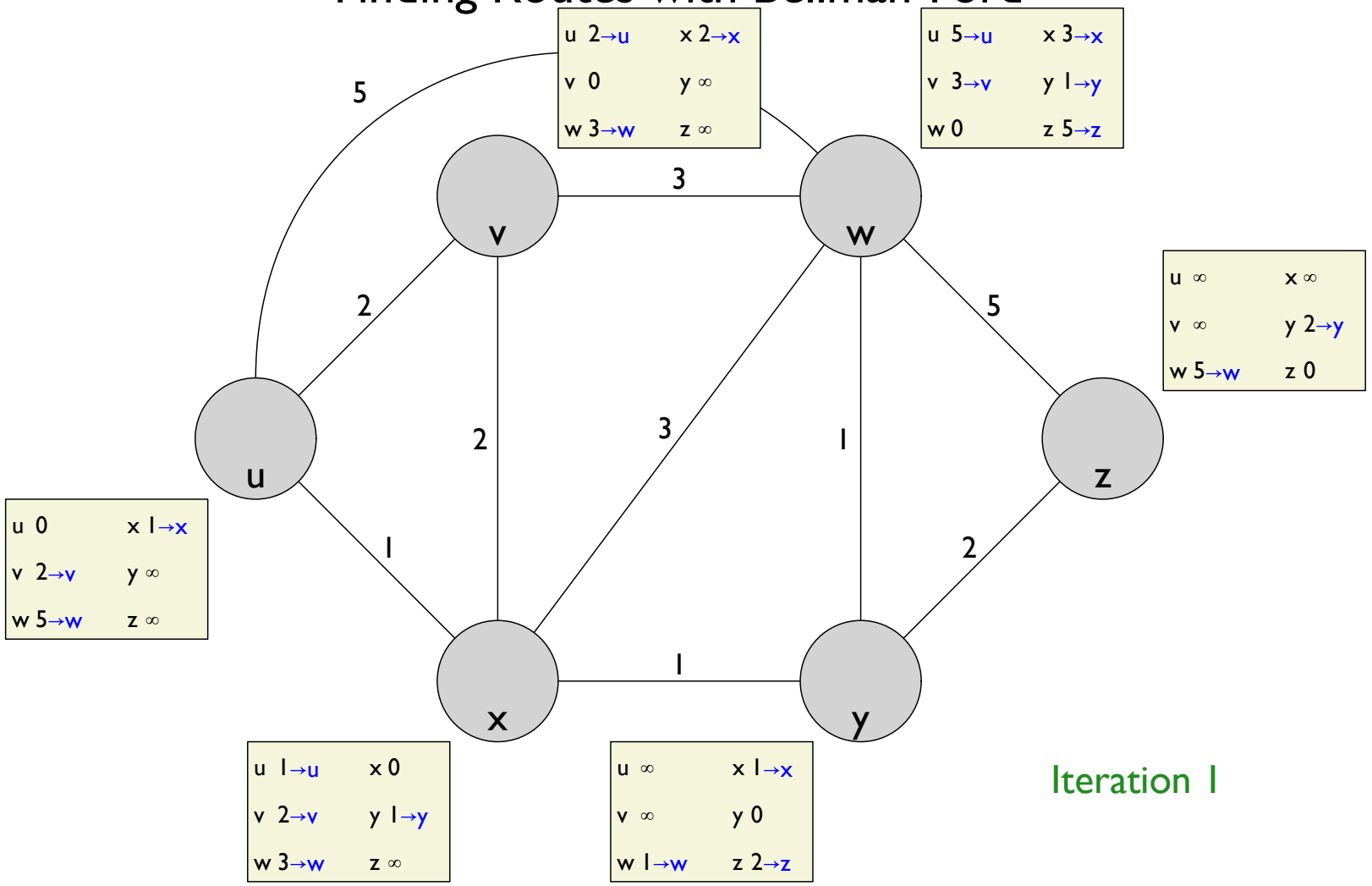
$c(x, v)$ reports the distance from x to immediately connected v

$$d_x(y) \quad = \quad \min_v \{ c(x, v) + d_v(y) \}$$

Start with $d_x(y) = c(x, y)$ where defined, $d_x(y) = \infty$ otherwise

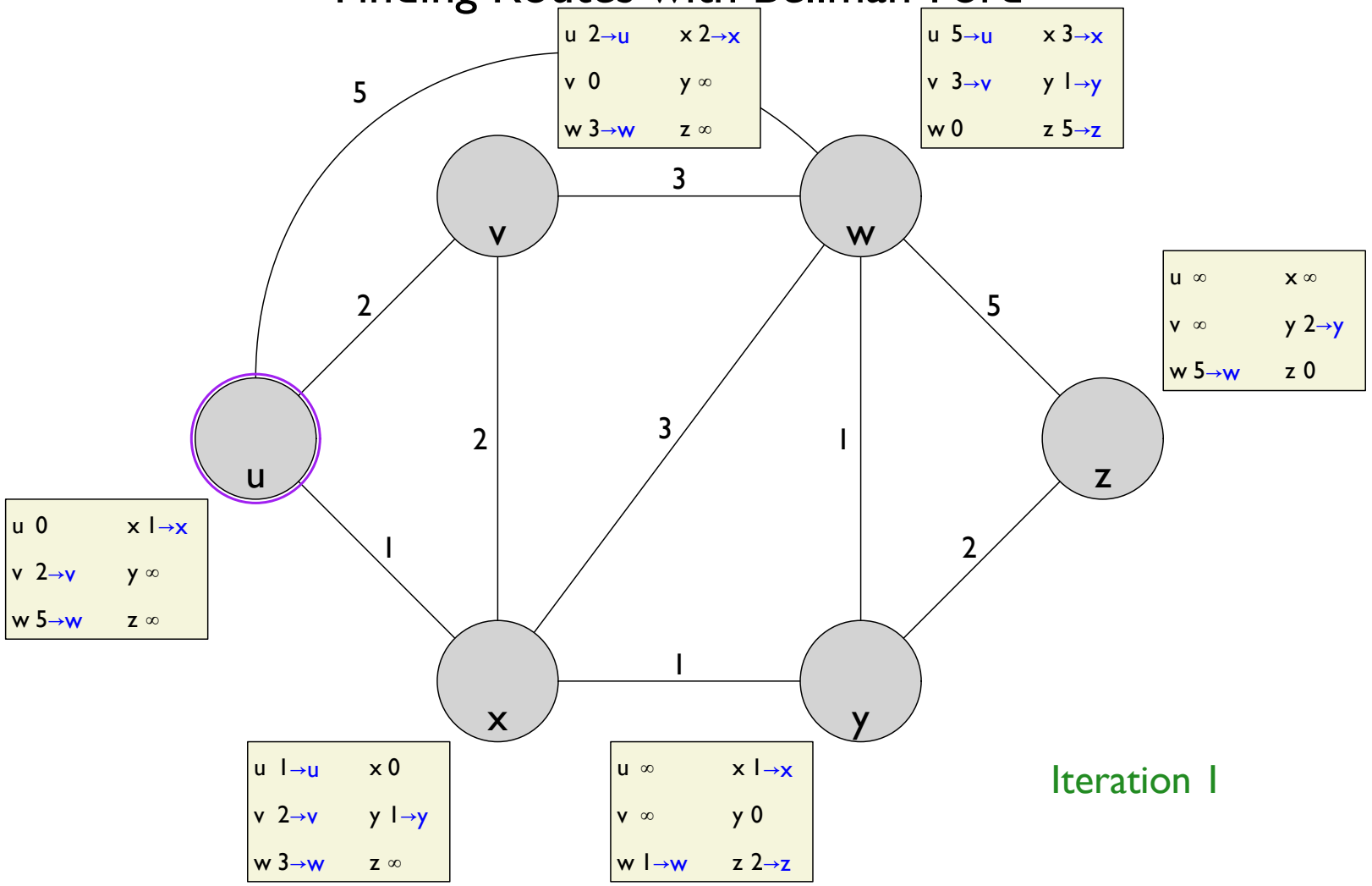
- check every $c(x, v) + d_v(y)$ and maybe update $d_x(y)$
- update can only get smaller,
  and steps down are a fixed size,
  so we'll eventually hit bottom everywhere

Finding Routes with Bellman-Ford



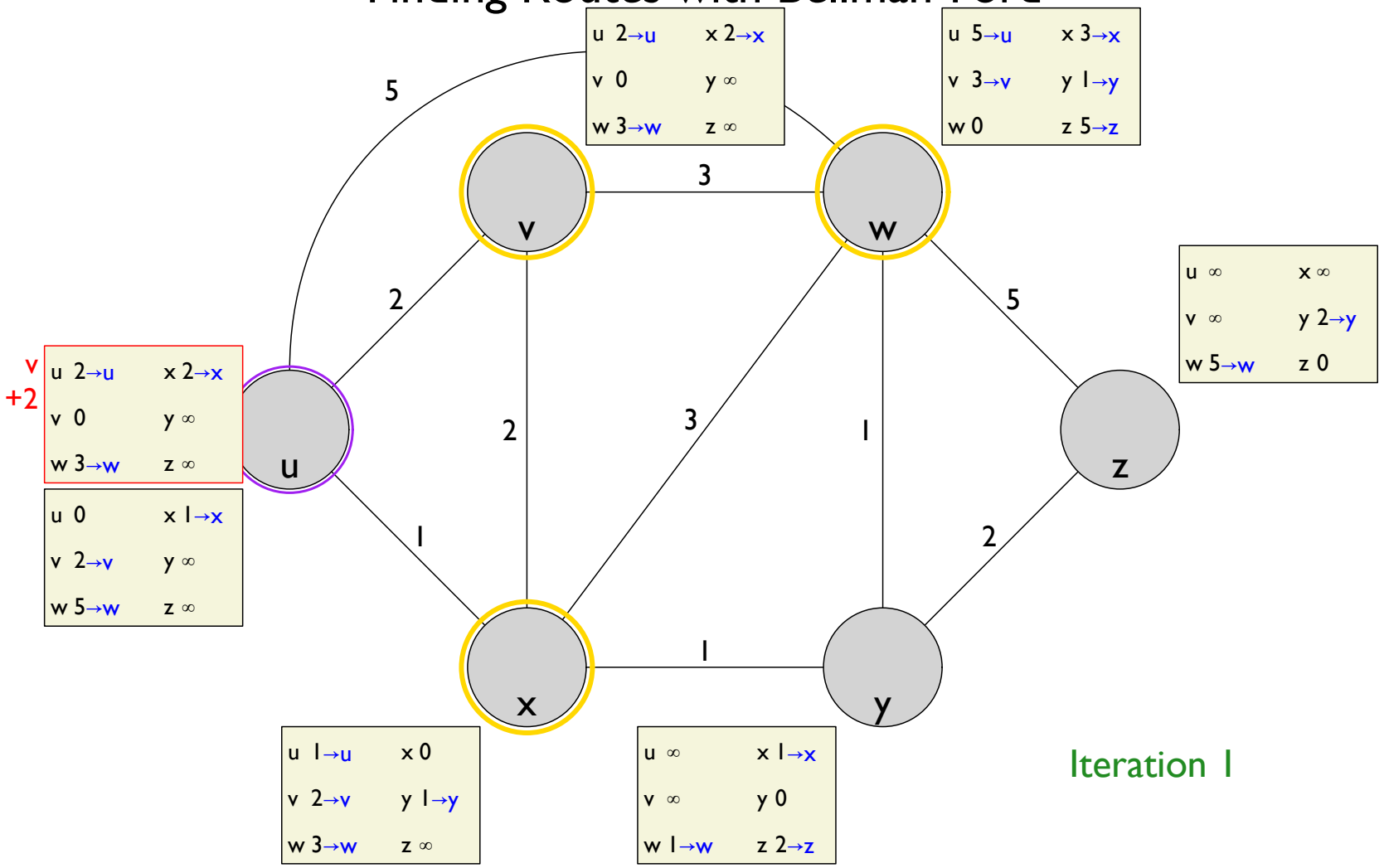
Iteration 1

Finding Routes with Bellman-Ford



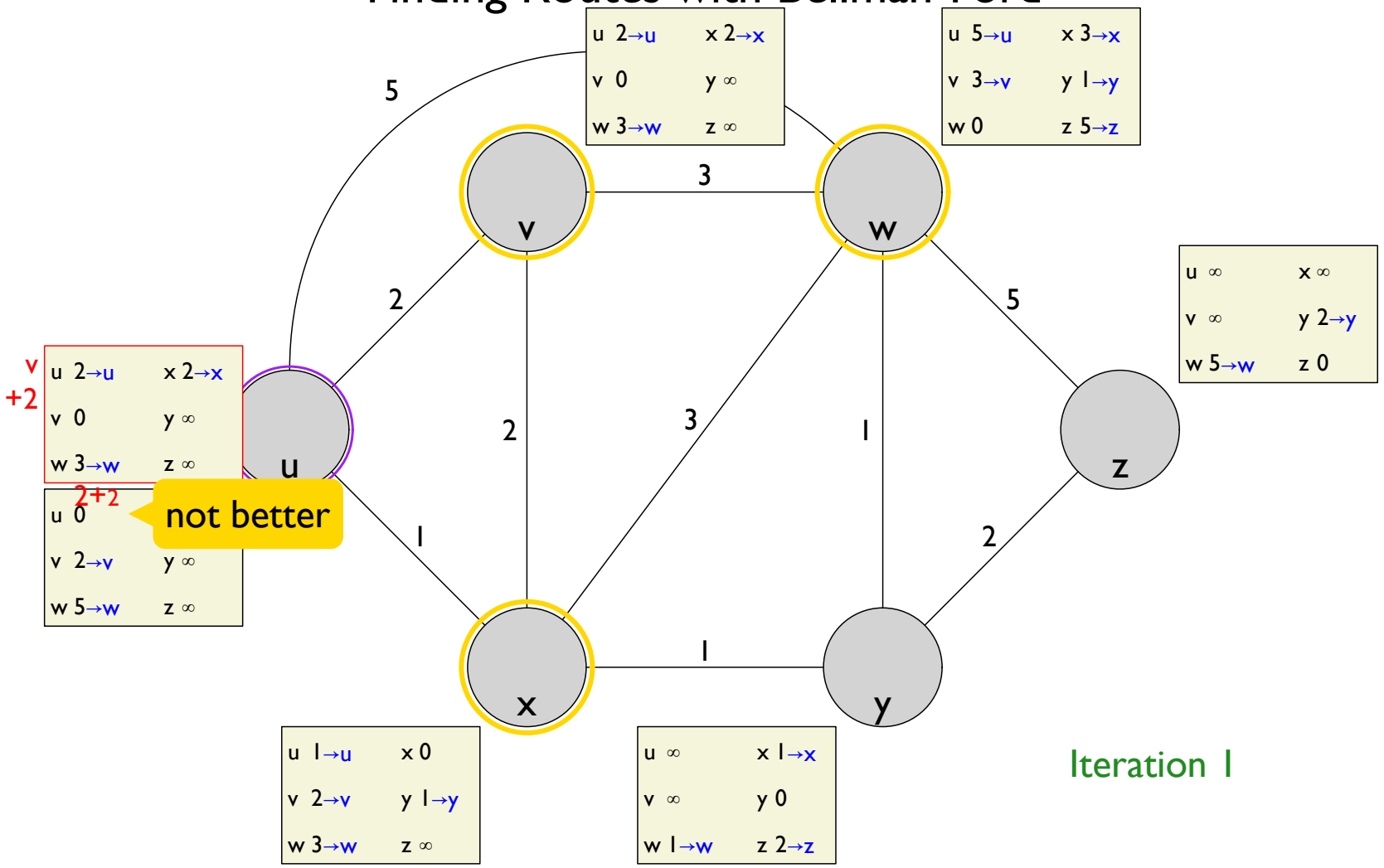
Iteration 1

Finding Routes with Bellman-Ford



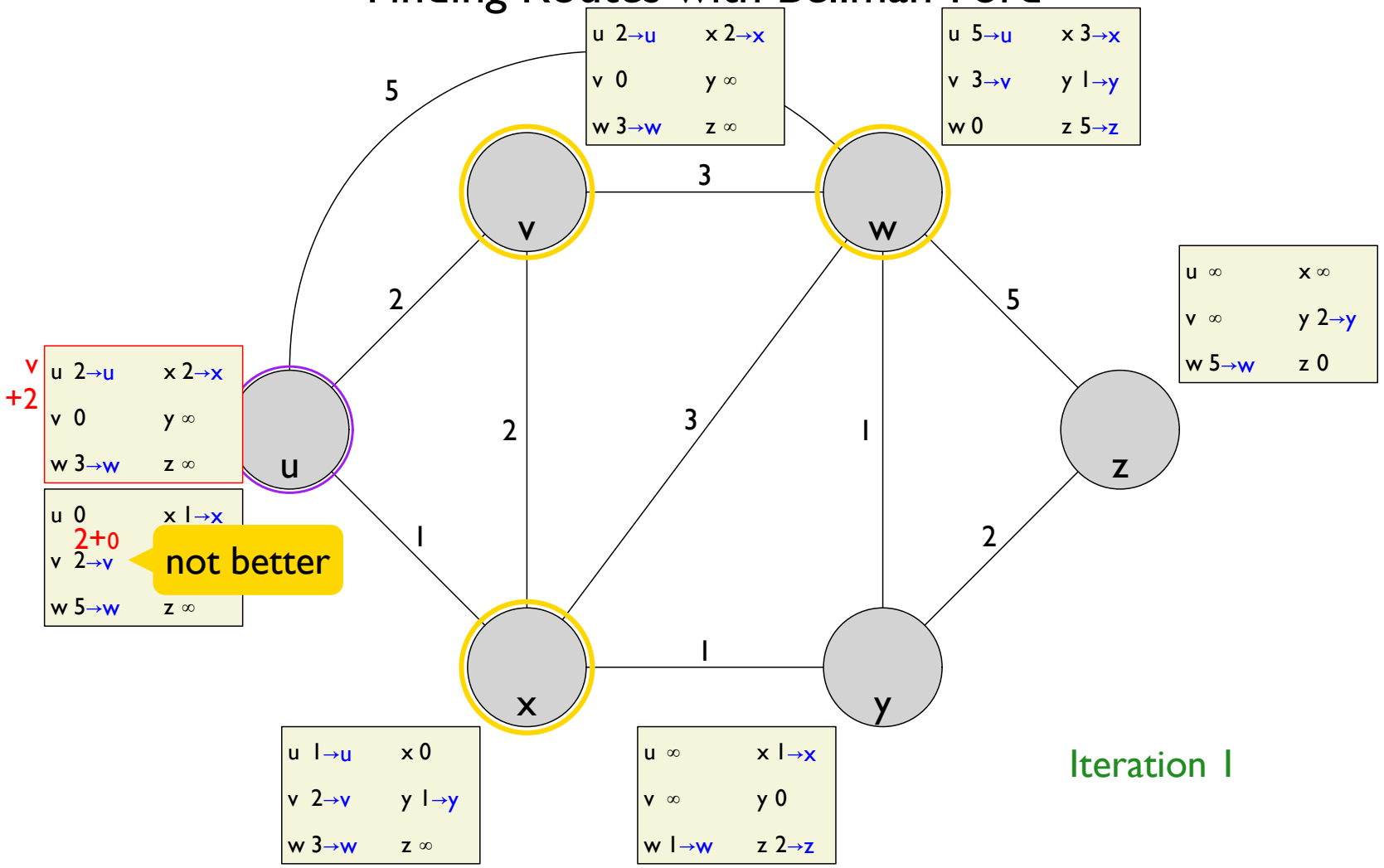
Iteration 1

Finding Routes with Bellman-Ford



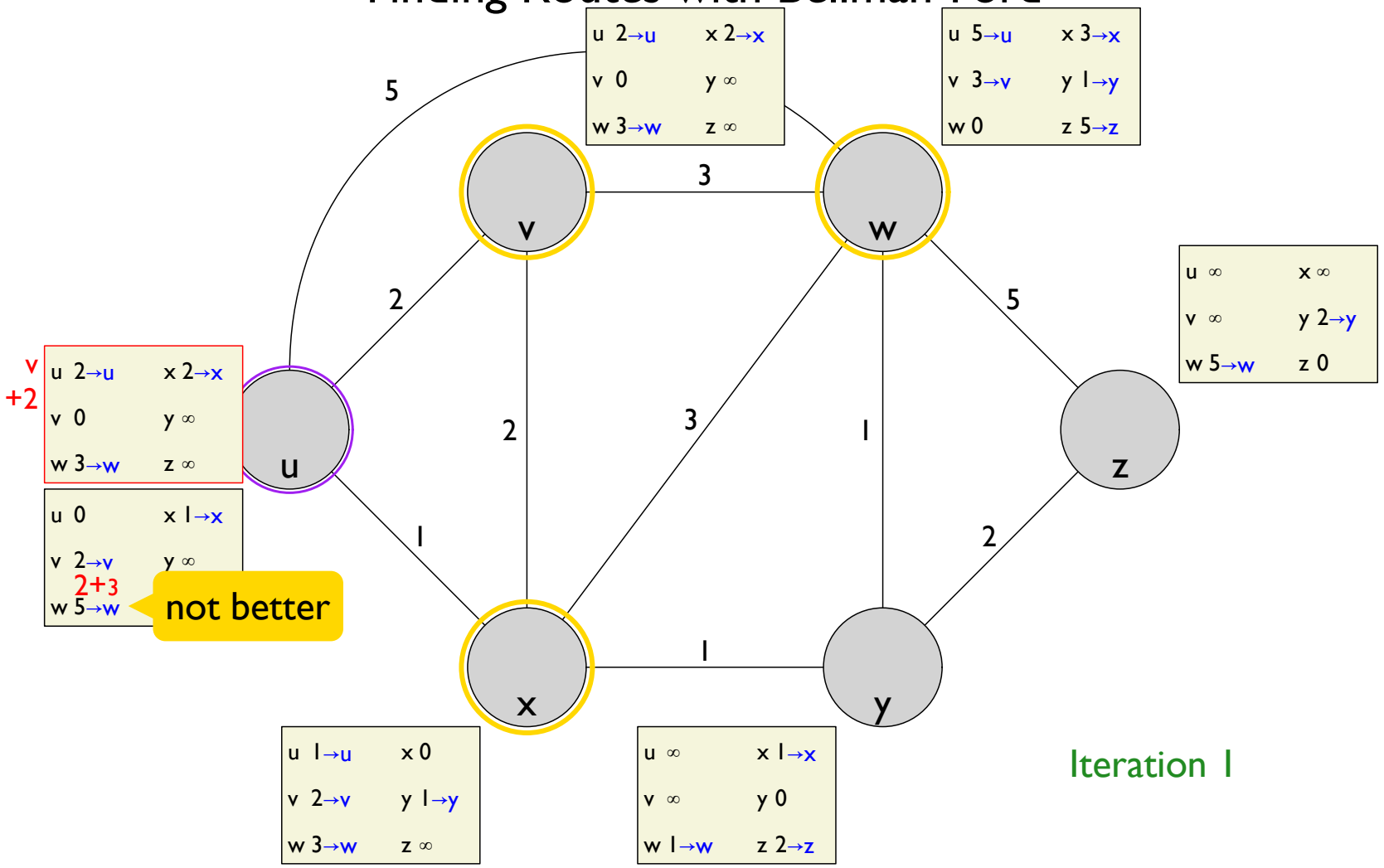
Iteration 1

Finding Routes with Bellman-Ford



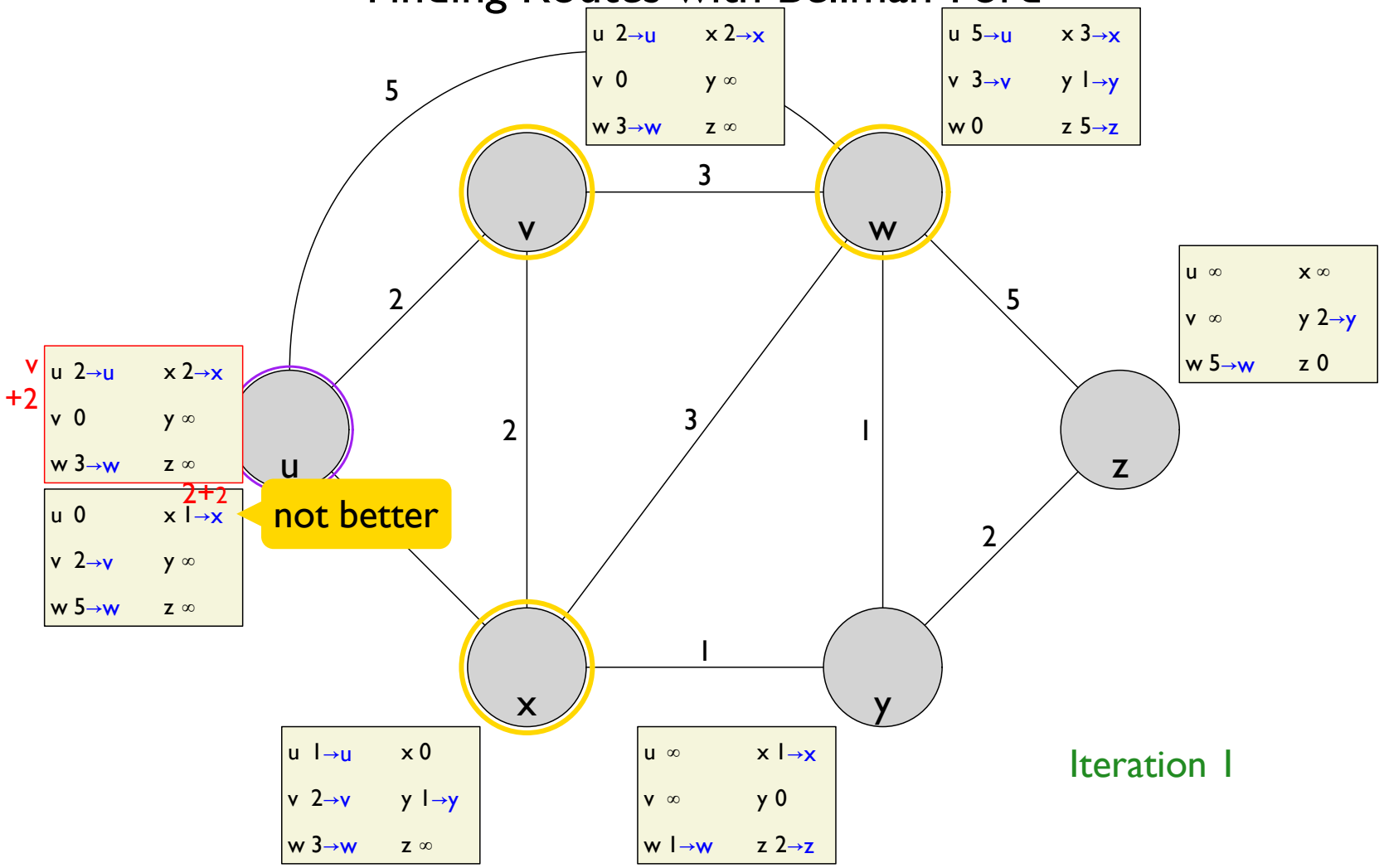
Iteration 1

Finding Routes with Bellman-Ford

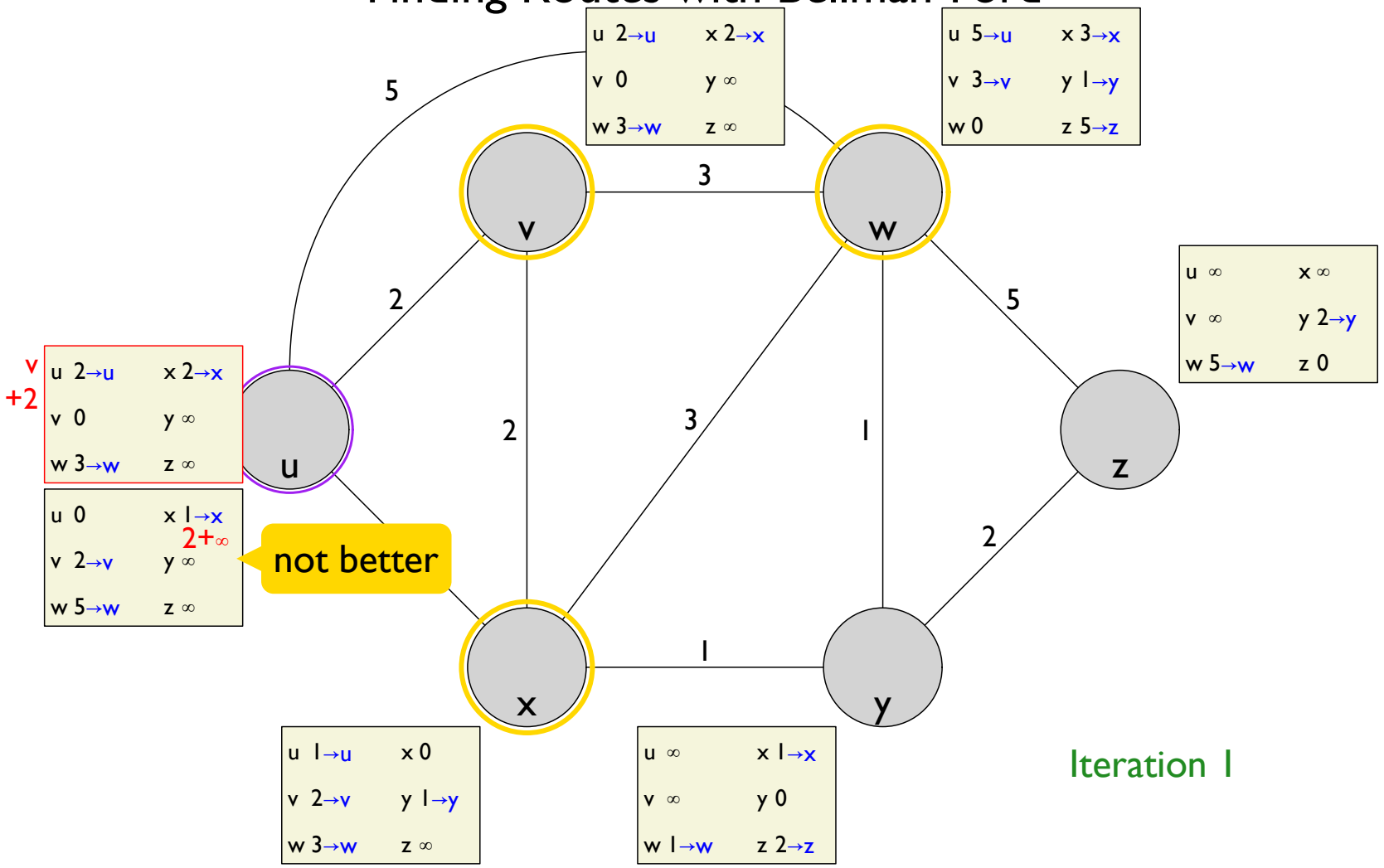


Iteration 1

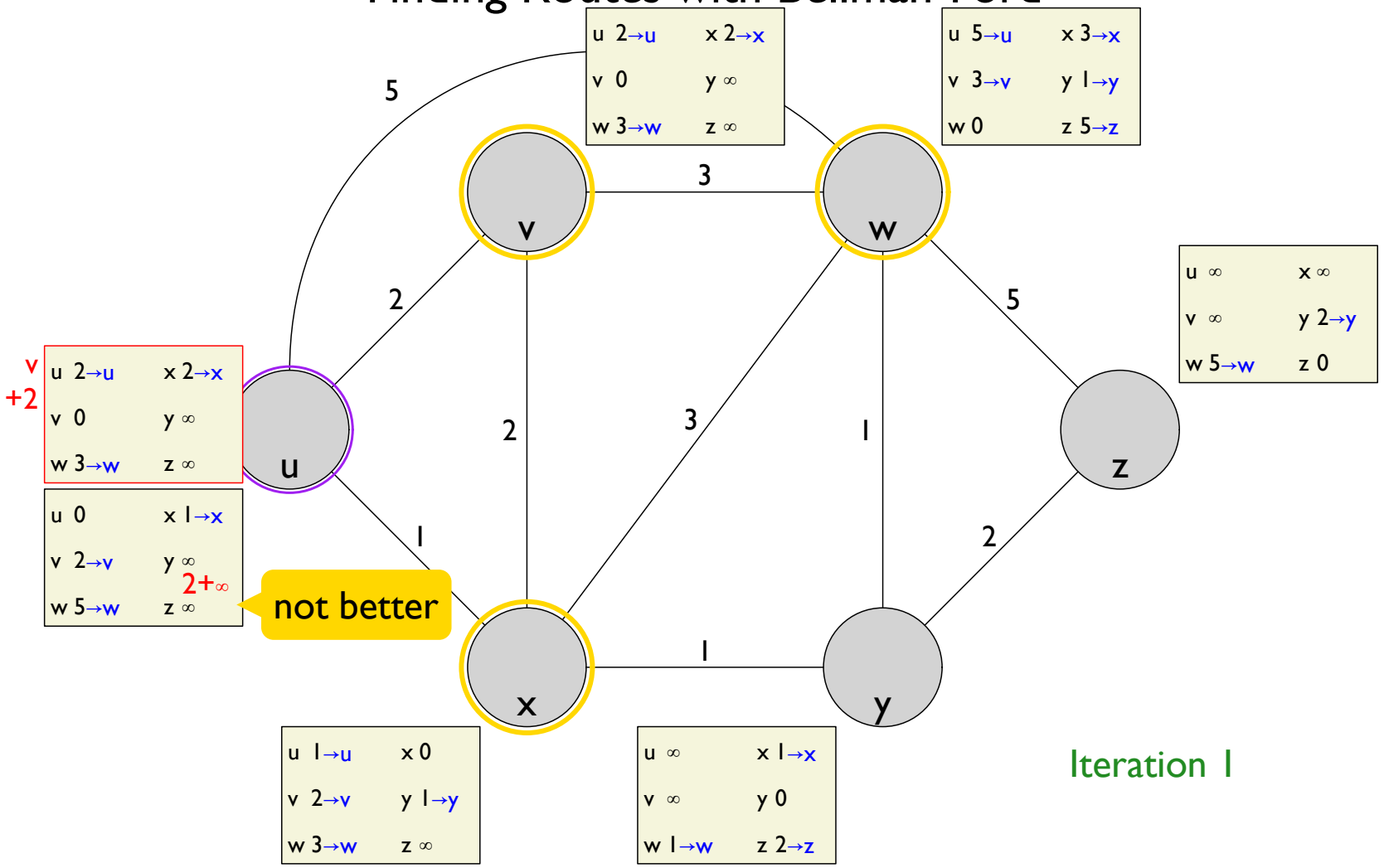
Finding Routes with Bellman-Ford



Finding Routes with Bellman-Ford

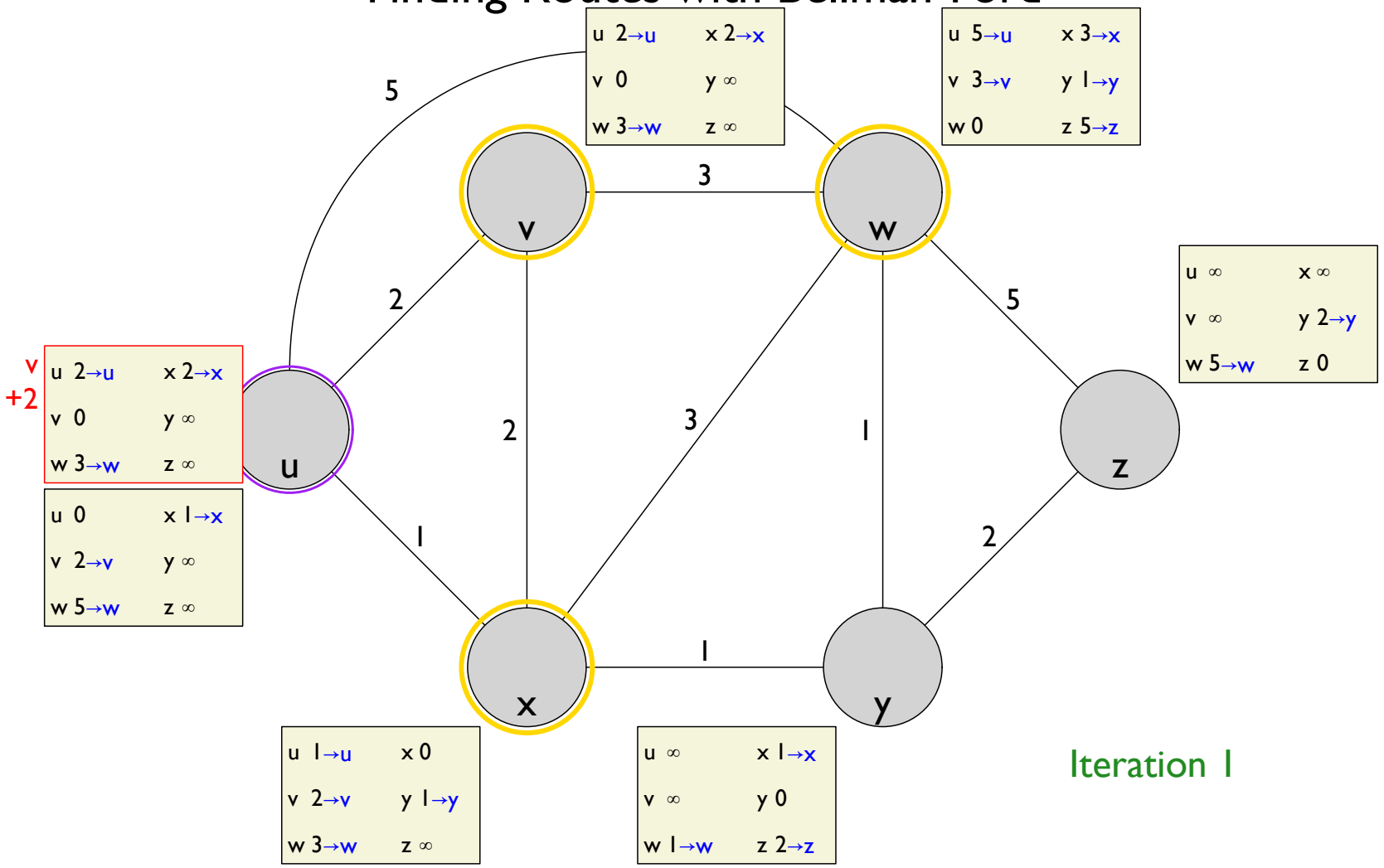


Finding Routes with Bellman-Ford



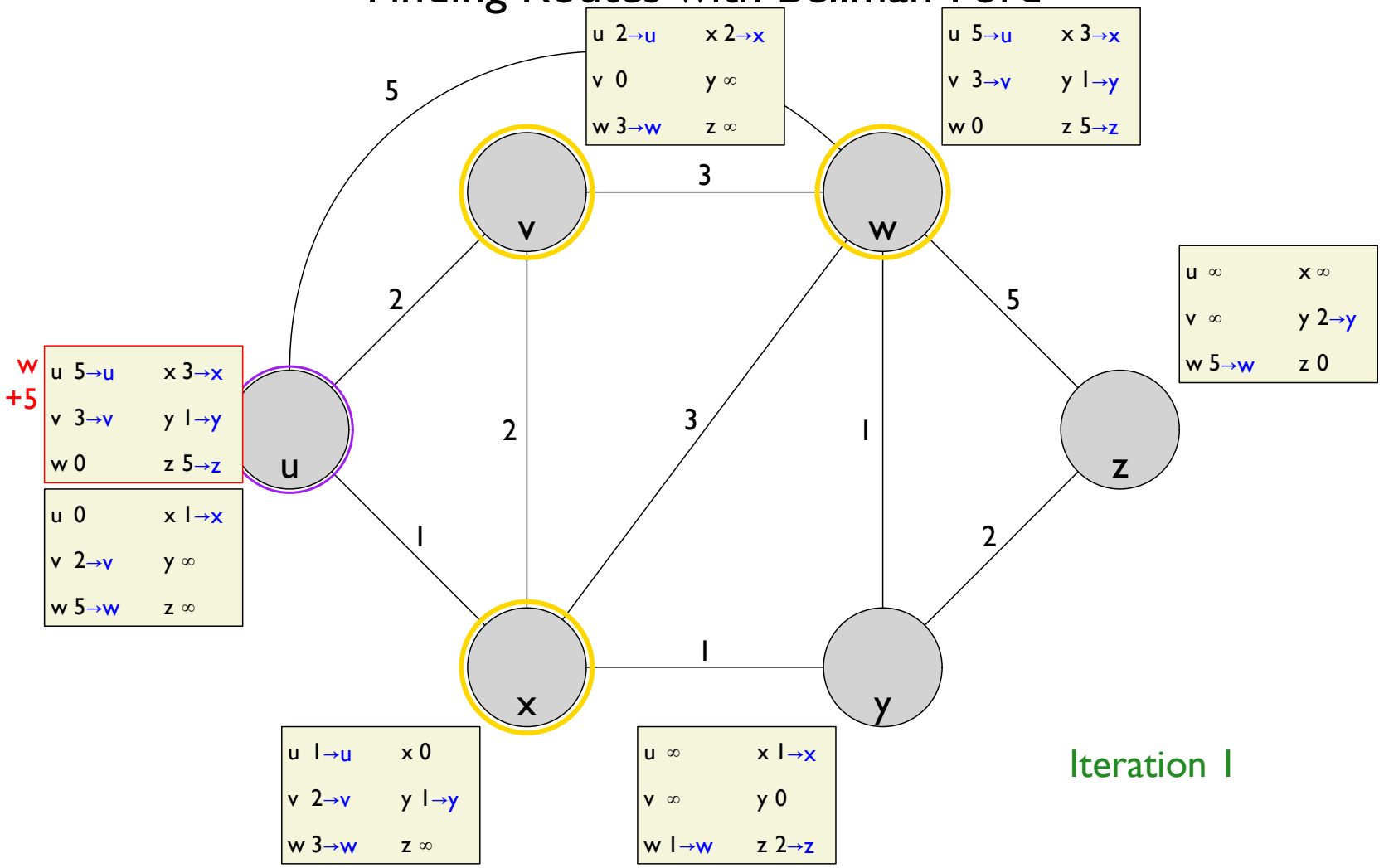
Iteration 1

Finding Routes with Bellman-Ford



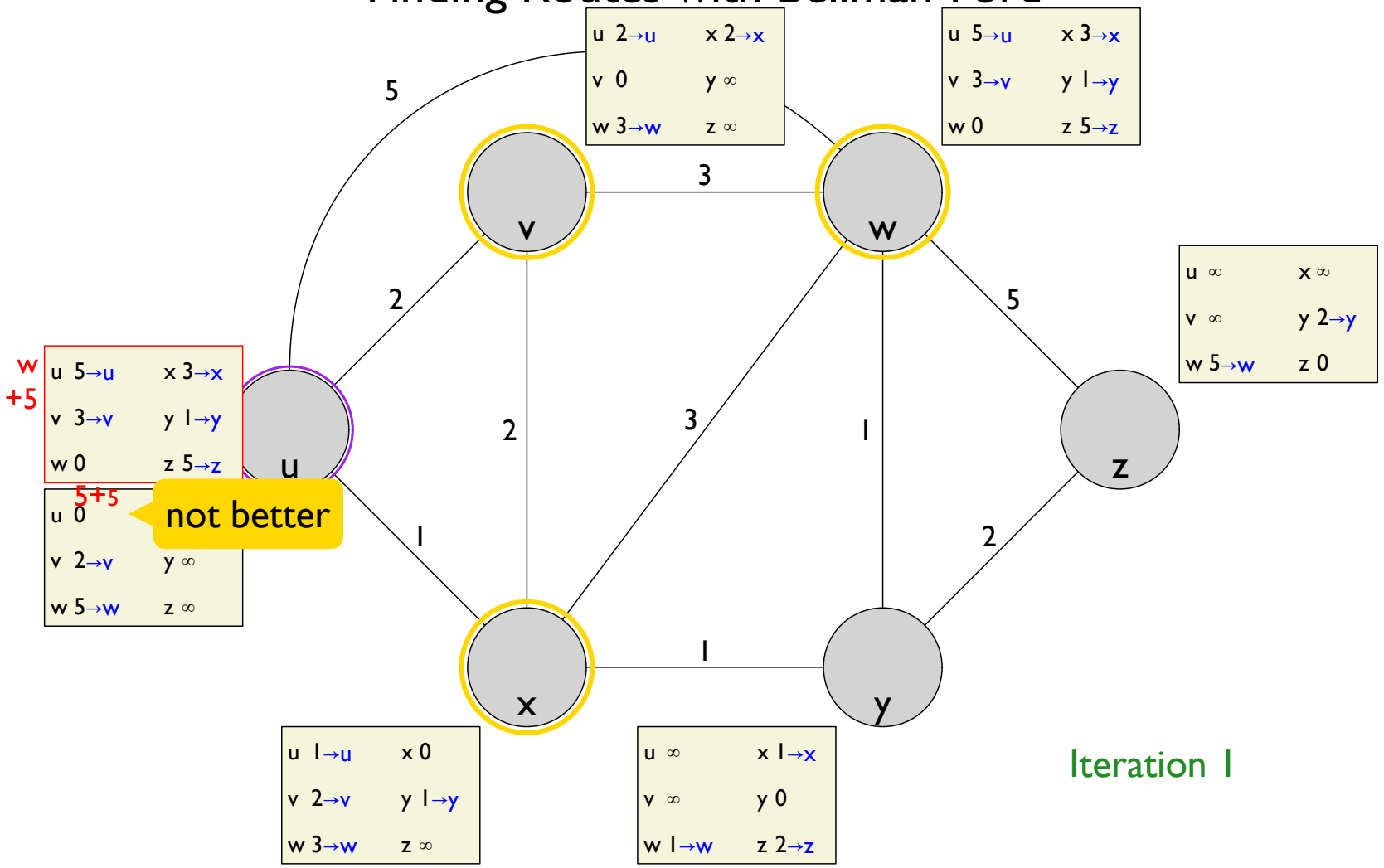
Iteration 1

Finding Routes with Bellman-Ford



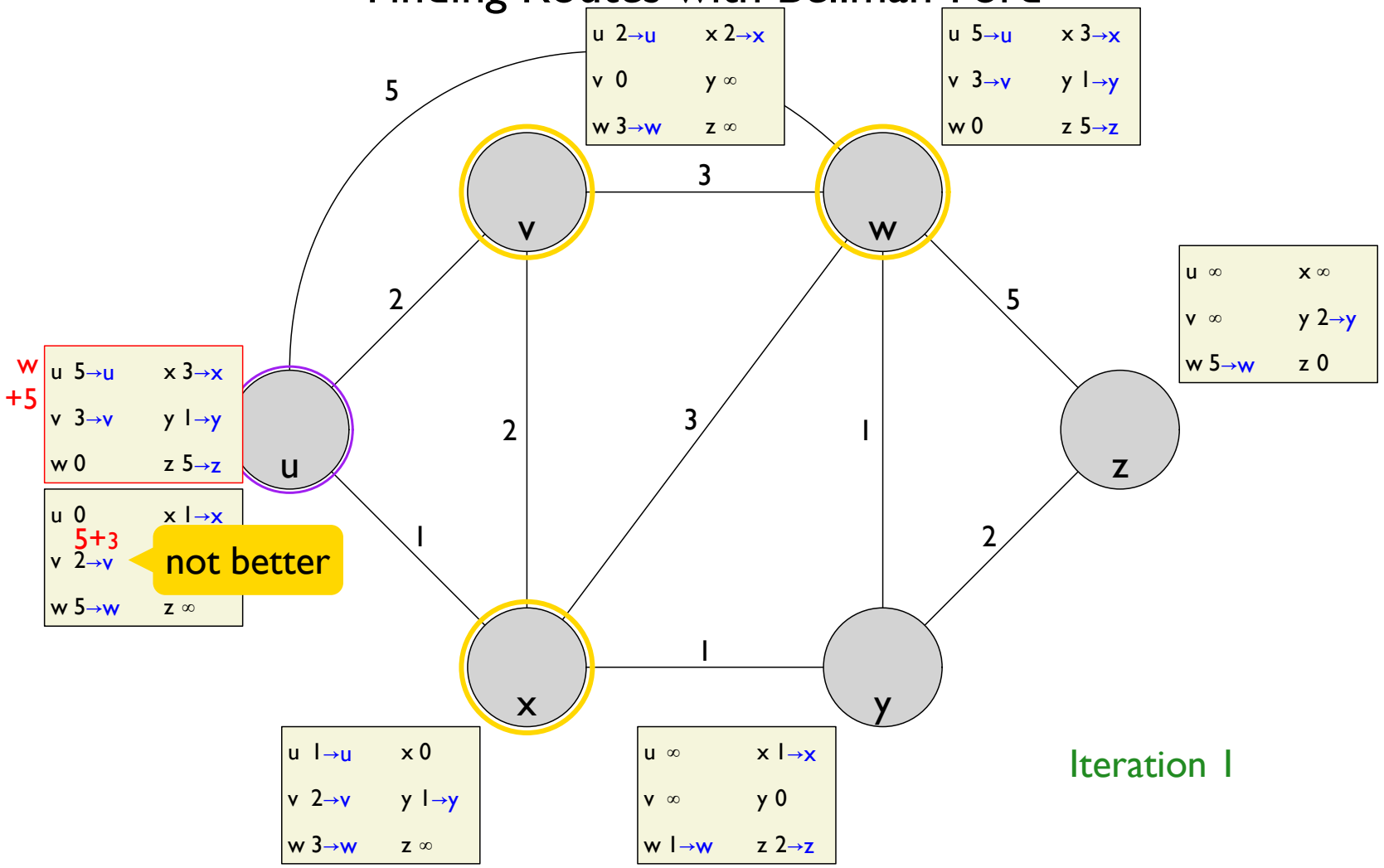
Iteration 1

Finding Routes with Bellman-Ford



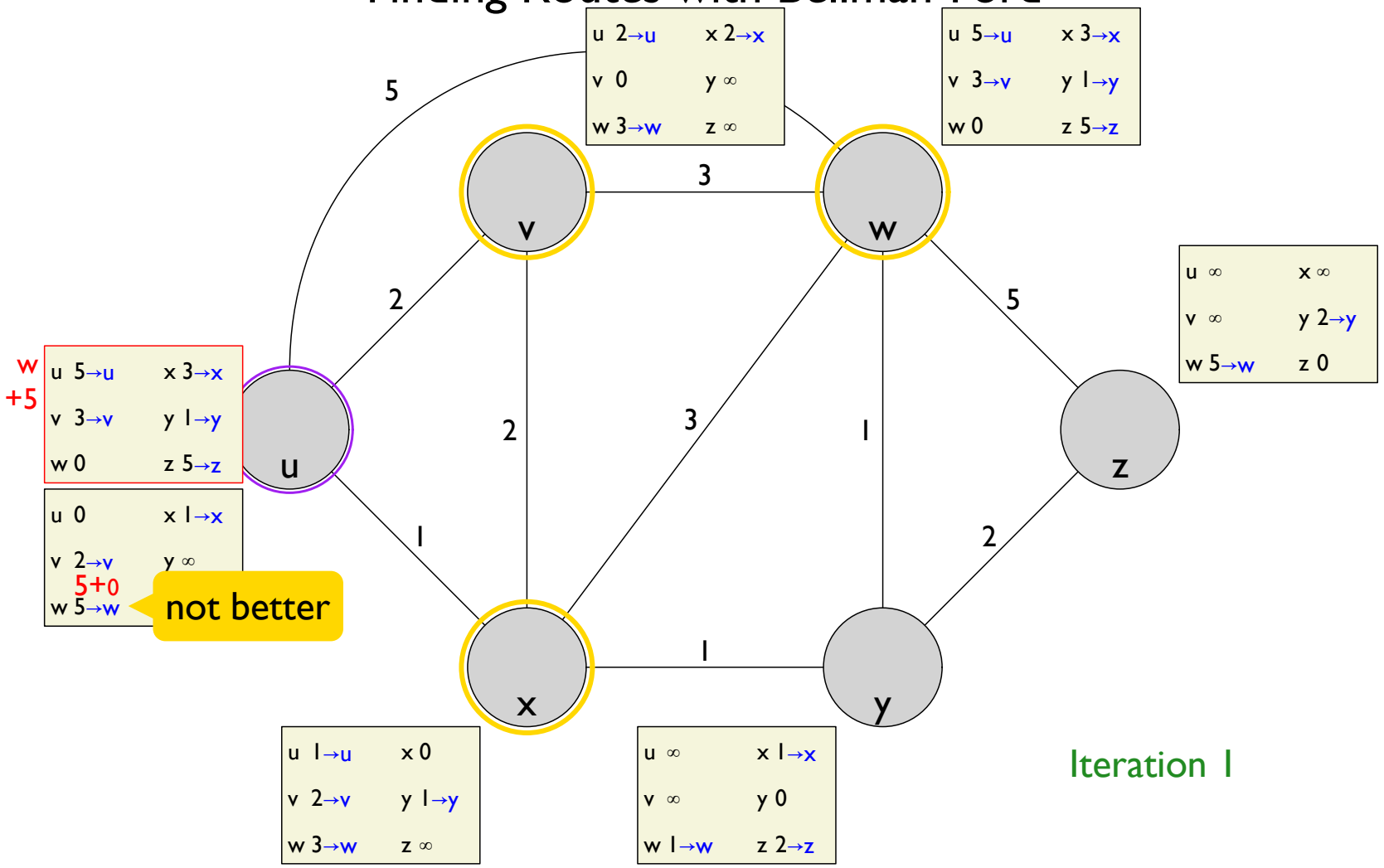
Iteration 1

Finding Routes with Bellman-Ford

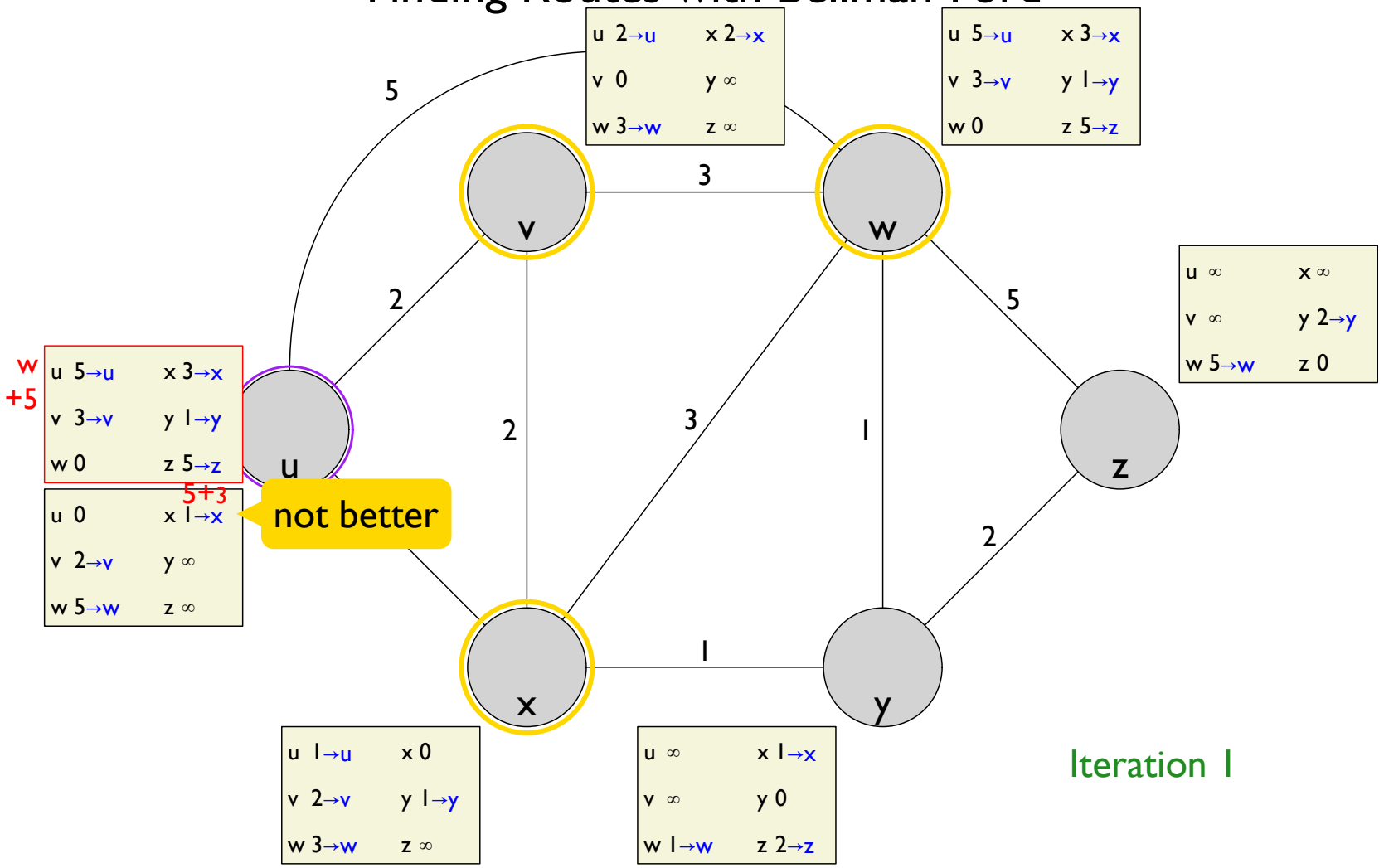


Iteration 1

Finding Routes with Bellman-Ford

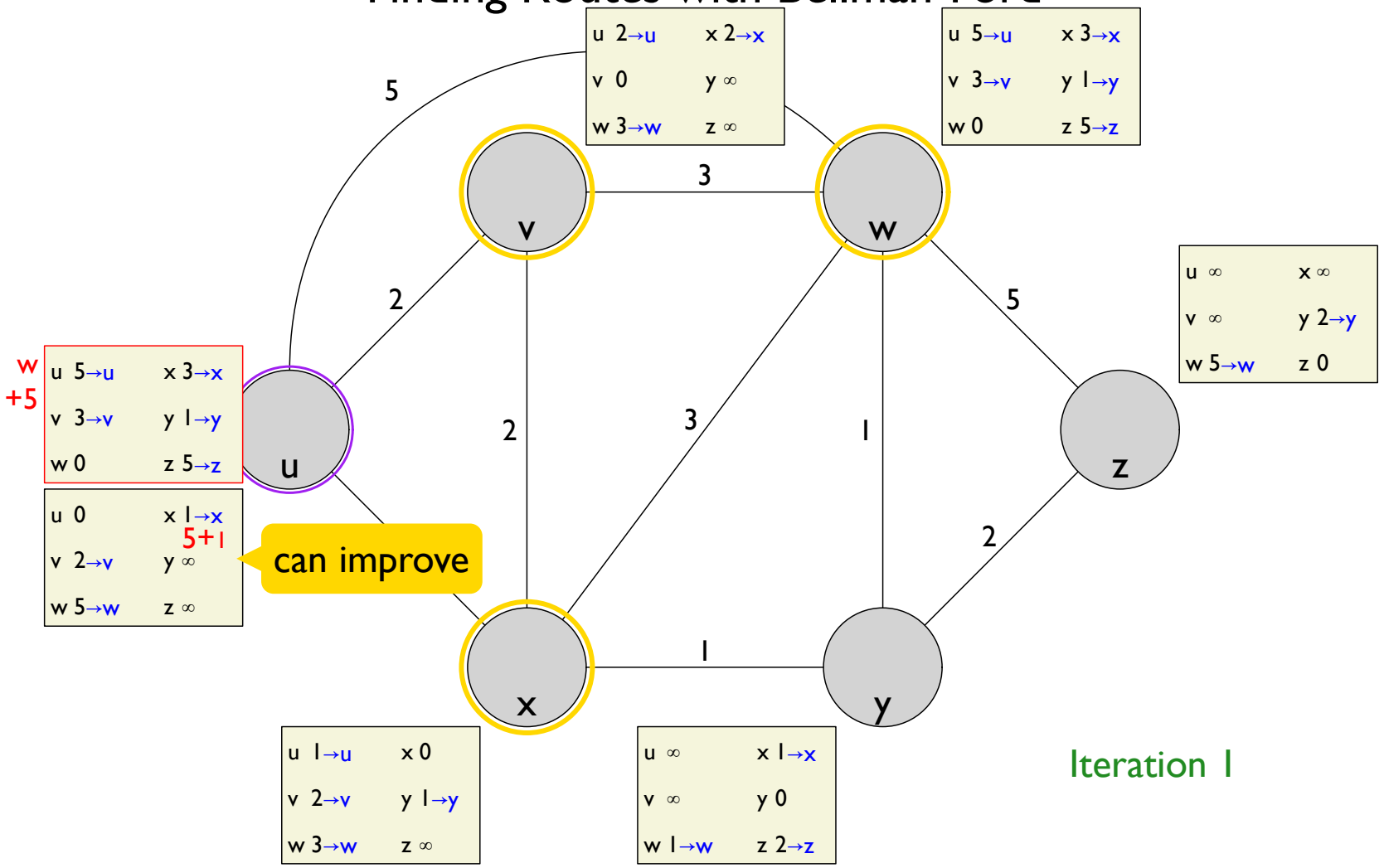


Finding Routes with Bellman-Ford



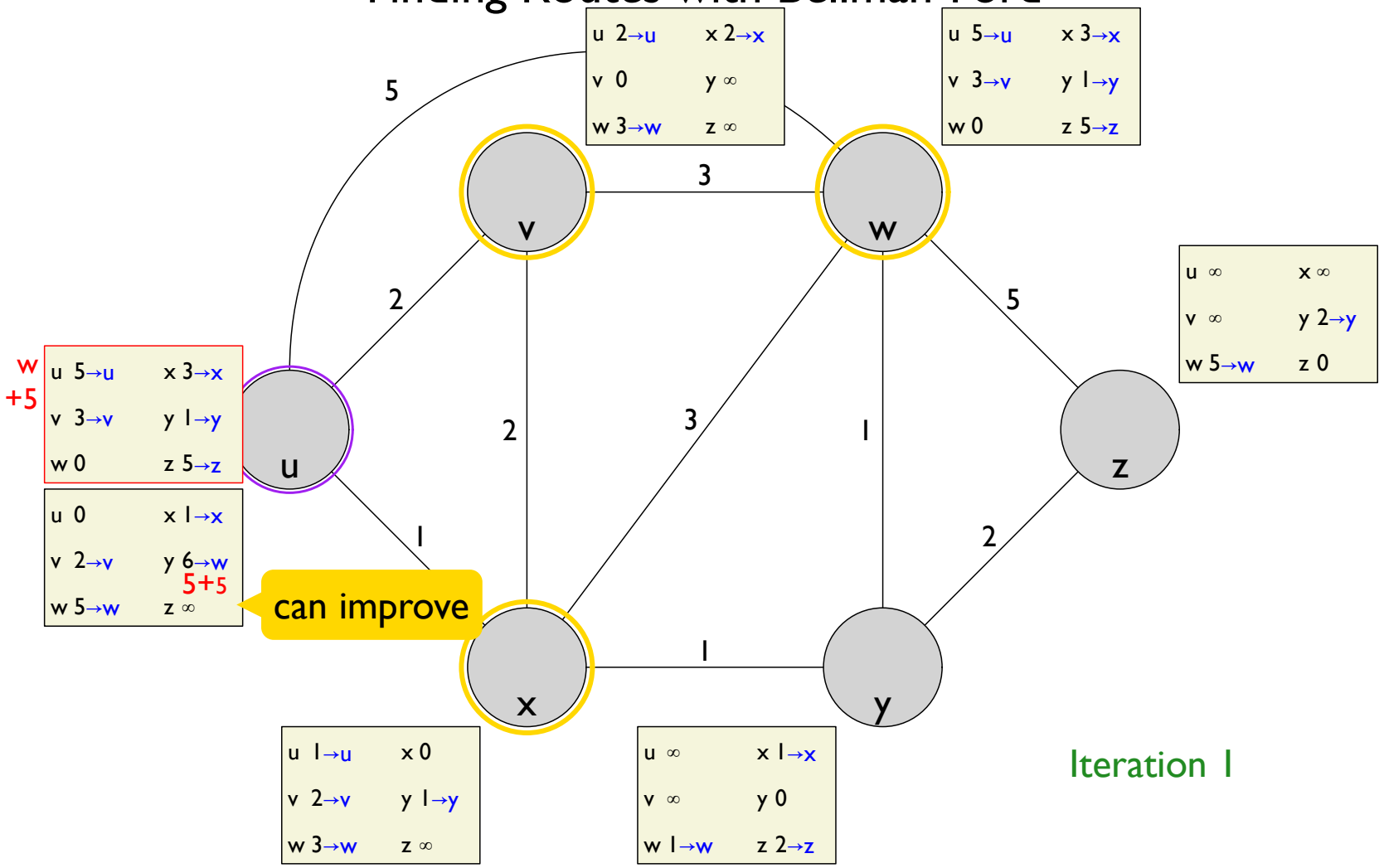
Iteration 1

Finding Routes with Bellman-Ford



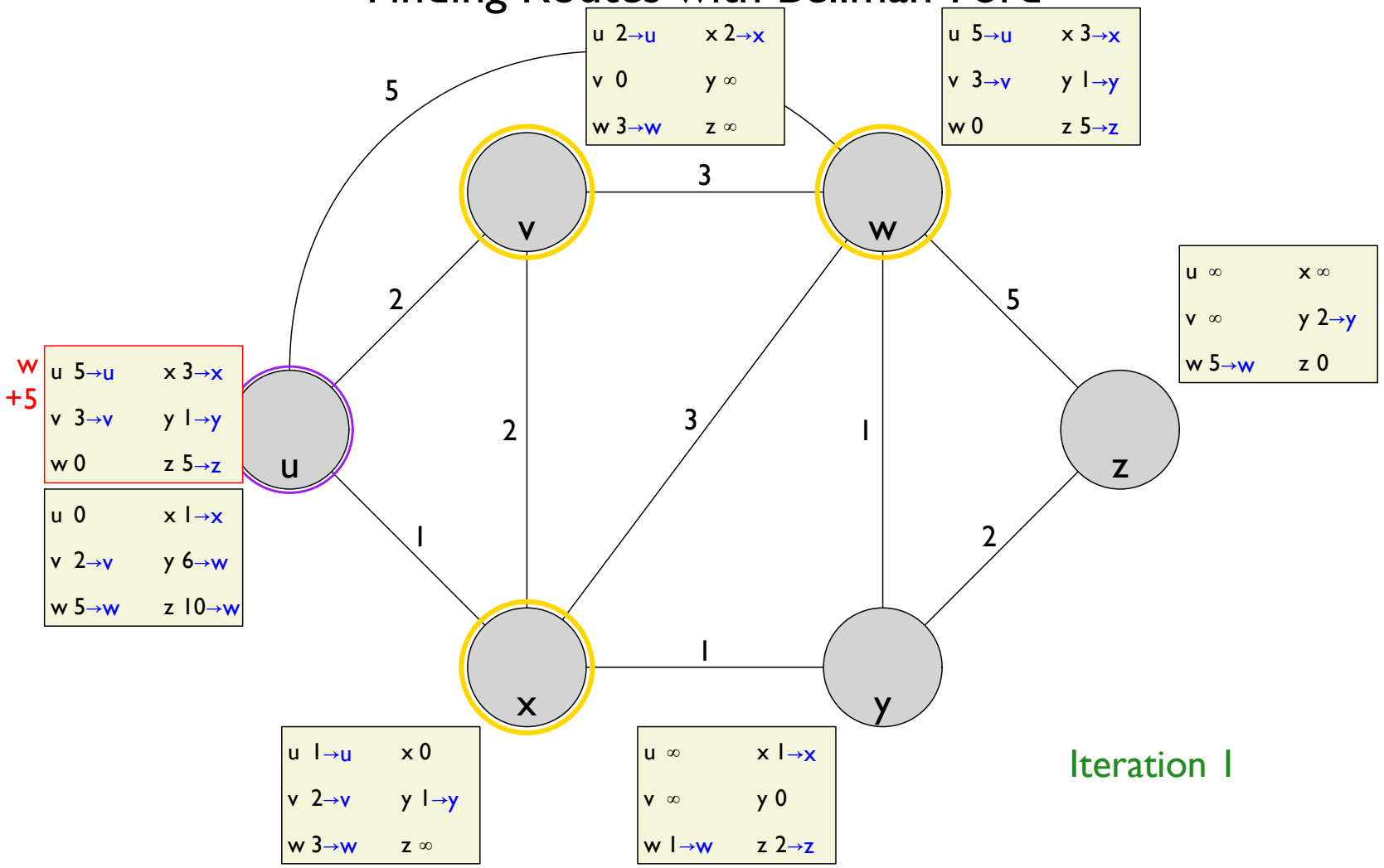
Iteration 1

Finding Routes with Bellman-Ford



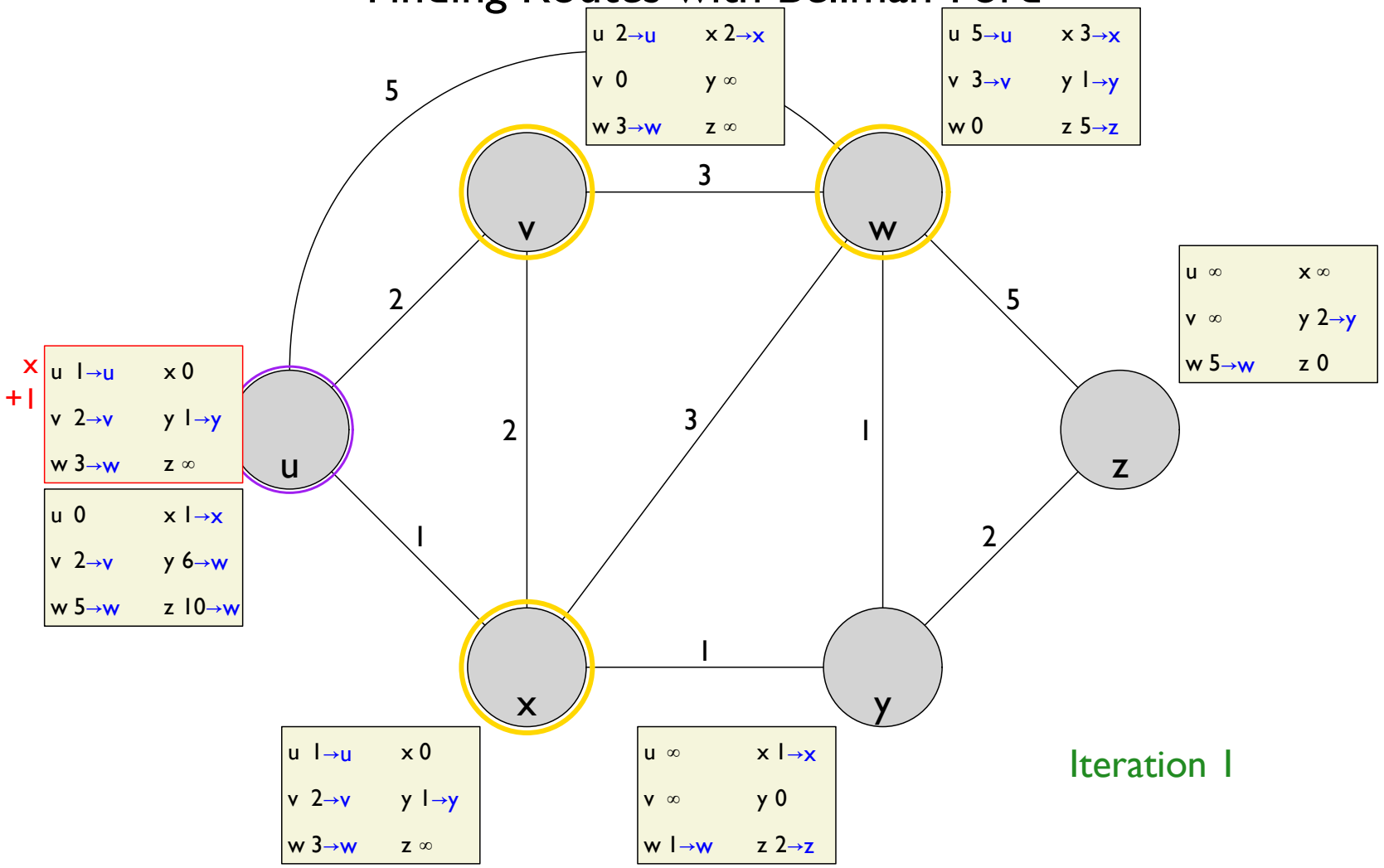
Iteration 1

Finding Routes with Bellman-Ford



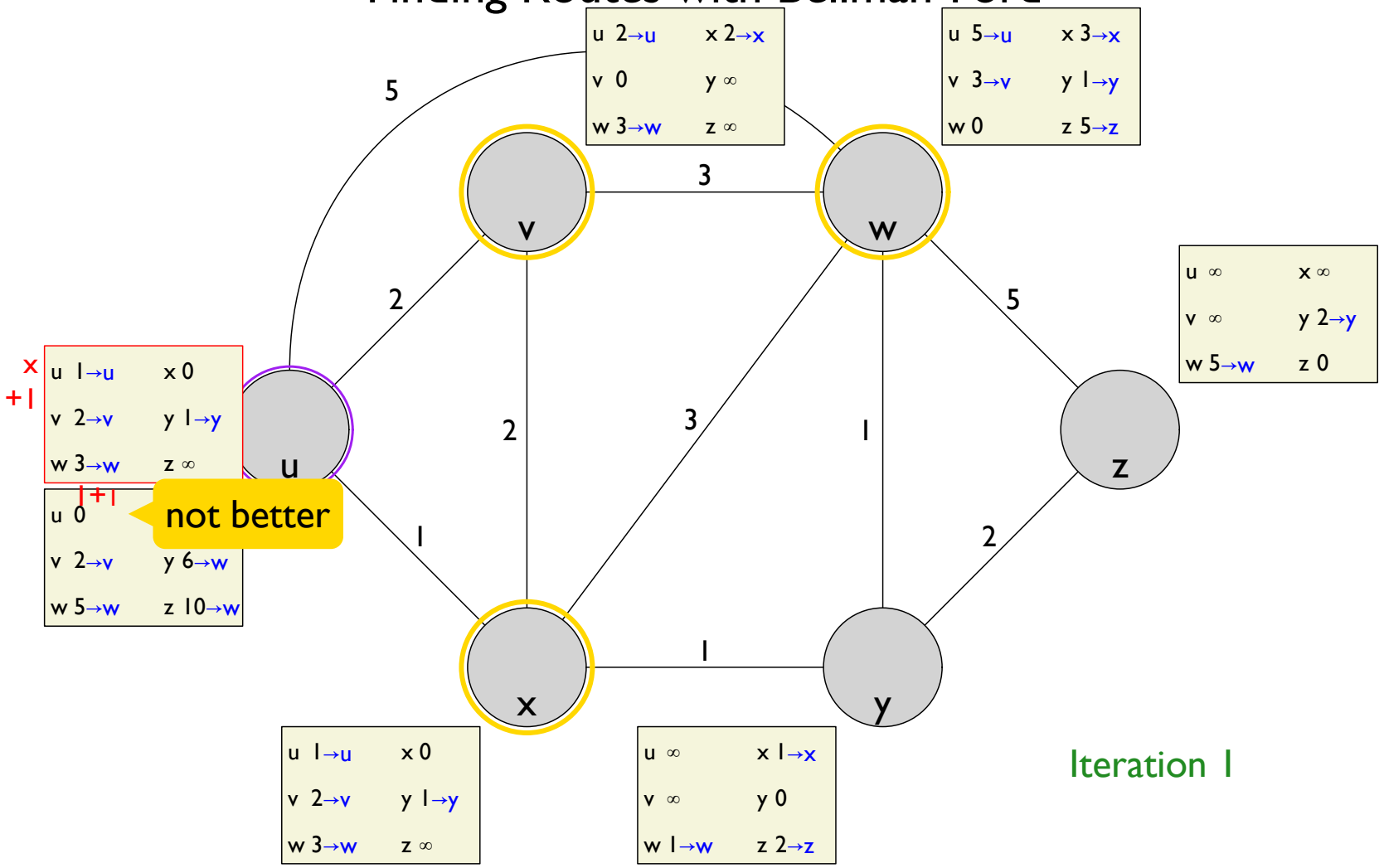
Iteration 1

Finding Routes with Bellman-Ford



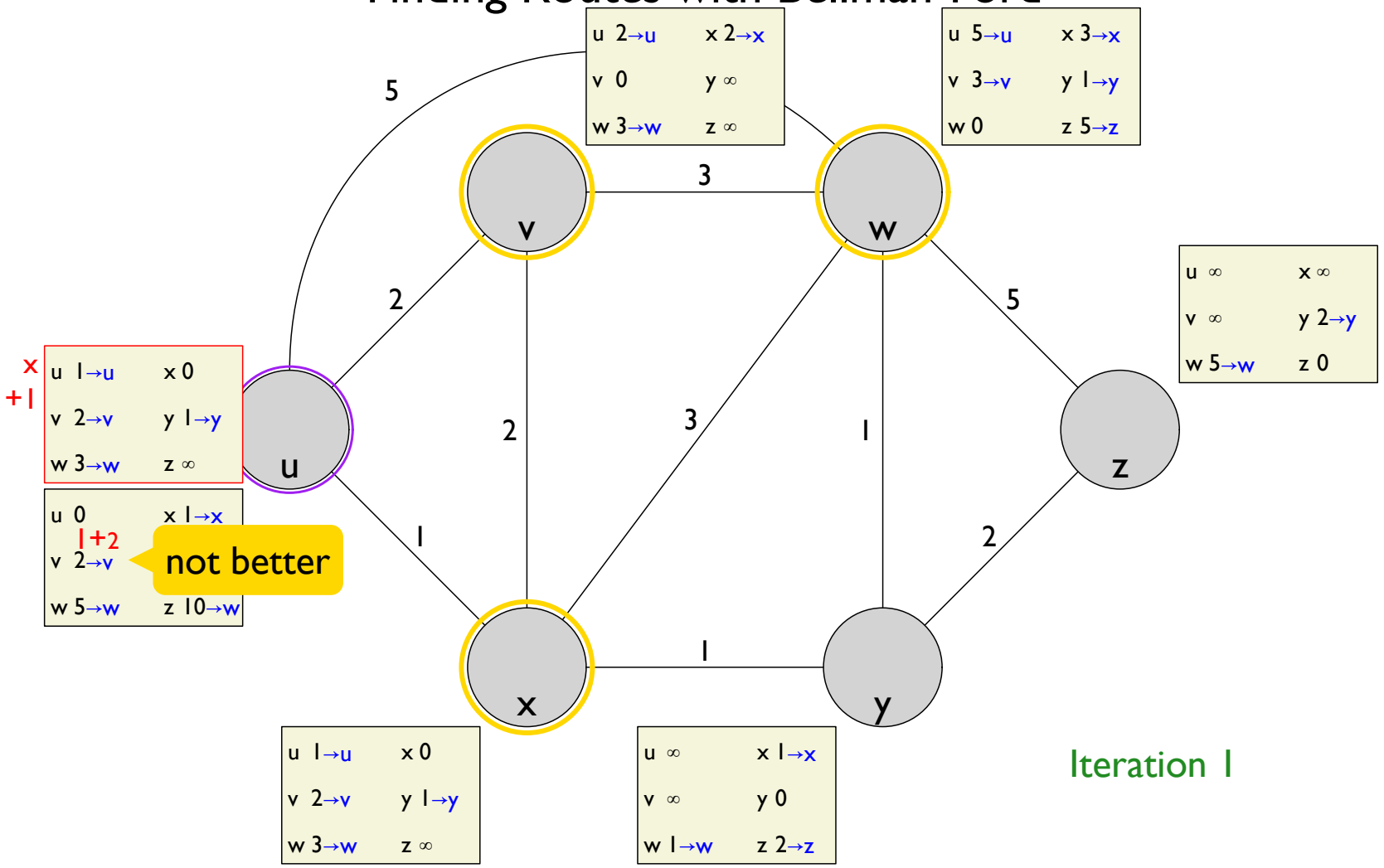
Iteration 1

Finding Routes with Bellman-Ford



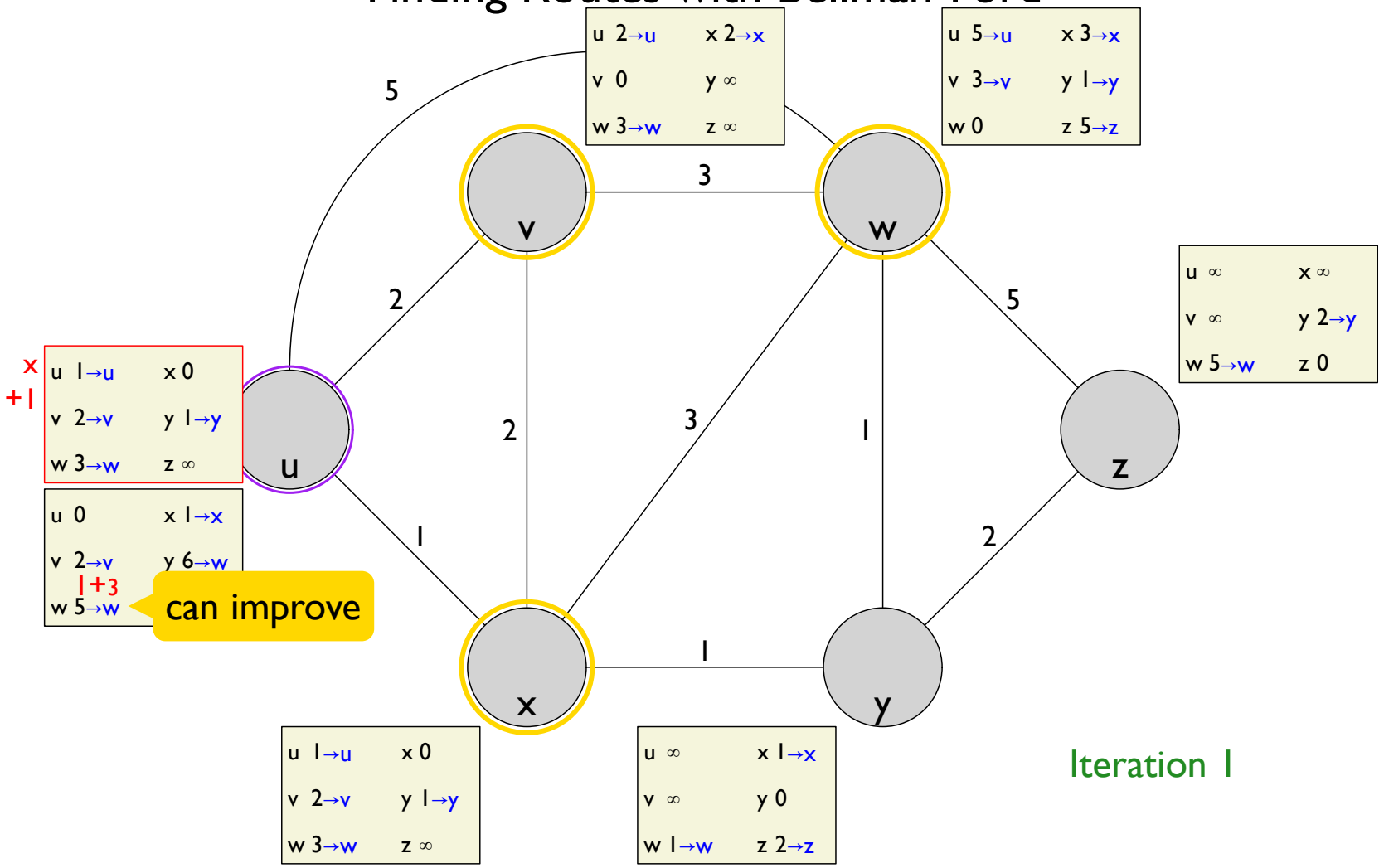
Iteration 1

Finding Routes with Bellman-Ford



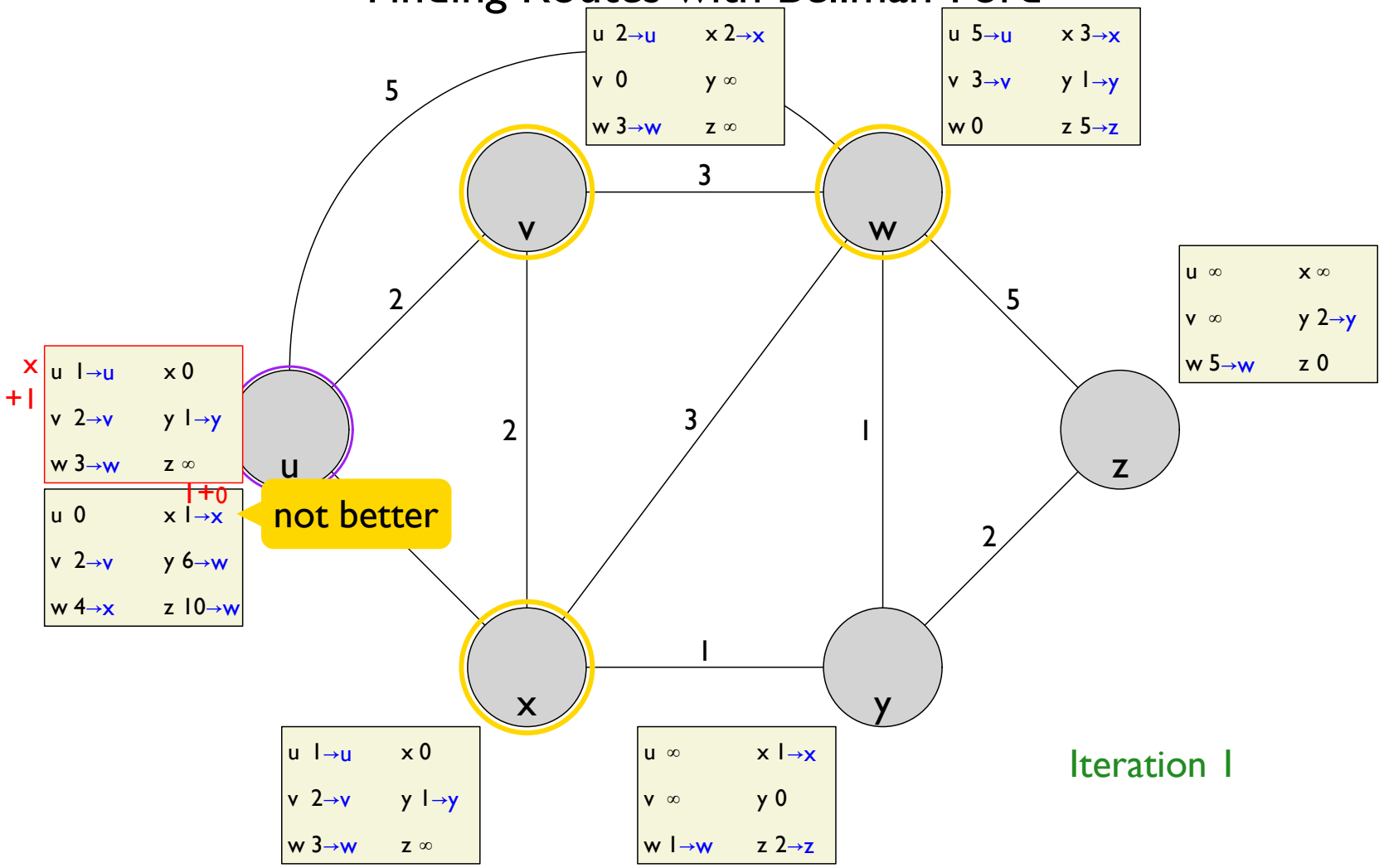
Iteration 1

Finding Routes with Bellman-Ford



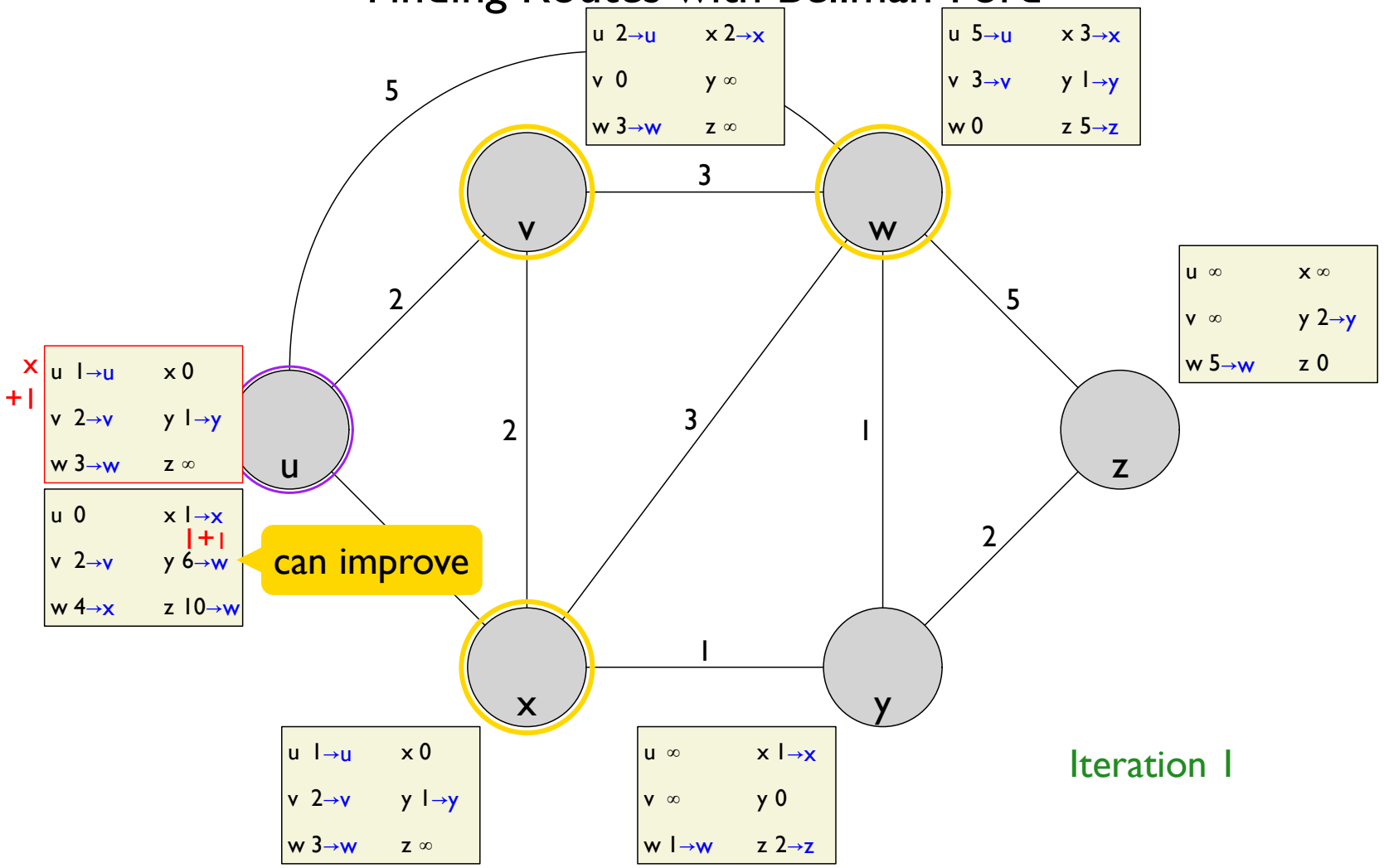
Iteration 1

Finding Routes with Bellman-Ford

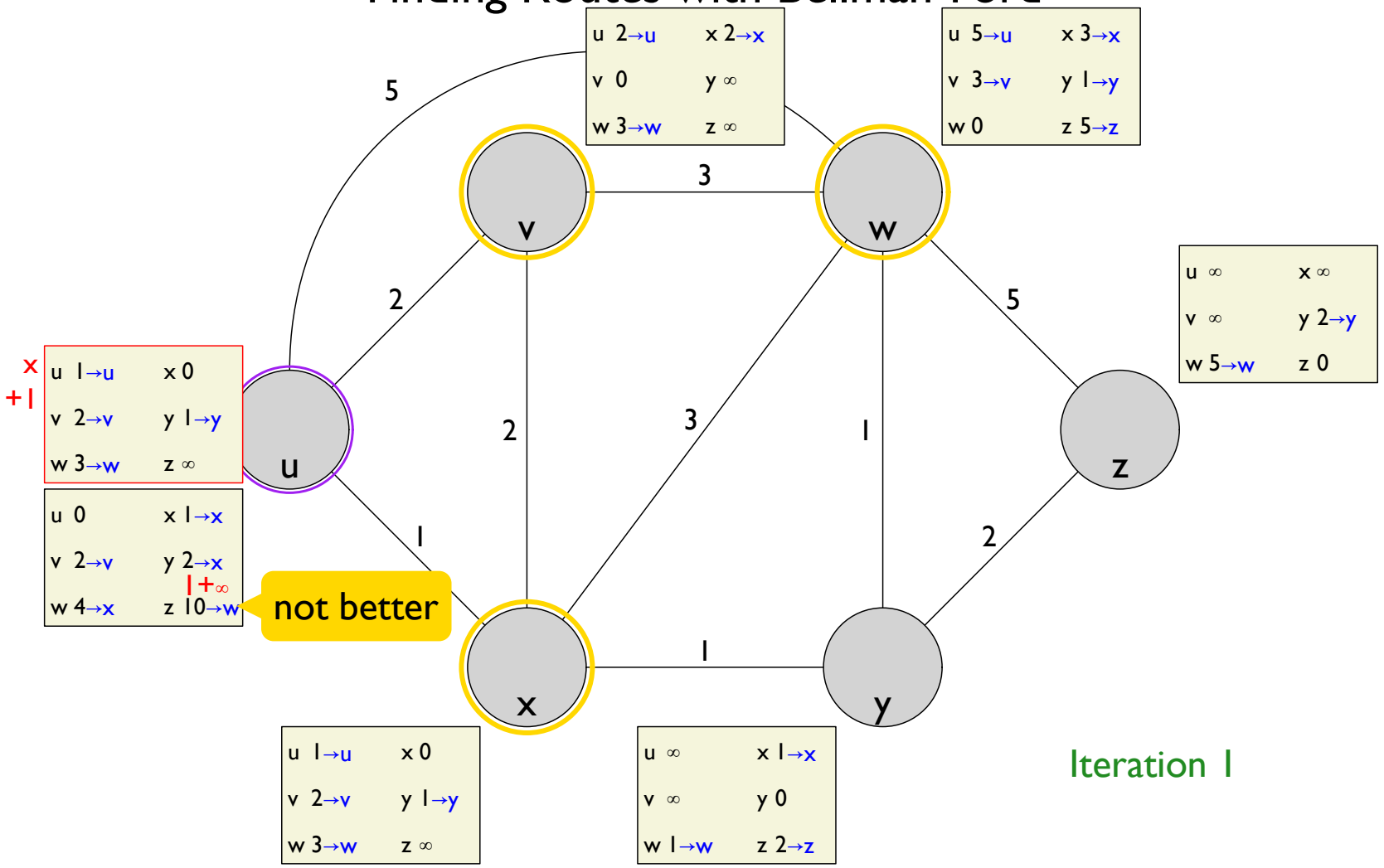


Iteration 1

Finding Routes with Bellman-Ford

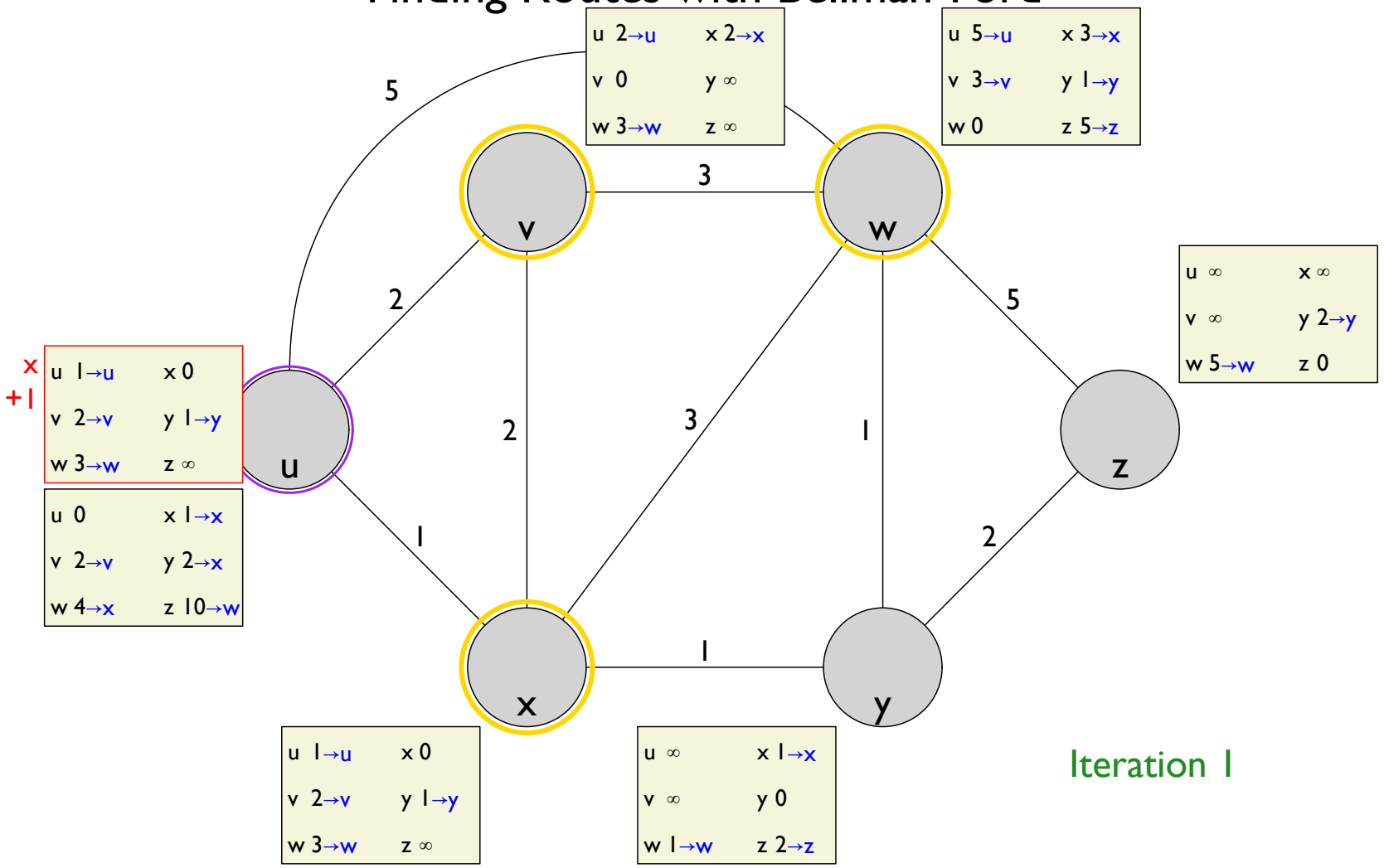


Finding Routes with Bellman-Ford



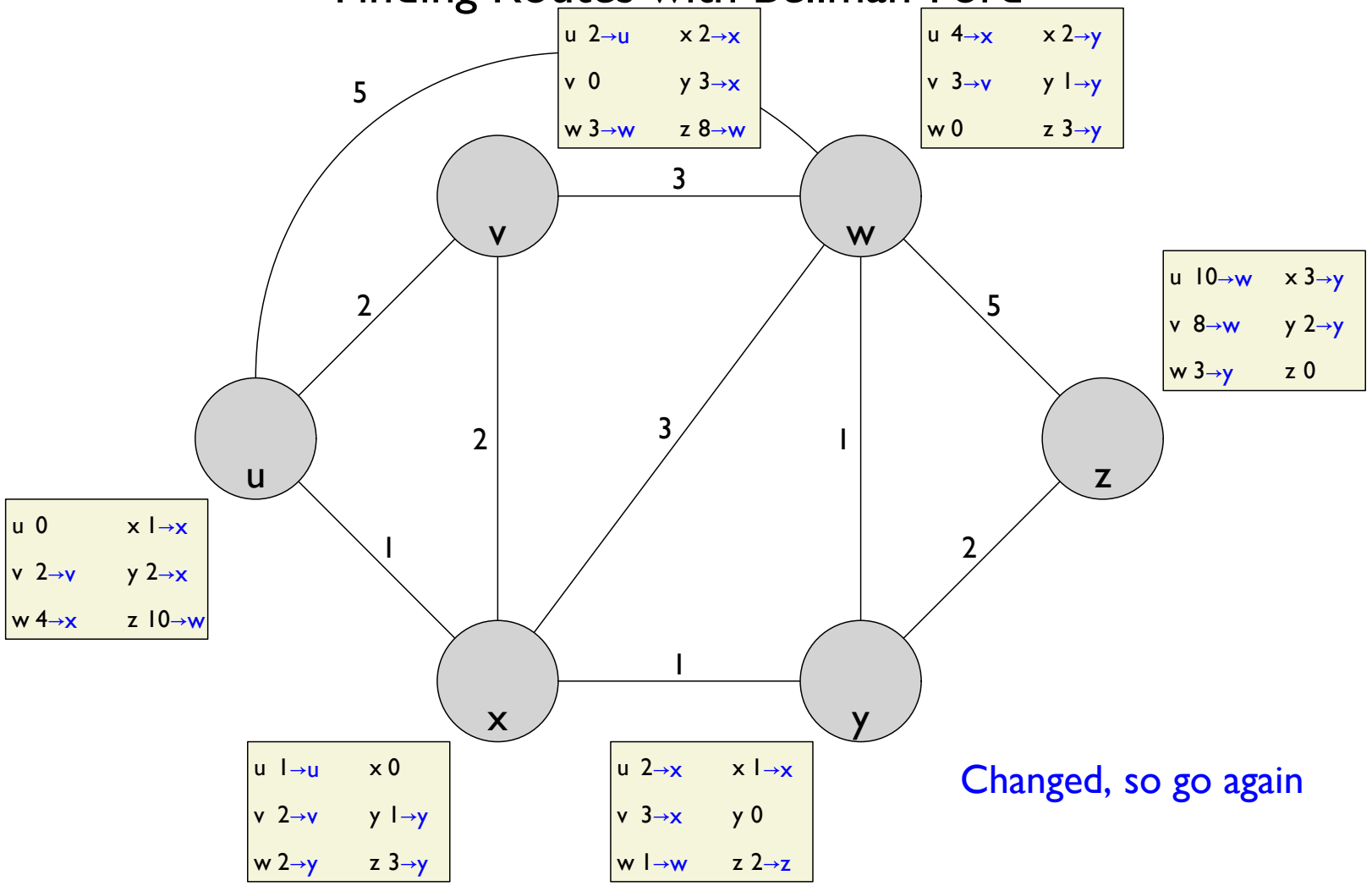
Iteration 1

Finding Routes with Bellman-Ford



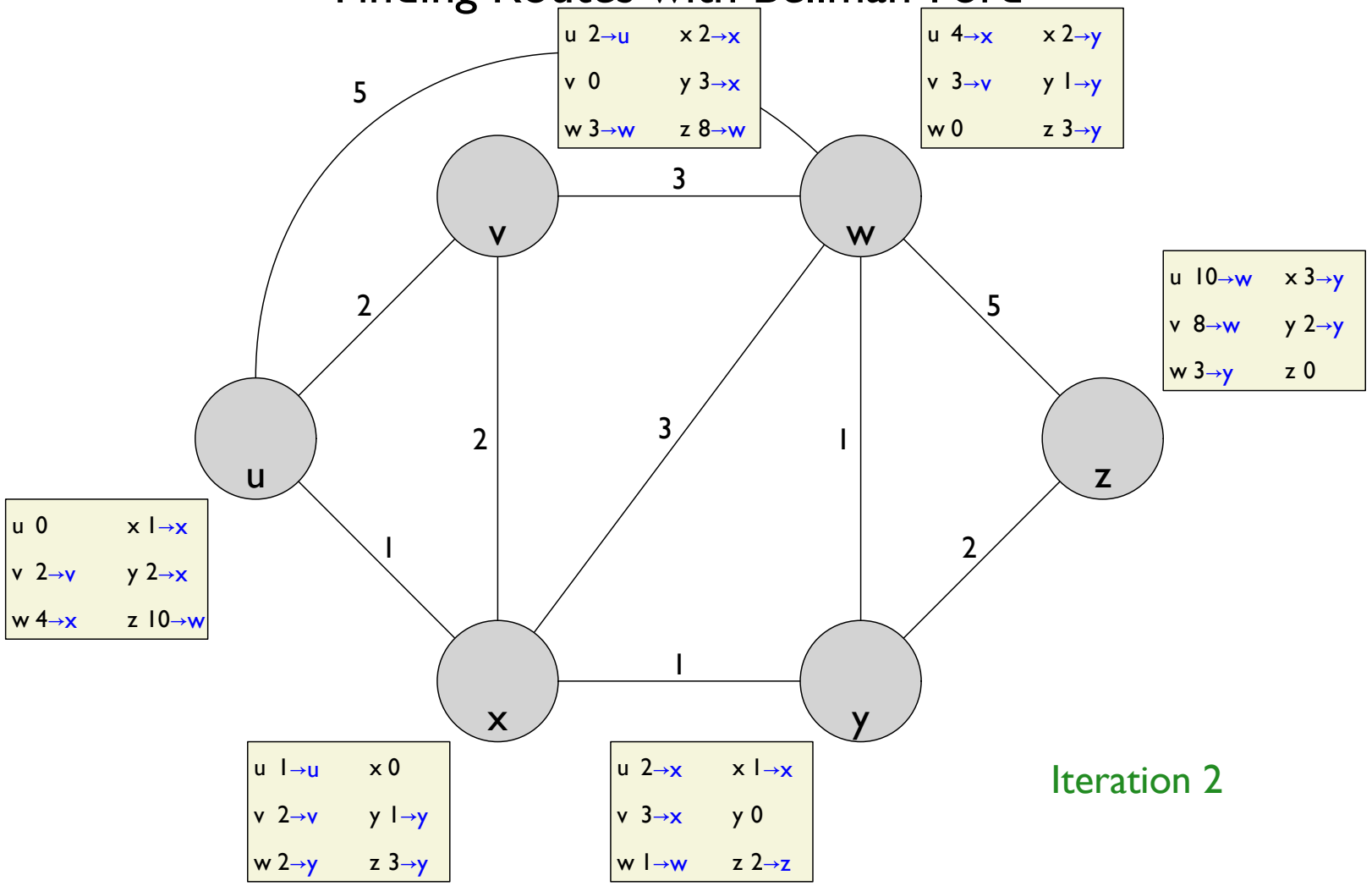
Iteration 1

Finding Routes with Bellman-Ford



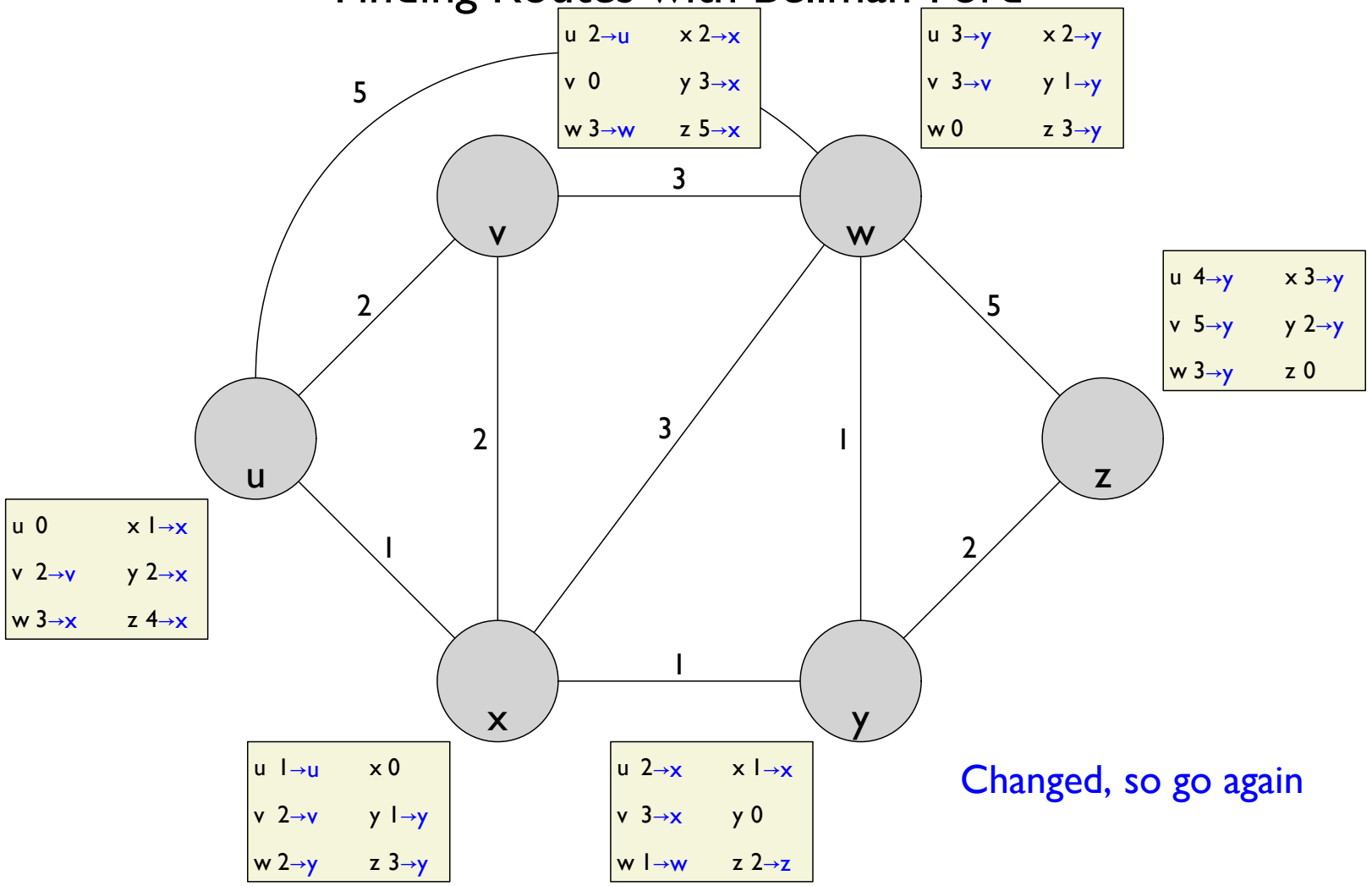
Changed, so go again

Finding Routes with Bellman-Ford



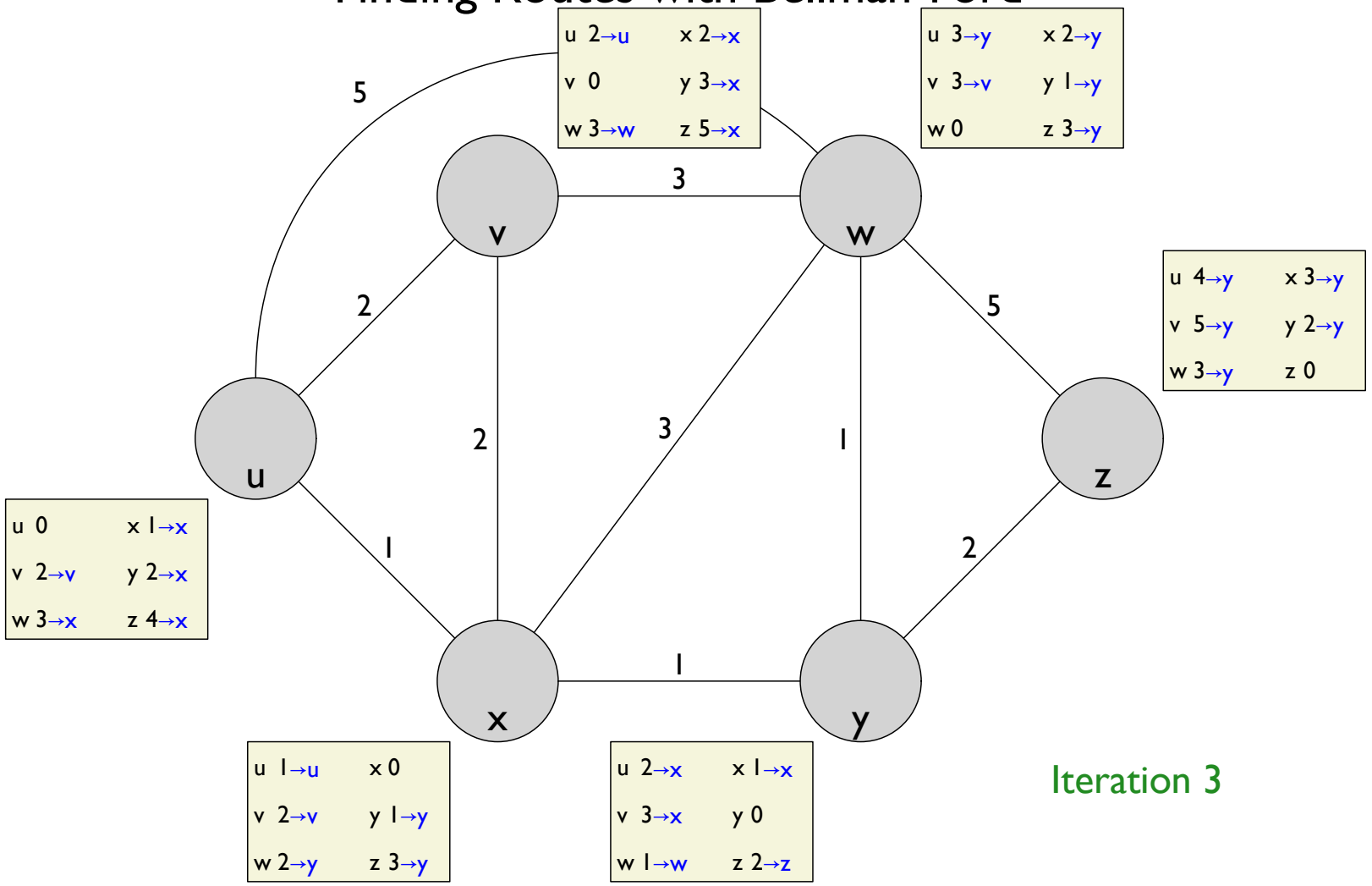
Iteration 2

Finding Routes with Bellman-Ford



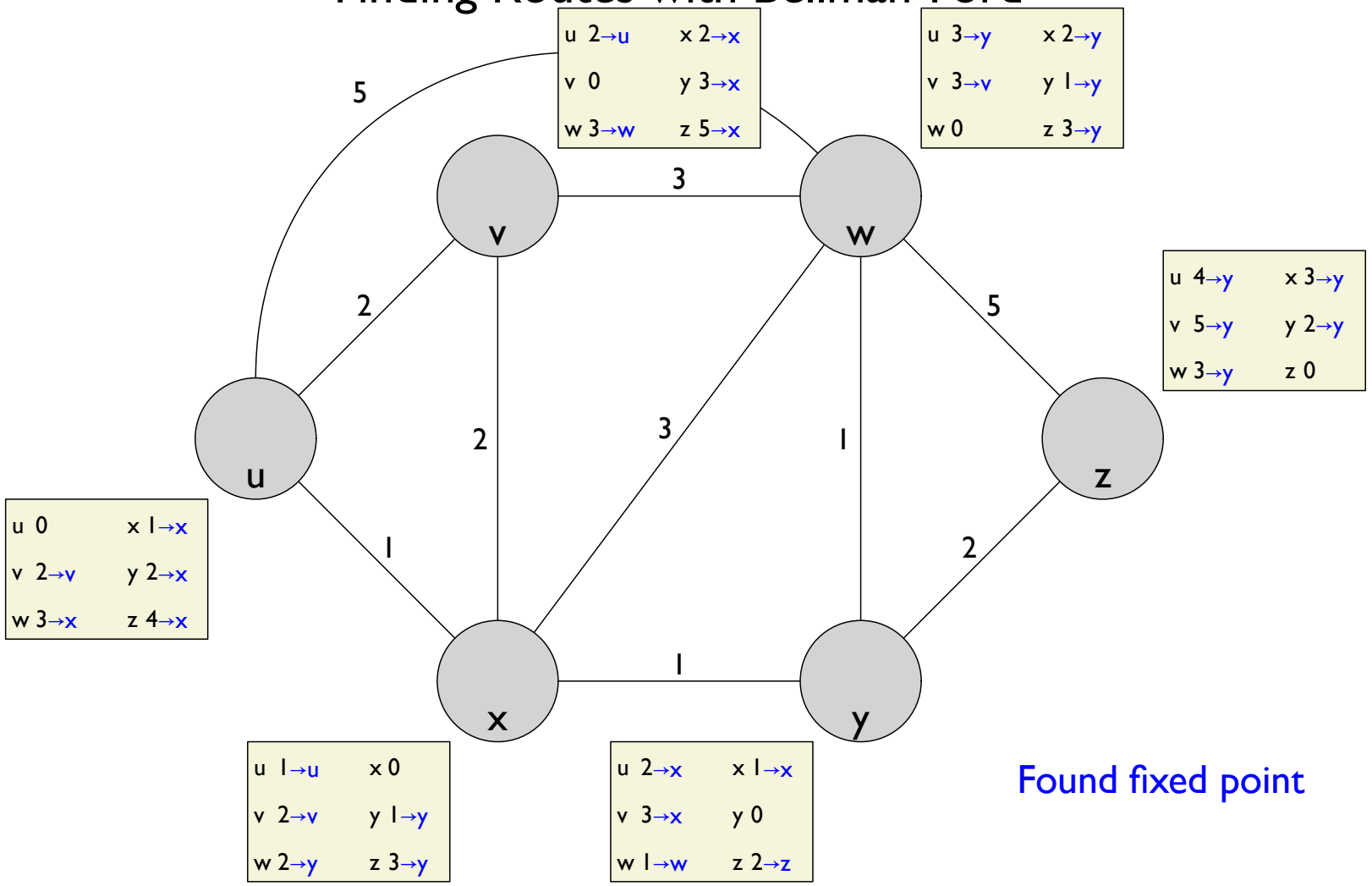
Changed, so go again

Finding Routes with Bellman-Ford



Iteration 3

Finding Routes with Bellman-Ford



Found fixed point

Finding Routes with Bellman-Ford

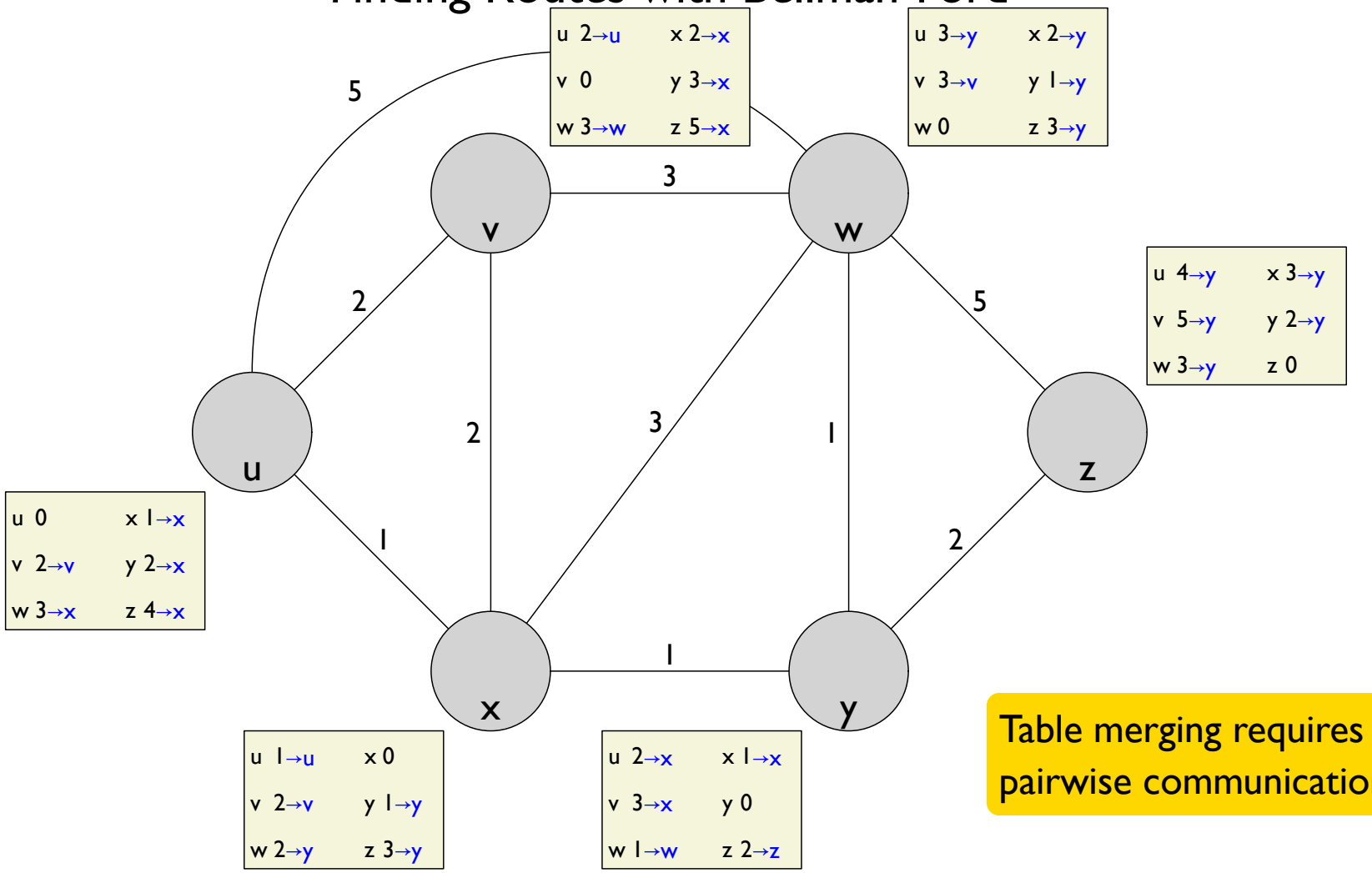
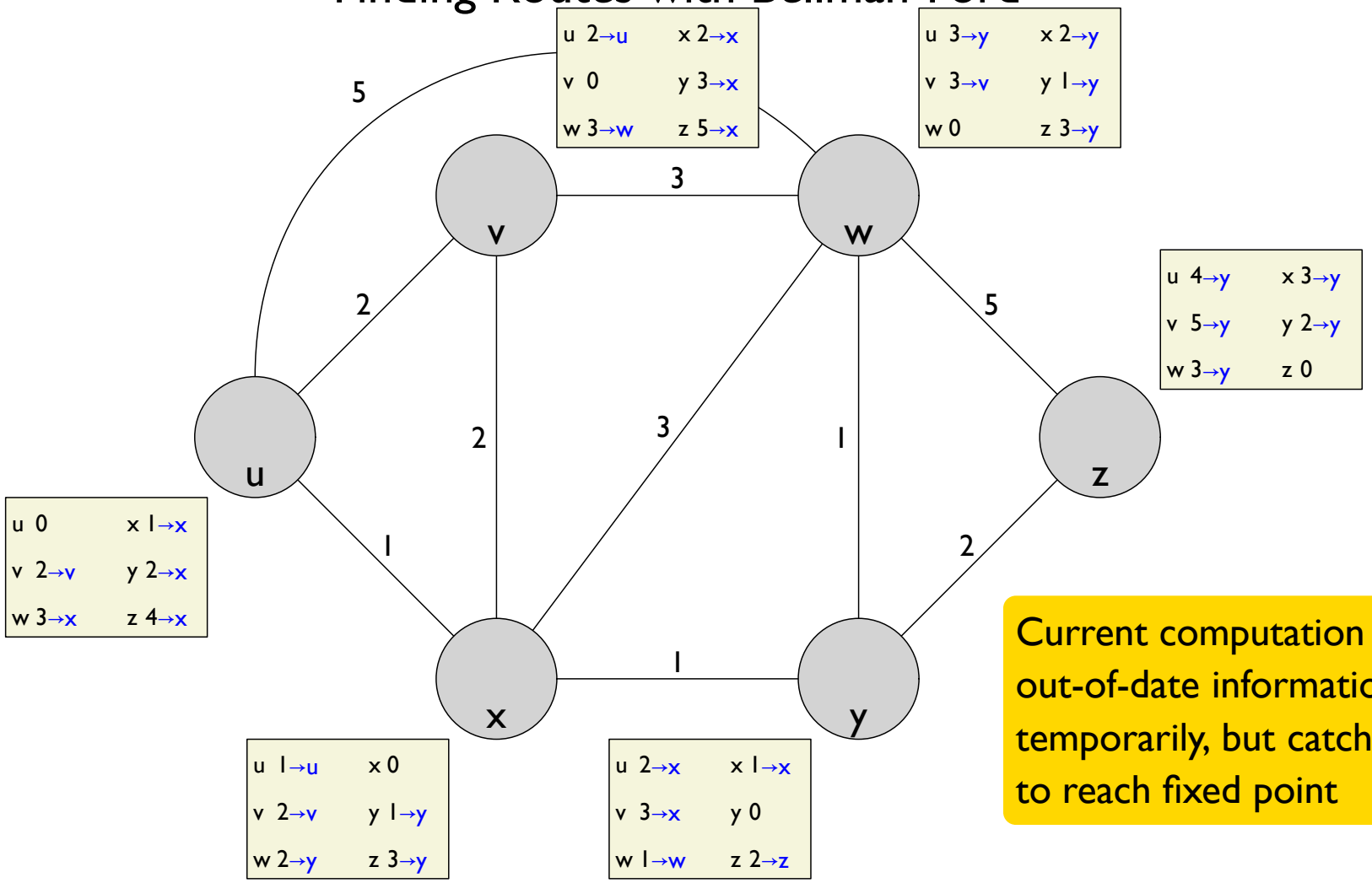


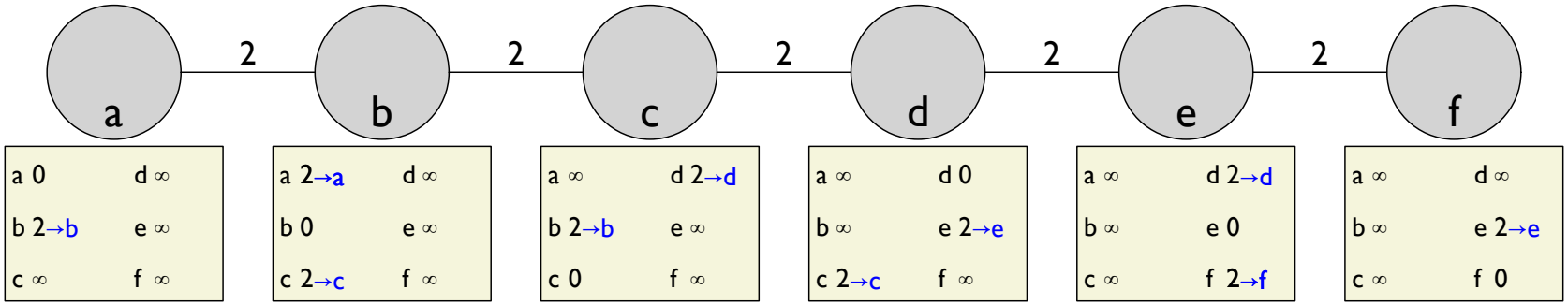
Table merging requires only pairwise communication

Finding Routes with Bellman-Ford



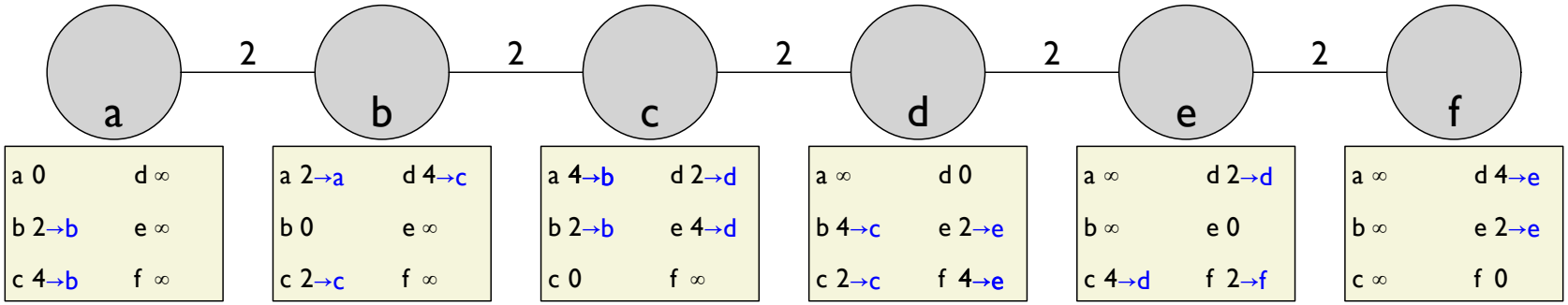
Current computation risks out-of-date information temporarily, but catches up to reach fixed point

Worst-Case Iterations for Bellman-Ford



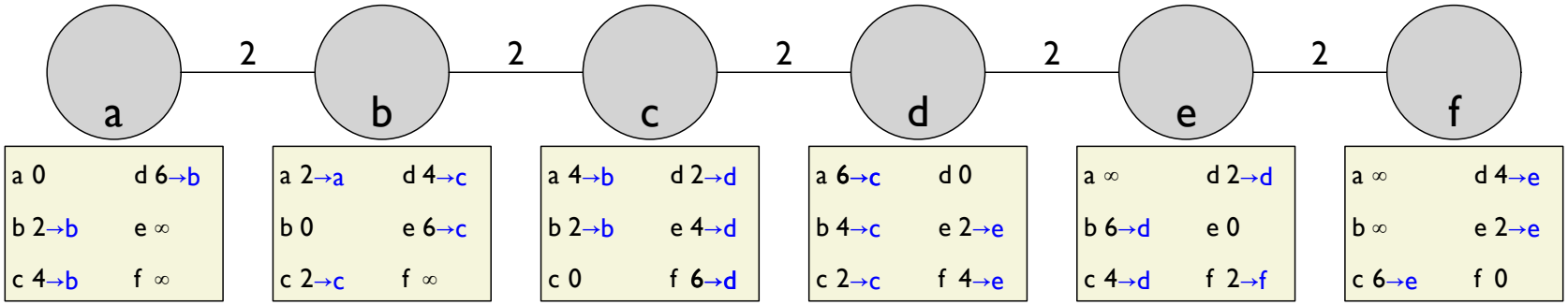
Iteration 1

Worst-Case Iterations for Bellman-Ford



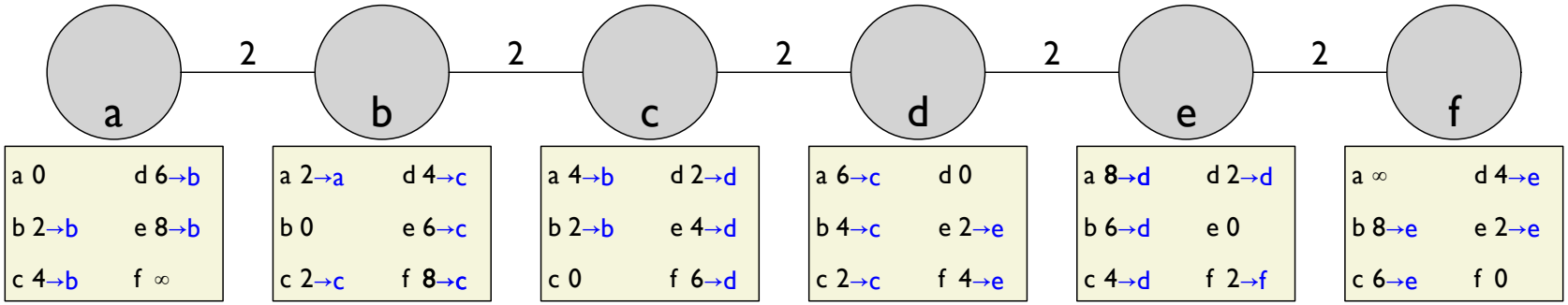
Changed, so go again

Worst-Case Iterations for Bellman-Ford



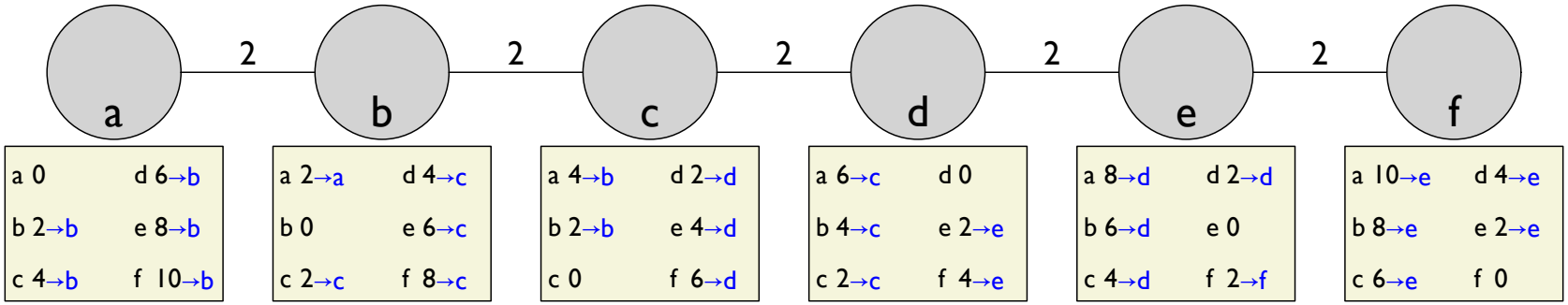
Changed, so go again

Worst-Case Iterations for Bellman-Ford



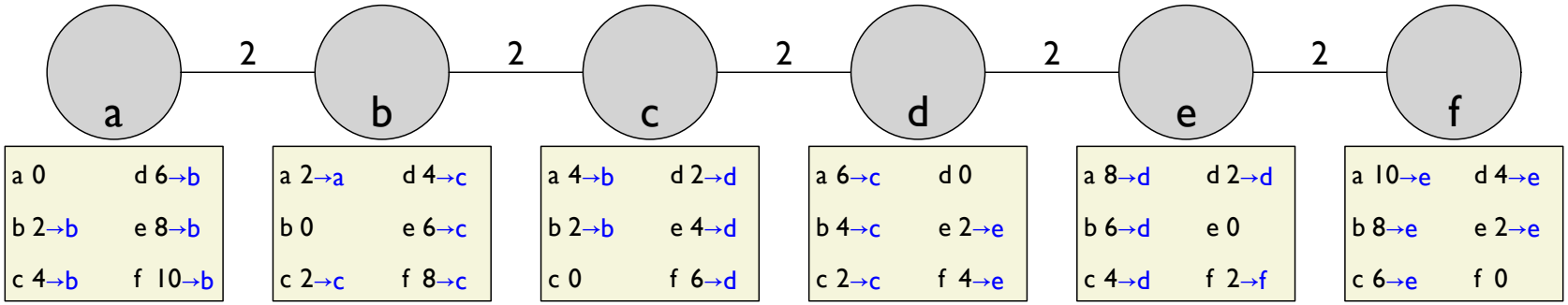
Changed, so go again

Worst-Case Iterations for Bellman-Ford



Changed, so go again

Worst-Case Iterations for Bellman-Ford



Found fixed point

Time Complexity

Dijkstra's:

for each of N nodes

visit each of $N-1$ other nodes

checking each of up to M adjacent nodes

where picking each other node takes $O(\log N)$ time

Time Complexity

Dijkstra's:

for each of N nodes

visit each of N-1 other nodes

checking each of up to M adjacent nodes

where picking each other node takes O(log N) time ⇒ $O(M N^2 \log N)$

Bellman-Ford:

for each of N nodes

check tables of up to M adjacent nodes

where each table has N entries

and iterate up to N times

Time Complexity

Dijkstra's:

for each of N nodes

visit each of N-1 other nodes

checking each of up to M adjacent nodes

where picking each other node takes O(log N) time ⇒ $O(M N^2 \log N)$

Bellman-Ford:

for each of N nodes

check tables of up to M adjacent nodes

where each table has N entries

and iterate up to N times ⇒ $O(M N^3)$

Time Complexity

Dijkstra's:

for each of N nodes

visit each of N-1 other nodes

checking each of up to M adjacent nodes

where picking each other node takes O(log N) time ⇒ $O(M N^2 \log N)$

Requires global communication

Bellman-Ford:

for each of N nodes

check tables of up to M adjacent nodes

where each table has N entries

and iterate up to N times ⇒ $O(M N^3)$

Time Complexity

Dijkstra's:

for each of N nodes

visit each of N-1 other nodes

checking each of up to M adjacent nodes

where picking each other node takes O(log N) time ⇒ $O(M N^2 \log N)$

Requires global communication

Bellman-Ford:

for each of N nodes

check tables of up to M adjacent nodes

where each table has N entries

and iterate up to N times ⇒ $O(M N^3)$

Can be local and in parallel

Time Complexity

Dijkstra's:

for each of N nodes

visit each of N-1 other nodes

checking each of up to M adjacent nodes

where picking each other node takes O(log N) time ⇒ $O(M N^2 \log N)$

Requires global communication

Bellman-Ford:

for each of N nodes

check tables of up to M adjacent nodes

where each table has N entries

and iterate up to N times ⇒ $O(M N^3)$

Can be local and in parallel

each node pays $\frac{1}{N}$ of cost

Time Complexity

Dijkstra's:

for each of N nodes

visit each of N-1 other nodes

checking each of up to M adjacent nodes

where picking each other node takes O(log N) time ⇒ $O(M N^2 \log N)$

Requires global communication

Bellman-Ford:

for each of N nodes

check tables of up to M adjacent nodes

where each table has N entries

and iterate up to N times ⇒ $O(M N^3)$

Can be local and in parallel

each node pays $\frac{1}{N}$ of cost

Diff can be much smaller than N

Applying Graph Algorithms in Networks

Dijkstra's = **link state (LS)**

Typical for local networks, single responsibility

Prominent example: Open Shortest Path First (OSPF)

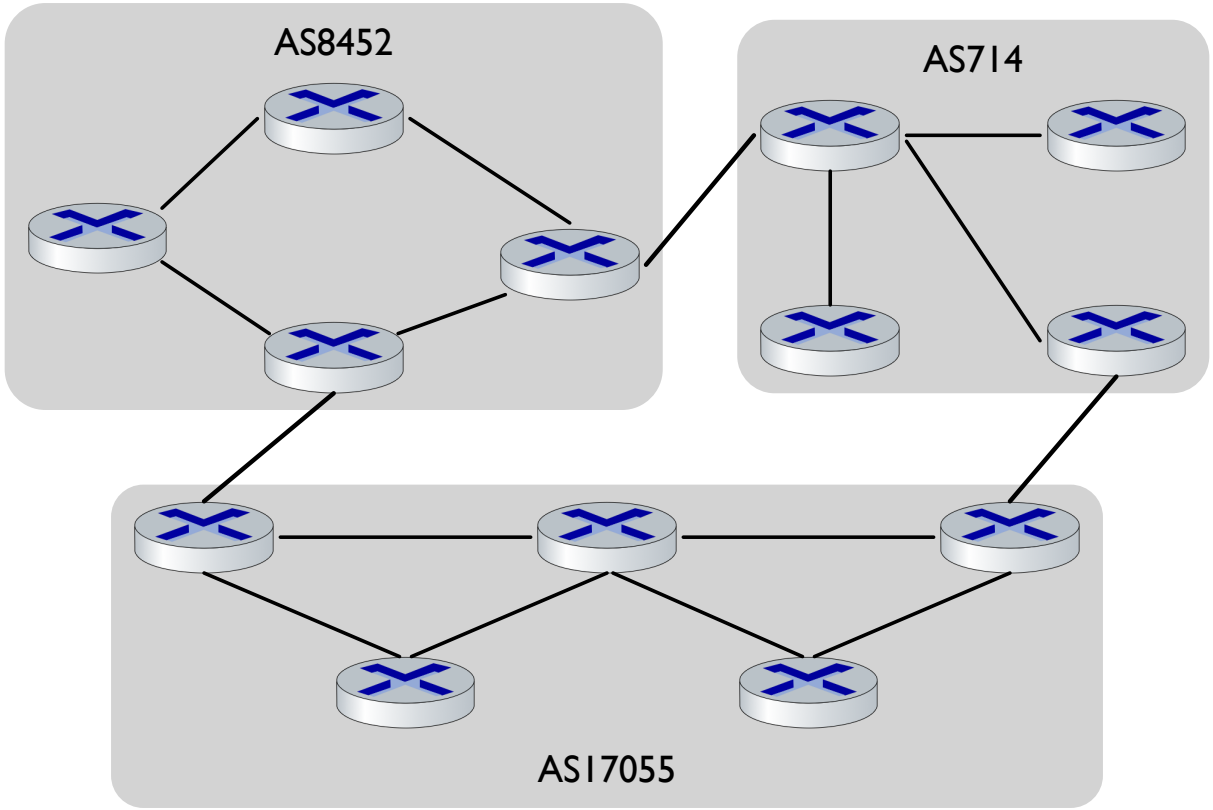
Bellman-Ford = **distance vector (DV)**

Typical for overall network, distributed responsibility

Prominent example: Border Gateway Protocol (BGP)

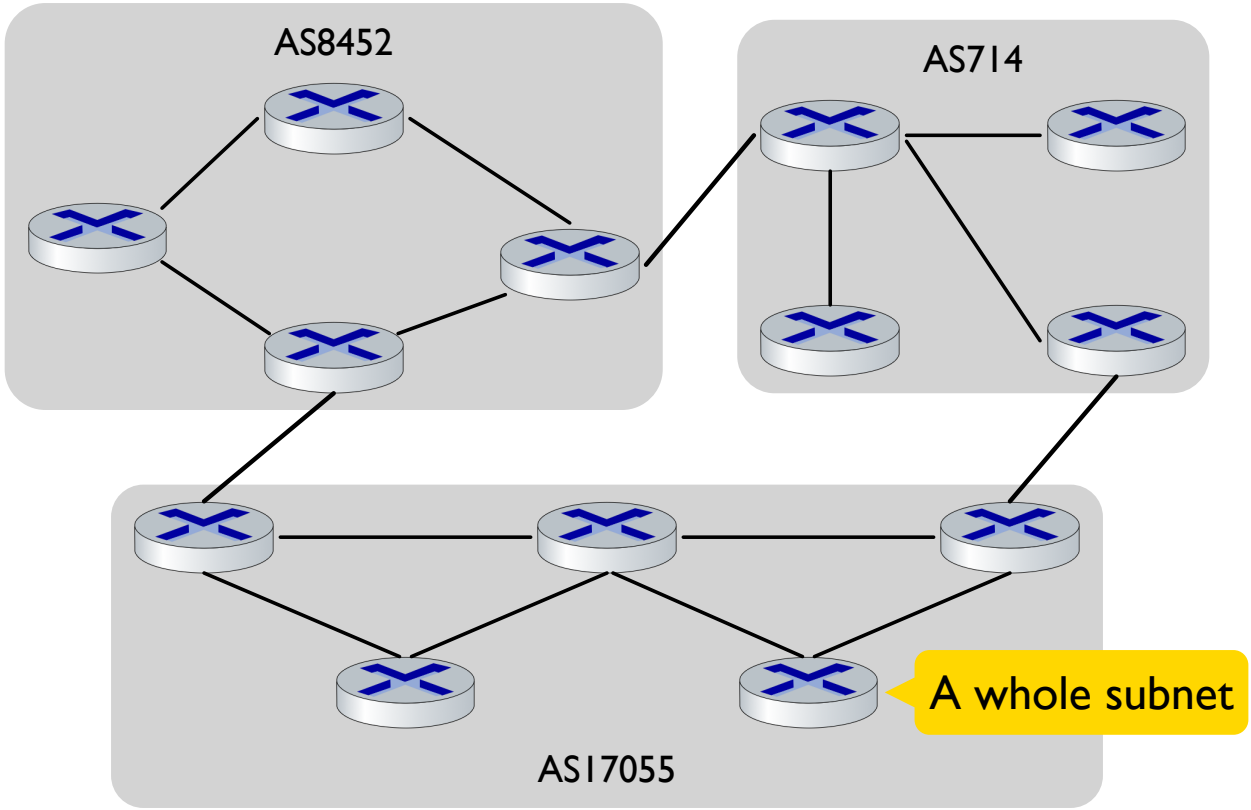
Applying Graph Algorithms in the Internet

A network of networks



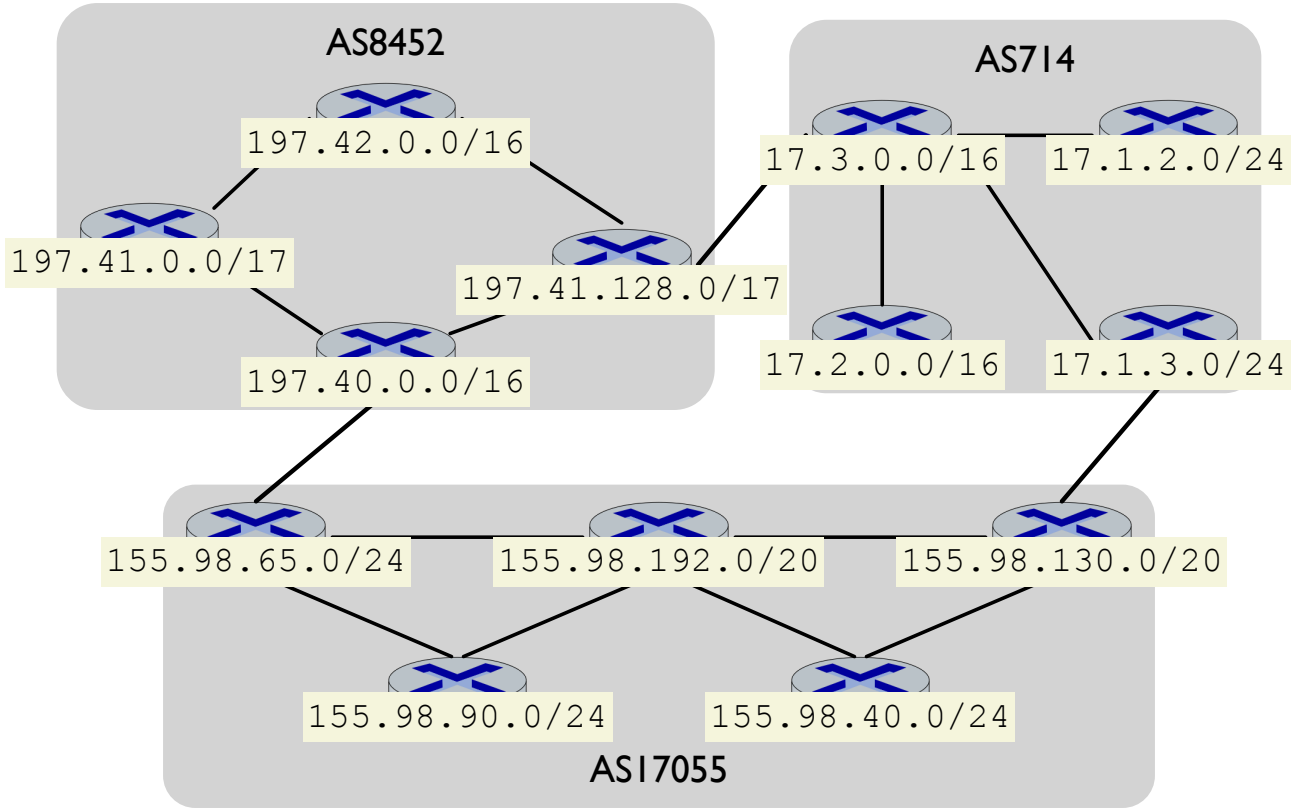
Applying Graph Algorithms in the Internet

A network of networks



Applying Graph Algorithms in the Internet

A network of networks

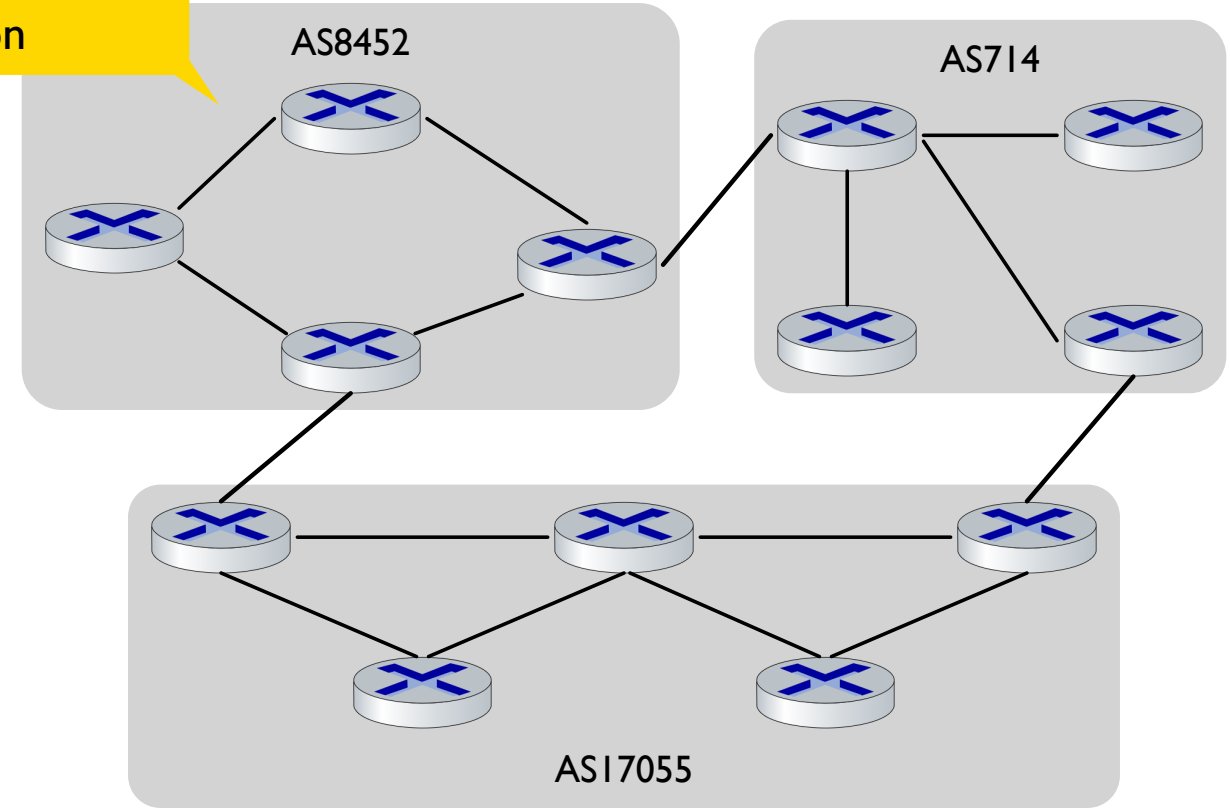


Applying Graph Algorithms in the Internet

A network of networks

Autonomous System (AS)

Managed as a whole by some organization



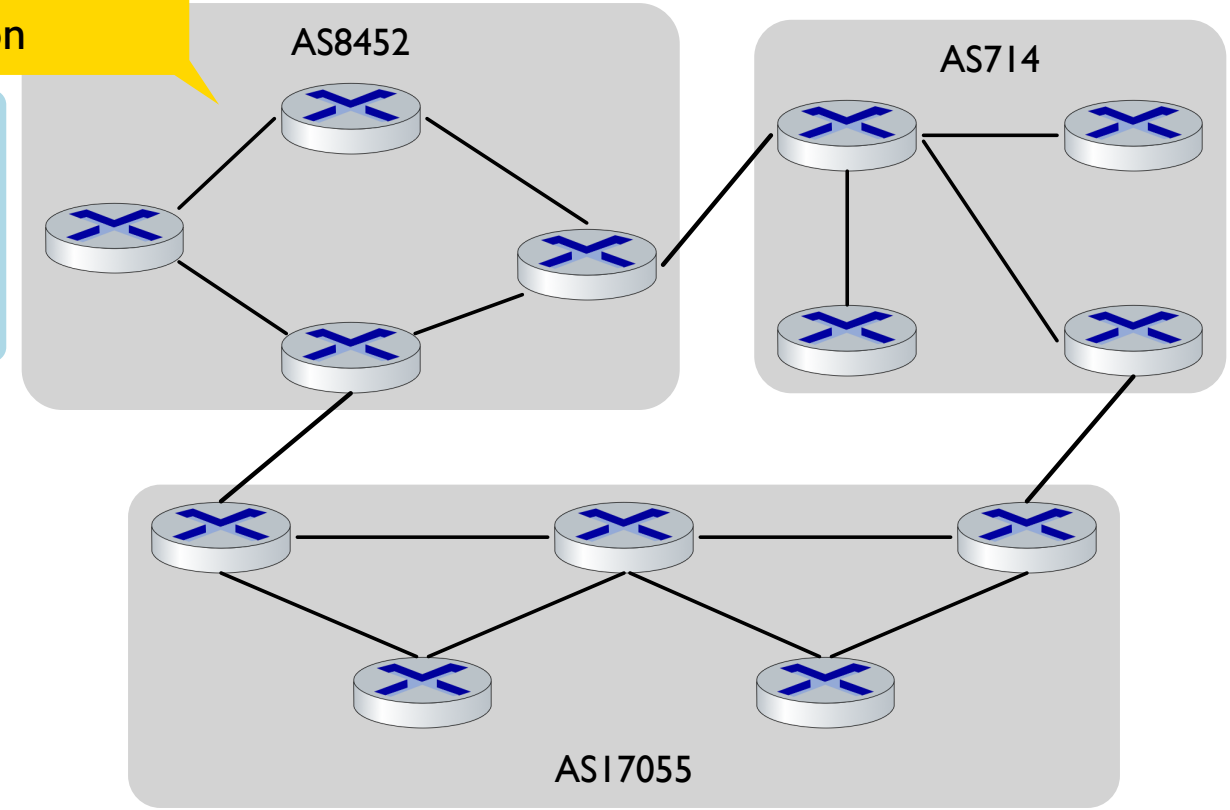
Applying Graph Algorithms in the Internet

A network of networks

Autonomous System (AS)

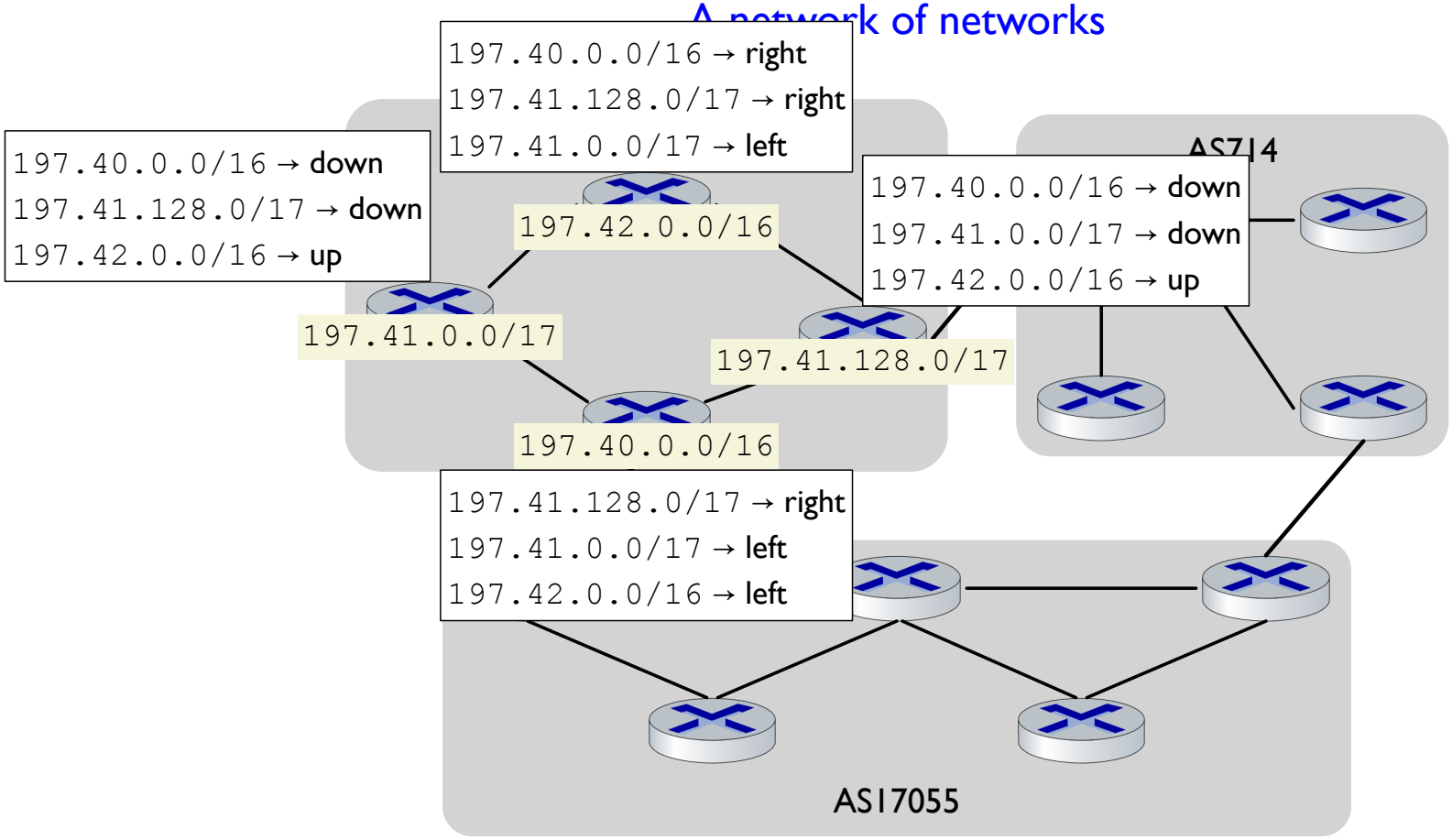
Managed as a whole by some organization

picks its own **intra-AS** routing configuration, often OSPF



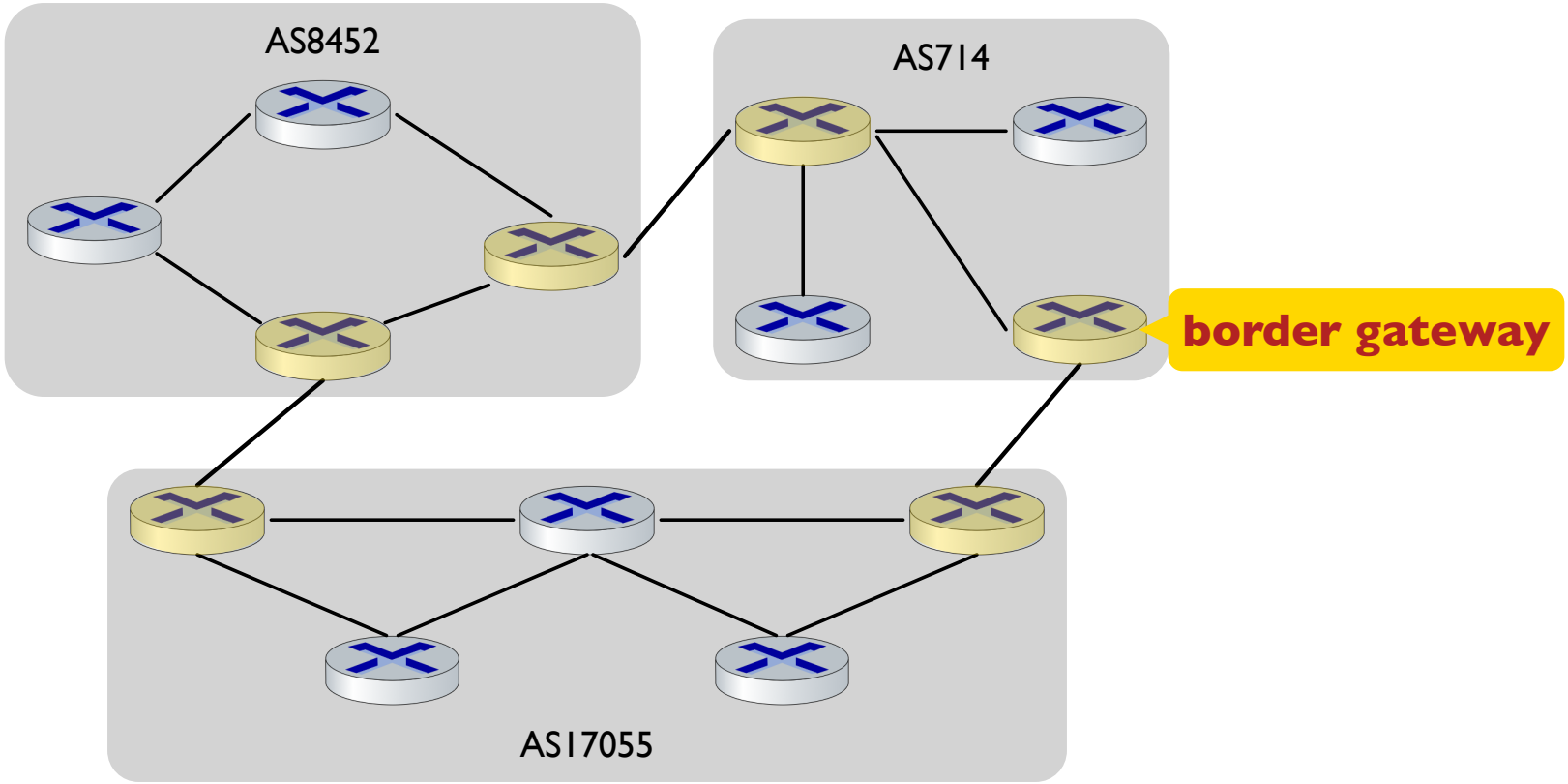
Applying Graph Algorithms in the Internet

A network of networks



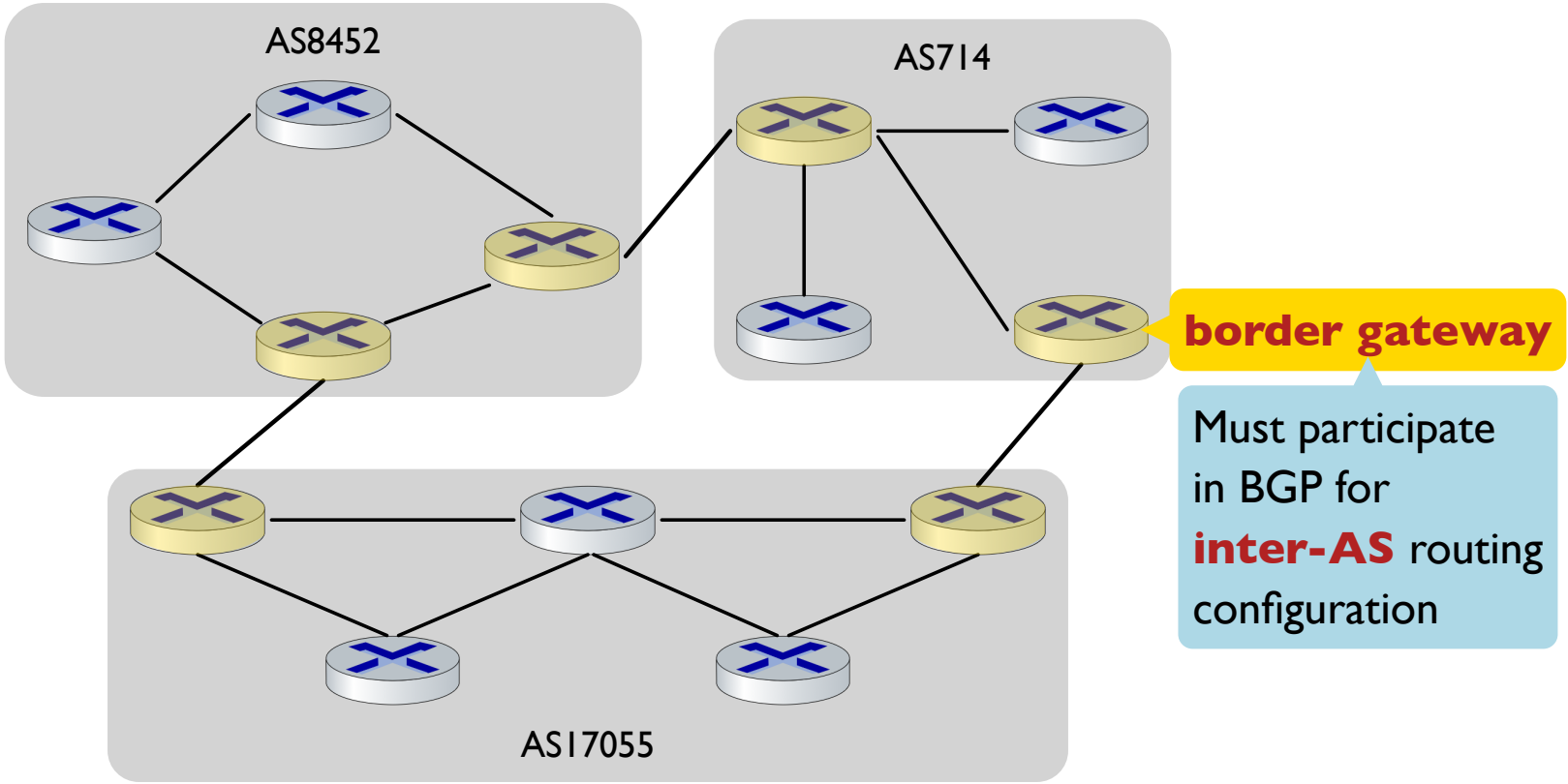
Applying Graph Algorithms in the Internet

A network of networks

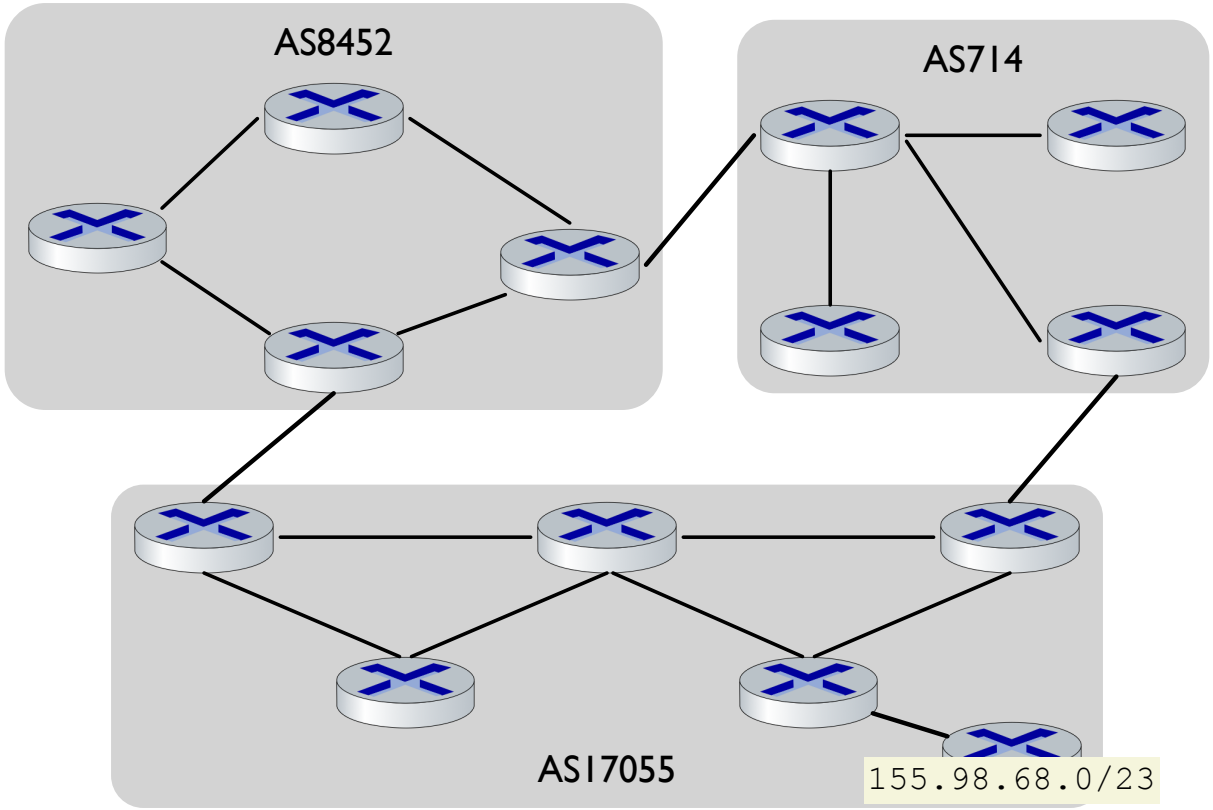


Applying Graph Algorithms in the Internet

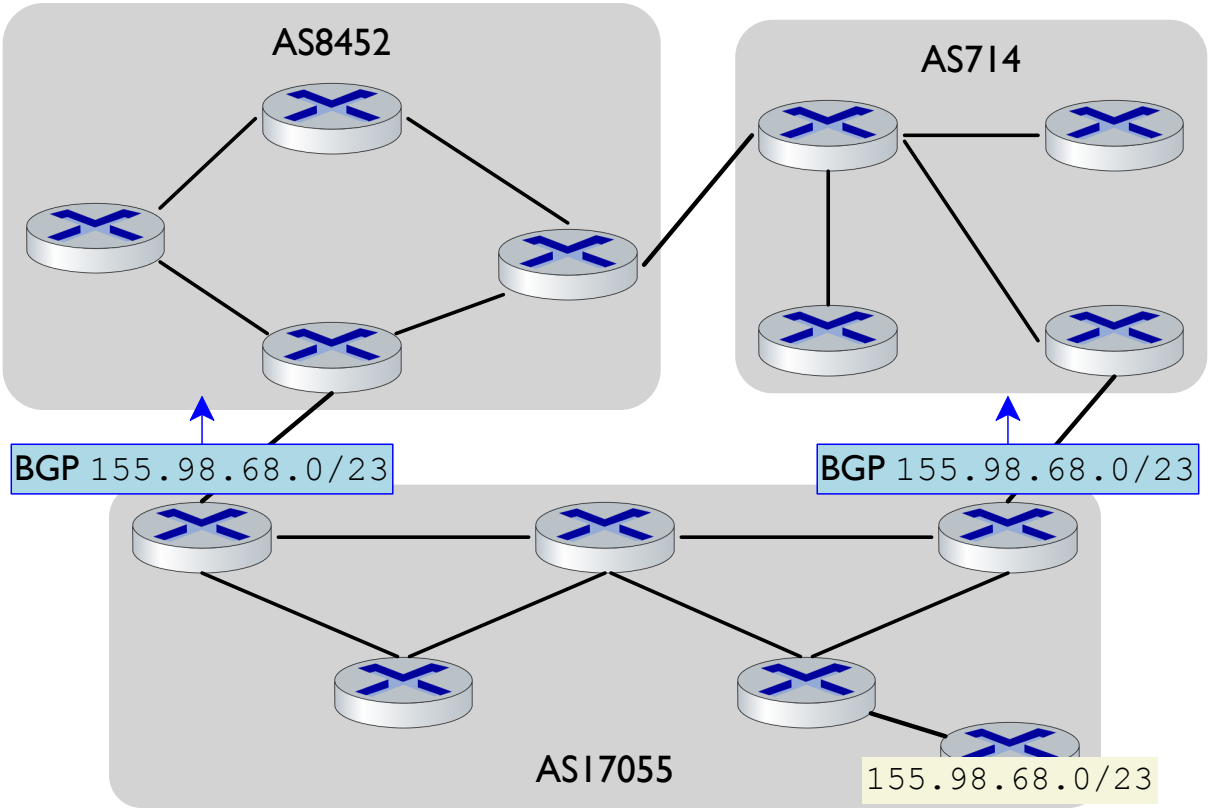
A network of networks



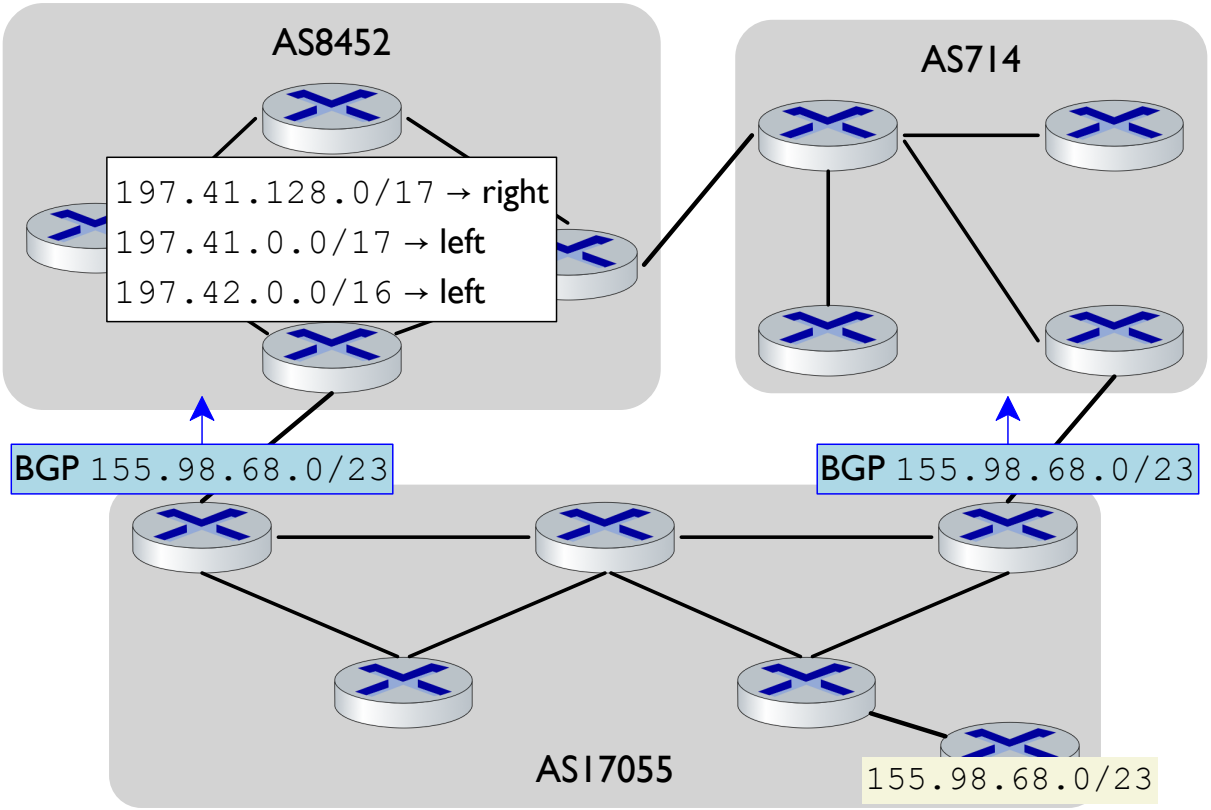
Adding a Subnet



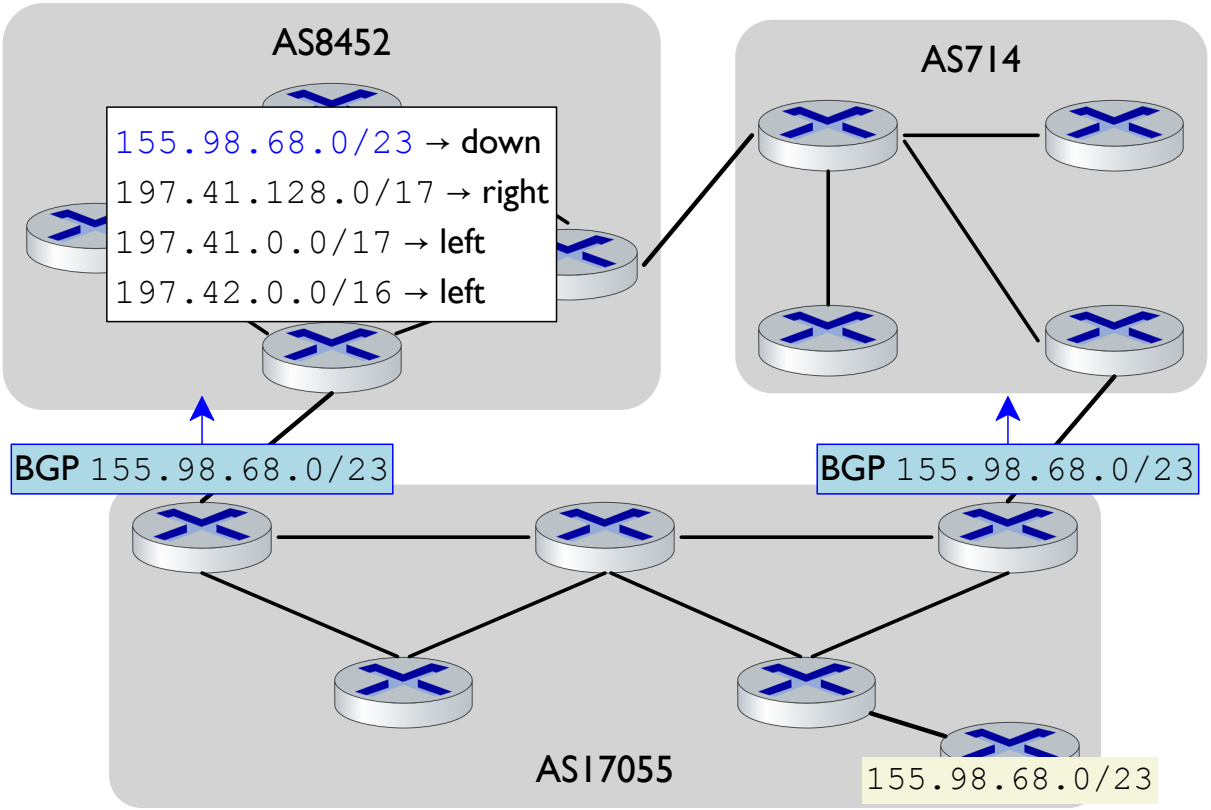
Adding a Subnet



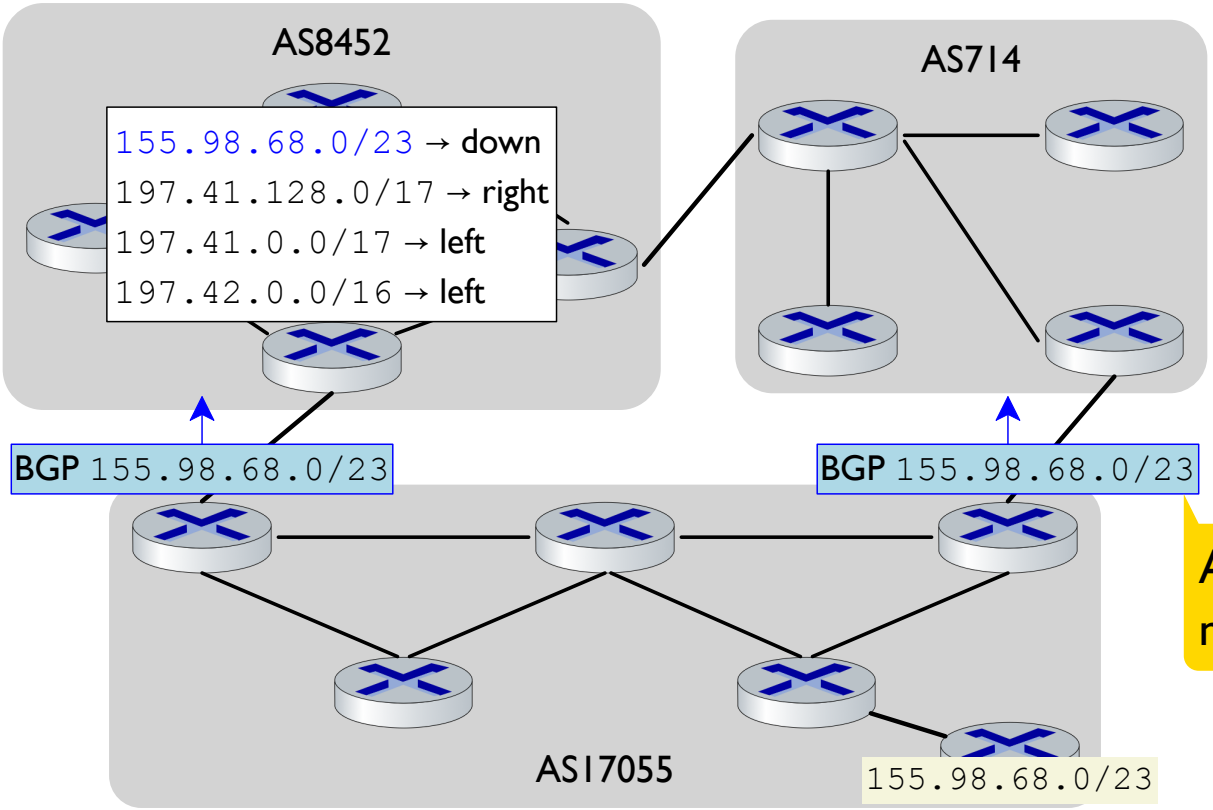
Adding a Subnet



Adding a Subnet

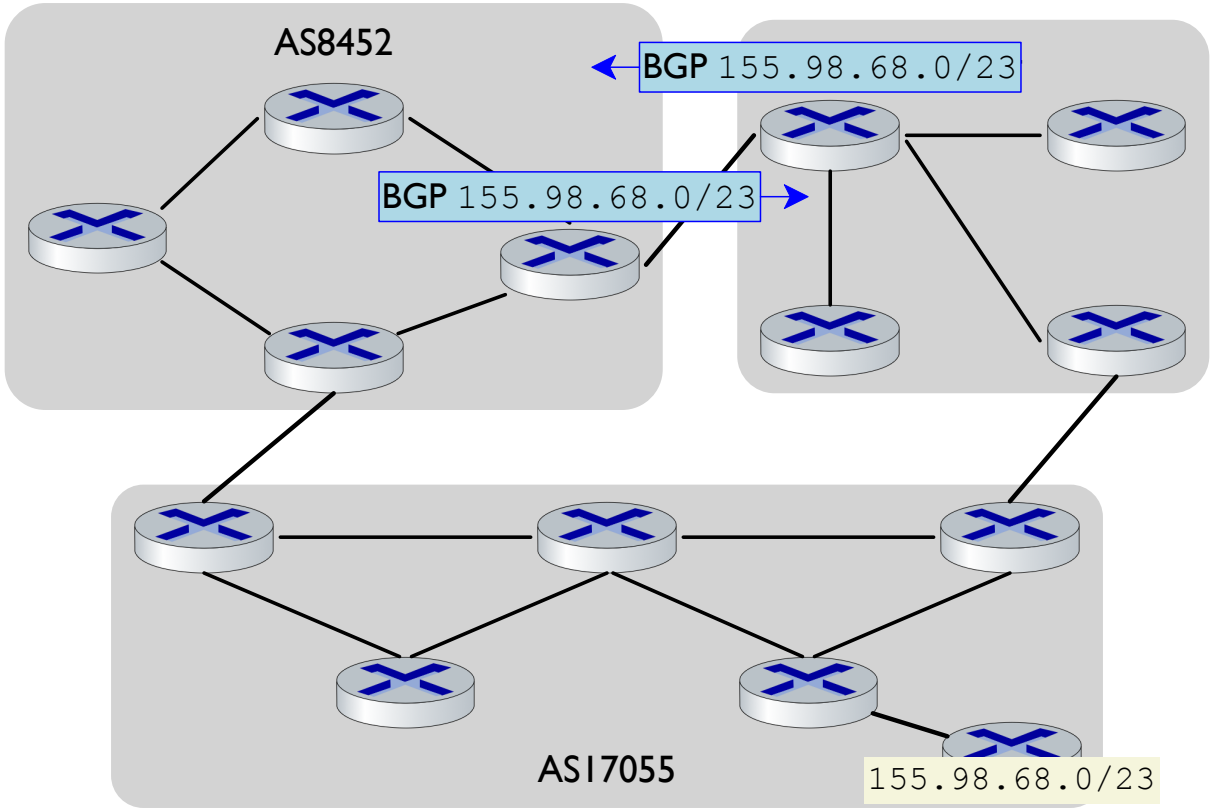


Adding a Subnet

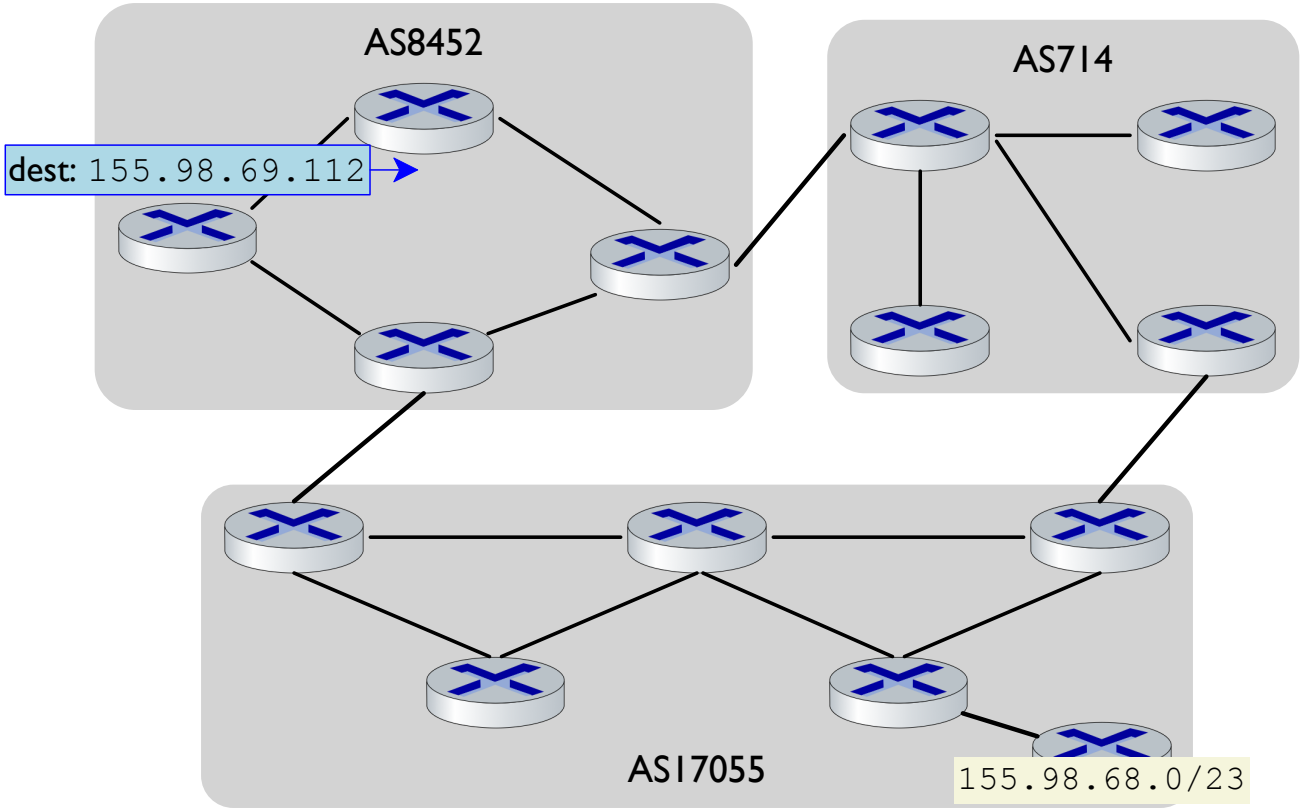


Acts as a diff on the node's distance table

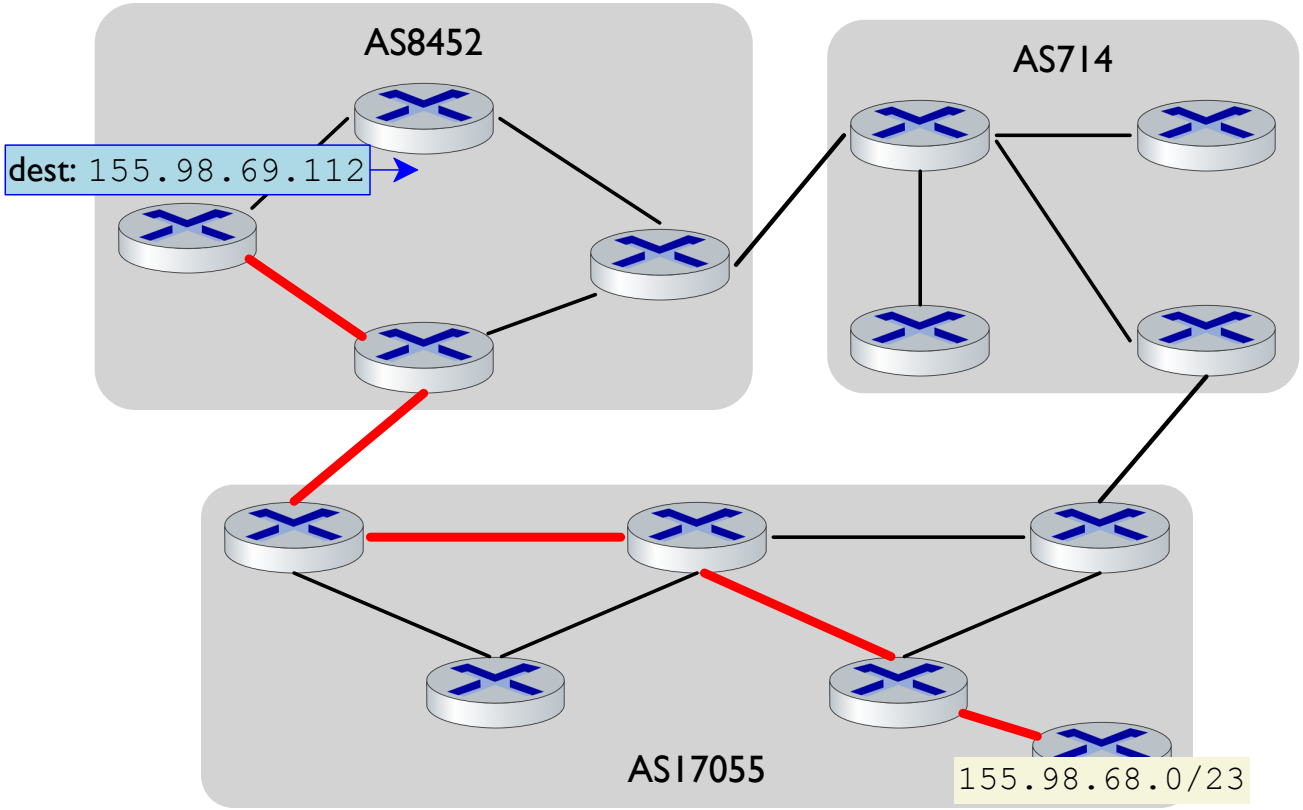
Adding a Subnet



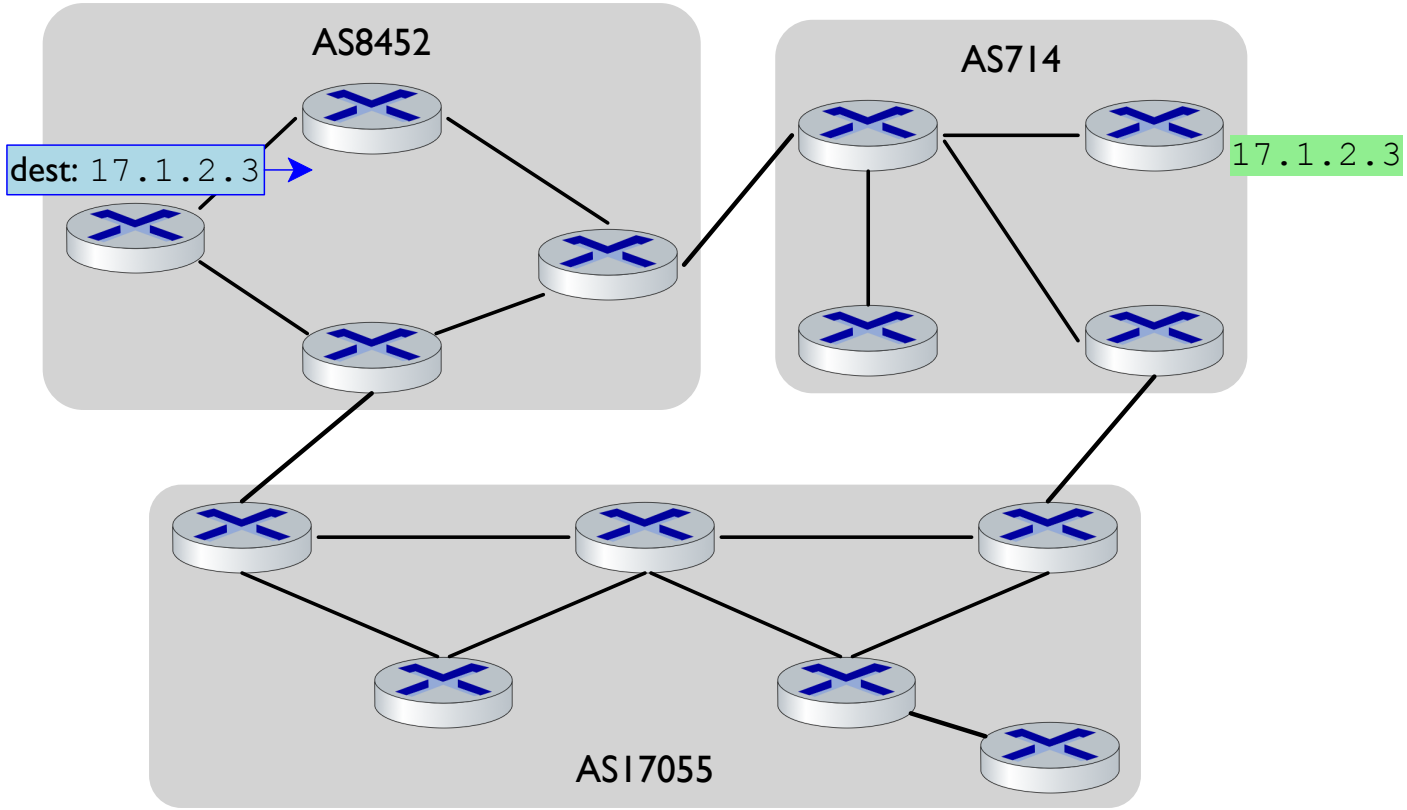
Adding a Subnet



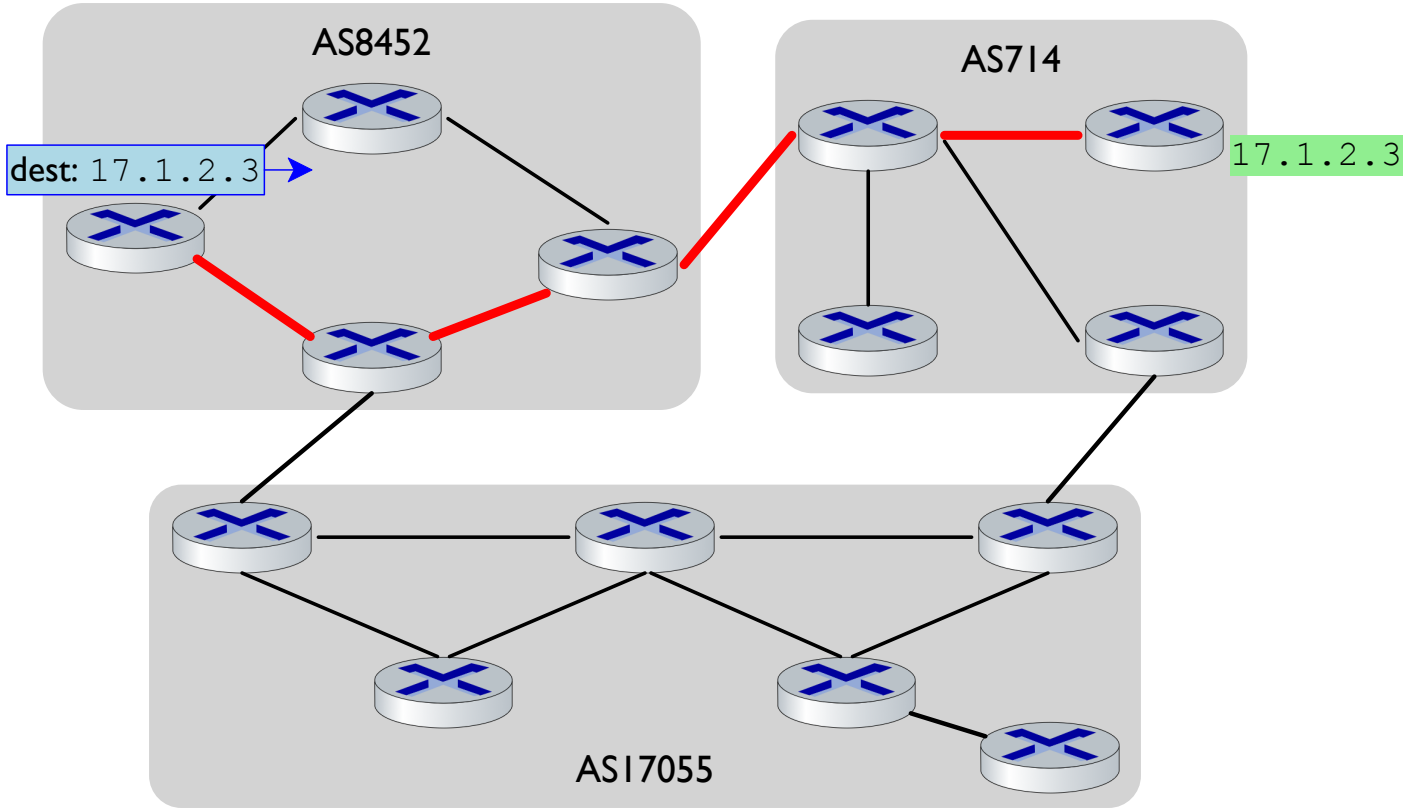
Adding a Subnet



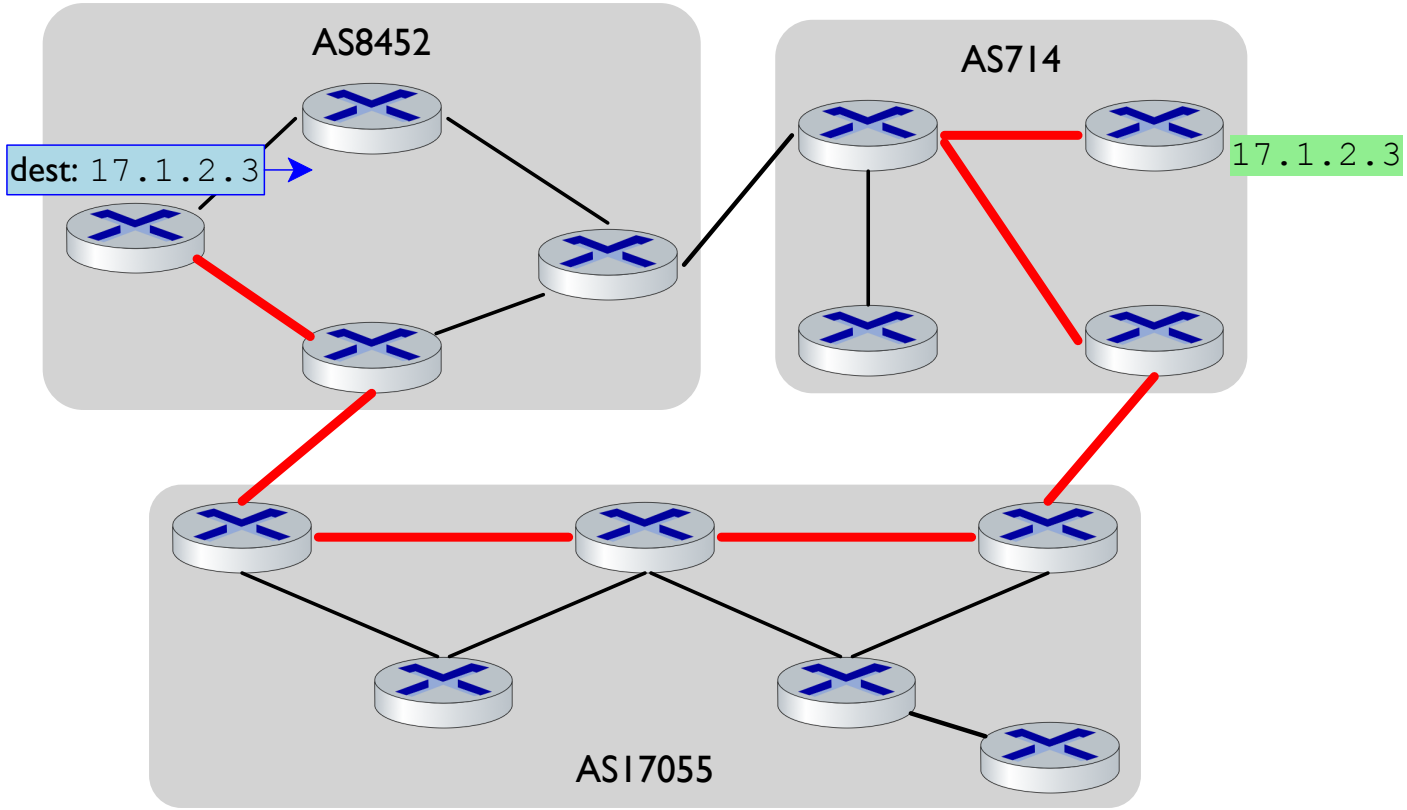
Hot Potato Routing



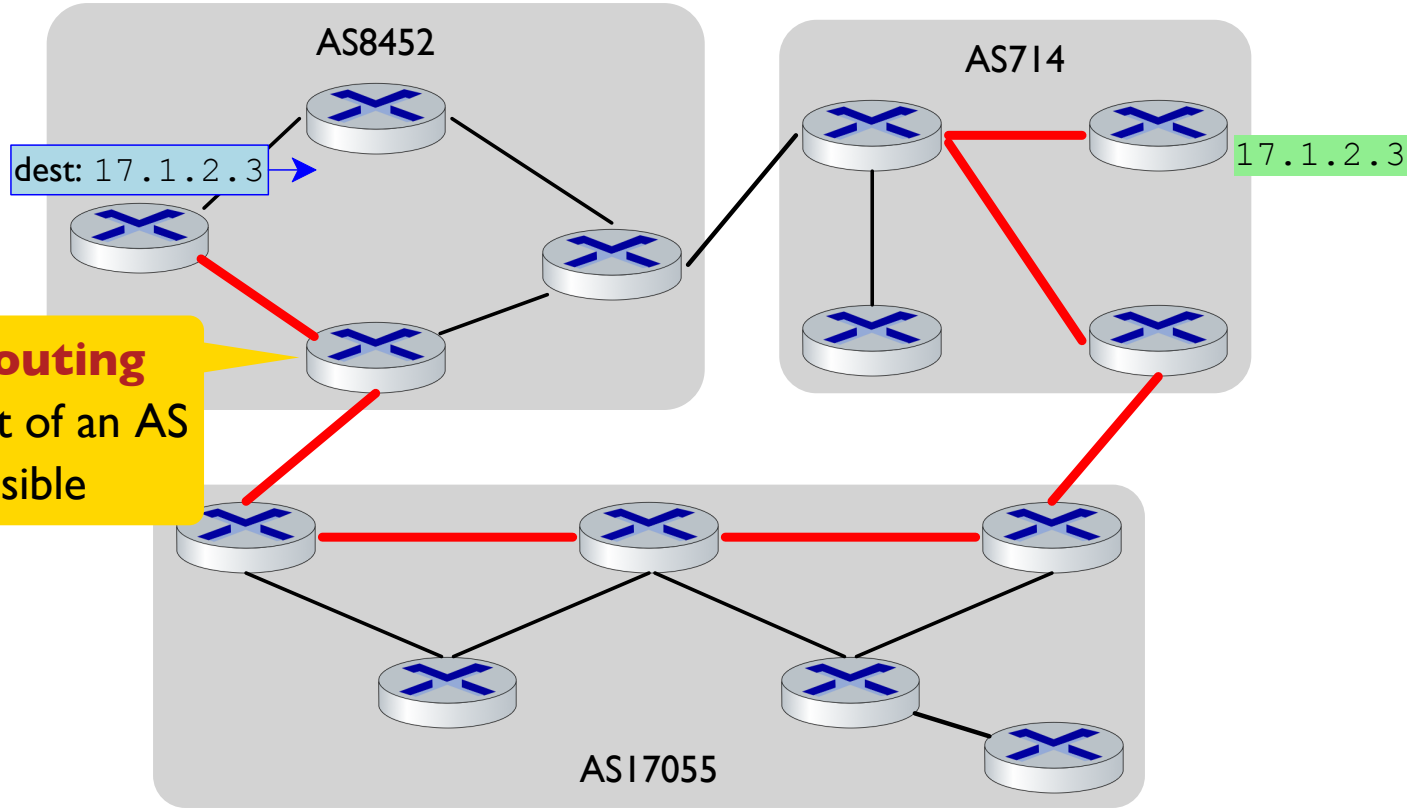
Hot Potato Routing



Hot Potato Routing

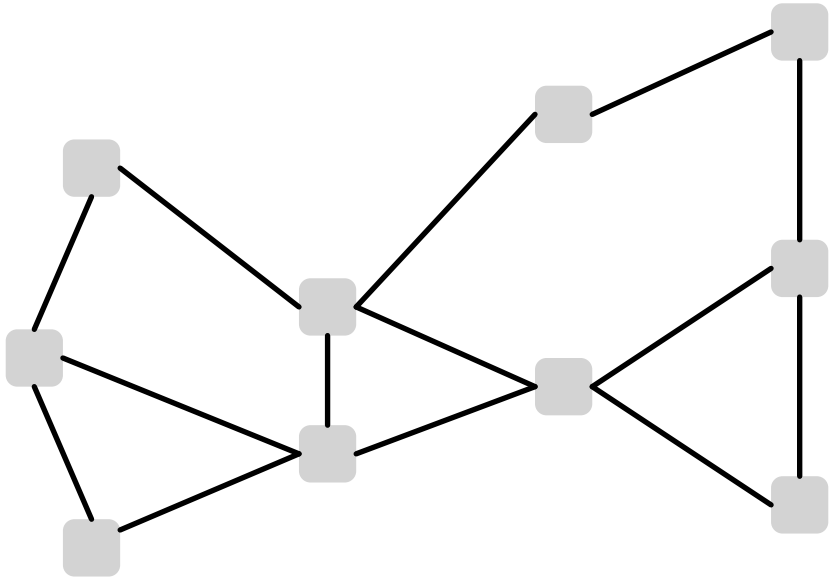


Hot Potato Routing

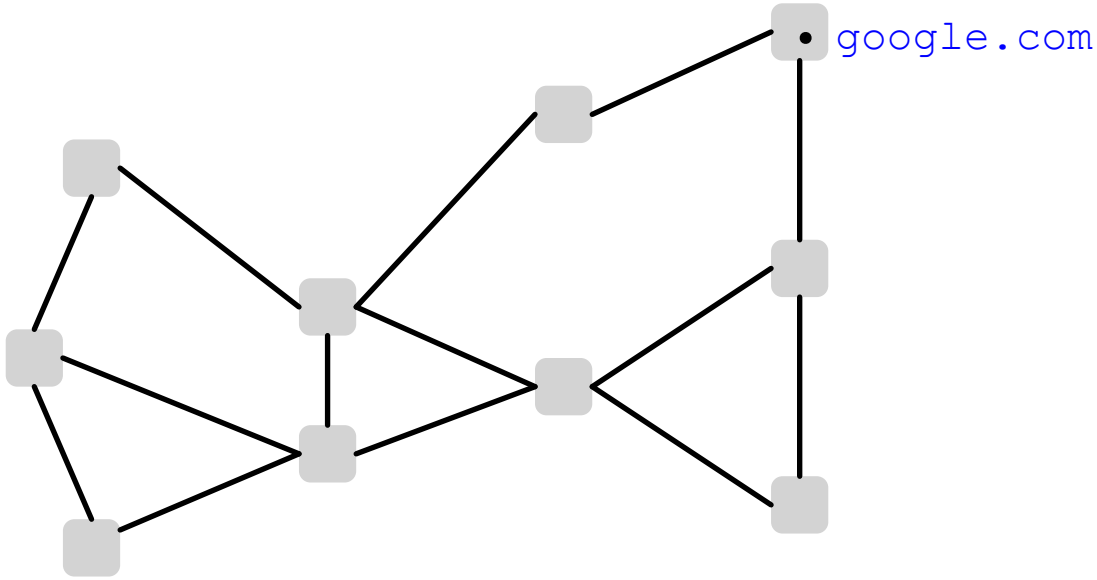


Hot potato routing
gets a packet out of an AS as quickly as possible

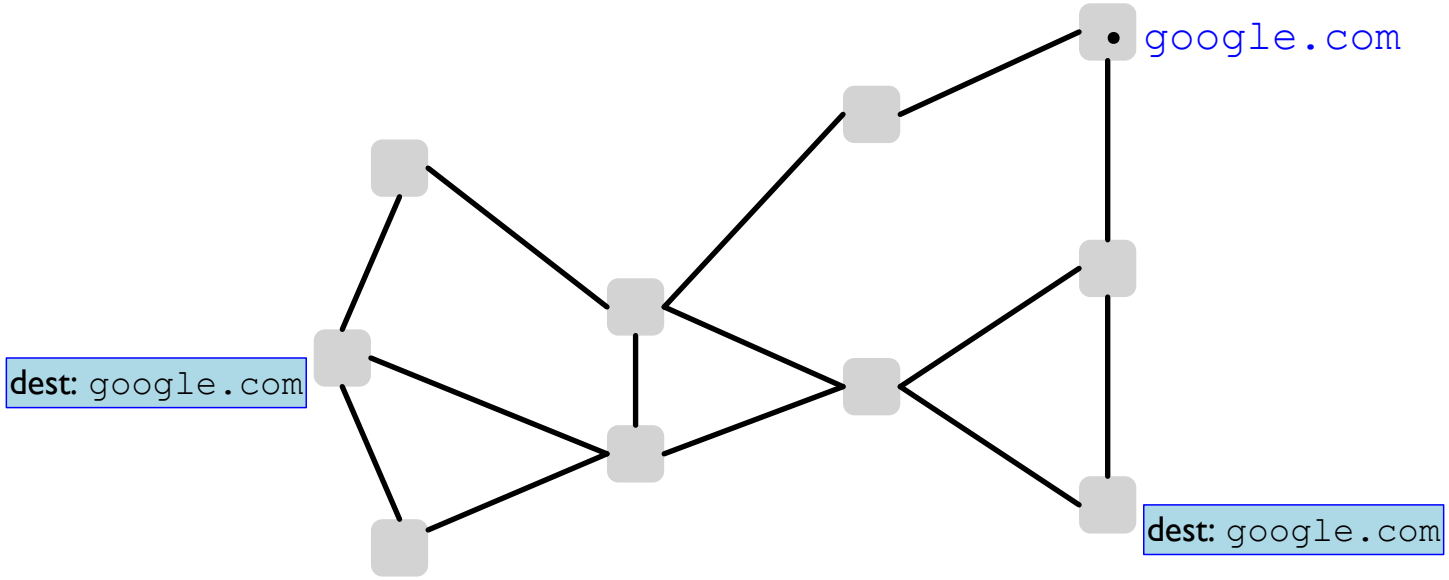
Finding Distributed Services via DNS



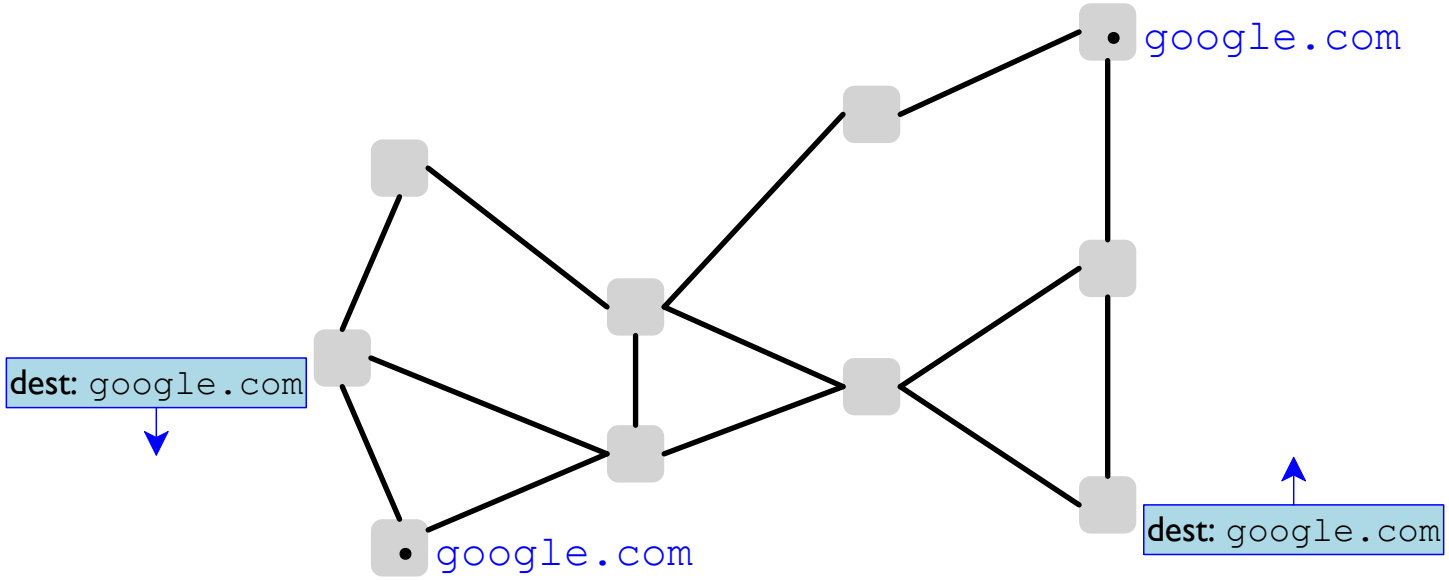
Finding Distributed Services via DNS



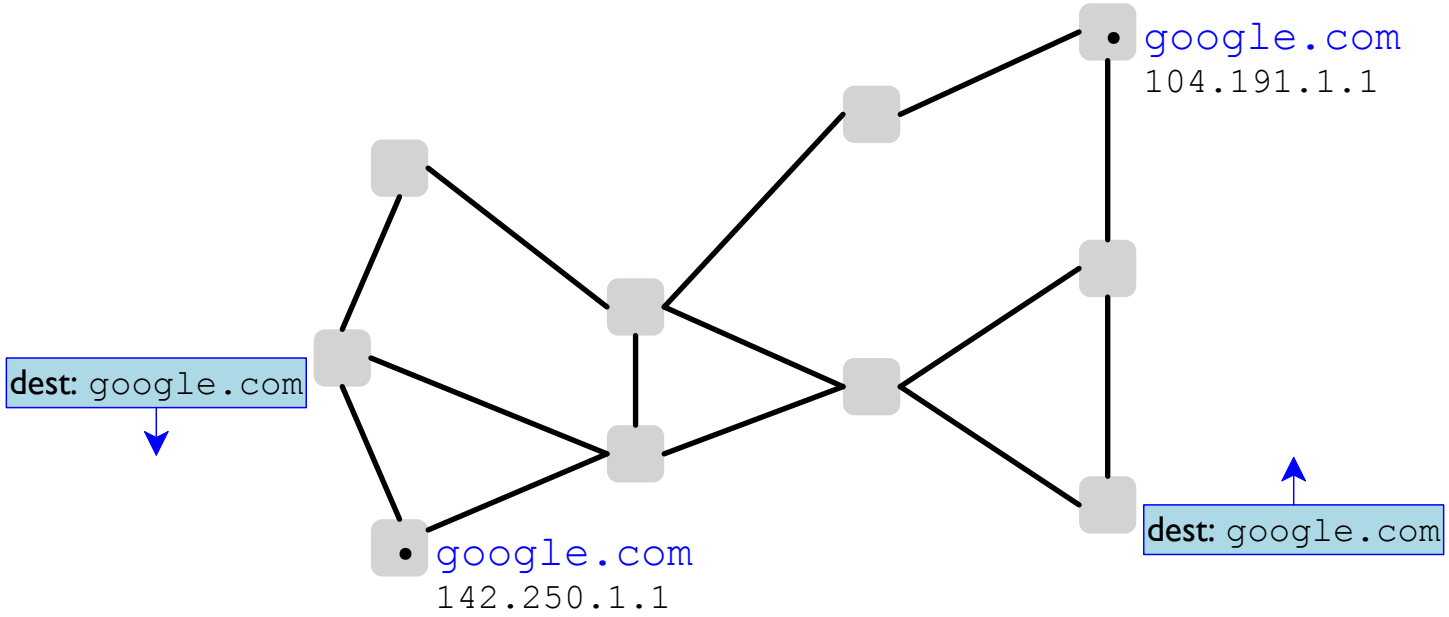
Finding Distributed Services via DNS



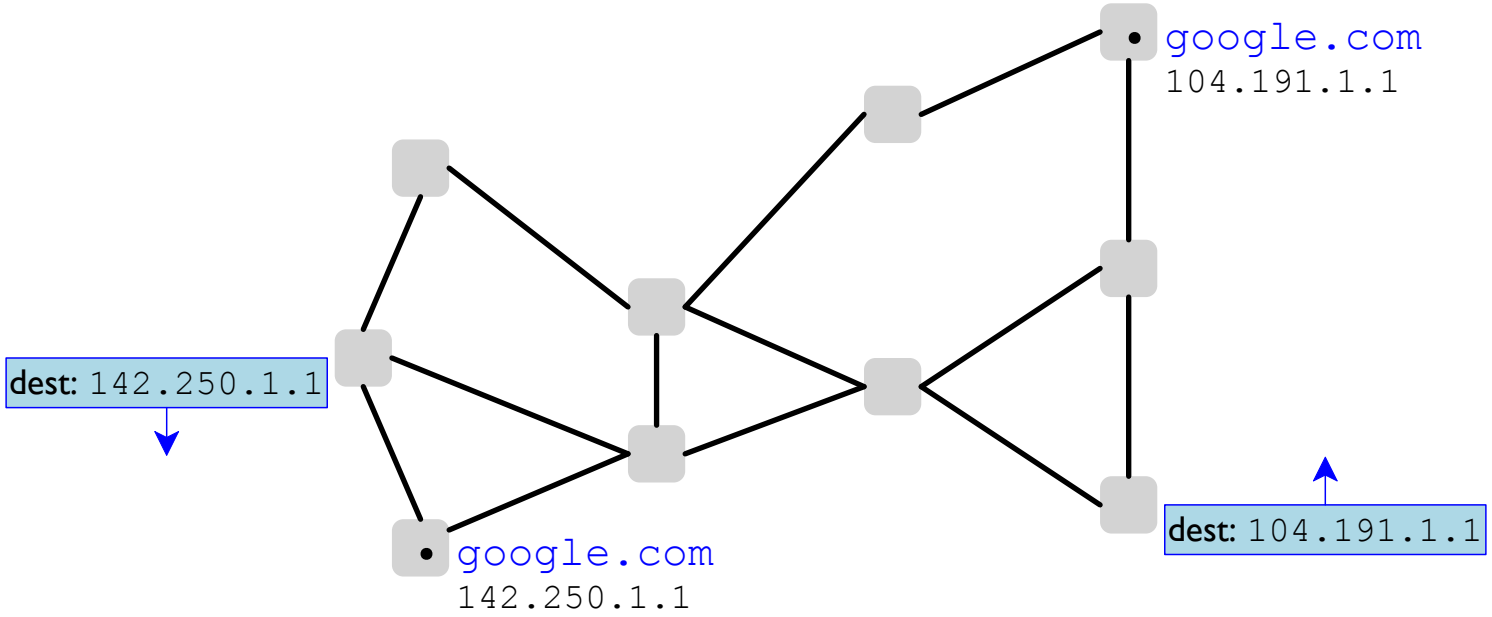
Finding Distributed Services via DNS



Finding Distributed Services via DNS

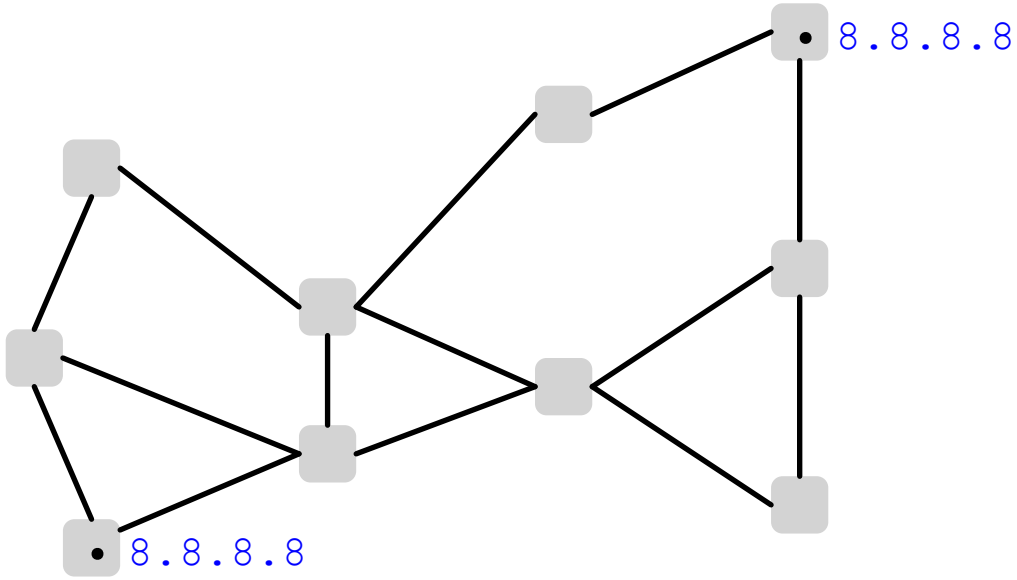


Finding Distributed Services via DNS

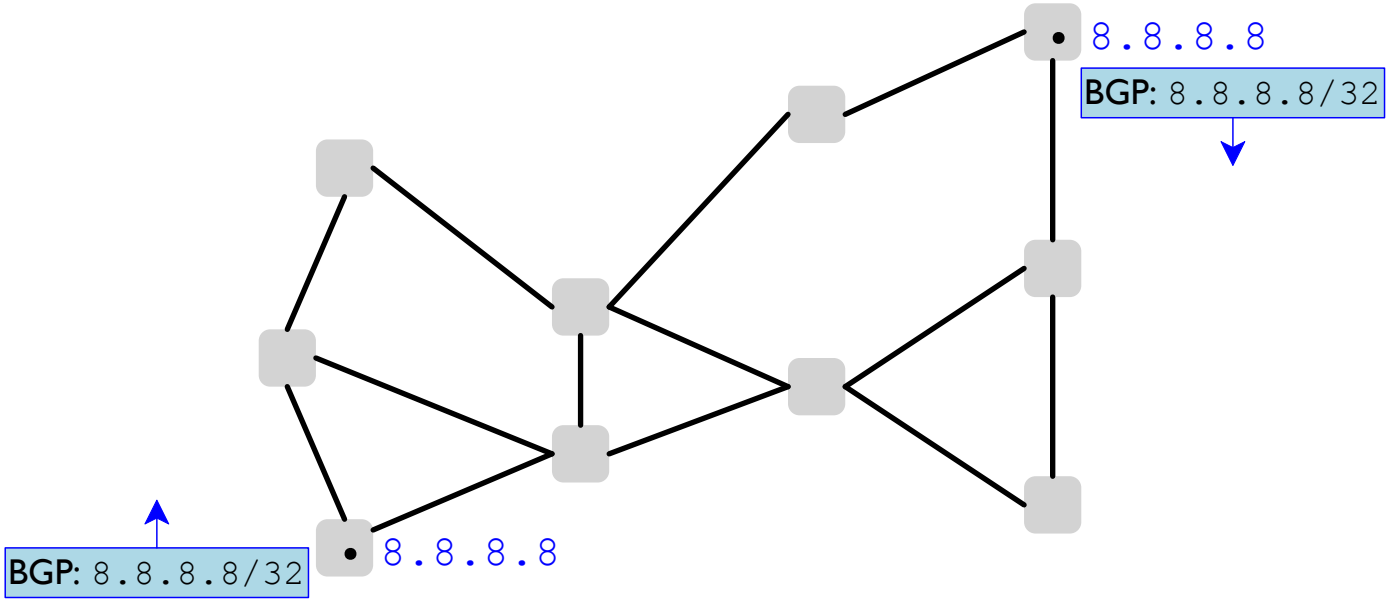


DNS servers can point different parts of the network in different directions

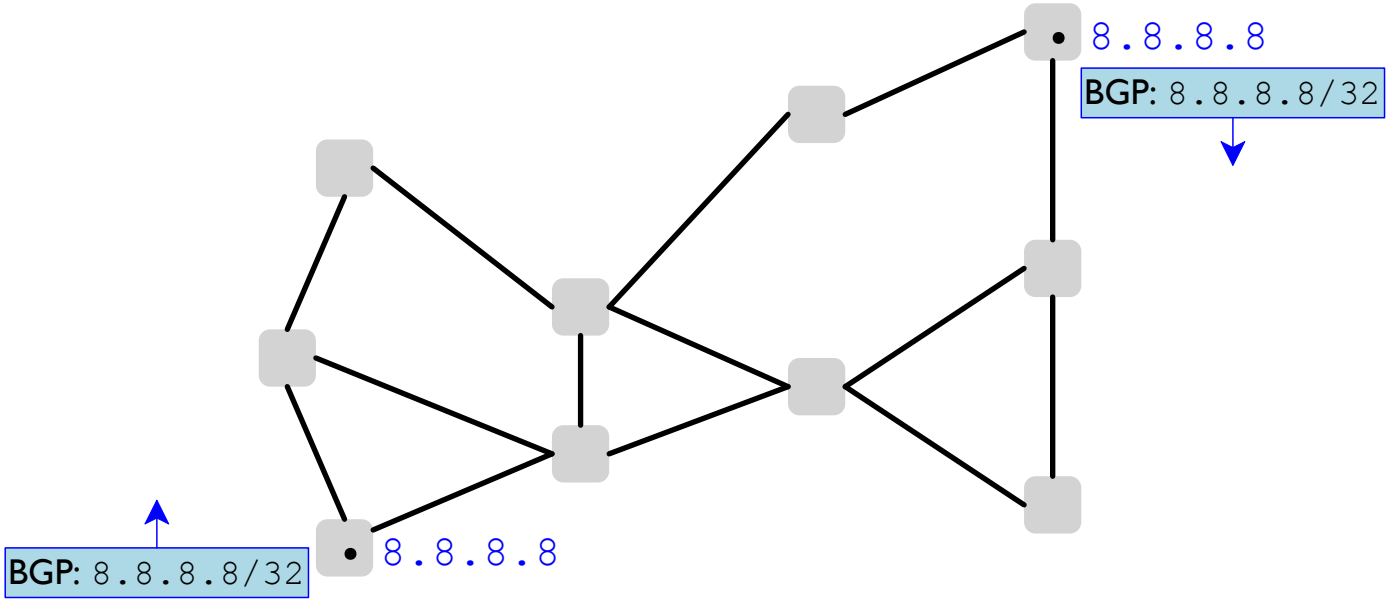
Finding Distributed Services via IP Anycast



Finding Distributed Services via IP Anycast

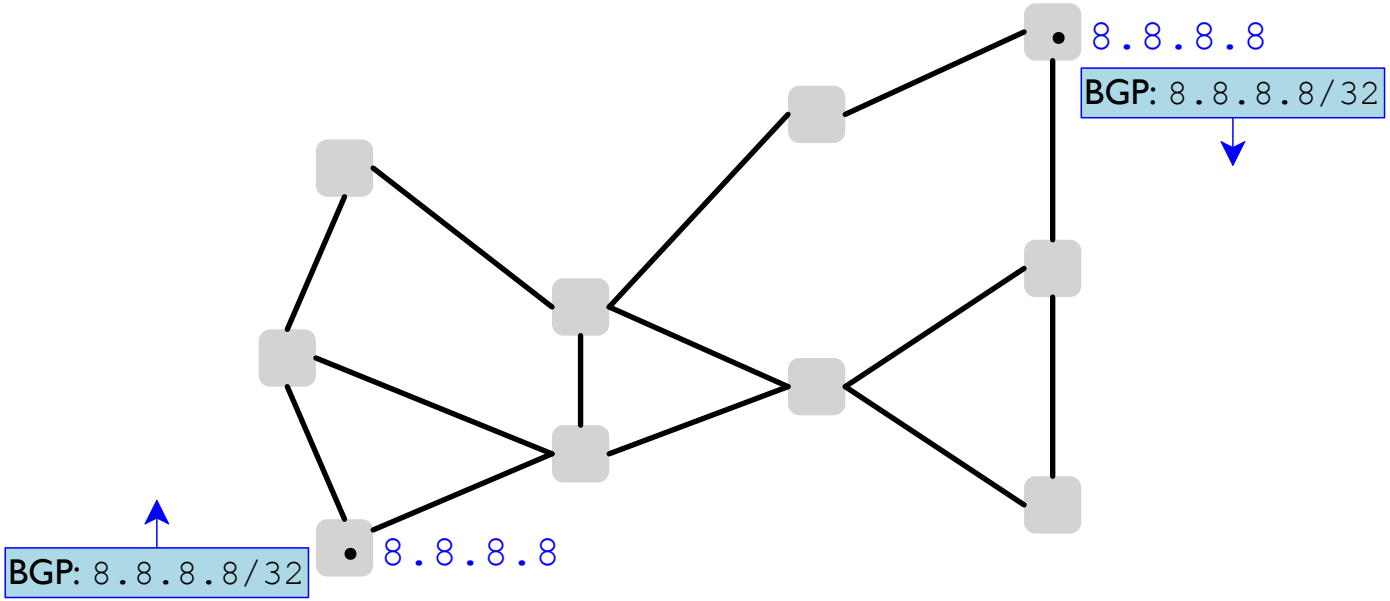


Finding Distributed Services via IP Anycast



Forwarding tables act as a level of indirection for IP addresses to support **IP anycast**

Finding Distributed Services via IP Anycast



IP anycast works best for DNS and similar, where there's no connection to maintain

Summary

The network layer's **control plane** configures routers

Dijkstra's algorithm as implemented in a
link state (LS) routing protocol
good for deriving local routing

The **Bellman-Ford algorithm** as implemented in a
distance vector (DV) routing protocol
good for deriving global routing

BGP is a DV routing protocol that is the inter-AS standard