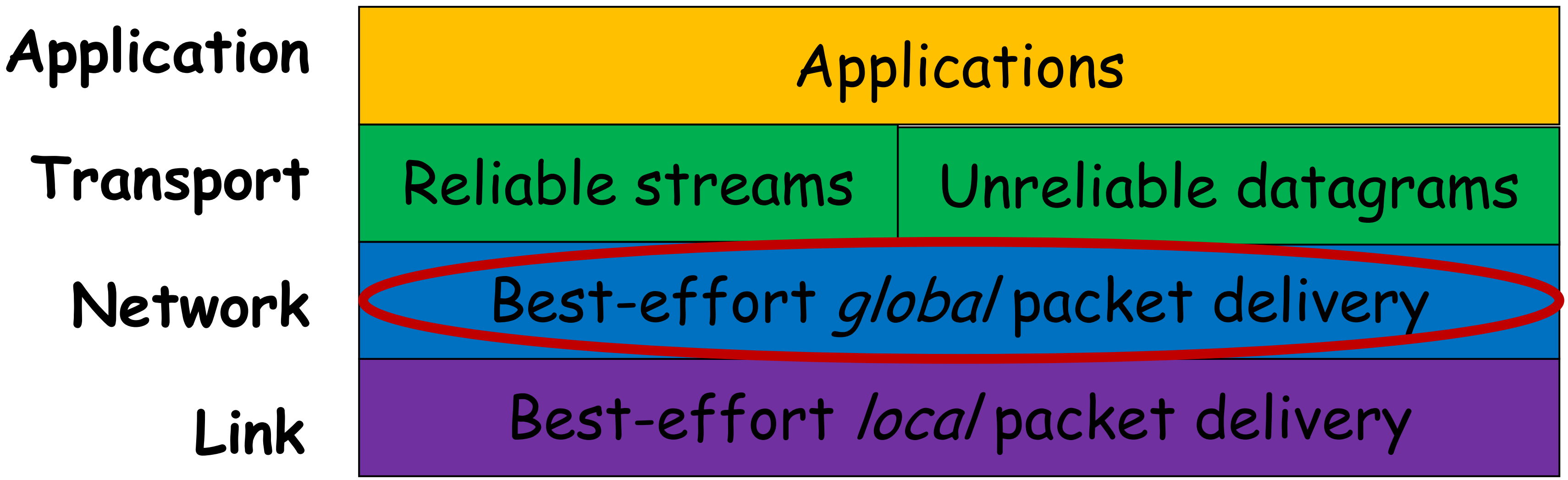
Forwarding and Routing

Richard T. B. Ma
School of Computing
National University of Singapore

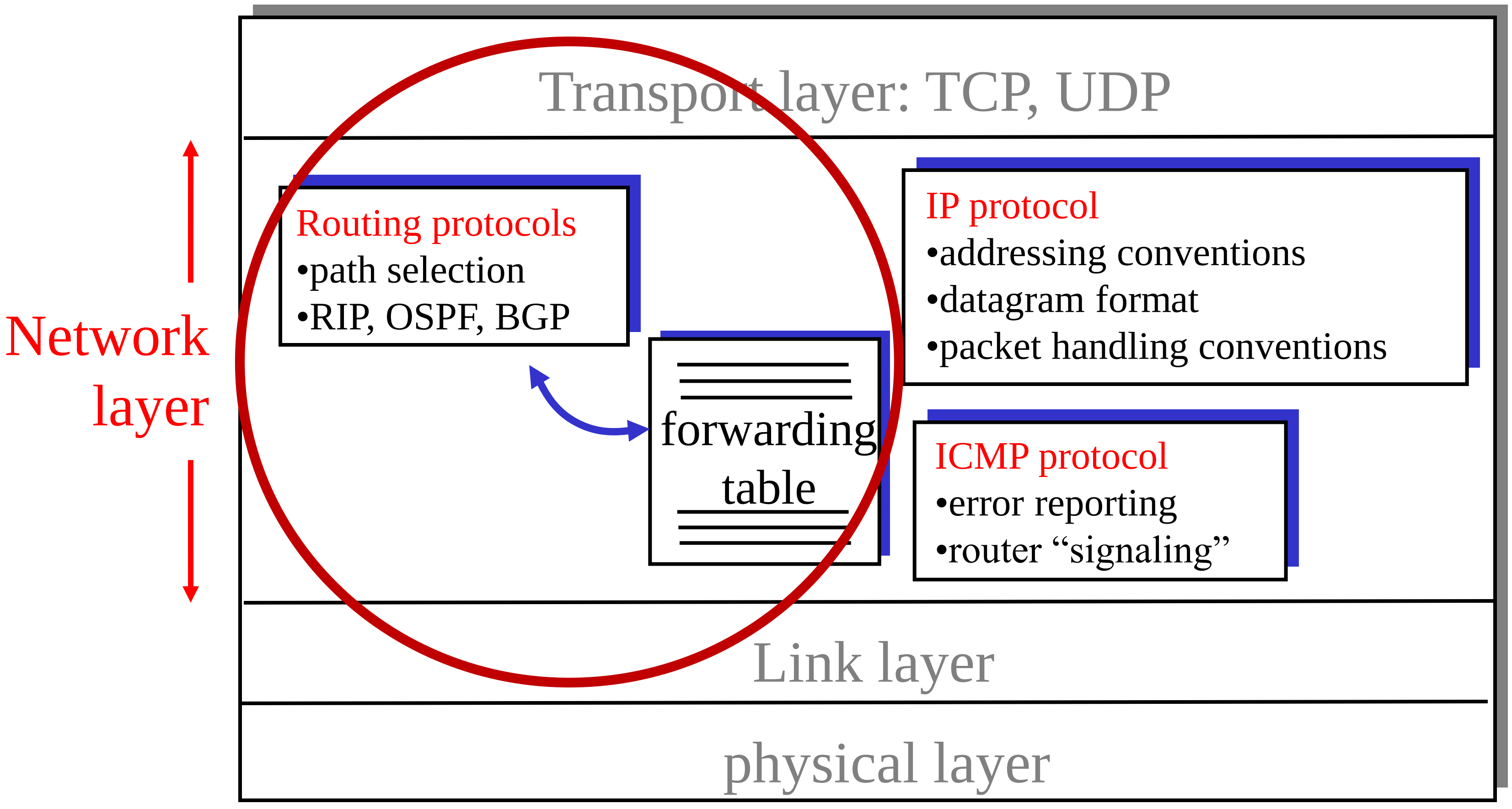
CS 3103: Compute Networks and Protocols

IP Protocol Stack: Key Abstractions



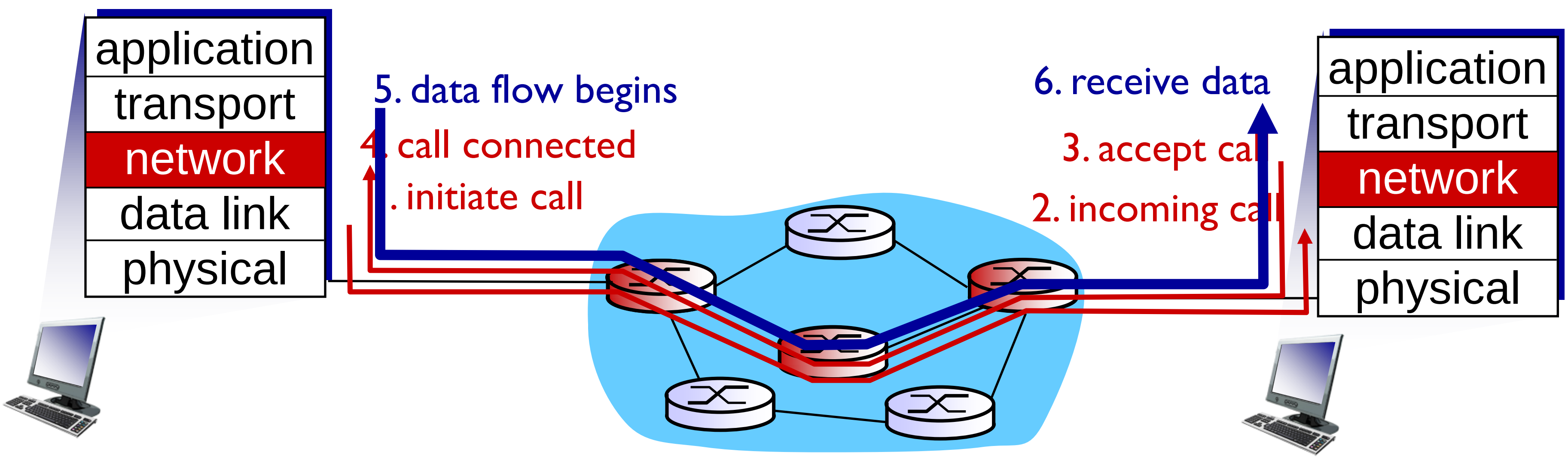
The Internet Network layer

Host, router network layer functions:



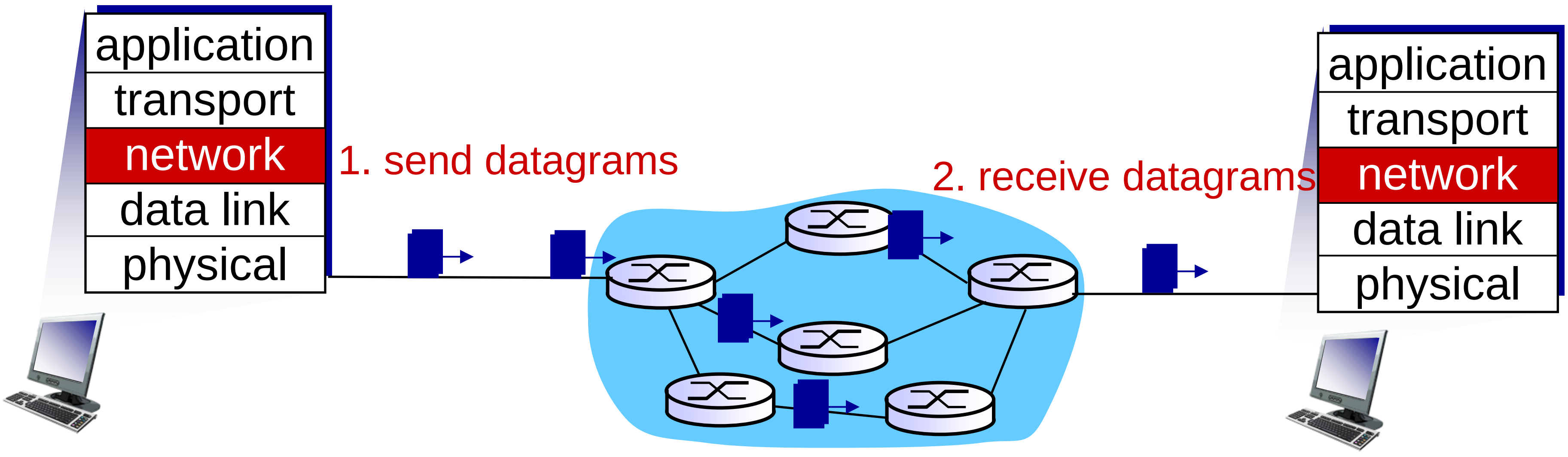
Routing in virtual circuits

- used to setup, maintain teardown VC
- used in ATM, frame-relay, X.25
- not used in today's Internet



Datagram networks

- no call setup at network layer
- routers: no end-to-end state information
  - no network-level concept of "connection"
- packets forwarded using destination host address



Two Key Network-Layer Functions

- *Forwarding (data plane):* move packets from an input interface to an appropriate output interface in a router
- *Routing (control plane):* determine route taken by packets from source to destination
  - *routing algorithms*

analogy:

- routing: process of planning trip from source to destination
- forwarding: process of getting through single interchange

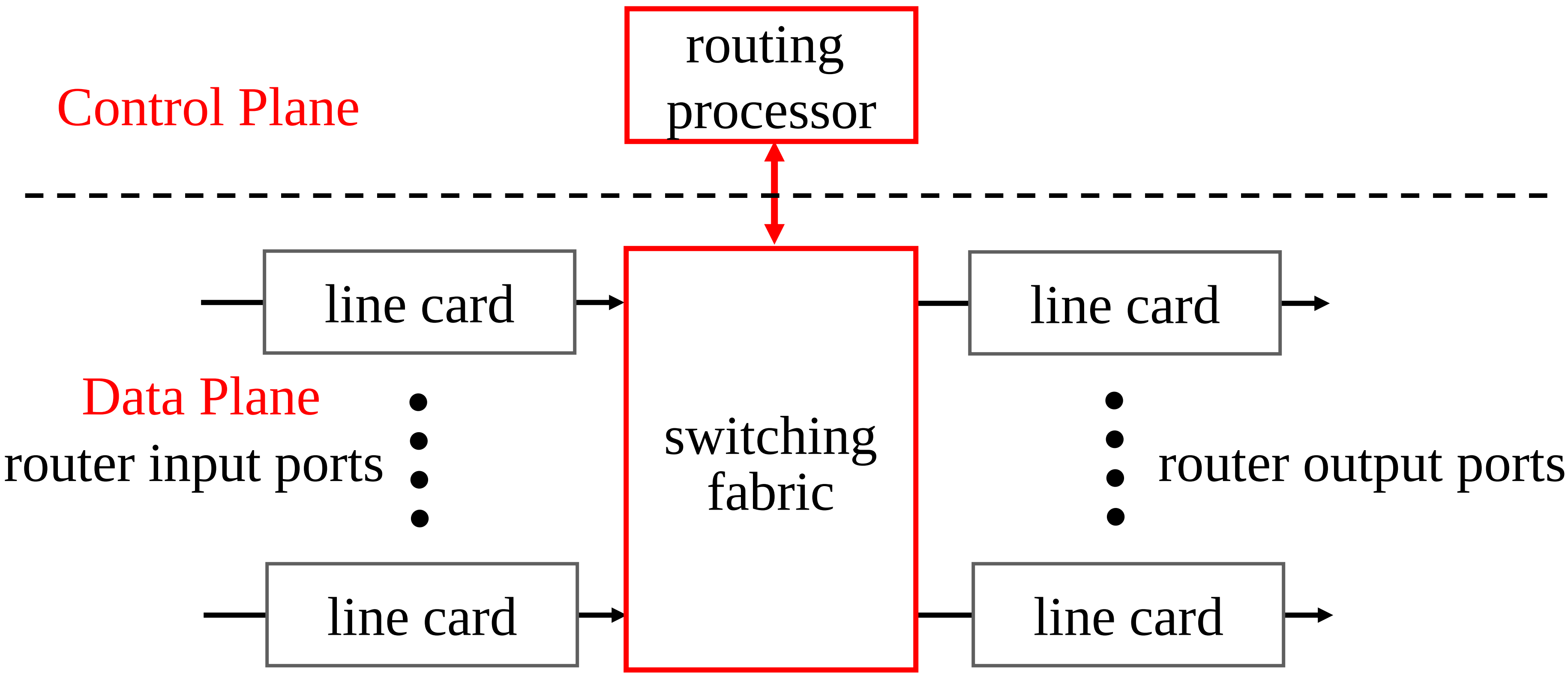
Forwarding: Hop-by-Hop

- Each router has a forwarding table
  - maps destination addresses to outgoing interfaces
- Upon receiving a packet
  - inspect the destination IP address in the header index into the table
  - determine the outgoing interface
  - forward the packet out that interface
- Then, the next router on the path repeats
  - and the packet travels along the path to the destination

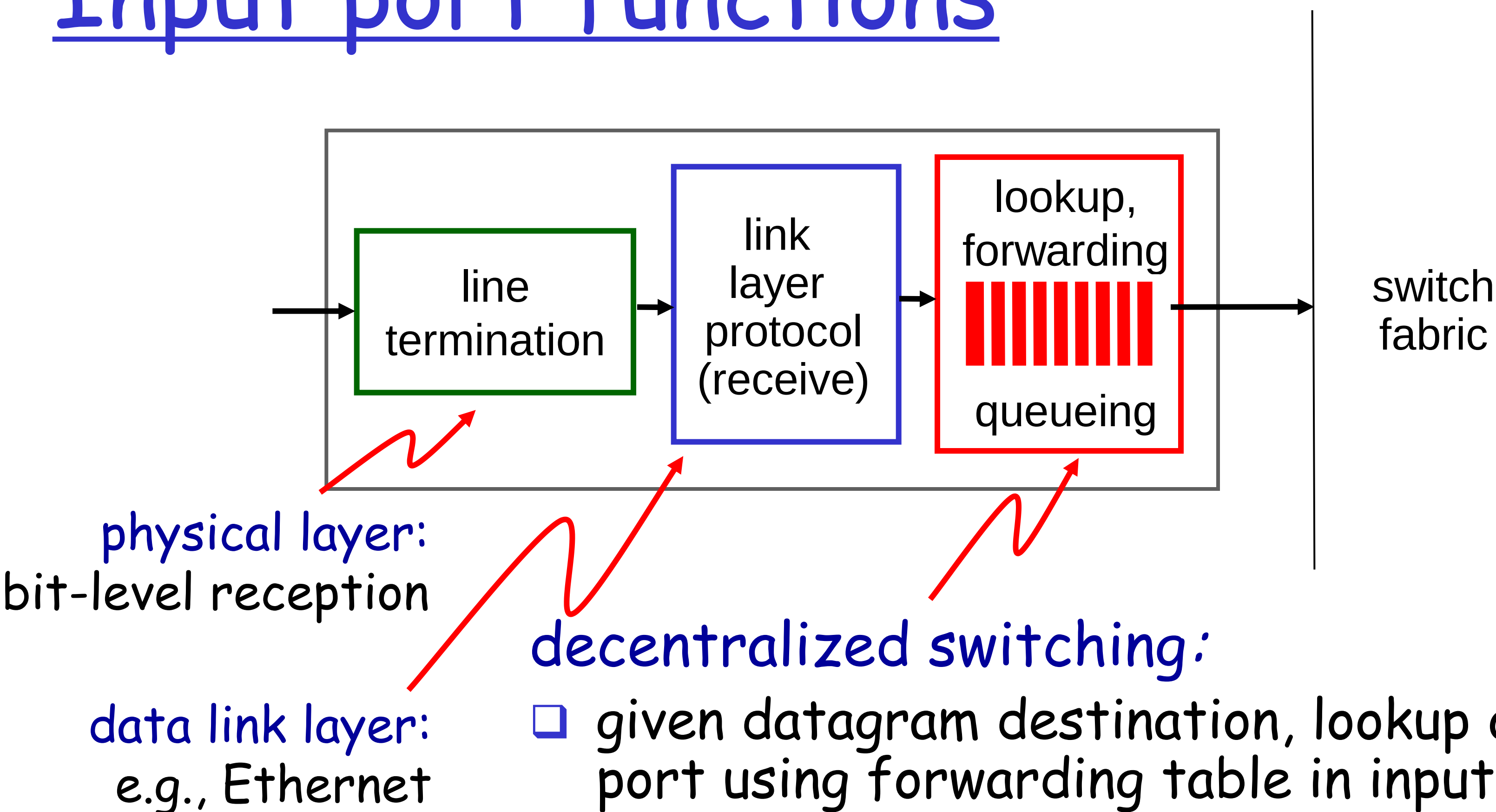
Router Architecture Overview

two key router functions:

- run routing algorithms/protocol (RIP, OSPF, BGP)
- *forwarding* datagrams from incoming to outgoing link



Input port functions

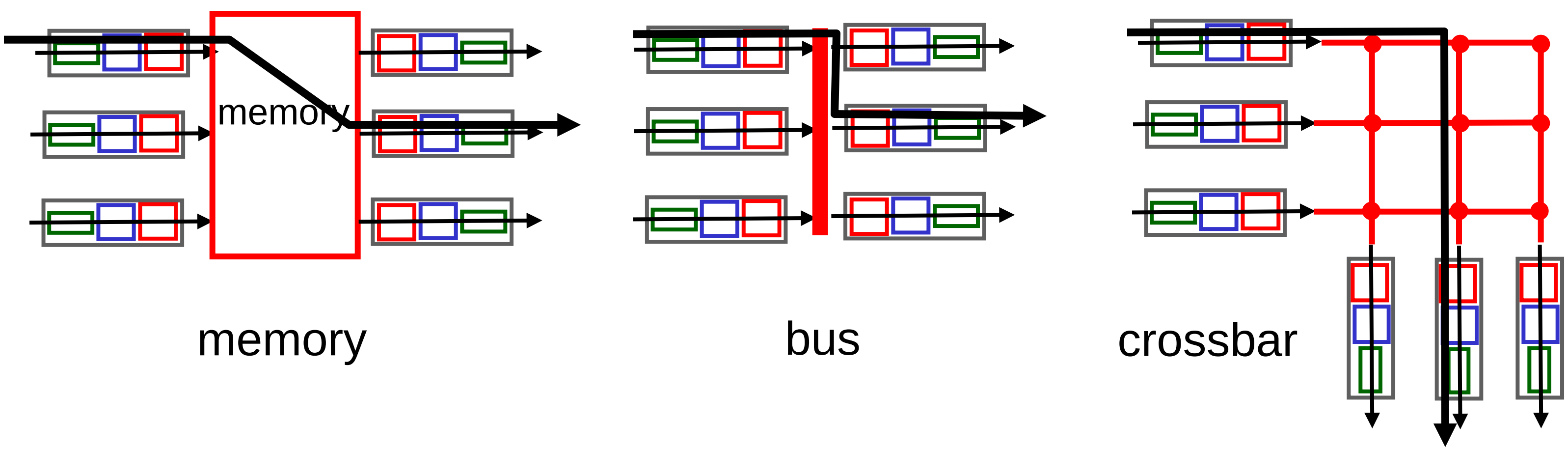


decentralized switching:

- given datagram destination, lookup output port using forwarding table in input port memory *("match plus action")*
- goal: complete input port processing at "line speed"
- queuing: if datagrams arrive faster than forwarding rate into switch fabric

Switching fabrics

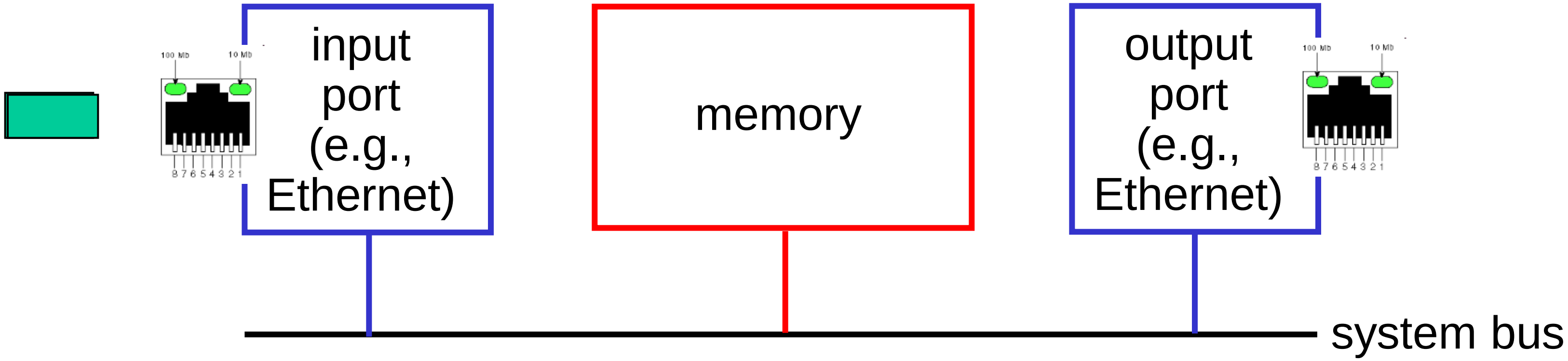
- transfer packet from input buffer to appropriate output buffer
- switching rate: rate at which packets can be transfer from inputs to outputs
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate desirable
- three types of switching fabrics



Switching via memory

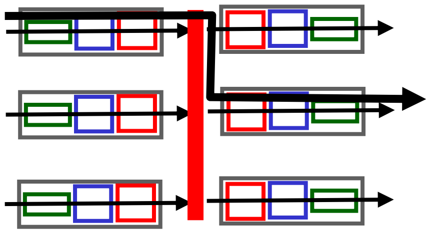
first generation routers:

- traditional computers with switching under direct control of CPU
- packet copied to system's memory
- speed limited by memory bandwidth (2 bus crossings per datagram)



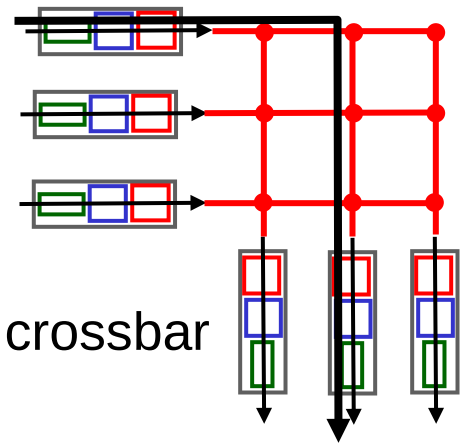
Switching via a bus

- datagram from input port memory to output port memory via a shared bus
- internal label is used to indicate output port
- *bus contention:* switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for access and enterprise routers

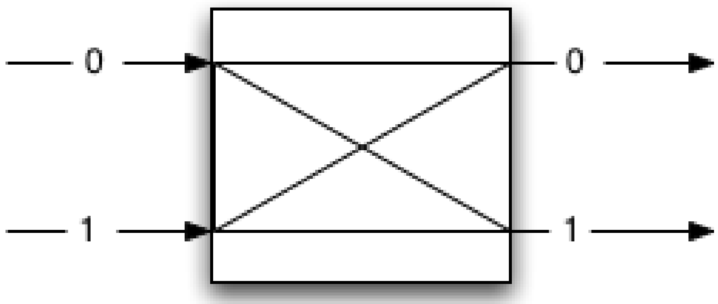


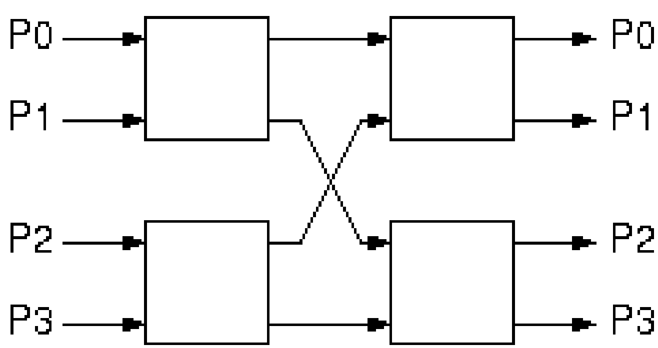
Switching via interconnection nets

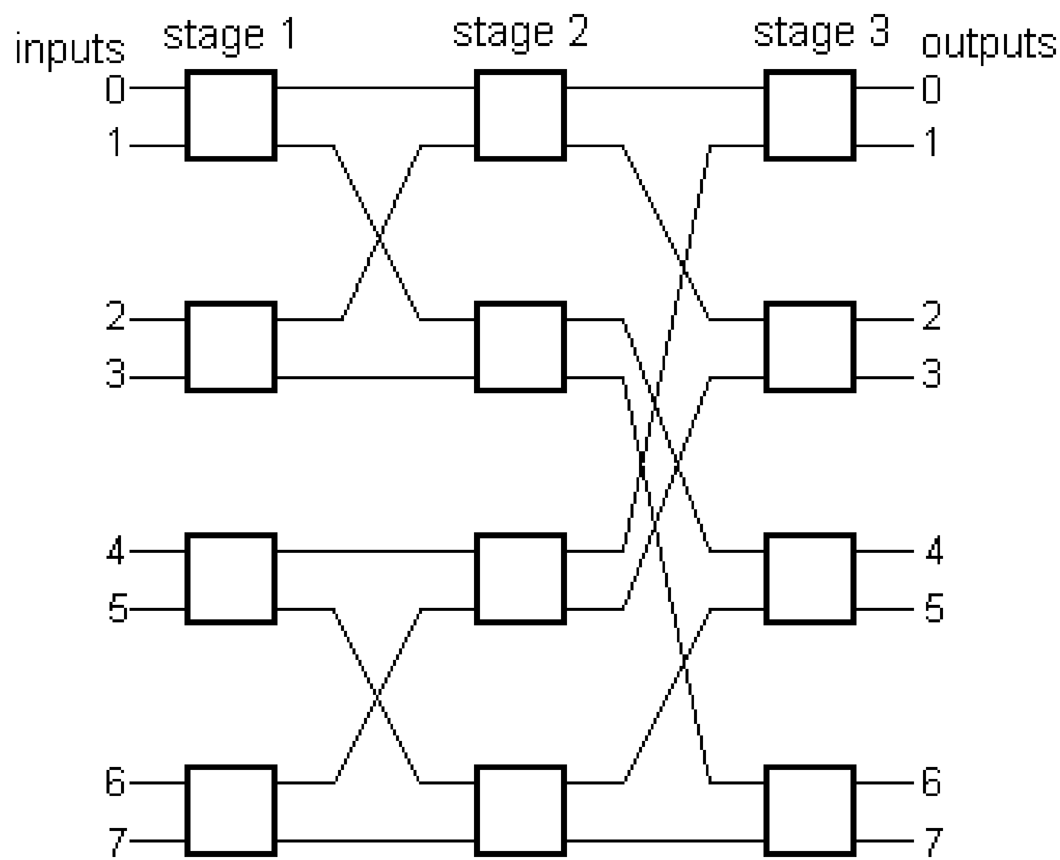
- overcome bus bandwidth limitations
- banyan networks, crossbar, other interconnection nets initially developed to connect processors in multiprocessor
- advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
- Cisco 12000: switches 60 Gbps through the interconnection network



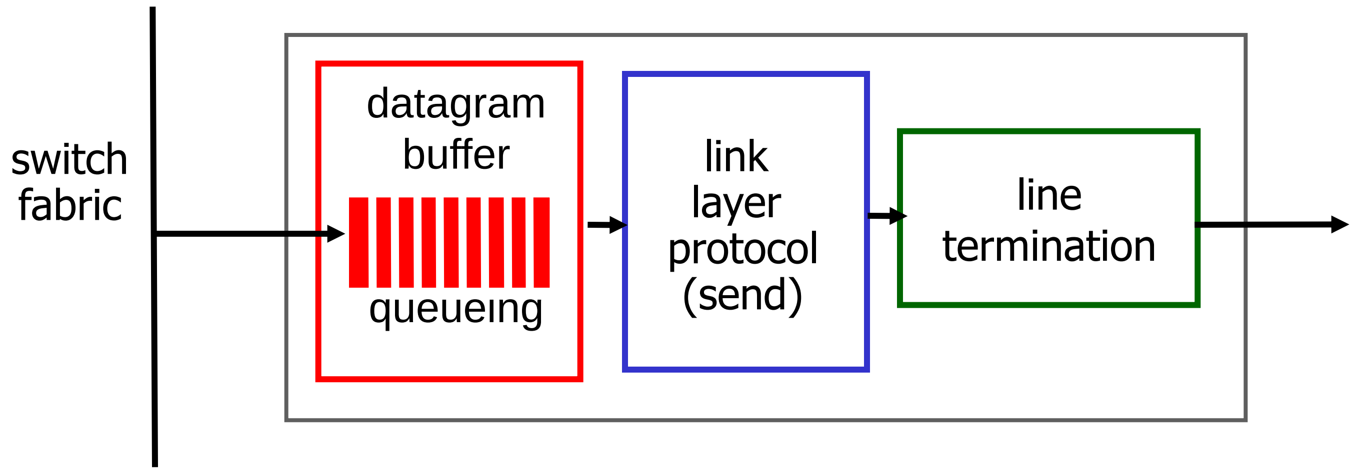
Banyan networks (*)





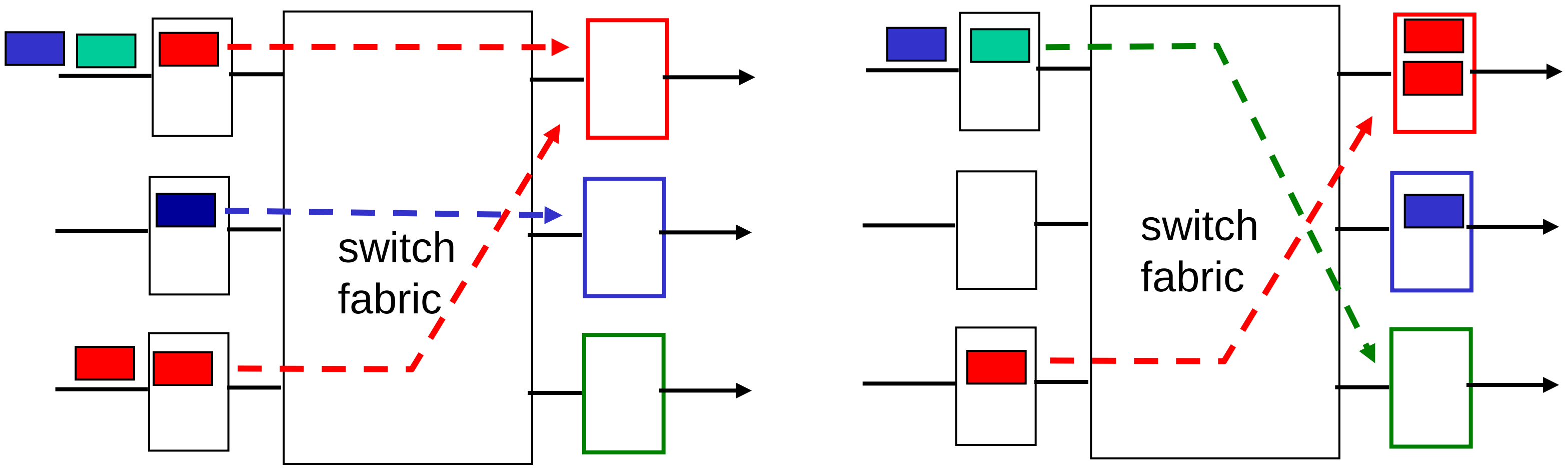


Output ports



- *buffering* required when datagrams arrive from fabric faster than the transmission rate
- *scheduling discipline* chooses among queued datagrams for transmission

Output port queueing



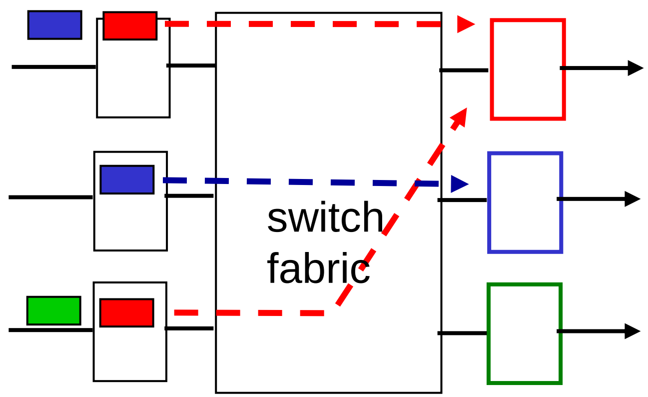
at *t*, packets more from input to output

one packet time later

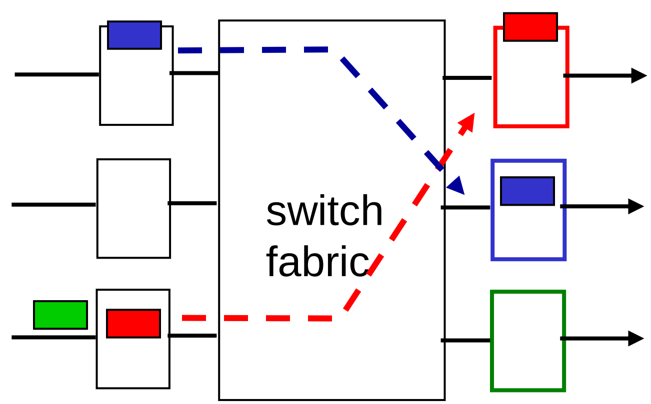
- buffering when arrival rate via switch exceeds output line speed
- *queueing (delay) and loss due to congestion and output port buffer overflow!*

Input port queuing

- fabric slower than input ports combined -> queueing may occur at input queues
  - *delay and loss due to input buffer overflow!*
- Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward

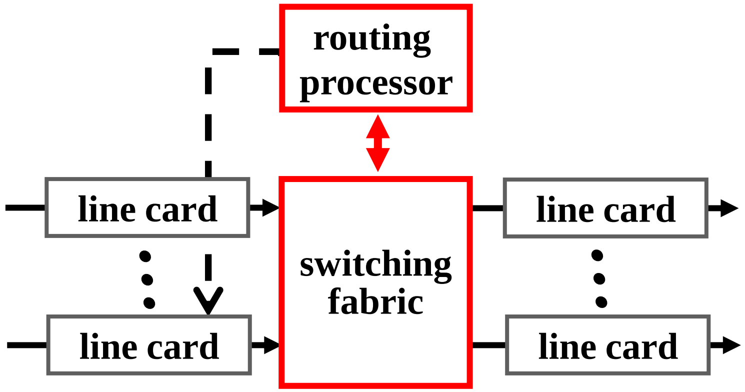


output port contention:
only one red packet can be transferred
(lower red packet is blocked)



one packet time later:
green packet experiences
HOL blocking

Routing Processor

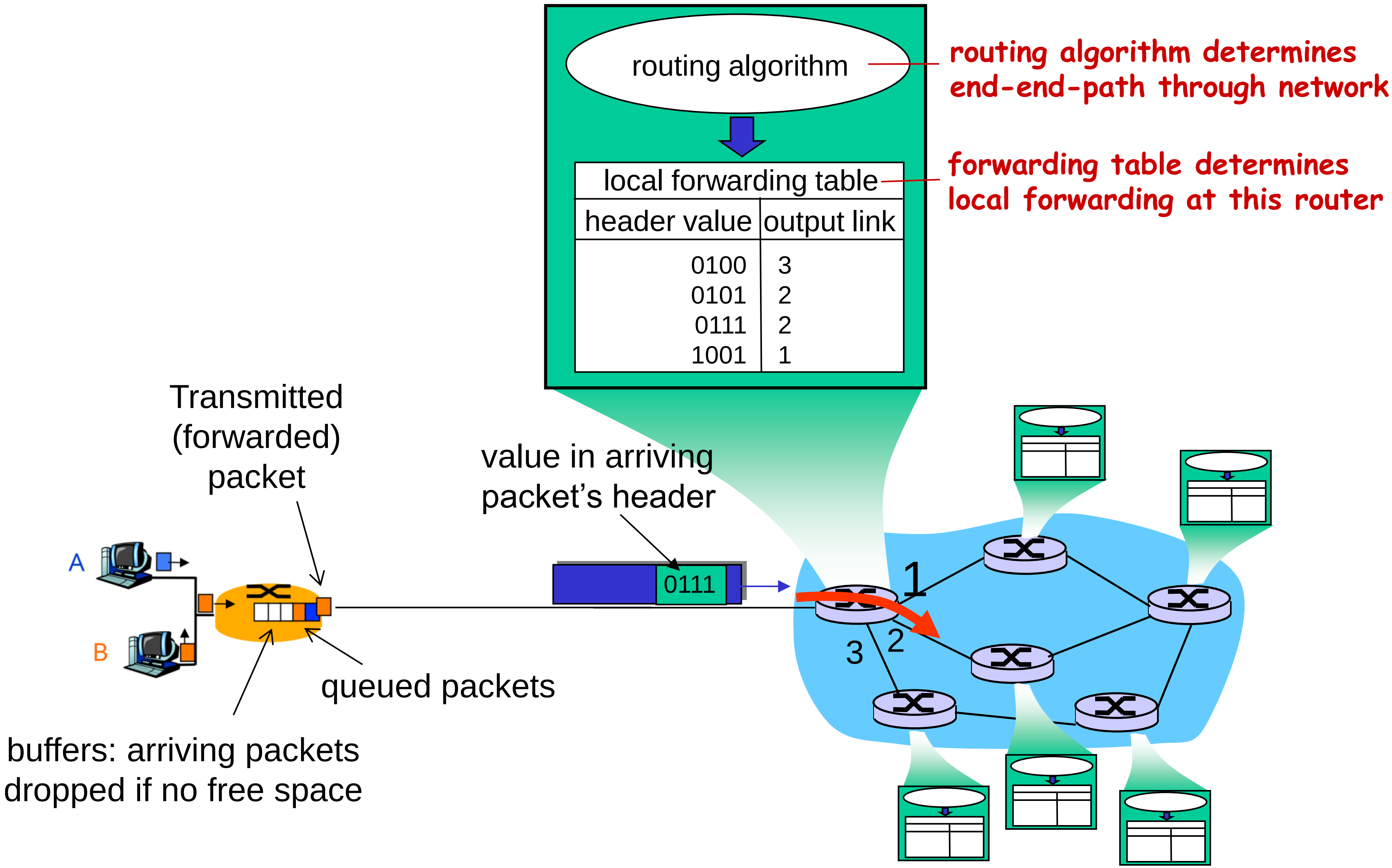


- "loopback" interface
  - IP address of the CPU on the router
- "Control-Plane" software
  - implementation of routing protocols
  - creation of forwarding table for the line cards
- Handling of special data packets
  - packets with IP options enabled
  - packets with expired Time-to-Live
- Network management functions
  - command-line interface (CLI) for configuration
  - Transmission of measurement statistics

Routing vs. Forwarding

- Routing: control plane
  - Computing paths the packets will follow
  - Routers talking amongst themselves
  - Creating the forwarding tables
- Forwarding: data plane
  - Directing a data packet to an outgoing link
  - Using the forwarding tables

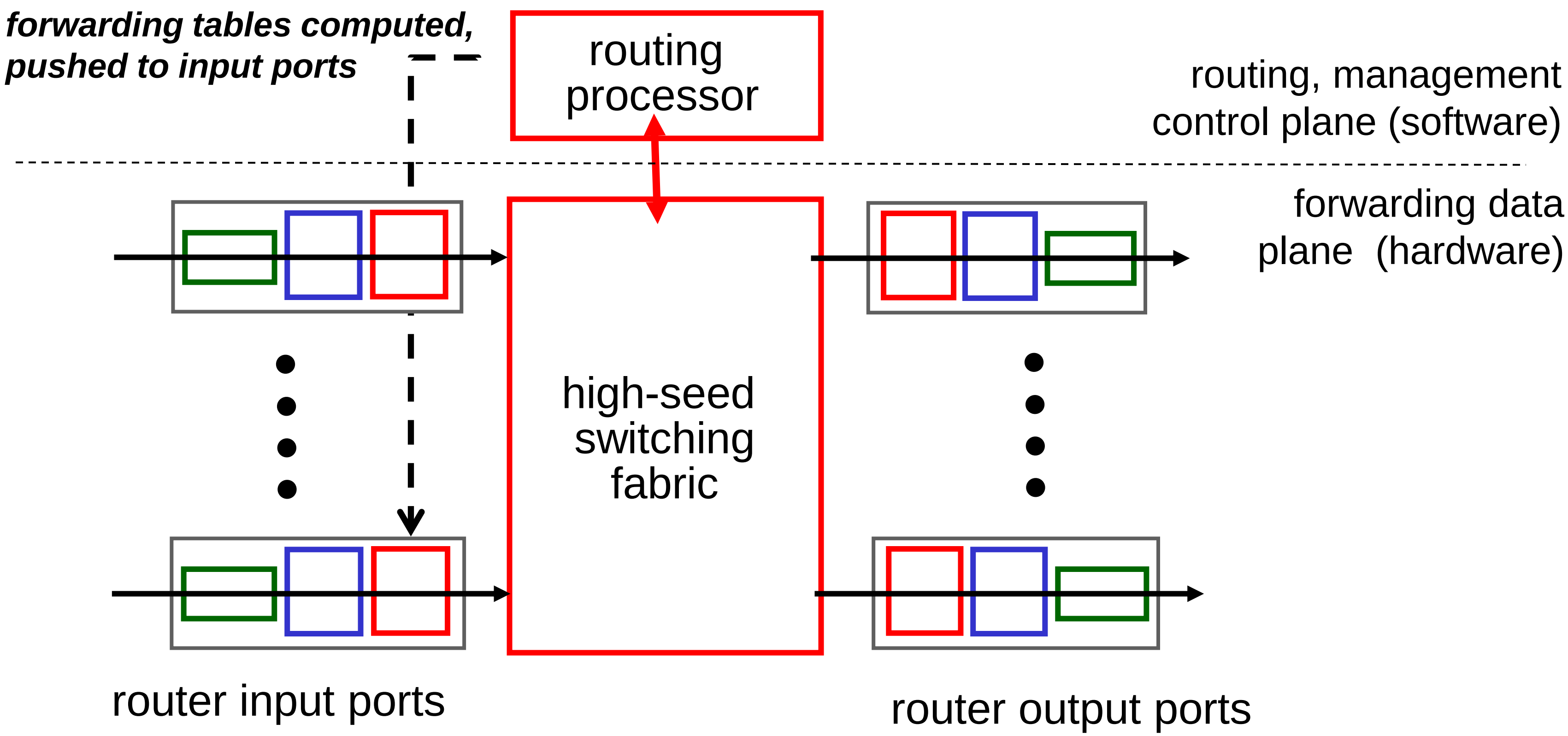
Interplay between routing and forwarding



Router architecture overview

two key router functions:

- run routing algorithms/protocol (RIP, OSPF, BGP)
- *forwarding* datagrams from incoming to outgoing link

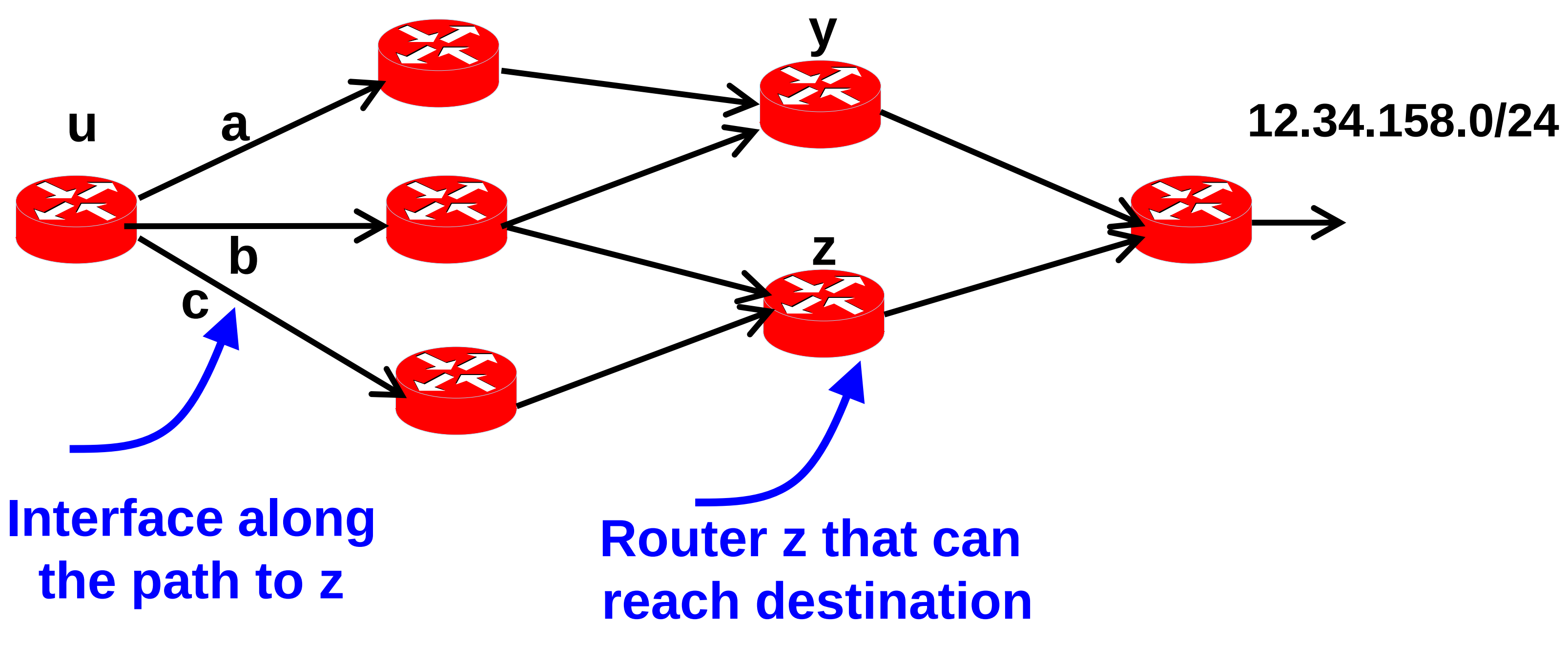


Goal of routing

- You may be able to send datagrams directly to the destination
- If not, the router attempts to send datagrams to a router that is nearer the destination.
- The *goal* of a routing protocol is very simple: Find a "good" path from source to destination.
  - Hosts often have default routers/gateways
  - We can focus on the routing from the source router to the destination router

Computing Paths between Routers

- Routers need to know two things
  - which router to use to reach a destination?
  - which interface to use to reach that router?

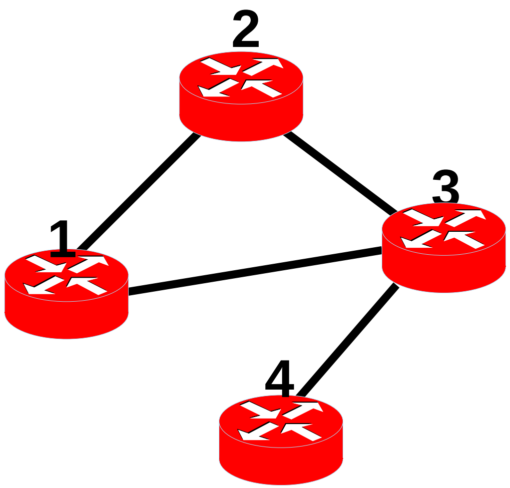


Routing in the Internet: Example

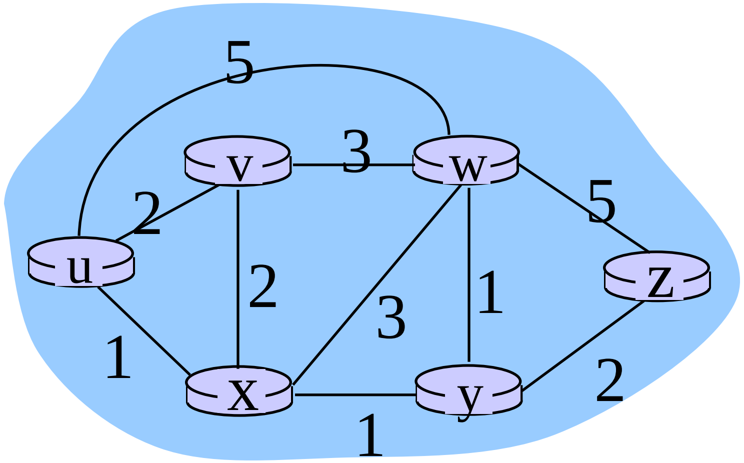
- How does a router construct its forwarding table?
- How does a router know which is the next hop towards a destination?
- Use a routing protocol to propagate (and update) reachability information

Router 1's table

Destination	Next Hop
2	2
3	3
4	3



Graph abstraction



Graph: G = (N,E)

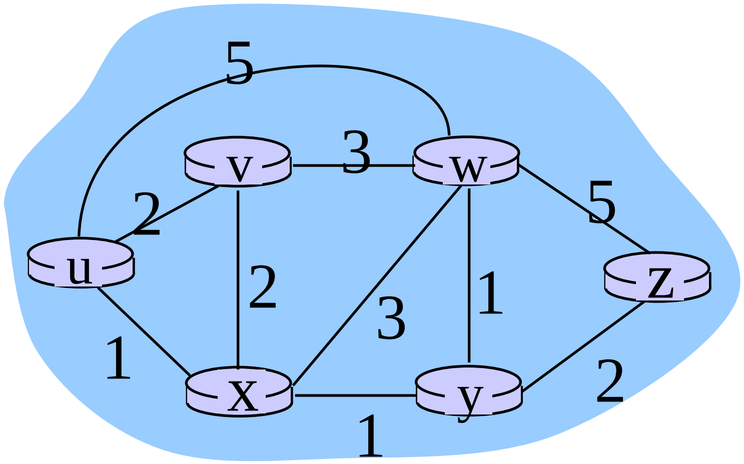
N = set of routers = { u, v, w, x, y, z }

E = set of links ={ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }

Remark: Graph abstraction is useful in other network contexts

Example: P2P, where N is set of peers and E is set of TCP connections

Graph abstraction: costs



• $c(x,x')$ = cost of link (x,x')

- e.g., $c(w,z) = 5$

• cost could always be 1, or inversely related to bandwidth, or inversely related to congestion

Cost of path $(x_1, x_2, x_3,\ldots, x_p) = c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$

Question: What's the least-cost path between u and z ?

Routing algorithm: algorithm that finds least-cost path

Routing algorithm classification

Q: global or decentralized information?

global:

- all routers have complete topology, link cost info
- "link state" algorithms

decentralized:

- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- "distance vector" algorithms

Q: static or dynamic?

static:

- routes change slowly over time

dynamic:

- routes change more quickly
  - periodic update
  - in response to link cost changes

A Link-State Routing Algorithm

Dijkstra's algorithm

- net topology, link costs known to all nodes
  - accomplished via "link state broadcast"
  - all nodes have same info
- computes least cost paths from one node ('source') to all others
  - gives *forwarding table* for that node
- after k iterations, know least-cost path to k destinations

notation:

- c(x,y): link cost from node x to y; = ∞ if not direct neighbors
- D(v): current value of cost of path from source to destination v
- p(v): predecessor node along path from source to v
- N': set of nodes whose least cost path definitively known

Dijkstra's Algorithm

Initialization:
N' = {u}
for all nodes v
if v adjacent to u
then D(v) = c(u,v)
else D(v) = ∞

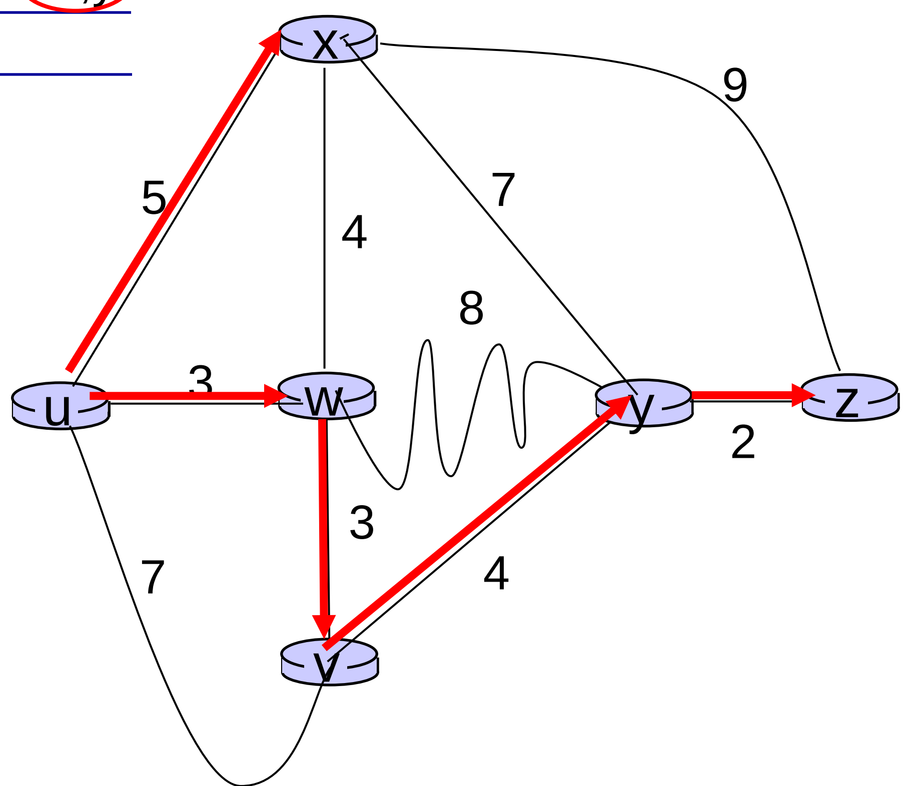
Loop
find w not in N' such that D(w) is a minimum
add w to N'
update D(v) for all v adjacent to w and not in N' :
D(v) = min(D(v), D(w) + c(w,v))
/* new cost to v is either old cost to v or known
shortest path cost to w plus cost from w to v */
until all nodes in N'

Dijsktra's algorithm: example

Step	N'	D(v) p(v)	D(w) p(w)	D(x) p(x)	D(y) p(y)	D(z) p(z)
0	u	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					

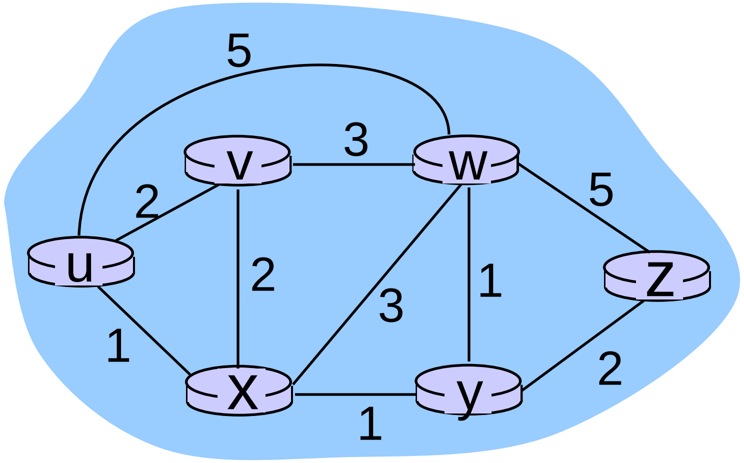
notes:

- construct shortest path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)



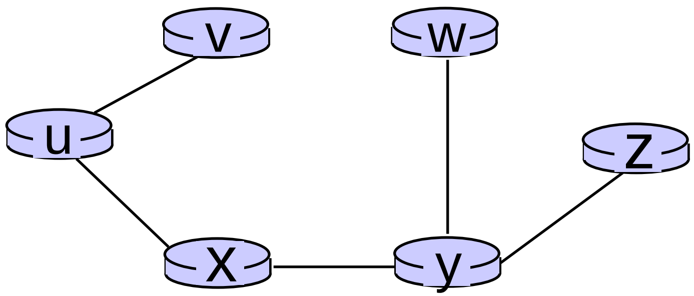
Dijkstra's algorithm: example 2

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



Dijkstra's algorithm: example 2

resulting shortest-path tree from u:



resulting forwarding table in u:

destination	link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

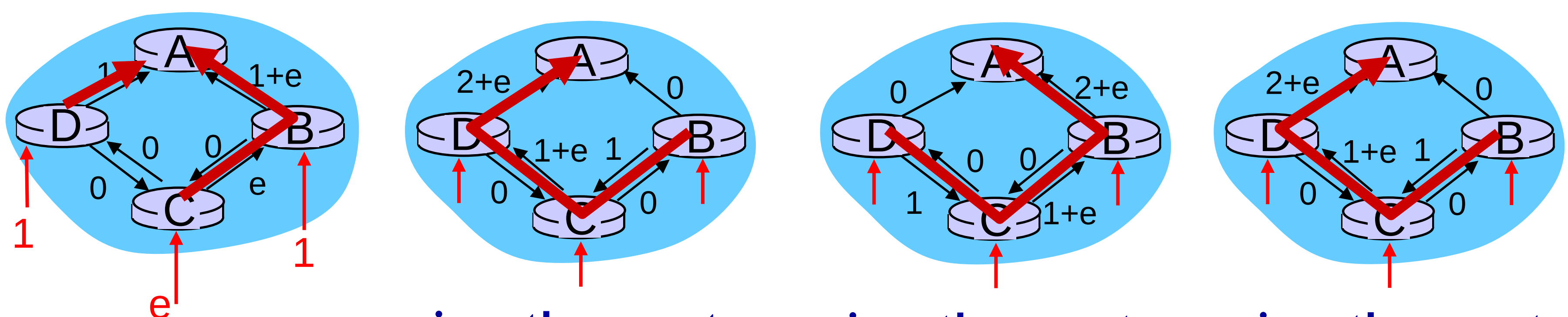
Dijkstra's algorithm, discussion

algorithm complexity: n nodes

- each iteration: need to check all nodes, w, not in N
- n(n+1)/2 comparisons: $O(n^2)$
- more efficient implementations possible: O(nlogn)

oscillations possible:

- e.g., suppose link cost = amount of carried traffic:



initially

given these costs, find new routing.... ➔ new costs

given these costs, find new routing.... ➔ new costs

given these costs, find new routing.... ➔ new costs

Distance Vector (DV) algorithm

- distributed, iterative, and asynchronous
  - receives info from directly attached neighbors
  - continues on until no more info is exchanged
  - nodes can update and send info asynchronously
- $D_x(y)$ = estimate of least cost from x to y
  - x maintains distance vector $\mathbf{D}_x = [D_x(y): y \in N]$
- For node x:
  - knows cost to each neighbor v: $c(x,v)$
  - also maintains its neighbors' DVs. For each neighbor v, x maintains $\mathbf{D}_v = [D_v(y): y \in N]$

Distance Vector Algorithm

Bellman-Ford Equation (dynamic programming)

Define

$d_x(y)$:= cost of least-cost path from x to y

Then

cost to neighbor v

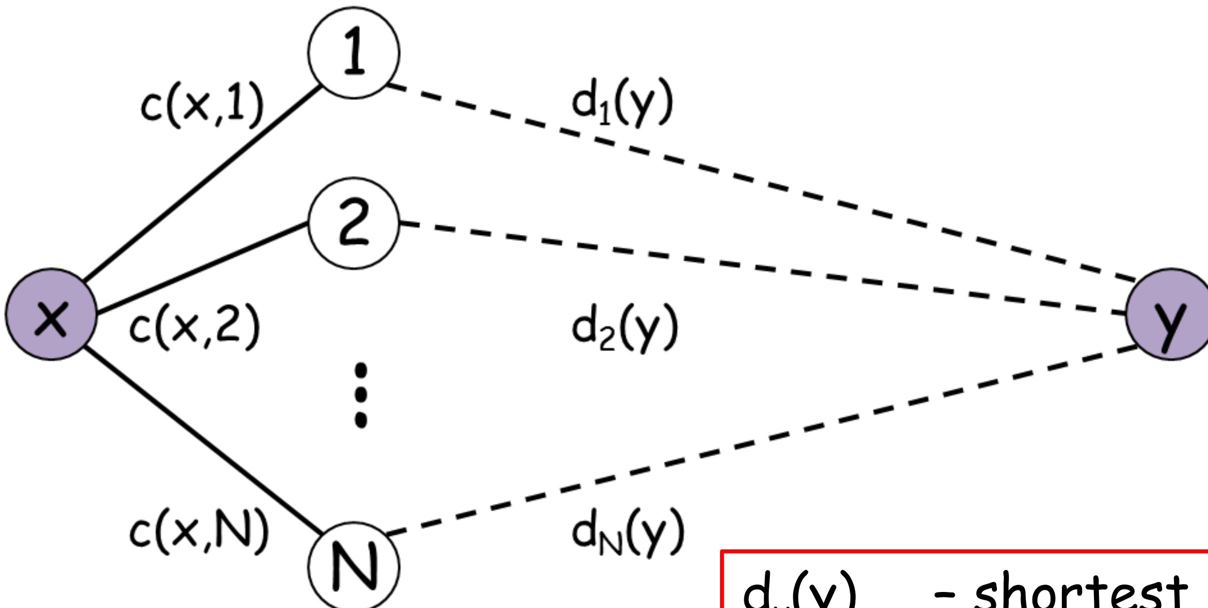
$$\boxed{d_x(y) = \min_v \{ c(x,v) + d_v(y) \}}$$

min taken over all neighbors v of x

cost from neighbor v to destination y

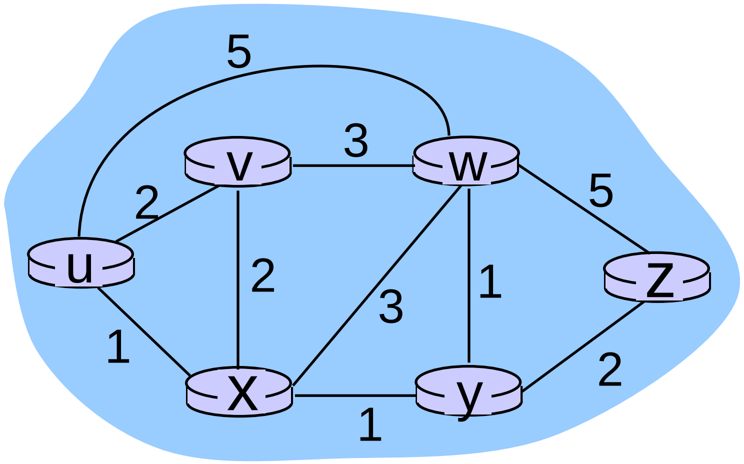
The fact behind Bellman-Ford

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



$d_v(y)$	- shortest distance between v and y
$c(x,v)$	- cost between x and v
N	- number of neighbor of x

Bellman-Ford example



clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

$$d_u(z) = \min \{ c(u,v) + d_v(z),\ c(u,x) + d_x(z),\ c(u,w) + d_w(z) \}$$
$$= \min \{ 2 + 5,\ 1 + 3,\ 5 + 3 \} = 4$$

- Practical contributions of B-F equation:
  - node achieving minimum is next hop in the shortest path, used in forwarding table
  - it suggests the form of the neighbor-to-neighbor communication used in the DV algorithm

Distance vector algorithm

key idea:

- from time-to-time, each node sends its own distance vector estimate to neighbors
- when x receives new DV estimate from neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow \min_v \{ c(x,v) + D_v(y) \} \quad \text{for each } y \in N$$

- under minor, natural conditions, the estimate Dx(y) converge to the actual least cost dx(y)

Distance vector algorithm

iterative, asynchronous:
each local iteration caused by
- local link cost change
- DV update message from neighbor

distributed:
- each node notifies neighbors *only* when its DV changes
  - neighbors then notify their neighbors if necessary

at each node:

wait for (change in local link cost or msg from neighbor)

↓

recompute estimates

↓

if DV to any destination has changed, *notify* neighbors

$$D_x(y) = \min\{c(x,y) + D_y(y),\ c(x,z) + D_z(y)\}$$
$$= \min\{2+0\ ,\ 7+1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z),\ c(x,z) + D_z(z)\}$$
$$= \min\{2+1\ ,\ 7+0\} = 3$$

node x table

from \ *cost to*	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

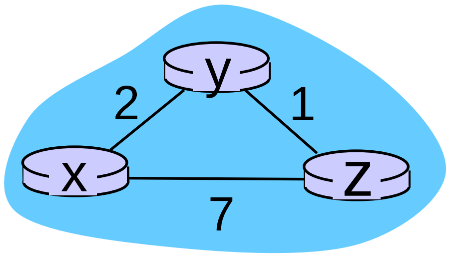
from \ *cost to*	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

node y table

from \ *cost to*	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

node z table

from \ *cost to*	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0



time

$D_x(y) = \min\{c(x,y) + D_y(y),\ c(x,z) + D_z(y)\}$
$= \min\{2+0\ ,\ 7+1\} = 2$

$D_x(z) = \min\{c(x,y) +$
$D_y(z),\ c(x,z) + D_z(z)\}$
$= \min\{2+1\ ,\ 7+0\} = 3$

node x table

from \ cost to	x	y	z
x	0	2	7
y	∞	∞	∞
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node y table

from \ cost to	x	y	z
x	∞	∞	∞
y	2	0	1
z	∞	∞	∞

from \ cost to	x	y	z
x	0	2	7
y	2	0	1
z	7	1	0

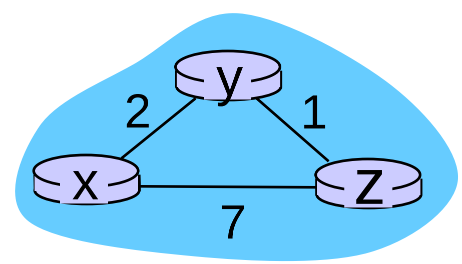
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node z table

from \ cost to	x	y	z
x	∞	∞	∞
y	∞	∞	∞
z	7	1	0

from \ cost to	x	y	z
x	0	2	7
y	2	0	1
z	3	1	0

from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

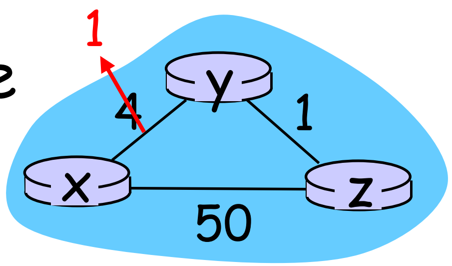


time

Distance vector: link cost changes

link cost changes:

- node detects local link cost decrease
- updates routing info, recalculates distance vector
- if DV changes, notify neighbors



"good news travels fast"

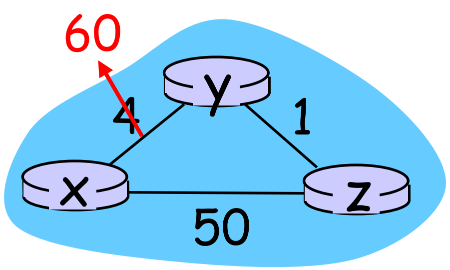
t_0 : *y* detects link-cost change, updates its DV, informs its neighbors.

t_1 : *z* receives update from *y*, updates its table, computes new least cost to *x* , sends its neighbors its DV.

t_2 : *y* receives *z*'s update, updates its distance table. *y*'s least costs do *not* change, so *y* does *not* send a message to *z*.

Distance vector: link cost changes

link cost changes:



- node detects local link cost increase
- $D_y(x) = 4$, $D_y(z) = 1$, $D_z(y) = 1$, and $D_z(x) = 5$
- $D_y(x) = \min\{c(y,x) + D_x(x), c(y,z) + D_z(x)\}$
  $= \min\{60 + 0, 1 + 5\} = 6$
- $D_z(x) = \min\{50 + 0, 1 + 6\} = 7$
- *bad news travels slow* - “count to infinity” problem!

poisoned reverse:

- If Z routes through Y to get to X :
  - Z tells Y its (Z’s) distance to X is infinite (so Y won’t route to X via Z)
- will this completely solve count to infinity problem?

Comparison of LS and DV algorithms

Message complexity

- LS: with n nodes, E links, O(nE) msgs sent; talk to all nodes, only about direct links
- DV: exchange between neighbors only, but about least-distance to all nodes
  - convergence time varies

Speed of Convergence

- LS: $O(n^2)$ algorithm requires O(nE) msgs
  - may have oscillations
- DV: convergence time varies
  - may be routing loops
  - count-to-infinity problem

Robustness: what if router malfunctions?

- LS: node can advertise incorrect *link* cost;
  each computes only its *own* table
- DV: node can advertise incorrect *path* cost;
  each table used by others (error propagate thru network)

Hierarchical routing

our routing study thus far - idealization
- all routers identical
- network "flat"

... *not* true in practice

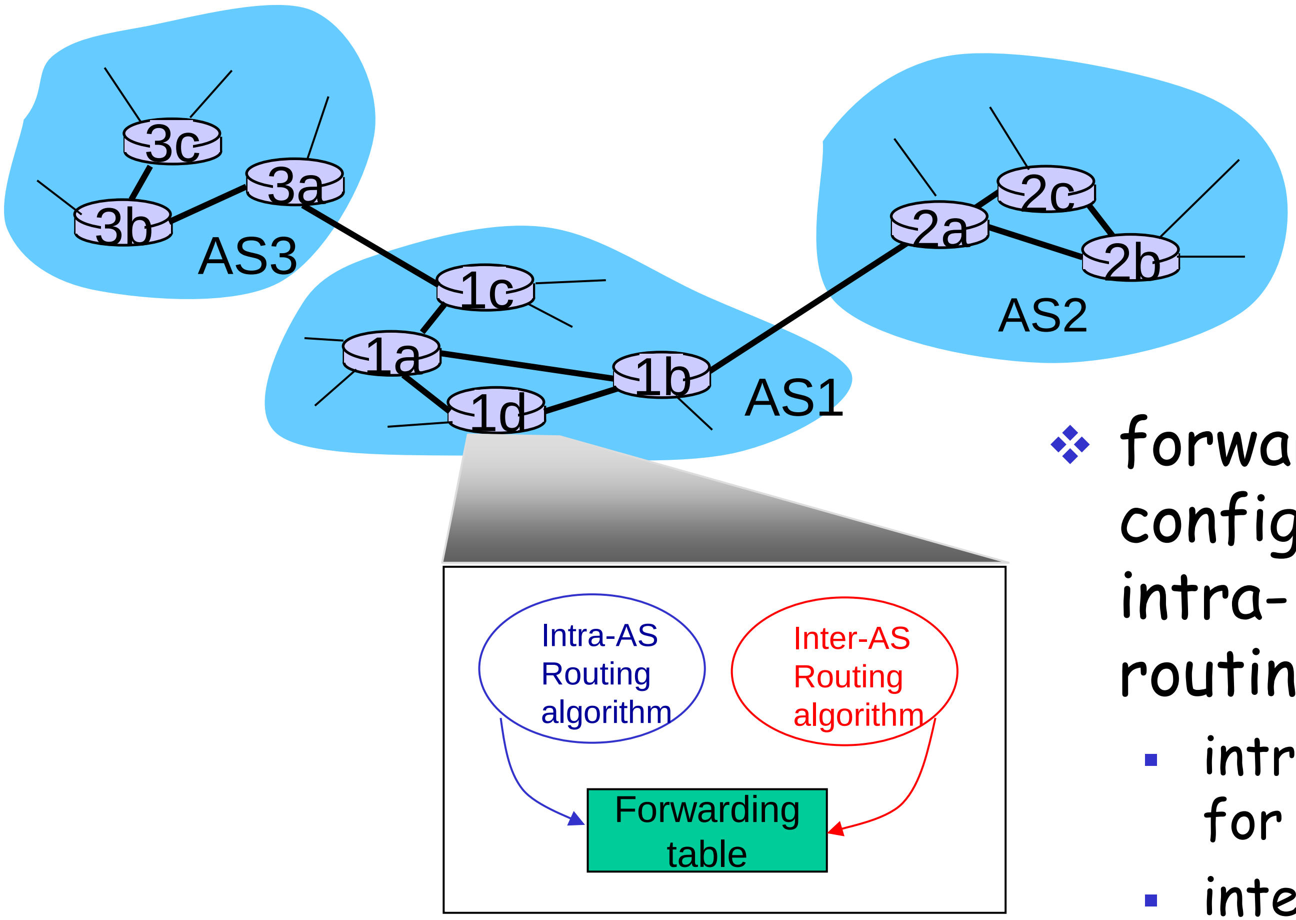
scale: with 600 million destinations:
- can't store all dest's in routing tables!
- routing table exchange would swamp links!

administrative autonomy
- internet = network of networks
- each admin may want to control routing in its own network, or hide topology

Hierarchical routing

- aggregate routers into regions,
  - "autonomous systems" (AS)
- routers in same AS run same protocol
  - "intra-AS" routing protocol
  - routers in different AS can run different intra-AS routing protocol
- *gateway router:*
  - at "edge" of its own AS
  - has link to router in another AS

Interconnected ASes



- forwarding table configured by both intra- and inter-AS routing algorithm
  - intra-AS sets entries for internal dests
  - inter-AS & intra-AS sets entries for external dests

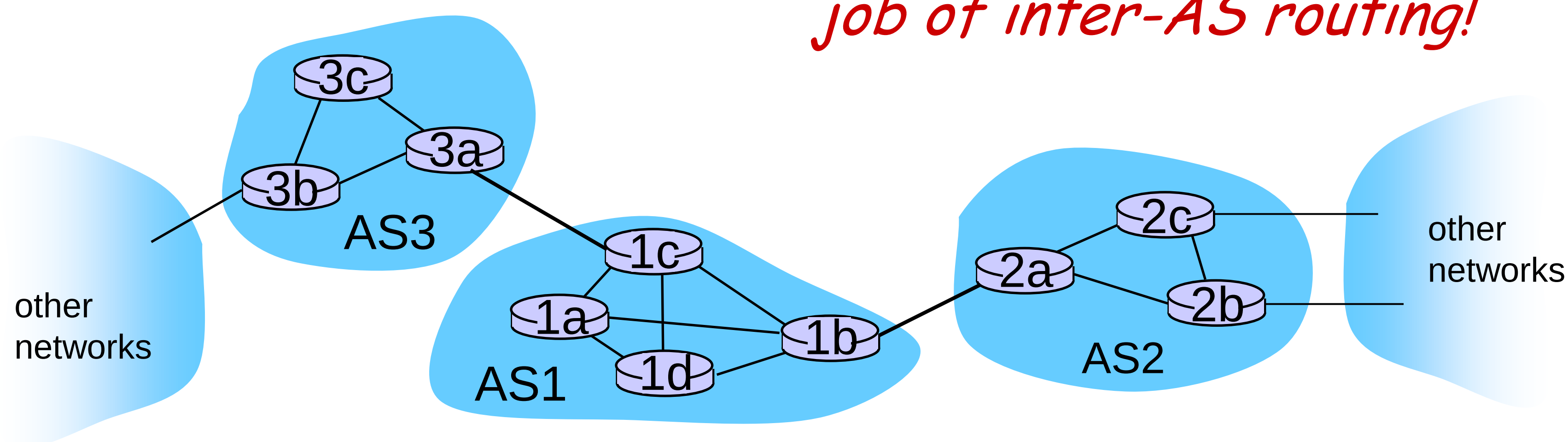
Inter-AS tasks

- suppose router in AS1 receives datagram destined outside of AS1
  - router should forward packet to gateway router, but which one?

AS1 must:

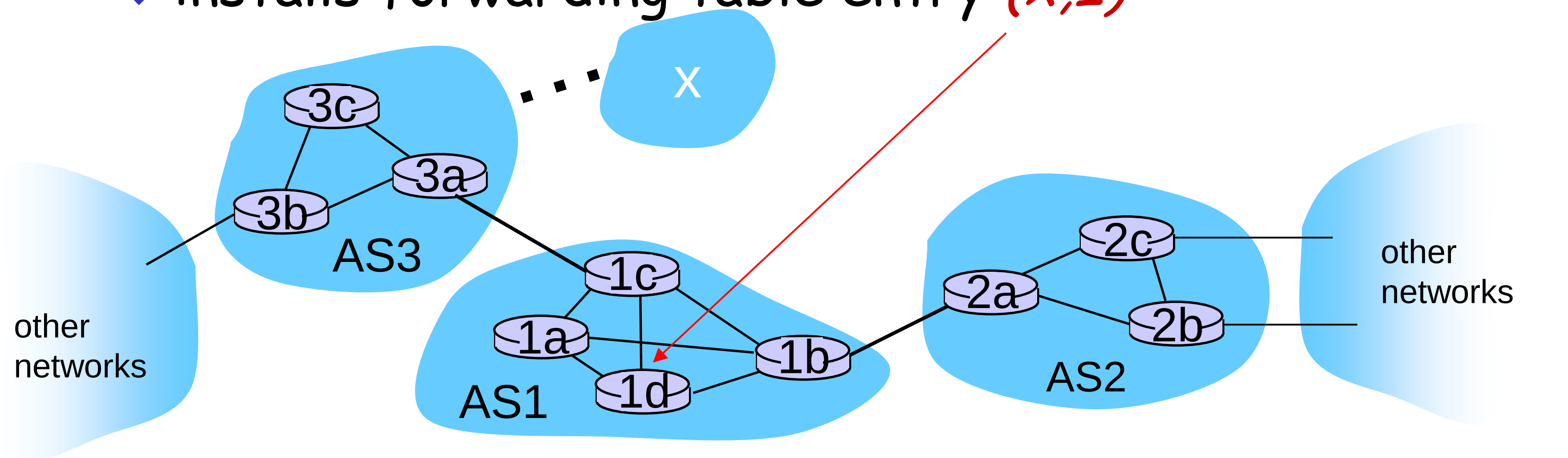
1. learn which dests are reachable through AS2, which through AS3
2. propagate this reachability info to all routers in AS1

job of inter-AS routing!



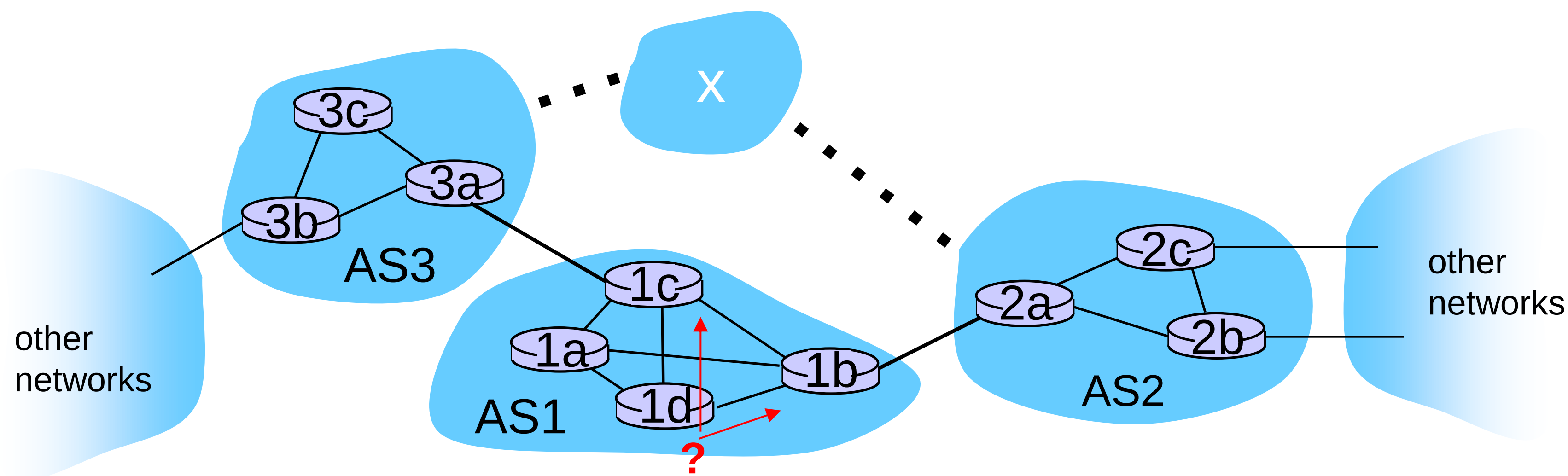
Set forwarding table in router 1d

- suppose AS1 learns (via inter-AS protocol) that subnet *x* reachable via AS3 (gateway 1c), but not via AS2
  - inter-AS protocol propagates reachability info to all internal routers
- router 1d determines from intra-AS routing info that its interface *I* is on the least cost path to 1c
  - installs forwarding table entry *(x,I)*



Example: choosing among multiple ASes

- now suppose AS1 learns from inter-AS protocol that subnet *x* is reachable from AS3 *and* from AS2.
- to configure forwarding table, router 1d must determine which gateway it should forward packets towards for dest x
  - this is also job of inter-AS routing protocol!



Example: choosing among multiple ASes

- now suppose AS1 learns from inter-AS protocol that subnet *x* is reachable from AS3 *and* from AS2.
- to configure forwarding table, router 1d must determine which gateway it should forward packets towards for dest *x*
  - this is also job of inter-AS routing protocol!
- *hot potato routing policy: send* packet towards closest of two routers.

learn from inter-AS protocol that subnet *x* is reachable via multiple gateways → use routing info from intra-AS protocol to determine costs of least-cost paths to each of the gateways → hot potato routing: choose the gateway that has the smallest least cost → determine from forwarding table the interface *I* that leads to least-cost gateway. Enter *(x,I)* in forwarding table