

# **Introduction to Linux Desktop and Command Line**

**11 Jan 2022**

Abhijit A. M.  
[abhijit.comp@coep.ac.in](mailto:abhijit.comp@coep.ac.in)

# **Why GNU/Linux ?**

## Why GNU/Linux ?

1. Programmer's Paradise : most versatile, vast, all pervasive programming environment
2. Free Software ( or Open Source?) : Free as in freedom. *Freely* Use, copy, modify, redistribute.
3. Highly Productive : Do more in less time
4. Better quality, more secure, very few crashes

## Why Command Line ?

1. Not for everyone ! Absolutely !
2. Those who do it are way more productive than others
3. Makes you think !
4. Portable. Learn once, use everywhere on all Unixes, Linuxes, etc.

# Few Key Concepts

- ***Files don't open themselves***
  - Always some application/program open()s a file.
- ***Files don't display themselves***
  - A file is displayed by the program which opens it. Each program has its own way of handling files

# Few Key Concepts

- **Programs don't run themselves**
  - You click on a program, or run a command --> equivalent to request to Operating System to run it. The OS runs your program
- **Users (humans) request OS to run programs, using Graphical or Command line interface**
  - and programs open files

# Path names

- Tree like directory structure
- Root directory called /
- A complete path name for a file
  - /home/student/a.c
- Relative path names
  - concept: every running program has a *current working directory*
  - . current directory
  - .. parent directory
  - ./Desktop/xyz/../.c

# A command

- **Name of an executable file**
  - For example: 'ls' is actually “/bin/ls”
- **Command takes arguments**
  - E.g. `ls /tmp/`
- **Command takes options**
  - E.g. `ls -a`

# A command

- **Command can take both arguments and options**
  - E.g. `ls -a /tmp/`
- **Options and arguments are basically argv[] of the main() of that program**

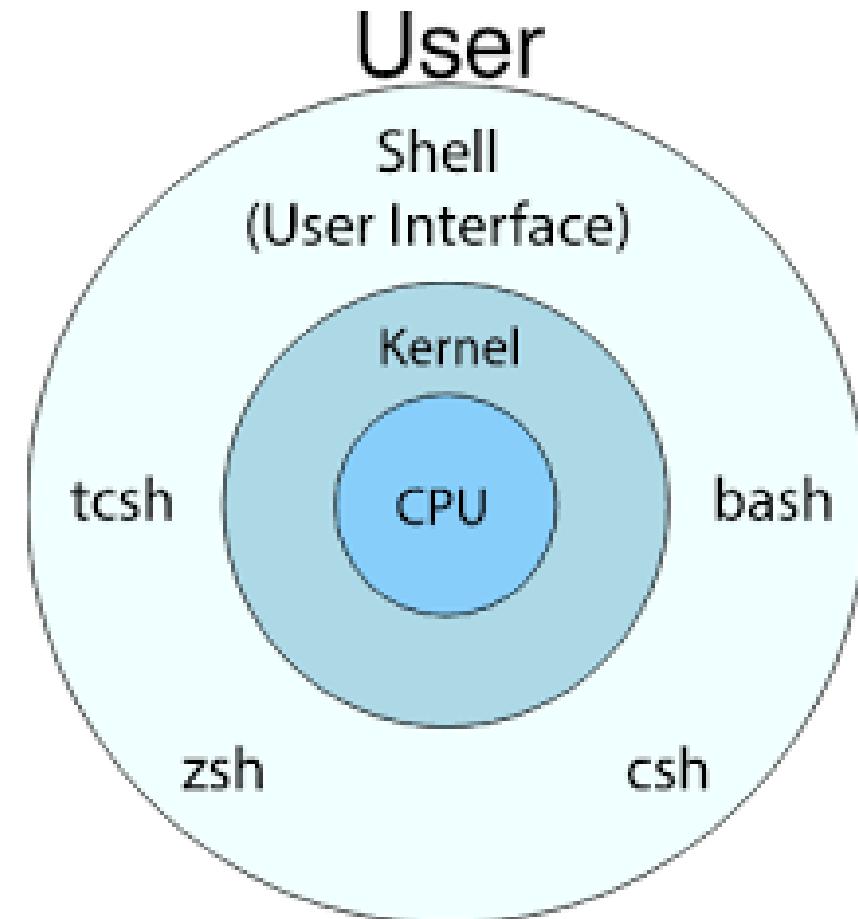
# Basic Navigation Commands

- `pwd`
- `ls`
  - `ls -l`
  - `ls -l /tmp/`
  - `ls -l /home/student/Desktop`
  - `ls -l ./Desktop`
  - `ls -a`
  - `\ls -F`
- `cd`
  - `cd /tmp/`
  - `cd`
  - `cd /home/student/Desktop`
- **notation: ~**
  - `cd ~`
  - `cd ~/Desktop`
  - `ls ~/Desktop`

Map these commands  
to navigation using a  
graphical file browser

# The Shell

- **Shell = Cover**
- **Covers some of the Operating System's "System Calls" (mainly fork+exec) for the Applications**
- **Talks with Users and Applications and does some talk with OS**



Not a very accurate diagram !

# The Shell

Shell waits for user's input

Requests the OS to run a program which the user  
has asked to run

Again waits for user's input

GUI is a Shell !

# Let's Understand fork() and exec()

```
#include <unistd.h>

int main() {
    fork();
    printf("hi\n");
    return 0;
}
```

```
#include <unistd.h>

int main() {
    printf("hi\n");
    execl("/bin/ls", "ls",
NULL);
    printf("bye\n");
    return 0;
}
```

# A simple shell

```
#include <stdio.h>
#include <unistd.h>
int main() {
    char string[128];
    int pid;
    while(1) {
        printf("prompt>");
        scanf("%s", string);
        pid = fork();
        if(pid == 0) {
            execl(string, string, NULL);
        } else {
            wait(0);
        }
    }
}
```

# File Permissions on Linux

- **Two types of users**
  - root and non-root
  - Users can grouped into 'groups'
- **3 sets of 3 permission**
  - Octal notation
  - Read = 4, Write = 2, Execute = 1
  - 644 means
    - Read-Write for owner, Read for Group, Read for others
- **chmod command uses these notations**
-

# File Permissions on Linux

-rw-r--r-- 1 abhijit abhijit 1183744 May 16 12:48 01\_linux\_basics.ppt

-rw-r--r-- 1 abhijit abhijit 341736 May 17 10:39 Debian Family Tree.svg

drwxr-xr-x 2 abhijit abhijit 4096 May 17 11:16 fork-exec

-rw-r--r-- 1 abhijit abhijit 7831341 May 11 12:13 foss.odp

3 sets of 3 permissions

3 sets = user (owner),  
group, others

3 permissions = read,  
write, execute

Owner

size

name

last-modification

hard link count

# File Permissions on Linux

- **r on a file : can read the file**
  - `open(..., O_RDONLY)` works
- **w on a file: can modify the file**
- **x on a file: can ask the os to run the file as an executable program**
- **r on a directory: can do 'ls'**
- **w on a directory: can add/remove files from that directory (even without 'r'!)**
- **x on a directory: can 'cd' to that directory**

# Access rights examples

- rw-r--r--

Readable and writable for file owner, only readable for others

- rw-r----

Readable and writable for file owner, only readable for users belonging to the file group.

- drwx-----

Directory only accessible by its owner

- r-x

File executable by others but neither by your friends nor by yourself.  
Nice protections for a trap...

# Man Pages

- **Manpage**
  - \$ man ls
  - \$ man 2 mkdir
  - \$ man man
  - \$ man -k mkdir
- **Manpage sections**
  - **1 User-level cmds and apps**
    - /bin/mkdir
  - **2 System calls**
    - int mkdir(const char \*, ...);
  - **3 Library calls**
    - int printf(const char \*, ...);
- **4 Device drivers and network protocols**
  - /dev/tty
- **5 Standard file formats**
  - /etc/hosts
- **6 Games and demos**
  - /usr/games/fortune
- **7 Misc. files and docs**
  - man 7 locale
- **8 System admin. Cmds**
  - /sbin/reboot

# GNU / Linux filesystem structure

Not imposed by the system. Can vary from one system to the other, even between two GNU/Linux installations!

/	Root directory
/bin/	Basic, essential system commands
/boot/	Kernel images, initrd and configuration files
/dev/	Files representing devices <b>/dev/hda</b> : first IDE hard disk
/etc/	System and application configuration files
/home/	User directories
/lib/	Basic system shared libraries

# GNU / Linux filesystem structure

<b>/lost+found</b>	Corrupt files the system tried to recover
<b>/media</b>	Mount points for removable media: <b>/media/usbdisk, /media/cdrom</b>
<b>/mnt/</b>	Mount points for temporarily mounted filesystems
<b>/opt/</b>	Specific tools installed by the sysadmin <b>/usr/local/</b> often used instead
<b>/proc/</b>	Access to system information <b>/proc/cpuinfo, /proc/version ...</b>
<b>/root/</b>	root user home directory
<b>/sbin/</b>	Administrator-only commands
<b>/sys/</b>	System and device controls (cpu frequency, device power, etc.)

# GNU / Linux filesystem structure

**/tmp/**

Temporary files

**/usr/**

Regular user tools (not essential to the system)  
**/usr/bin/, /usr/lib/, /usr/sbin...**

**/usr/local/**

Specific software installed by the sysadmin  
(often preferred to **/opt/**)

**/var/**

Data used by the system or system servers  
**/var/log/, /var/spool/mail** (incoming  
mail), **/var/spool/lpd** (print jobs)...

# Files: cut, copy, paste, remove,

- **cat <filenames>**
  - cat /etc/passwd
  - cat fork.c
  - cat <filename1> <filename2>
- **cp <source> <target>**
  - cp a.c b.c
  - cp a.c /tmp/
  - cp a.c /tmp/b.c
  - cp -r ./folder1 /tmp/
  - cp -r ./folder1 /tmp/folder2
- **mv <source> <target>**
  - mv a.c b.c
  - mv a.c /tmp/
  - mv a.c /tmp/b.c
- **rm <filename>**
  - rm a.c
  - rm a.c b.c c.c
  - rm -r /tmp/a
- **mkdir**
  - mkdir /tmp/a /tmp/b
- **rmdir**
  - rmdir /tmp/a /tmp/b

# Useful Commands

- **echo**
  - echo hi
  - echo hi there
  - echo "hi there"
  - j=5; echo \$j
- **sort**
  - sort
  - sort < /etc/passwd
- **firefox**
- **libreoffice**
- **grep**
  - grep bash /etc/passwd
  - grep -i display /etc/passwd
  - egrep -i 'a|b' /etc/passwd
- **less <filename>**
- **head <filename>**
  - head -5 <filename>
  - tail -10 <filename>

# Useful Commands

- **alias**  
`alias ll='ls -l'`
- **tar**  
`tar cvf folder.tar folder`
- **gzip**  
`gzip a.c`
- **touch**  
`touch xy.txt`  
`touch a.c`
- **strings**  
`strings a.out`
- **adduser**  
`sudo adduser test`
- **su**  
`su administrator`

# Useful Commands

- **df**  
    **df -h**
- **du**  
    **du -hs .**
- **bc**
- **time**
- **date**
- **diff**
- **wc**

# Network Related Commands

- **ifconfig**
- **ssh**
- **scp**
- **telnet**
- **ping**
- **w**
- **last**
- **whoami**

# Unix job control

- Start a background process:
  - gedit a.c &
  - gedit
    - hit ctrl-z*
  - **bg**
- Where did it go?
  - **jobs**
  - **ps**
- Terminate the job: kill it
  - **kill %jobid**
  - **kill pid**
- Bring it back into the foreground
  - **fg %1**

# Shell Wildcards

- **? (question mark)**
  - Any one character
  - ls a?c
  - ls ??c
- **\***
  - any number of characters
  - ls \*
  - ls d\*
  - echo \*
- **[ ]**
  - Matches a range
  - ls a[1-3].c
- **{}**
  - ls pic[1-3].{txt,jpg}

# Configuration Files

- Most applications have configuration files in TEXT format
- Most of them are in `/etc`
- `/etc/passwd` and `/etc/shadow`
  - Text files containing user accounts
- `/etc/resolv.conf`
  - DNS configuration
- `/etc/network/interfaces`
  - *Network configuration*
- `/etc/hosts`
  - Local database of Hostname-IP mappings
- `/etc/apache2/apache2.conf`
  - Apache webserver configuration

# `~/.bashrc` file

- **`~/.bashrc`**

**Shell script read each time a `bash` shell is started**

- **You can use this file to define**

- Your default environment variables (**`PATH`, `EDITOR`...**).
- Your aliases.
- Your prompt (see the **`bash`** manual for details).
- A greeting message.
- **Also `~/.bash_history`**

# Special devices (1)

## Device files with a special behavior or contents

- **/dev/null**

The data sink! Discards all data written to this file.

Useful to get rid of unwanted output, typically log information:

**mplayer black\_adder\_4th.avi &> /dev/null**

- **/dev/zero**

Reads from this file always return \b0 characters

Useful to create a file filled with zeros:

**dd if=/dev/zero of=disk.img bs=1k count=2048**

See **man null** or **man zero** for details

# Special devices (2)

- **/dev/random**

Returns random bytes when read. Mainly used by cryptographic programs. Uses interrupts from some device drivers as sources of true randomness (“entropy”).

Reads can be blocked until enough entropy is gathered.

- **/dev/urandom**

For programs for which pseudo random numbers are fine.

Always generates random bytes, even if not enough entropy is available (in which case it is possible, though still difficult, to predict future byte sequences from past ones).

See **man random** for details.

# Special devices (3)

- **/dev/full**

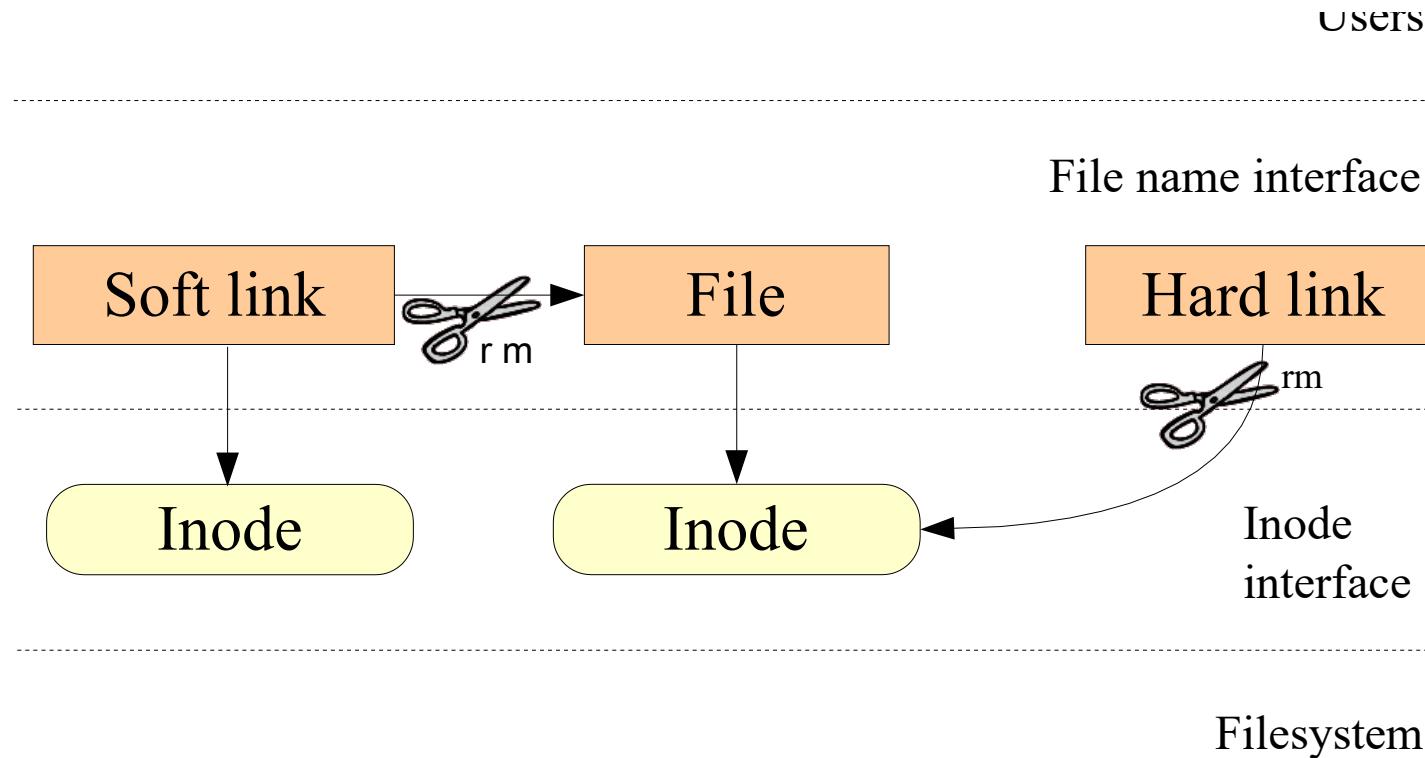
Mimics a full device.

Useful to check that your application properly handles this kind of situation.

See **man full** for details.

# Files names and inodes

## Hard Links Vs Soft Links



# Shell Programming

# Shell Programming

- **is “programming”**
- **Any programming: Use existing tool to create new utilities**
- **Shell Programming: Use shell facilities to create better utilities**
  - Know “commands” --> More tools
  - Know how to combine them

# Shell Variables

- **Shell supports variables**

- Try:
  - `j=5; echo $j`
  - No space in `j=5`
- Try
  - `set`
  - Shows all set variables
- Try
  - `a=10; b=20; echo $a$b`
- What did you learn?
- All variables are strings

# Shell's predefined variables

- **USER**
  - Name of current user
- **HOME**
  - Home directory of current \$USER
- **PS1**
  - The prompt
- **LINES**
  - No. of lines of the screen
- **HOSTNAME**
  - Name of the computer
- **OLDPWD**
  - Previous working directory
- **PATH**
  - List of locations to search for a command's executable file
- **\$?**
  - Return value of previous command

# Redirection

- **cmd > filename**
  - Redirects the output to a file
  - Try:
    - `ls > /tmp/xyz`
    - `cat /tmp/xyz`
    - `echo hi > /tmp/abc`
    - `cat /tmp/abc`
- **cmd < filename**
  - Reads the input from a file instead of keyboard
  - Think of a command now!

# Pipes

- Try
  - last | less
  - grep bash /etc/passwd | head - 1
  - grep bash /etc/passwd | head - 2 | tail -1
- Connect the output of LHS command as input of RHS command
- Concept of *filters* – programs which read input only from stdin (keyboard, e.g. scanf, getchar), and write output to stdout (e.g. printf, putchar)
- Programs can be connected using pipes if they are filters
- Most Unix/Linux commands are filters !

# The *test* command

- **test**
    - `test 10 -eq 10`
    - `test "10" == "10"`
    - `test 10 -eq 9`
    - `test 10 -gt 9`
    - `test "10" >= "9"`
    - `test -f /etc/passwd`
    - `test -d ~/desktop`
    - ...
- Shortcut notation for calling test**
- [ ]
- [ 10 -eq 10 ]
- Note the space after '[' and before ']'**

# The *expr* command and backticks

- **expr**
  - `expr 1 + 2`
  - `a=2; expr $a + 2`
  - `a=2; b=3; expr $a + $b`
  - `a=2;b=3; expr $a \* $b`
  - `a=2;b=3; expr $a | $b`
- **Used for mathematical calculations**
- **backticks ``**
  - `j=`ls`; echo $j`
  - `j=`expr 1 + 2`; echo $j`

# **if then else**

```
if [ $a -lt $b ]  
then  
    echo $a  
  
else  
    echo $b  
fi
```

```
if [ $a -lt $b ];then  
echo $a; else echo $b;  
fi
```

0	TRUE
Nonzero	FALSE

# while    do    done

```
while [ $a -lt $b ]
do
    echo $a
    a=`expr $a + 1`
done
```

```
while [ $a -lt $b ]; do
    echo $a;  a=`expr $a +
1`; done
```

```
while [ $a -lt $b ]
do
    echo $a
    a=$((a + 1))
done
```

**for x in ... do done**

**for i in {1..10}  
do  
echo \$i  
done**

**for i in \*; do echo \$i;  
done**

**for i in \*  
do  
echo \$i  
done**

read space

case \$space in

[1-6]\*)

Message="one to 6"

;;

[7-8]\*)

Message="7 or 8"

;;

9[1-8])

Message="9 with a number"

;;

\*)

Message="Default"

;;

esac

echo \$Message

**case ... esac**

**Syntax**

;;

) after option

\* for default

esac

# Try these things

- Print 3<sup>rd</sup> line from /etc/passwd, which contains the word bash
- Print numbers from 1 to 1000
- Create files named like this: file1, file2, file3, ... filen where n is read from user
  - Read i%5<sup>th</sup> file from /etc/passwd and store it in filei
- Find all files ending in .c or .h and create a .tar.gz file of these files
-

# The Golden Mantra

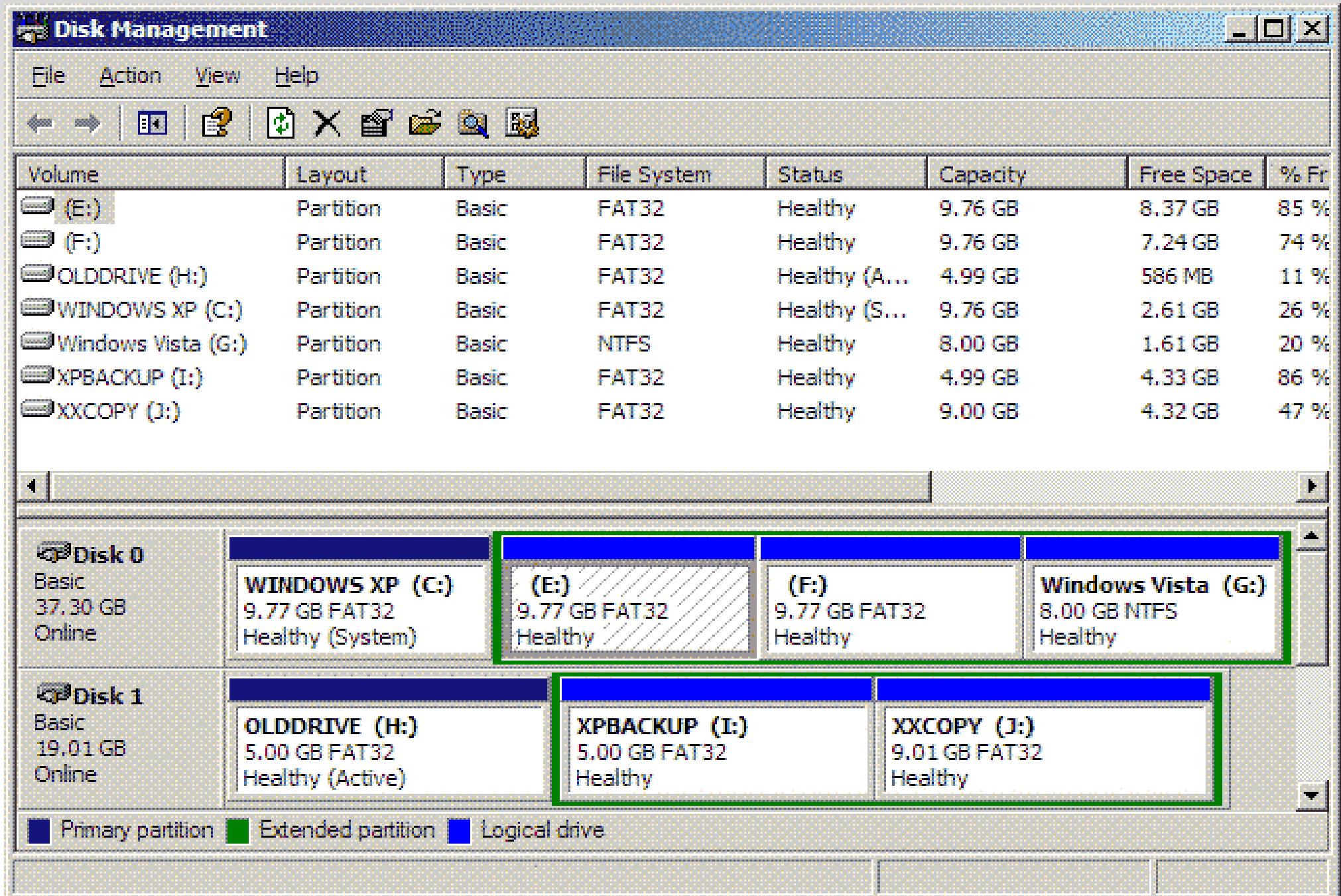
Everything can be done from command line !

Command line is far more powerful than graphical interface

Command line makes you a better programmer

# **Mounting**

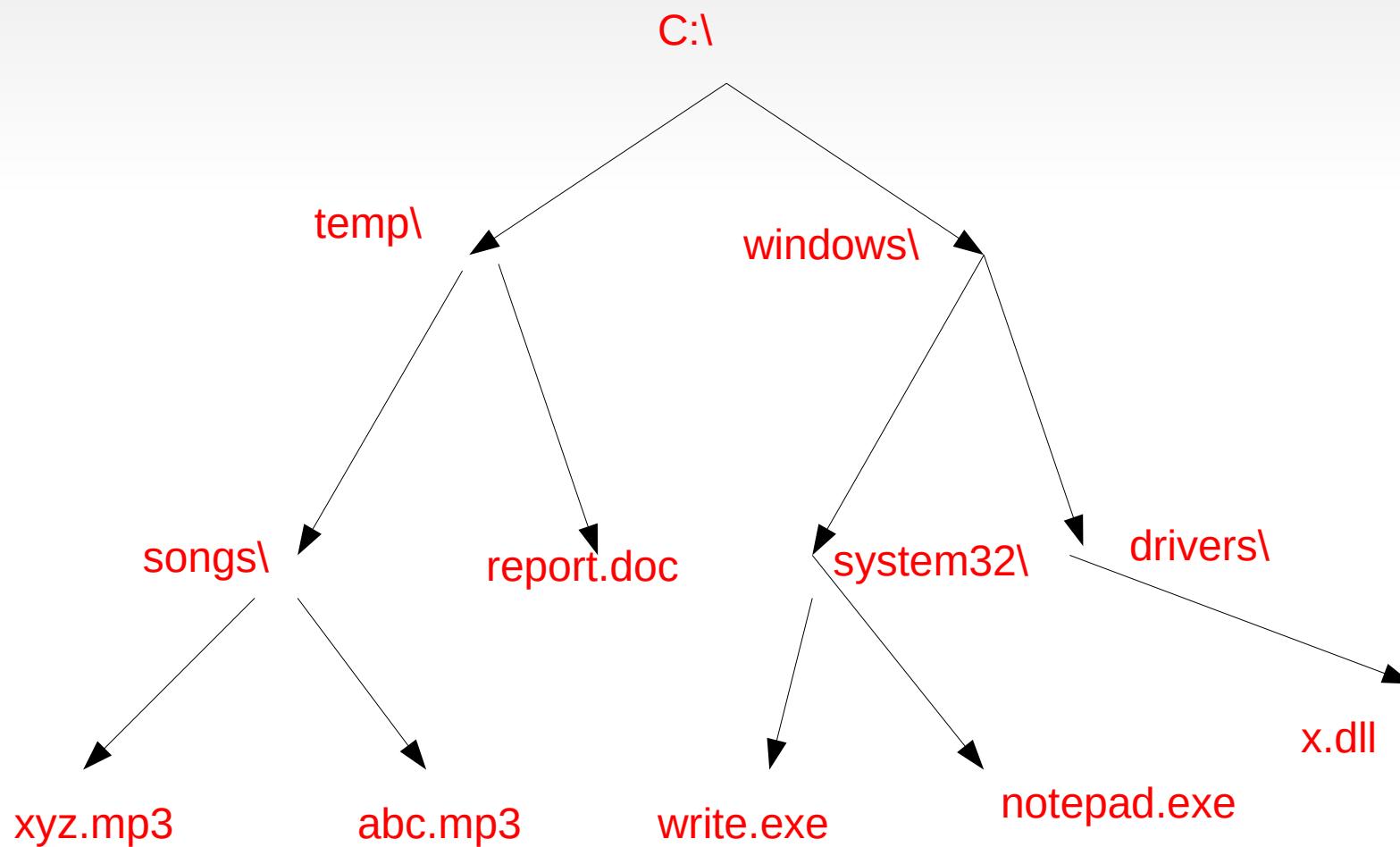
# Partitions



# Windows Namespace

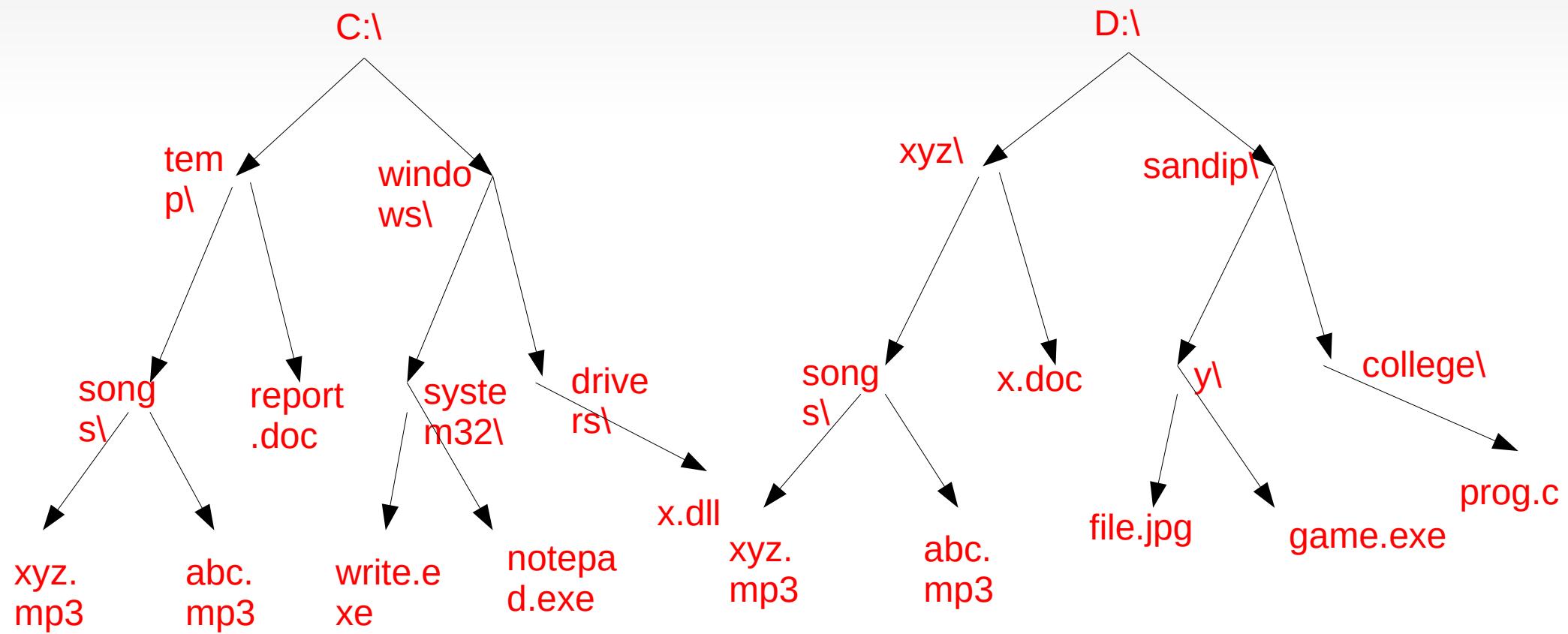
c:\temp\songs\xyz.mp3

- Root is C:\ or D:\ etc
- Separator is also “\”



# Windows Namespace

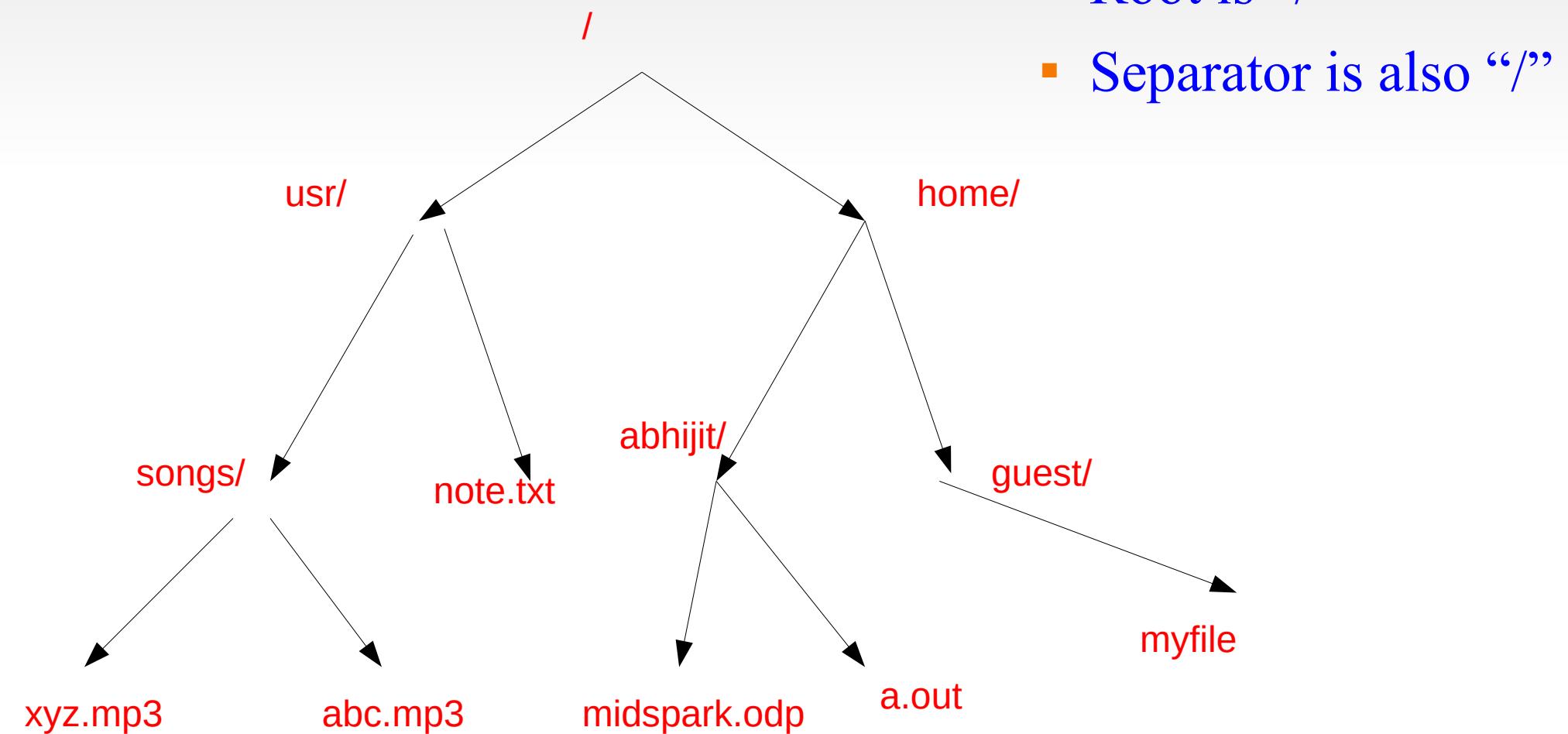
- C:\ D:\ Are partitions of the disk drive
- Typical convention: C: contains programs, D: contains data
- One “tree” per partition
  - Together they make a “forest”



# Linux Namespace: On a partition

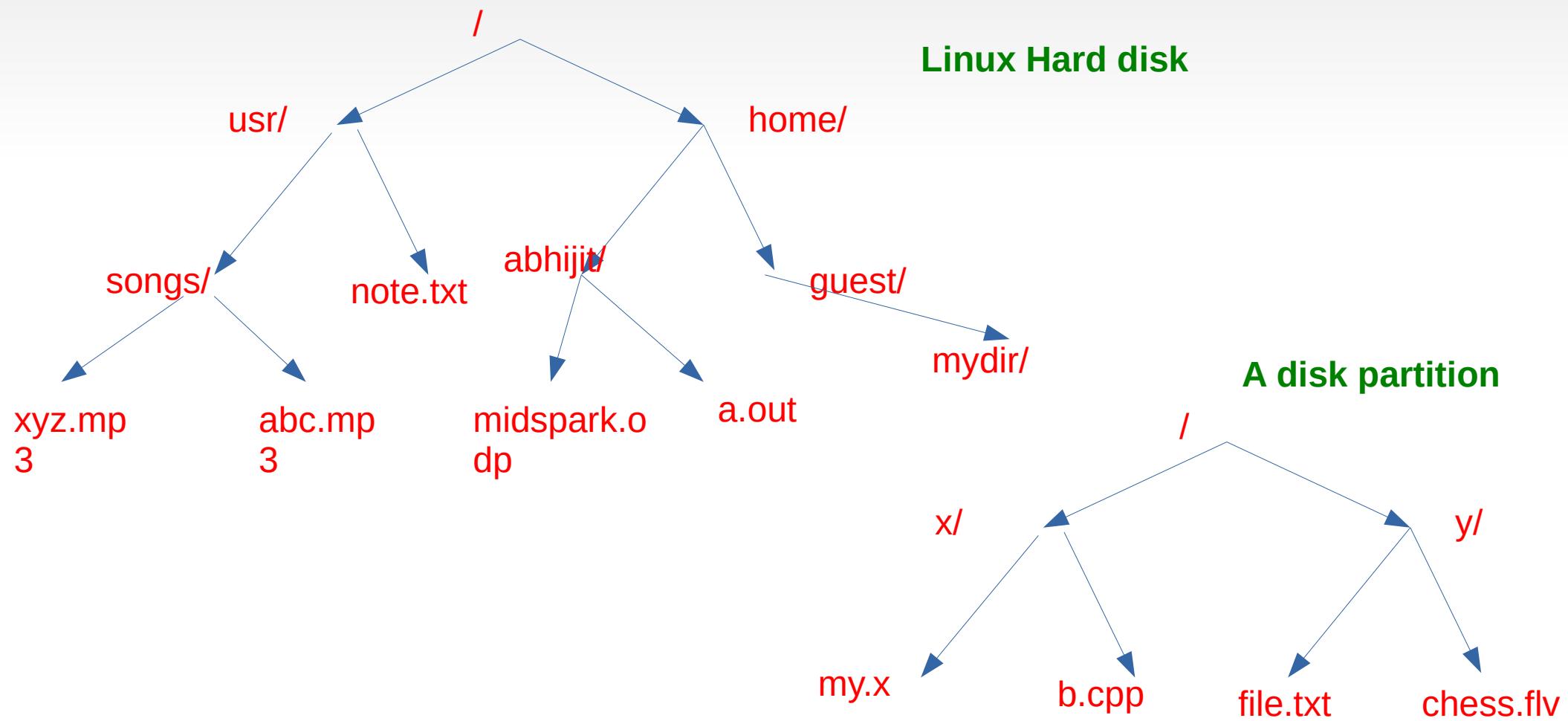
/usr/songs/xyz.mp3

- On every partition:
  - Root is “/”
  - Separator is also “/”

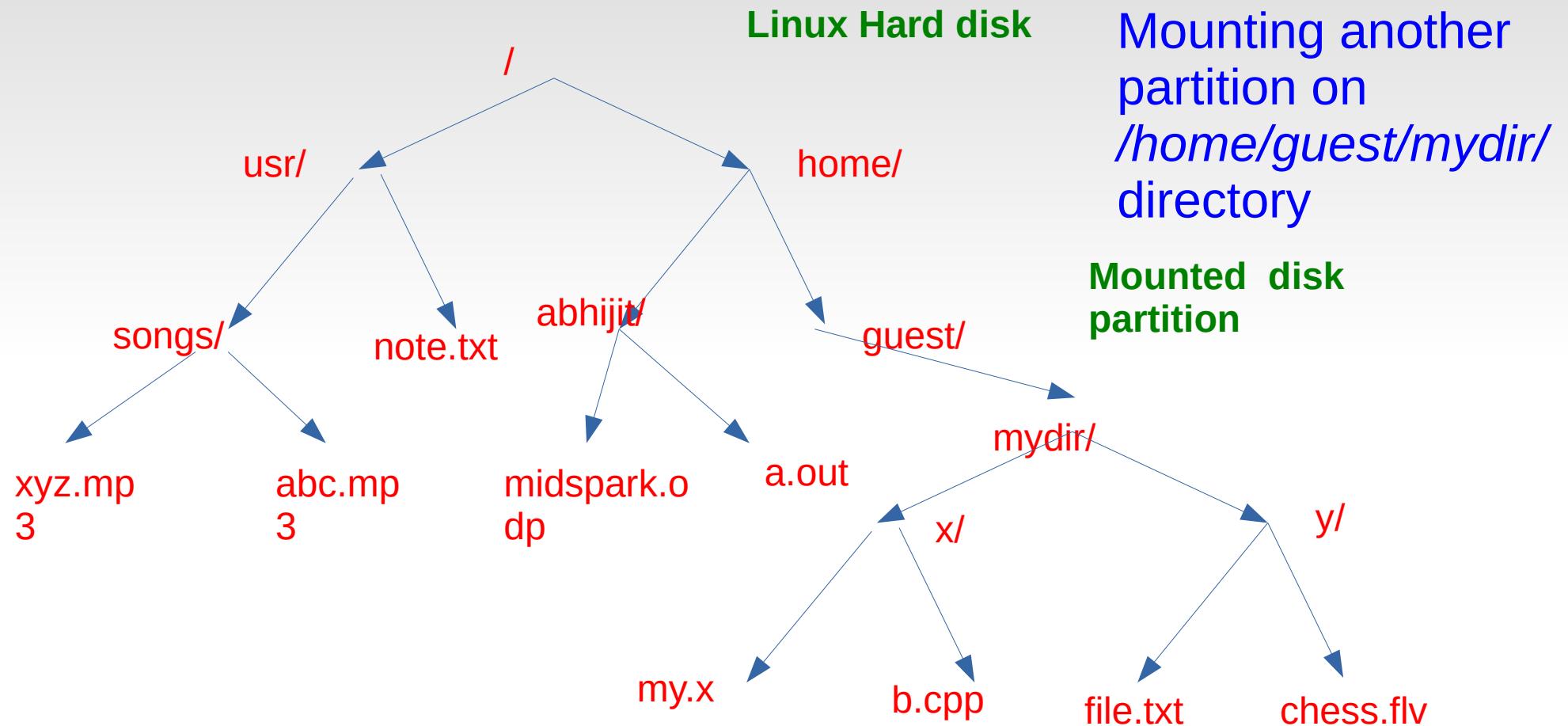


# Linux namespace: Mount

- Linux namespace is a single “tree” and not a “forest” like Windows
- Combining of multiple trees is done through “mount”



# Linux namespace Mounting a partition



*/home/guest/mydir/x/b.cpp* → way to access the file on the other disk partition

**Let's go for a live installation Demo !**

# **Some Shell Gimmicks**

# Terminal Tricks

**Ctrl + n : same as Down arrow.**

**Ctrl + p : same as Up arrow.**

**Ctrl + r : begins a backward search through command history.(keep pressing Ctrl + r to move backward)**

**Ctrl + s : to stop output to terminal.**

**Ctrl + q : to resume output to terminal after Ctrl + s.**

# Terminal Tricks

**Ctrl + a : move to the beginning of line.**

**Ctrl + e : move to the end of line.**

**Ctrl + d : if you've type something, Ctrl + d deletes the character under the cursor, else, it escapes the current shell.**

**Ctrl + k : delete all text from the cursor to the end of line.**

**Ctrl + t : transpose the character before the cursor with the one under the cursor**

# Terminal Tricks

**Ctrl + w : cut the word before the cursor;  
then Ctrl + y paste it**

**Ctrl + u : cut the line before the cursor;  
then Ctrl + y paste it**

**Ctrl + \_ : undo typing.**

**Ctrl + l : equivalent to clear.**

**Ctrl + x + Ctrl + e : launch editor defined by  
\$EDITOR to input your command.**

# Run from history

First: What's history?

Ans: Run 'history

\$ history

\$ !53

\$ !!

\$ !cat

\$ !c

# Math

```
$ echo $(( 10 + 5 )) #15
```

```
$ x=1
```

```
$ echo $(( x++ )) #1 , notice that it is still 1,  
since it's post-incremen
```

```
$ echo $(( x++ )) #2
```

# More Math

```
$ seq 10|paste -sd+|bc
```

```
# How does that work ?
```

Using expr

```
$ expr 10+20 #30
```

```
$ expr 10\*20 #600
```

```
$ expr 30 \> 20 #1
```

# More Math

Using bc

\$ bc

obase=16

ibase=16

AA+1

AB

# More Math

Using bc

```
$ bc
```

```
ibase=16
```

```
obase=16
```

```
AA+1
```

```
07 17
```

```
# what went wrong?
```

# Fun with grep

```
$ grep -Eo '[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}'
```

# above will only search for IP addresses!

```
$ grep -v bbo filename
```

```
$ grep -w 'abhijit' /etc/passwd
```

```
$ grep -v '^#' file.txt
```

```
$ grep -v ^$ file.txt
```

# xargs: convert stdin to args

```
$ find . | grep something | xargs rm
```

xargs is highly powerful

# **rsync**

**The magic tool to sync your folders!**

```
$ rsync -rvupt ~/myfiles  
/media/abhijit/PD
```

```
$ rsync -rvupt --delete ~/myfiles  
/media/abhijit/PD
```

# **find**

```
$ find .
```

```
$ find . -type f
```

```
$ find . -type d
```

```
$ find . -name '*.*.php'
```

```
$ find / -type f -size +4G
```

```
$ find . -type f -empty -delete
```

```
$ find . -type f | wc -l
```

# Download : wget, curl

```
$ wget foss.coep.org.in
```

```
$ wget -r foss.coep.org.in
```

```
$ wget -r --convert-links foss.coep.org.in
```

```
$ wget -r --convert-links --no-parent  
foss.coep.org.in/fossmeet/
```

```
$ curl  
https://raw.githubusercontent.com/onceupon/Bash-Oneliner/master/README.md | pandoc -f markdown -t man | man -l -
```

```
# curl is more powerful than wget. Curl can upload.  
Curl supports many protocols, wget only HTTP/FTP.
```

# Random data

# shuffle numbers from 0-100, then pick 15 of them randomly

\$ shuf -i 0-100 -n 15

# Random pick 100 lines from a file

\$ shuf -n 100 filename

#generate 5 password each of length 13

\$ pwgen 13 5

echo \$((RANDOM % 10))

# Run commands remotely

\$ ssh administrator@foss.coep.org.in

\$ ssh -X administrator@foss.coep.org.in

\$ ssh -X administrator@foss.coep.org.in

**firefox**

# System Information

```
# Show memory usage,. # print 10 times, at 1 second interval  
$ free -c 10 -mhs 1  
  
# Display CPU and IO statistics for devices and partitions.  
# refresh every second  
$ iostat -x -t 1  
  
# Display bandwidth usage on an network interface (e.g.  
enp175sofo)  
$ sudo iftop -i wlo1  
  
# Tell how long the system has been running and number  
of users  
$ uptime
```

# Surf the web

\$ w3m

\$ links

# Add a user without commands

**Know how to edit the */etc/passwd* and  
*/etc/shadow* files**

# More tricks

# Show 10 Largest Open Files

```
$ lsof / | awk '{ if($7 > 1048576) print  
$7/1048576 "MB" " " $9 " " $1 }' | sort -n -u |  
tail
```

# Generate a sequence of numbers

```
$ echo {01..10}
```

# More tricks

# Rename all items in a directory to lower case

```
$ for i in *; do mv "$i" "${i,,}"; done
```

# List IP addresses connected to your server on port 80

```
$ netstat -tn 2>/dev/null | grep :80 | awk '{print $5}' | cut -d: -f1 | sort | uniq -c | sort -nr | head
```

**Credits:**

**[https://onceupon.github.io/Bash-Oneliner](https://onceupon.github.io/Bash-Oneliner/)**

**<http://www.bashoneliners.com/>**

# User Administration

# Users and Groups

- There is a privileged user called “root”
  - Can do anything, like “administrator” on Windows
  - Can't login in graphical mode !
- Other users are normal users
- Some users are given “sudo” privileges: called *sudoers*
  - Sudo means “do as a superuser”
  - Password is asked, when the otherwise normal user tries to do administrative task
  - The first user account created on Ubuntu, is by default with sudo privileges

# **Adding/Deleting/Changing users**

- **System → Administration → Users and Groups**
- **Click on “Add” to add a user**
  - Asks for password !
  - Provide the details asked for
  - Verify the user was created, by doing 'switch user'
- **Try 'deleting' the user created**
- **Groups: Various groups of users meant for different purposes**
  - Every user by default belongs to her own group
  - Add the user explicitly to other groups

# Software installation

# Some terms

- **.deb**
  - The “setup” file. The installer package. Similar to Setup.exe on windows.
  - Contains all *binary* files and some *shell scripts*
- **repository**
  - A collection of .deb files, categorized according to type (security, main, etc.)
- **Software source/ ubuntu mirror**
  - A computer on internet having all .deb files for ubuntu

# **Software Installation Concept**

- **Online installation**
  - Use the “Ubuntu Software Center” to select the software, click and install !
  - Software is fetched automatically and installed !
  - Much easier than Windows !
- **Offline installation**
  - Collect ALL .deb files for your application
  - Select all, and install using package manager

# Software Installation

- **When we install using Software Center**
  - .deb files are stored in /var/cache/apt/archives folder
- **One needs to be a *sudoer* to install software**
- **Try installing some software on your own and try them out !**

# Network configuration

# Setting up network for a desktop

- **DHCP**
  - Nothing needs to be done !
  - Default during installing Linux
- **Static I/P**
  - System → Preferences → Network connections
  - System → Administration → Network
  - System → Administration → Network tools

# **Network setup for wireless**

- **Just plug and Play !**
- **Network icon shows available wireless network, just click and connect.**

# **Reliance/Tata/Idea USB devices**

- **Each has a different procedure**
- **One needs to search the web for setting it up**
  - **Most of the devices work plug and play on Ubuntu 12.04**
  - **Some do not work, as fault of the providers, they have not given an installer CD for Linux!**
    - **Still Linux community have found ways to work around it !**
    - **My Reliance netconnect worked faster and more steadily on Linux than Windows !**

# Disk Management

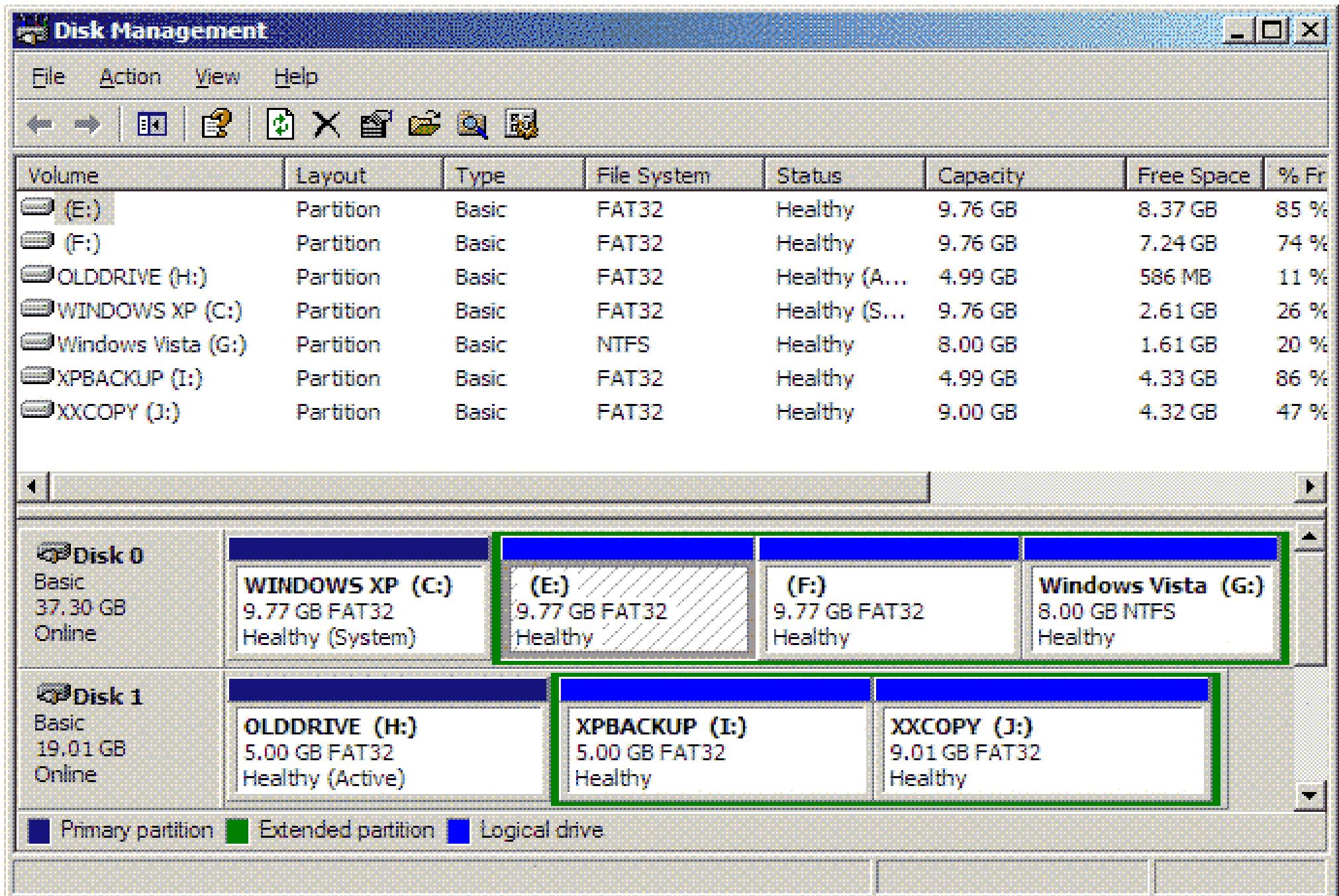
# Partition

- **What is C:\ , D:\, E:\ etc on your computer ?**
  - “Drive” is the popular term
  - Typically one of them represents a CD/DVD RW
- **What do the others represent ?**
  - They are “partitions” of your “hard disk”

# Partition

- Your hard disk is one contiguous chunk of storage
  - Lot of times we need to “logically separate” our storage
  - Partition is a “logical division” of the storage
  - Every “drive” is a partition
- A logical chunk of storage is partition
  - Hard disk partitions (C:, D:), CD-ROM, Pen drive, ...

# Partitions



# Managing partitions and hard drives

- **System → Administration → Disk Utility**
- **Had drive partition names on Linux**
  - `/dev/sda` → Entire hard drive
  - `/dev/sda1, /dev/sda2, /dev/sda3, ....` Different partitions of the hard drive
  - Each partition has a *type* – ext4, ext3, ntfs, fat32, etc.
- **Pen drives can also be managed from here**
- **Formatting can also be done from here**

[Containers recap](#)

[What is Kubernetes](#)

[Kubernetes Architecture](#)

[Components of Kubernetes cluster](#)

[Pod](#)

[Kube-apiserver](#)

[KubeControllerManager](#)

[Kube scheduler](#)

[Kubelet](#)

[Hands-on](#)

[Pod replication](#)

[Deployments](#)

[Create custom image with required contents \(optional\)](#)

[Namespace](#)

[Resource limit on Pod](#)

[Resource limit on namespace](#)

[Services in Kubernetes](#)

[ClusterIP](#)

[NodePort](#)

[LoadBalancer](#)

[Services summarized](#)

[Scheduling](#)

[Storage](#)

[Lab setup](#)

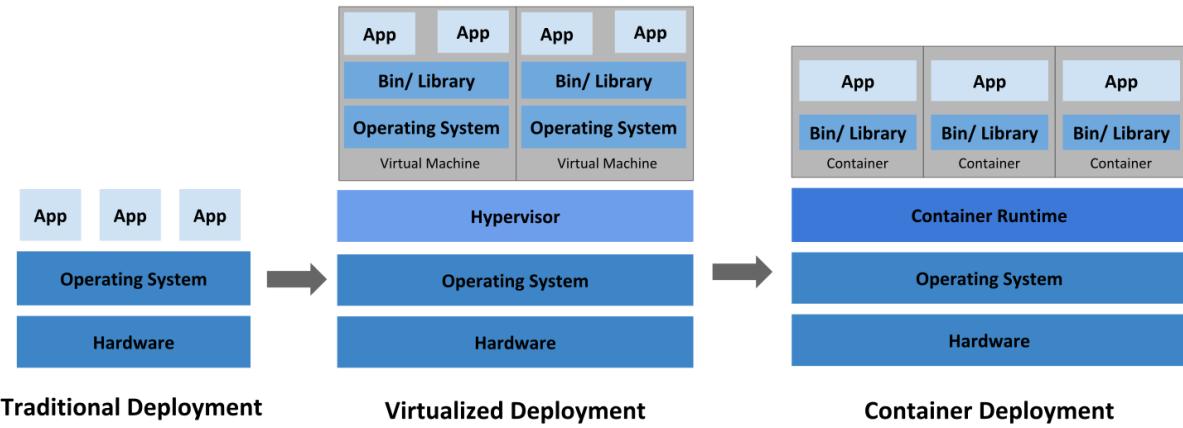
[Minikube installation on Fedora OS](#)

[Install kubectl](#)

[Multi node kubernetes setup](#)

[Resources:](#)

## Containers recap

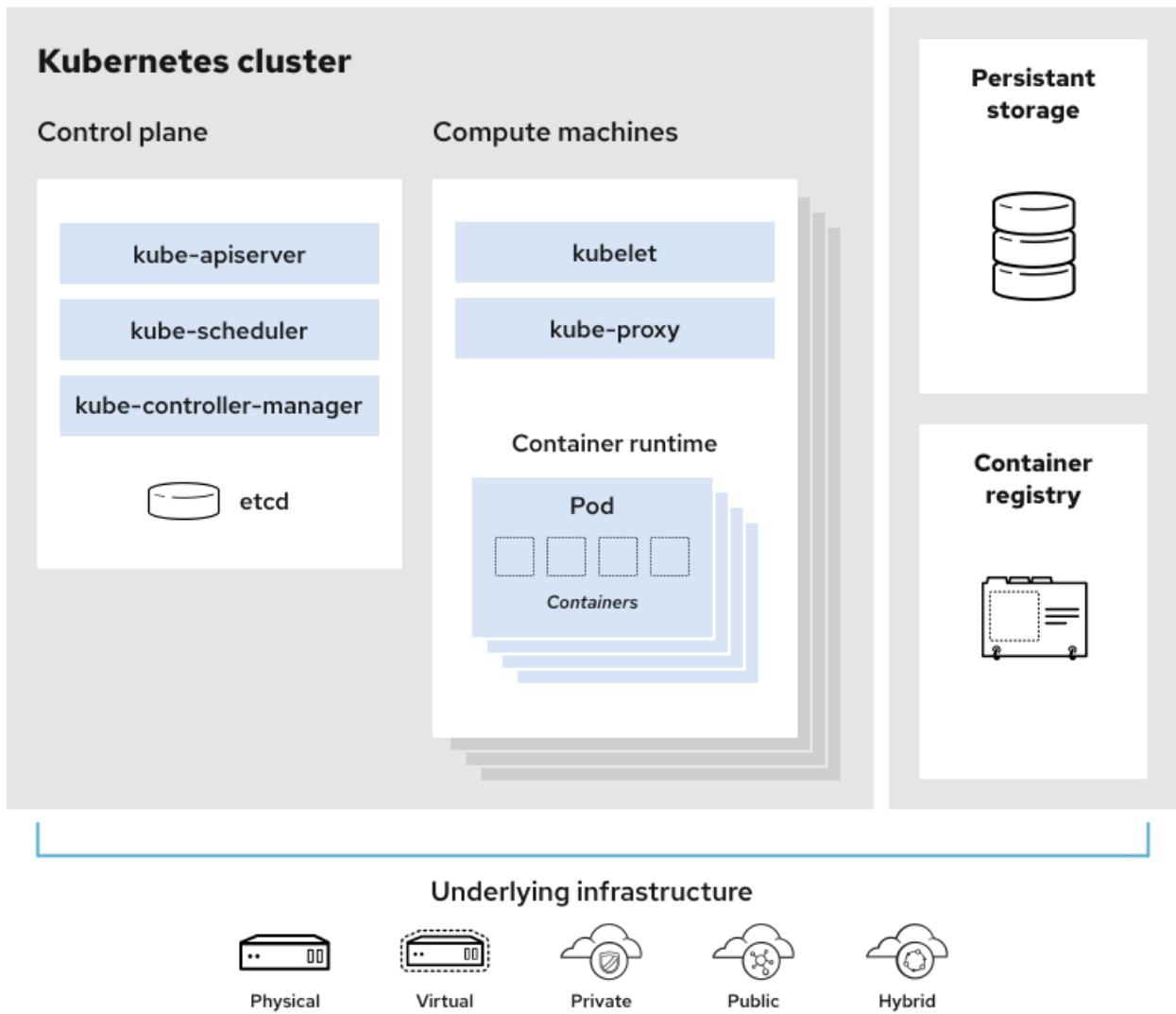


## What is Kubernetes

Kubernetes (also known as k8s or “kube”) is an open source container orchestration platform that automates many of the manual processes involved in deploying, managing, and scaling containerized applications.

Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation. It's based on Google's internal project - Borg.

## Kubernetes Architecture



- Control plane (master nodes)  
Manages the workload

- Compute plane (worker nodes)  
Runs the workload

## Components of Kubernetes cluster

**Master node - manages the cluster setup**

- Etcd

Etcd is a key-value store database. Configuration data and information about the state of the cluster lives in etcd,

Fault-tolerant and distributed, etcd is designed to be the ultimate source of truth about your cluster.

- kube-scheduler

The scheduler considers the resource needs of a pod, such as CPU or memory, along with the health of the cluster. Then it schedules the pod to an appropriate compute node.

- Controller manager

Controllers take care of actually running the cluster, and the Kubernetes controller-manager contains several controller functions. For Example:

- Node controller - Manages node related tasks like onboarding nodes, managing node failures, etc
  - Replication controller - Manages container. Make sure the desired number of containers run all the time.
- Kube-apiserver

Facilitates the interaction with the Kubernetes cluster.

The is the front end of the Kubernetes control plane, handling internal and external requests. The API server determines if a request is valid and, if it is, processes it. You can access the API through REST calls, through the kubectl command-line interface, or through other command-line tools such as kubeadm.

## **Worker node - Run the container workload in the cluster setup**

- Container runtime Engine

To run the containers, each compute node has a container runtime engine. Docker is one example, but Kubernetes supports other Open Container Initiative-compliant runtimes as well, For example:

- Docker

- Containerd
  - Rocket
  - CRI-O
- Kubelet

Each compute node contains a kubelet, a tiny application that communicates with the control plane. The kubelet makes sure containers are running in a pod. When the control plane needs something to happen in a node, the kubelet executes the action.

- Kube-proxy

Each compute node also contains kube-proxy, a network proxy for facilitating Kubernetes networking services. The kube-proxy handles network communications inside or outside of your cluster—relying either on your operating system’s packet filtering layer, or forwarding the traffic itself.

## Pod

Pod is the atomic unit of scheduling in kubernetes cluster

Pod is a group of containers. Normally one container is run in one pod.  
When two applications are tightly coupled, then only two containers may run in a single pod.  
Usually it is avoided to run more than one container in one pod.

Pod is a wrapper to distribute the container.

Consider running multiple containers with port mapping from host:

```
docker run -d -p 80:80 httpd  
docker run -d -p 80:80 httpd
```

The second run fails as port 80 on the host is already occupied with the earlier container. Hence another container needs to be run with different port mapping on host.

```
docker run -d -p 81:80 httpd
```

We need to manually track the free ports.

Container abstraction helps to handle this.

Pod gets its own network namespace and virtual ethernet connection to connect underlying infrastructure. (We will look actual hands-on with this using kubernetes later)

Basic operations to run a workload in Kbernetes:

Package application as container

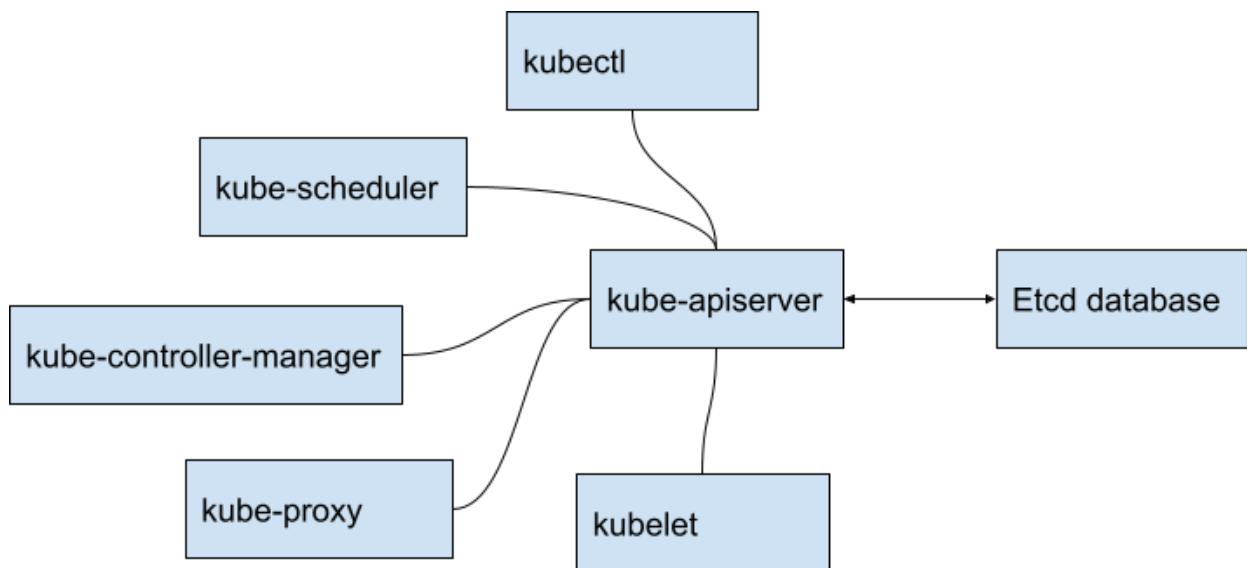
Wrap it in the pod

Deploy it using manifest file

Pods help for abstraction of container engine.

With use of pods as basic atomic unit, the underlying container engine can be easily changed.

## Kube-apiserver



Consider any kubectl get command

- 1) **kubectl** generates authentication request to **Kube-apiserver**
- 2) **Kube-apiserver** validates the request
- 3) **Kube-apiserver** retrieve data from **etcd**
- 4) Data is displayed on the console by **kubectl**

APIs can also be invoked directly by sending HTTP request instead of running the kubectl command because api-server uses HTTP protocol for all communication

During the request for creating pod:

- 1) **kubectl** generates authentication request to **kube-apiserver**
- 2) **Kube-apiserver** validates the request
- 3) **kube-apiserver** creates a pod object without assigning it to a node
- 4) **kube-apiserver** updates the pod info in **etcd** database
- 5) **Kube-apiserver** updates the user that pod has been created
- 6) **kube-scheduler** continuously monitors the **kube-apiserver** and realizes there is new pod with no node assigned
- 7) **kube-scheduler** identifies the right node for the pod and communicated that back to **kube-apiserver**
- 8) **kube-apiserver** updates the info in **etcd**
- 9) **kube-apiserver** passes that info to the **kubelet** in the appropriate worker node
- 10) **kubelet** creates the pod in the node and instructs the container runtime engine to deploy the application image
- 11) once done **kubelet** updates the status back to the **kube-apiserver**
- 12) **kube-apiserver** updates the data back in the **etcd**

**kube-apiserver** at the center of all the tasks that need to be performed on the cluster.

## KubeControllerManager

Manages various controllers in the setup.

Continuously monitors the status of different components on the cluster

In case of any failure, it takes corrective actions to bring the cluster back to its desired state.

For Example:

node-controller continuously monitors the status of the nodes via kube-apiserver

Heartbeat is sent every 5 seconds to monitor the health of the node

If the heartbeat is missed, the controller waits for 40 seconds before marking the node unreachable.

After the node is marked unreachable, the controller waits for another 5 mins to see if the node is coming back.

If a node does not come up after 5 mins, the controller manager moves the pods assigned to the broken node to the healthy nodes.

There are many controllers in a kubernetes cluster e.g. node-controller, replication controller, namespace controller, deployment controller, etc. All these controllers are packaged in to single kube-controller-manager process

## Kube scheduler

Kube scheduler does not create the pod on the worker node.

Kube scheduler just decides which pod goes on which node. Kubelet creates the pod on the node.

Scheduler tries to identify the best node for the pod based on:  
resource requirements

Taints

Affinity rules

Etc

## Kubelet

Kubelet in the worker node registers the node with the cluster

When kube-apiserver sends request to load a container it sends request to container runtime engine e.g. docker to run the instance

Kubelet then regularly monitors the status of the pods and reports the status to kube-apiserver

Kubelet is the sole point of contact on the worker node for the master node.

Load or unload the pods, send status

## Hands-on

```
kubectl get pods
```

```
kubectl get pods -A
```

```
kubectl run --image IMAGE_NAME POD_NAME
```

```
kubectl run --image nginx nginx
```

```
kubectl create deployment NAME --image=IMAGE
```

```
kubectl create -f definition.yml
```

```
kubectl create -f definition.yml
```

```
kubectl delete pod NAME
```

```
kubectl delete deployment NAME
```

```
kubectl run NAME --image=NAME --dry-run=client -o yaml
```

```
kubectl edit pod NAME
```

```
kubectl apply -f file.yml
```

```
kubectl describe pod PODNAME
```

```
kubectl logs PODNAME
```

```
kubectl logs PODNAME CONTAINERNAME
```

```
pod.yml:  
apiVersion: v1  
kind: Pod  
metadata:  
  name: myapp  
  labels:  
    app: myapp  
    type: prod  
spec:  
  containers:  
    - name: container-1  
      image: redis  
    - name: container-2  
      image: nginx
```

## Pod replication

Replication controller helps create multiple instances of a pod to provide high availability.

It also helps with scaling of application workload using load balancing.

What is Scale up (vertical scaling) VS scale out (horizontal scaling)?

Labels and Selectors are used to identify and use the pods in a specific replica set.

This can be used for pods outside of the replica set definition file as well.

```
apiVersion: apps/v1  
kind: ReplicaSet  
metadata:  
  name: myapp-ha  
  labels:  
    app: myapp  
    type: front-end
```

```
spec:  
template:  
  metadata:  
    name: myapp  
  labels:  
    app: myapp  
    type: frontend  
spec:  
  containers:  
    - name: container-1  
      image: redis  
replicas: 3  
selector:  
  matchLabels:  
    type: frontend
```

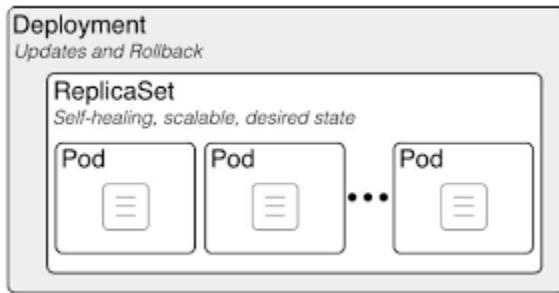
```
Kubectl create -f file.yml  
Kubectl explain replicaset  
Kubectl get replicaset  
Kubectl get pods  
Kubectl delete pod NAME  
Kubectl get pods  
Kubectl describe replicaset myapp-ha
```

Scale by changing the replicas number in above config and then use below command  
kubectl replace -f file.yml

Alternate options:

```
Kubectl scale --replicas=4 -f file.yml  
Kubectl scale --replicas=5 replicaset myapp-ha
```

## Deployments



Typical requirements form any production application:

- Replications - for high availability
- Seamless upgrade
- Rolling updates - updates one after other for seamless user experience
- Rollback updates - in case of failure, updates can be rolled back
- Pause - update resume capability

Same config as replicaset except  
kind: Deployment

Kubectl explain deployment

Generate the sample config file using:

```
Kubectl create deployment --image=nginx nginx --dry-run=client -o yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.22
```

```
Kubectl apply -f file.yml  
Kubectl get deployments  
Kubectl get replicaset  
Kubectl get pods  
Kubectl get all  
Kubectl describe deployment NAME  
Kubectl describe pod name
```

Watch “kubectl get all”

```
kubectl set image deployment.v1.apps/nginx-deployment nginx=nginx:1.16.1  
Kubectl describe pod name
```

[https://hub.docker.com/\\_/nginx/tags](https://hub.docker.com/_/nginx/tags)

Edit file and change to the newer version  
Kubectl apply -f file.yml

```
Kubectl describe pod name  
kubectl rollout status deployment nginx-deployment  
kubectl rollout history deployment nginx-deployment  
kubectl rollout undo deployment nginx-deployment --to-revision 1
```

## Create custom image with required contents (optional)

- 1) Create a program (tobe used in custom image) which will consume system memory:

```
cat mem.c  
/*  
 * https://stackoverflow.com/a/1865536  
 */  
#include <stdio.h>  
#include <stdlib.h>  
#include <string.h>  
int main()  
{  
    while(1)  
    {  
        printf ("\nAllocating... ");  
        void *m = malloc(1024*1024);  
        memset(m,0,1024*1024);  
        printf("done");  
        printf ("\nPress any key to continue...");  
    }  
}
```

```
//sleep(2);
getchar();
}
return 0;
}
```

```
$ gcc mem.c -o mem
```

- 2) Create a Dockerfile to generate a custom container with required packages and tools.

```
cat Dockerfile
from fedora
run dnf -y install iputils iproute
copy mem /usr/local/bin/mem
CMD sleep 100000
```

- 3) Create docker image and test is by running docker container

```
docker build -t my-fedora .
```

```
docker run --rm -ti --name new-fedora my-fedora bash
```

- 4) Login to <https://hub.docker.com/> with your credentials and create a public repo with tag

- 5) Back to the system: tag newly created custom image and push it to docker hub:

```
$ docker login
$ docker image tag my-fedora:latest ashishkshah/my-fedora:test
$ docker push ashishkshah/my-fedora:test
```

- 6) Create kubernetes pod using this new image:

```
$ kubectl run --image ashishkshah/my-fedora:test mypod
```

- 7) Connect to the pod with bash shell and verify the tools embedded in it:

```
kubectl exec --stdin --tty mypod -- /bin/bash
$ mem
$ ping
$ ip
```

## Namespace

```
kubectl create namespace myspace
```

```

kubectl get pods --namespace=myspace
kubectl create -f file.yml --namespace=myspace
kubectl get ns
kubectl get pods --all-namespaces
kubectl config current-context
kubectl config set-context kubernetes-admin@kubernetes --namespace=myspace

cat namespace.yml
apiVersion: v1
kind: Namespace
metadata:
  creationTimestamp: null
  name: myspace
spec:
status: {}

```

## Resource limit on Pod

- 1) Run the pod with custom image

```

cat pod-limits.yml
---
apiVersion: v1
kind: Pod
metadata:
  name: testpod
spec:
  containers:
    - name: app
      image: ashishkshah/myrepo:test
      resources:
        requests:
          memory: "8Mi"
        limits:
          memory: "16Mi"

```

- 2) Connect to the pod via bash shell (two sessions) and execute memory consumption utility /usr/local/bin/mem
- 3) Keep increasing the program's memory consumption and observe it in another session.  
\$ watch “ps aux | grep mem | grep -v mem”

#### 4) Watch the program getting terminated upon hitting limit

A request is the amount of that resource that the system will guarantee for the container, and Kubernetes will use this value to decide on which node to place the pod.

A limit is the maximum amount of resources that Kubernetes will allow the container to use.

### Resource limit on namespace

```
cat resource-quota.yml
apiVersion: v1
kind: ResourceQuota
metadata:
  name: quota
spec:
  hard:
    cpu: "2"
    memory: "32Mi"
    pods: "4"
    replicationcontrollers: "2"
    resourcequotas: "1"
    services: "3"
```

```
kubectl describe quota
```

```
kubectl --namespace=myspace describe quota
```

```
# kubectl apply -f resource-quota.yml --namespace=myspace
```

```
kubectl --namespace=myspace describe quota
```

```
kubectl config set-context minikube --namespace=myspace
```

```
kubectl config get-contexts
```

```
kubectl describe quota
```

```
cat pod-limits-nginx.yml
```

```
---
apiVersion: v1
kind: Pod
metadata:
  name: testpod
spec:
  containers:
    - name: app
      image: nginx
```

```
resources:  
  requests:  
    memory: "8Mi"  
    cpu: 2  
  limits:  
    memory: "16Mi"  
    cpu: 2
```

## Services in Kubernetes

In kubernetes, services are different mechanisms available for accessing pods. By Default, pods created on kubernetes cluster are not accessible outside of the cluster and they can not communicate among themselves as well.

```
kubectl run --image nginx nginx --port 80  
kubectl describe pod nginx | grep 'Node\|IP'
```

```
curl IP  
curl IP <-- from node
```

```
kubectl port-forward pod-nginx 30005:80 --address 0.0.0.0
```

NOTE: port-forwarding is used throughout this hands-on activity for the demonstration purpose only. This is not a recommended way to expose any service from kubernetes cluster.

There are different types of services available in kubernetes cluster which enables the communication among the pods and communication outside the cluster.

- 1) ClusterIP
- 2) NodePort
- 3) LoadBalancer

### ClusterIP

ClusterIP is the default service type.

This is used for inter service communication within the cluster.

Consider two deployment sets, front-end and back-end.

Communication between these two deployment sets is established using clusterip service.

Why do we need it? or why do we prefer this for inter pod communication within the cluster?

If we are using a pod's ip address for communication among the pods, when any pod is deleted or crashed due to some reason, the deployment set or replica set will spawn another pod to maintain the replica count.

The new pod spawned will have different ip addresses and hence it will not be easy to maintain the communication among the pods with new pods running with different ips.

To resolve this, we are creating a clusterip service which attaches itself to pods or replicas or daemonsets using labels and selectors.

In our example we will have two clusterip services for frontend and backend each.

The IP address of clusterip service is now being used for communication among the services instead of pods ip address.

Now even if the pods in replicaset fail and new pods are spawned with a new ip address, communication between the two services is not broken as it is done using clusterip service.

Kubernetes assigns a cluster-internal IP address to ClusterIP service.

Due to this the service is only reachable within the cluster.

The IP address of ClusterIP service can not be accessed from outside the cluster.

```
### clusterip service enable connectivity between group of pods
```

```
kubectl create -f pod-client.yml
```

```
### Try to access pod ip using curl
```

```
kubectl get all -o wide
```

```
### Note the node where pod is running
```

```
### From the node try to access service using curl
```

```
cat deployment-nginx.yml
```

```
apiVersion: apps/v1
```

```
kind: Deployment
```

```
metadata:
```

```
  name: nginx-server-dep
```

```
  labels:
```

```
    app: nginx-server
```

```
spec:
```

```
  template:
```

```
    metadata:
```

```
      labels:
```

```
        app: nginx-server
```

```
spec:  
  containers:  
    - name: frontend  
      image: nginx  
replicas: 3  
selector:  
  matchLabels:  
    app: nginx-server
```

```
cat cip-service-nginx-dep.yml
```

```
apiVersion: v1  
kind: Service  
metadata:  
  labels:  
    app: cip-nginx-server-dep  
    name: cip-nginx-server-dep  
spec:  
  ports:  
    - name: "80"  
      port: 80  
      protocol: TCP  
      targetPort: 80  
  selector:  
    app: nginx-server  
  type: ClusterIP
```

```
kubectl apply -f deployment-nginx.yml
```

```
kubectl apply -f cip-service-nginx-dep.yml
```

```
kubectl port-forward services/cip-nginx-server-dep 30005:80 --address 0.0.0.0
```

```
curl kubemaster:30005
```

```
### Kill the pods one by one and make sure service is accessible via service ip
```

## NodePort

NodePort service is an extension of ClusterIP service.

It exposes the service outside of the cluster by adding a cluster-wide port on top of ClusterIP.

Limitation of clusterip service was it can not be used for communication outside the cluster.

Noteport service can be used for enabling access to the service outside of the cluster.

As the name may suggest, the NodePort service exposes the service(port) on each Node's IP. The service can be accessed from outside the cluster using nodeip:port. Port configured to listen on the node is mapped to the service port and it is further mapped to the port on the pod.

Node port must be in the range of 30000–32767. The ports can be allocated manually or kubernetes will take care of it if they are not manually assigned.

```
cat np-service-nginx-server.yml
apiVersion: v1
kind: Service
metadata:
  labels:
    app: np-service-server
    name: np-service-server
spec:
  ports:
  - name: "80"
    port: 80
    protocol: TCP
    targetPort: 80
    nodePort: 30009
  selector:
    app: nginx-server
  type: NodePort
```

```
kubectl apply -f np-service-nginx-server.yml
kubectl describe service np-service-server
```

Check the node where pod is provisioned

run curl kubeworker1:30005 from outside

## LoadBalancer

The nodeport service has enabled external connectivity for the service. But still there is one problem with it.

The service is listening on the specified port on node's ip. That means if the pods in your replicaset are distributed among different nodes, all the nodes ip addresses will listen on the nodeport. Hence there will be a number of IP:PORT combinations available to access your service.

Needless to say there is no server side distribution of the traffic with this approach of accessing service. This approach is not practically used for providing external access to the service but is used as an intermediate step.

LoadBalancer service is an extension of NodePort service.

Loadbalancer integrates NodePort with cloud-based load balancers and provides a single ip:port combination for accessing service from external networks.

The load is then distributed by load balancer service to different nodes underneath.

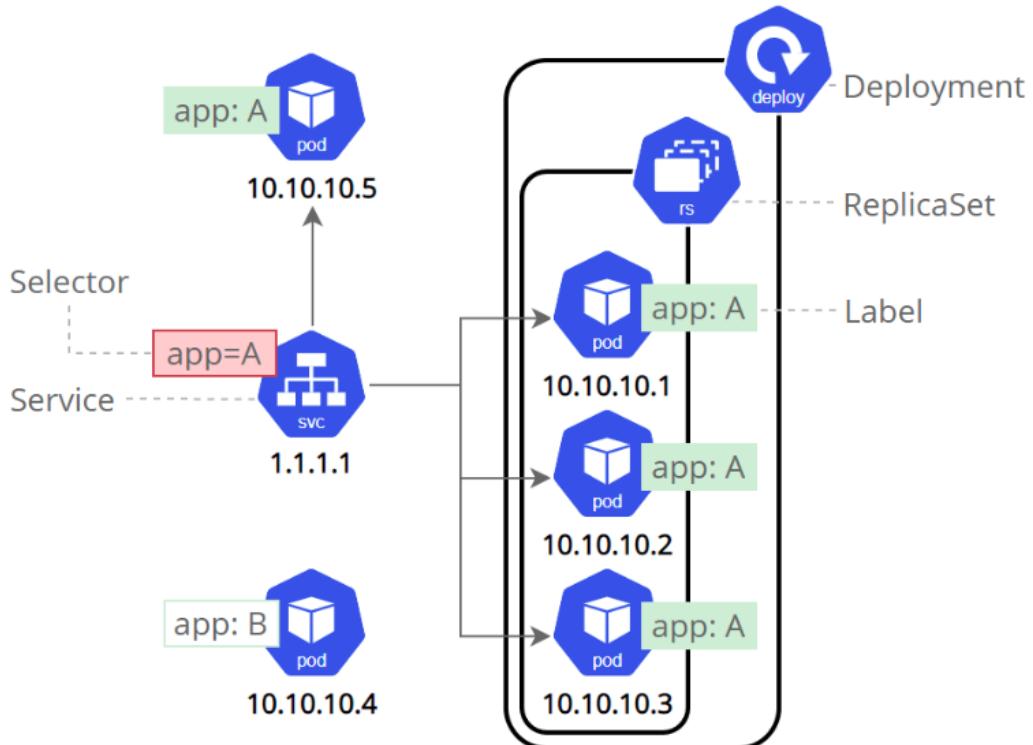
Usually cloud-provide's load balancer service offering is used to exposes this service externally. Each cloud provider like e.g. AWS, Azure, GCP, etc has its own native load balancer implementation.

This type of service is heavily dependent on the cloud provider.

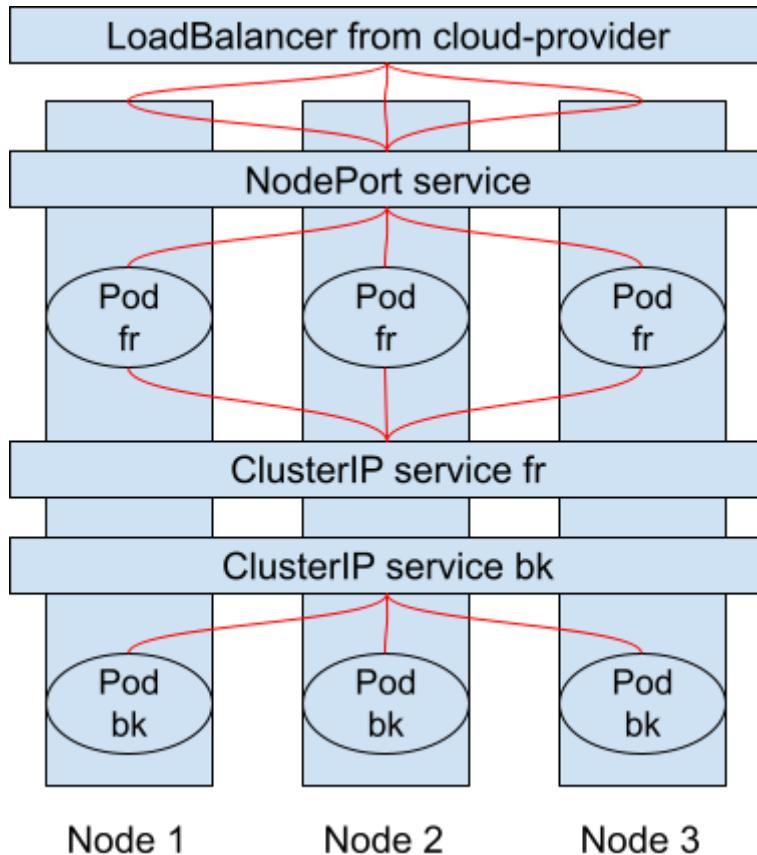
Load Balancer created by cloud provider routes the incoming requests to kubernetes services which sends the traffic further to backend pods.

Services summarized

Function:



Implementation:



## Scheduling

### Scheduling

A production kubernetes cluster consists of a number of worker nodes.

We run various types of workloads in any production environment.

There could be applications which need to be run on hardware with specific configuration, some applications may require specific tweaks in the OS for it to run effectively.

In the kubernetes cluster how can we guarantee that the pod for a specific application will always pick the suitable node to run.

Kubernetes has different scheduling capabilities to achieve this.

### Manual Scheduling

Add below in pod's specs section

nodeName: node01

We cannot move a running pod from one system to another.

We need to delete the pod from one node and re-create in other.

Selectively list the desired pods.

```
kubectl get pods --selector app=nginx-server
```

Pods with label app=nginx-server will only be listed with above command.

## Taints and Tolerations

```
kubectl taint nodes node-name key=val:taint-effect
```

taint effects:

noschedule - pods will not be scheduled on the node

prefer no schedule - system will try to avoid placing pod on node but it's not guaranteed

noexecute - new pods will not be scheduled on the node and existing pods will be deleted from it

```
kubectl taint nodes node1 app=redis:NoSchedule
```

Toleration:

added to pods

under specs, add section

tolerations:

```
- key: "app"  
  operator: "Equal"  
  value: "redis"  
  effect: "NoSchedule"
```

taints and tolerations are not to tell the pod to go to a specific node but it is for nodes to accept only a certain pods.

If you want to restrict pods to certain nodes it can be node with node affinity.

taint on master node prevents any pods from being scheduled on it

```
kubectl describe nodes kubemaster | grep Taint
```

```
kubectl taint node node01 app=nginx:NoSchedule  
kubectl run nginx --image=nginx
```

```
kubectl describe pod nginx  
see why pod is in pending state
```

```
kubectl run nginx2 --image=nginx --dry-run=client -o yaml
```

add section called

tolerations:

```
- key: "app"  
  operator: "Equal"  
  value: "redis"  
  effect: "NoSchedule"
```

check taint on ctrl plain node

remove the taint on it

copy taint form describe command

```
kubectl taint node controlplane PasteTheTaintAndAppendDashToRemovelt-
```

check the taint is removed now

```
cat deployment.yml  
apiVersion: apps/v1  
kind: Deployment  
metadata:  
  name: myapp-ha  
  labels:  
    app: myapp  
    type: front-end  
spec:  
  template:  
    metadata:  
      name: myapp  
    labels:  
      app: myapp  
      type: frontend  
  spec:  
    containers:  
      - name: container-1  
        image: nginx  
    nodeName: node01
```

```
replicas: 4
selector:
  matchLabels:
    type: frontend

cat pod-nginx.yml
apiVersion: v1
kind: Pod
metadata:
  name: nginx-server-1
  labels:
    app: nginx-server
spec:
  containers:
    - name: container-1
      #image: ashishkshah/myrepo:test
      image: nginx
      ports:
        - containerPort: 80
  tolerations:
    - key: "app"
      operator: "Equal"
      value: "nginx-server"
      effect: "NoExecute"
```

## Storage

As discussed during the docker session, docker images are layered one above other. A new layer is created for every line (which modifies the image) in the Dockerfile.

Old layers are reused from the cache when the action to be performed is the same in different Dockerfile. Thus it's faster and saves space as well.

Storage files in docker can be found at /var/lib/docker on host

Image layers are read only and can only be modified with image build.

Creating container creates a writable layer on top of it  
the rw layers are available only till the life of the container. When container is deleted this layer is lost hence data is ephemeral.

Docker uses a copy on write mechanism to modify the files in the image.

Any image file being modified is copied to the read-write layer first and then it is modified. Files in the image layer will not be modified as its read only layer.

This layer will only be modified upon docker build command to build the image.

To make the files modified persistent across container deletion we can use persistent volumes.

Two types of mounting

- Volume mount
- Bind mount

docker volume create myvol

creates vol directory at /var/lib/docker/volumes/myvol

docker run -v myvol:/var/lib/mysql mysql

mounts volume on host to mysql dir in container

if vol is not already created, docker will auto create it upon first use.

To use external storage on host instead of volume run:

docker run -v /data/mysql:/var/lib/mysql mysql

The path of mounted storage or directory on host is mentioned in the command instead of volume name.

alternate command (more preferred):

docker run --mount type=bind,source=/data/mysql,target=/var/lib/mysql mysql

Storage drivers responsible of doing all storage related operations like managing layers, maintaining files across the layers, etc.

aufz  
zfs  
btrfs  
device mapper  
overlay  
overlay2

Like Storage drivers manage the storage we have volume driver for volume management in docker

default volume driver plugin is local

There are other storage vendor specific drivers

e.g.

Azure FS storage, Convoy, gcw-docker, flusterfa, netapp, vmware vsphere storage, etc

Like CRI and CNI - CSI is developed to support multiple storage solutions. different storage vendors have their own csi drivers. this is not k8s specific standard. It is a universal standard meant for any container orchestration tool to work with any storage vendor with supported plugins.

## Volumes in k8s

```
cat pod-storage.yml
apiVersion: v1
kind: Pod
metadata:
  name: myapp
  labels:
    app: nginx
    type: frontend
    subject: devops
spec:
  containers:
    - name: container-1
      image: nginx
      volumeMounts:
        - mountPath: /usr/share/nginx/html/
          name: nginx-home
          readOnly: false
  volumes:
    - name: nginx-home
      hostPath:
        path: /data
```

Check where pod is running and then create /data/index.html in that node (or use preferredNode option)

## Persistent Volumes

In the above method, the user has to know the availability of the storage volume to be used. This must not be the case. Users should not be bothered about finding the suitable storage space but just ask for it.

System administrator should be responsible for making the storage space available.

One level of abstraction is achieved in allocation of storage space using persistent volumes and claims.

System administrators create persistent volumes on the kubernetes cluster.  
Users just claim the storage space from the available pool of persistent volumes.

PVs are visible throughout the entire cluster. They are not namespace specific.  
Claims are specific to a namespace.

```
cat pv.yml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: nginx-pv
spec:
  capacity:
    storage: 1Gi
  accessModes:
    - ReadWriteMany
  persistentVolumeReclaimPolicy: Recycle
  hostPath:
    path: /data
```

```
cat pvc.yml
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: nginx-claim
spec:
  accessModes:
    - ReadWriteMany
  resources:
    requests:
      storage: 50Mi
```

```
cat pod-pvc.yml
apiVersion: v1
kind: Pod
metadata:
  name: myapp
  labels:
    app: nginx
    type: frontend
```

```
subject: devops
spec:
  containers:
    - name: container-1
      image: nginx
      volumeMounts:
        - mountPath: /usr/share/nginx/html/
          name: mydrive
  volumes:
    - name: mydrive
      persistentVolumeClaim:
        claimName: nginx-claim
```

## Lab setup

### Minikube installation on Fedora OS

```
curl -LO https://storage.googleapis.com/minikube/releases/latest/minikube-latest.x86_64.rpm
sudo rpm -Uvh minikube-latest.x86_64.rpm
```

```
dnf install docker
systemctl enable docker.service --now
Useradd kube
Passwd kube

usermod -aG docker $USER && newgrp docker
echo -e "kube  ALL=(ALL)    ALL" >> /etc/sudoers
Su - kube

$ minikube start --driver=docker
```

### Install kubectl

```
cat <<EOF | sudo tee /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-\$basearch
```

```
enabled=1
gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF
sudo yum install -y kubectl
kubectl get pods -A
```

### **Bash auto completion:**

```
kubectl completion bash | sudo tee /etc/bash_completion.d/kubectl > /dev/null
source /usr/share/bash-completion/bash_completion
Kubectl <tab> <tab>
```

Multi node kubernetes setup

Execute steps 1 to 6 on all the nodes

- 1) Set static ip, gateway and dns configuration:

```
nmcli connection show
nmcli
```

```
sudo nmcli con modify NAME ifname NAME ipv4.method manual ipv4.addresses
192.168.122.10/24 gw4 192.168.122.1
```

```
sudo nmcli con mod NAME ipv4.dns 192.168.122.1
```

```
nmcli con down NAME
nmcli con up NAME
```

```
sudo hostnamectl set-hostname master-node
```

```
reboot
```

- 2) Disable swap

```
rpm -e zram-generator-defaults zram-generator
swapoff -a
free
```

### 3) Configure repo

```
cat <<EOF > /etc/yum.repos.d/kubernetes.repo
[kubernetes]
name=Kubernetes
baseurl=https://packages.cloud.google.com/yum/repos/kubernetes-el7-x86_64
enabled=1
gpgcheck=1
repo_gpgcheck=1
gpgkey=https://packages.cloud.google.com/yum/doc/yum-key.gpg
https://packages.cloud.google.com/yum/doc/rpm-package-key.gpg
EOF
```

### 4) Install packages and enable service

```
yum install -y docker kubelet kubeadm kubectl crio
```

```
systemctl enable docker --now
systemctl enable kubelet --now
systemctl enable crio --now
```

### 5) Configure firewall rules

```
sudo firewall-cmd --permanent --add-port=6443/tcp
sudo firewall-cmd --permanent --add-port=2379-2380/tcp
sudo firewall-cmd --permanent --add-port=10250/tcp
sudo firewall-cmd --permanent --add-port=10251/tcp
sudo firewall-cmd --permanent --add-port=10252/tcp
sudo firewall-cmd --permanent --add-port=10255/tcp
sudo firewall-cmd --reload
```

### 6) Enable bridge-nf-call-iptables in sysctl config file

```
cat <<EOF > /etc/sysctl.d/k8s.conf
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
EOF
sysctl --system
```

### 7) Execute steps 1 to 6 on Master node and then run below command:

```
sudo kubeadm init --pod-network-cidr=10.244.0.0/16
```

```
mkdir -p $HOME/.kube  
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config  
sudo chown $(id -u):$(id -g) $HOME/.kube/config  
  
sudo kubectl apply -f  
https://raw.githubusercontent.com/coreos/flannel/master/Documentation/kube-flannel.yml
```

8) Run steps 1 to 6 on worker node and then join the node to master

```
kubeadm join 192.168.122.10:6443 --token fm2fd.940uvjc3tqi6pgjx  
--discovery-token-ca-cert-hash  
sha256:a8989efa29ec4e6f4c343d104b3f66f84da855992d795ddc9082522bc39b9346
```

## Resources:

<https://kubernetes.io/docs/concepts/>  
<https://www.redhat.com/en/topics/containers/what-is-kubernetes>  
<https://www.redhat.com/en/topics/containers/kubernetes-architecture>  
<https://minikube.sigs.k8s.io/docs/start/>  
<https://kubernetes.io/docs/tasks/tools/install-kubectl-linux/#install-using-native-package-management>  
<https://kubernetes.io/docs/tasks/tools/included/optional-kubectl-configs-bash-linux/>  
<https://phoenixnap.com/kb/how-to-install-kubernetes-on-centos>

# Chapter 4

## Network Layer:

# Data Plane

A note on the use of these PowerPoint slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part.

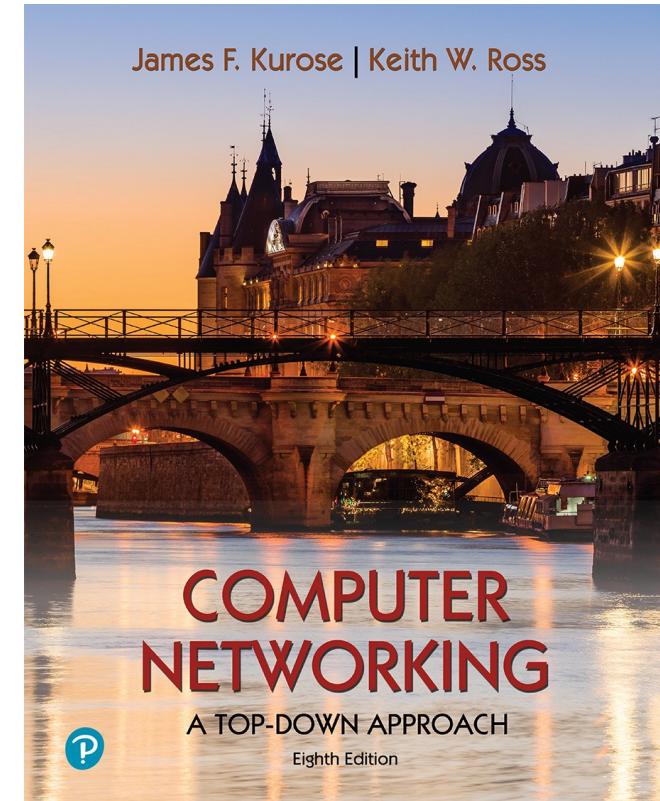
In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2020  
J.F Kurose and K.W. Ross, All Rights Reserved



**Computer Networking: A  
Top-Down Approach**  
8<sup>th</sup> edition  
Jim Kurose, Keith Ross  
Pearson, 2020

# Network layer: our goals

- understand principles behind network layer services, focusing on data plane:
  - network layer service models
  - forwarding versus routing
  - how a router works
  - addressing
  - generalized forwarding
  - Internet architecture
- instantiation, implementation in the Internet
  - IP protocol
  - NAT, middleboxes

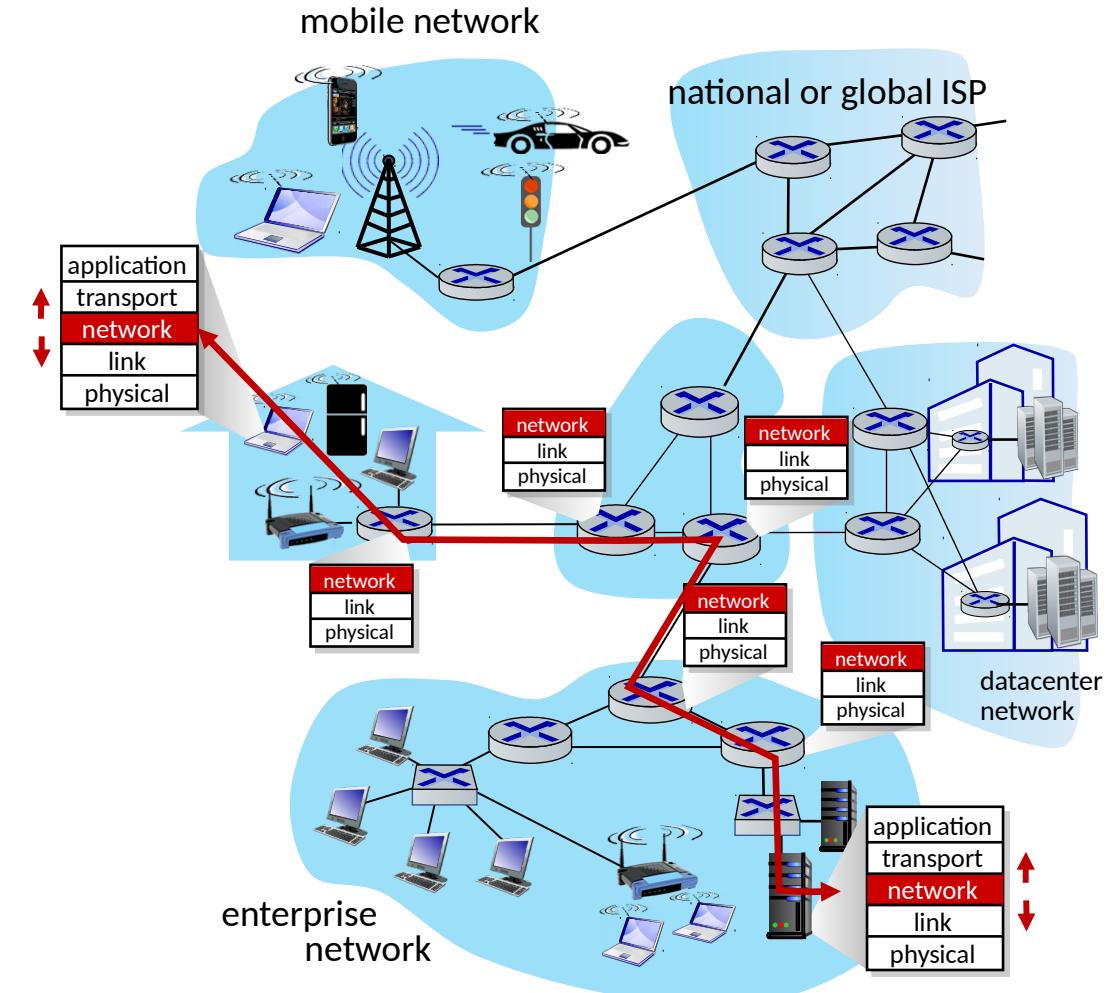
# Network layer: “data plane” roadmap

- Network layer: overview
  - data plane
  - control plane
- What's inside a router
  - input ports, switching, output ports
  - buffer management, scheduling
- IP: the Internet Protocol
  - datagram format
  - addressing
  - network address translation
  - IPv6
- Generalized Forwarding, SDN
  - Match+action
  - OpenFlow: match+action in action
- Middleboxes



# Network-layer services and protocols

- transport segment from sending to receiving host
  - **sender:** encapsulates segments into datagrams, passes to link layer
  - **receiver:** delivers segments to transport layer protocol
- network layer protocols in *every Internet device*: hosts, routers
- **routers:**
  - examines header fields in all IP datagrams passing through it
  - moves datagrams from input ports to output ports to transfer datagrams along end-end path



# Two key network-layer functions

network-layer functions:

- *forwarding*: move packets from a router's input link to appropriate router output link
- *routing*: determine route taken by packets from source to destination
  - *routing algorithms*

analogy: taking a trip

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



forwarding

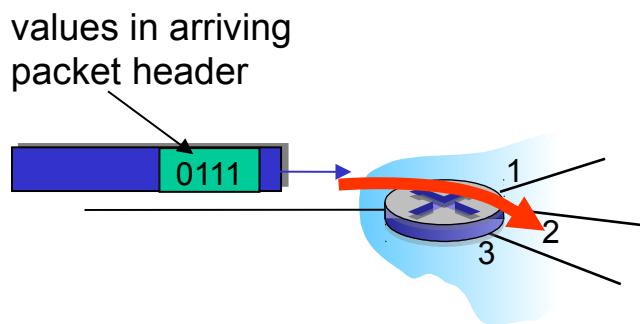


routing

# Network layer: data plane, control plane

## Data plane:

- *local*, per-router function
- determines how datagram arriving on router input port is forwarded to router output port

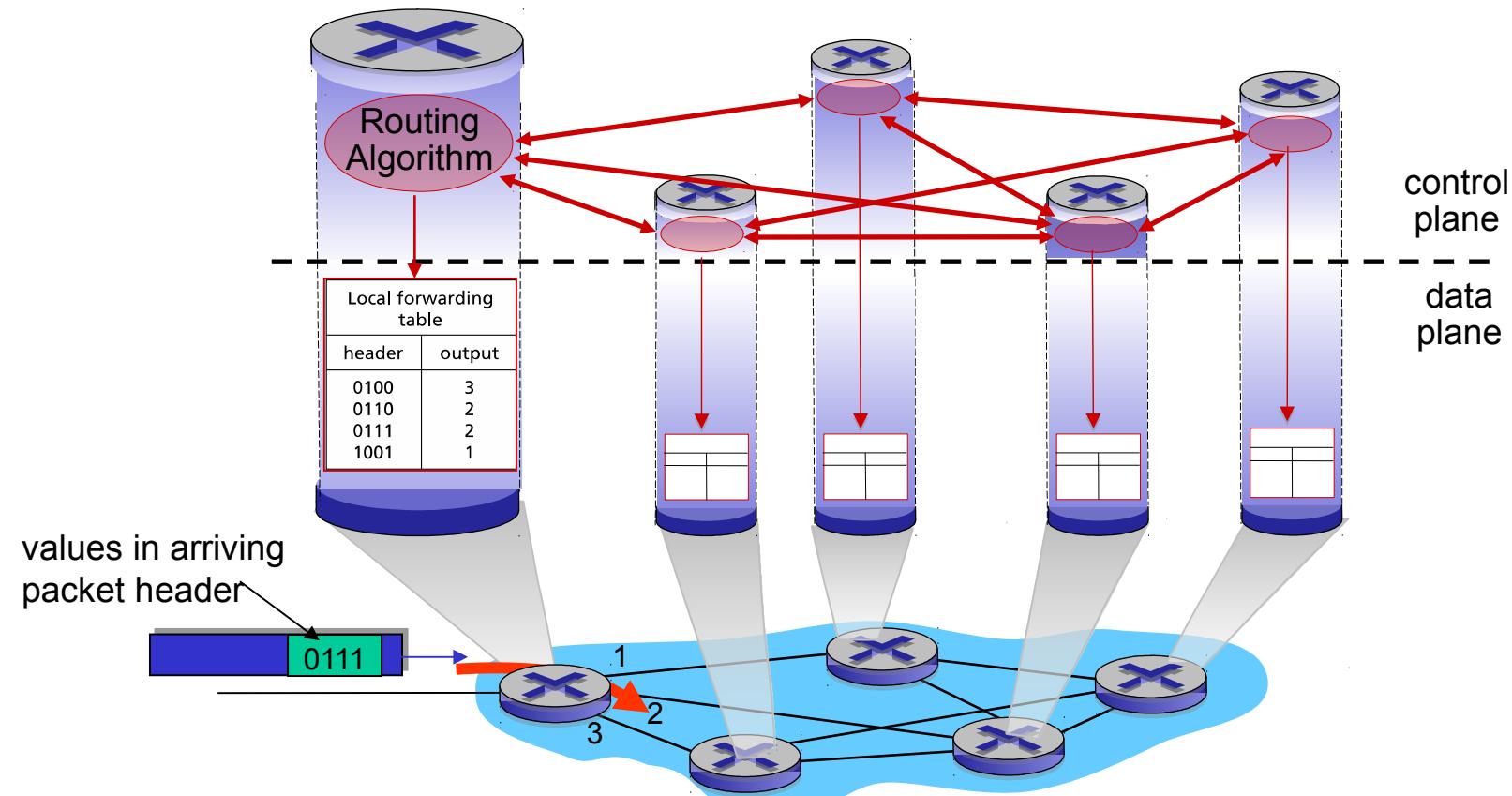


## Control plane

- *network-wide* logic
- determines how datagram is routed among routers along end-end path from source host to destination host
- two control-plane approaches:
  - *traditional routing algorithms*: implemented in routers
  - *software-defined networking (SDN)*: implemented in (remote) servers

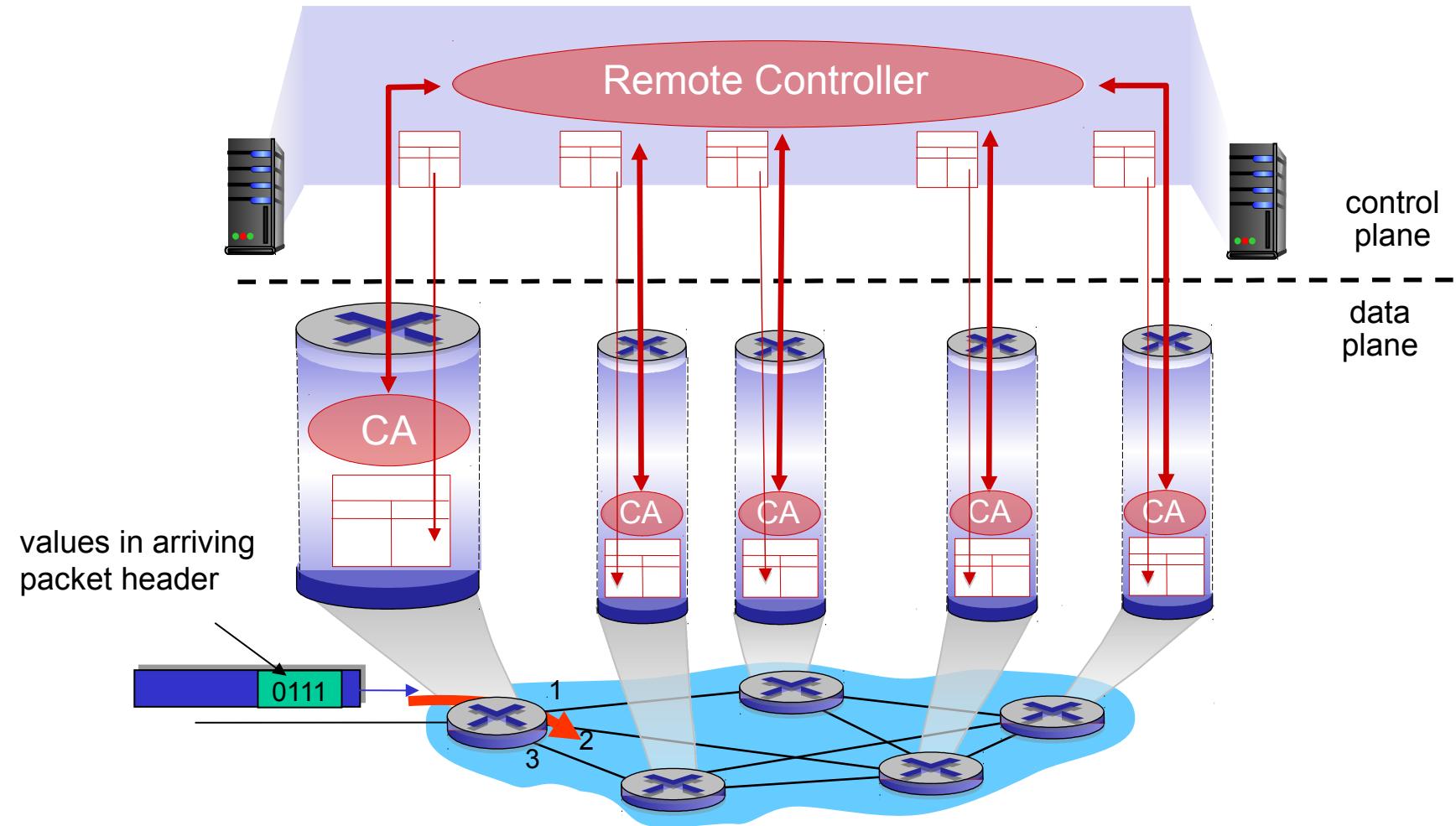
# Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane



# Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers



# Network service model

Q: What *service model* for “channel” transporting datagrams from sender to receiver?

example services for  
*individual* datagrams:

- guaranteed delivery
- guaranteed delivery with less than 40 msec delay

example services for a *flow* of datagrams:

- in-order datagram delivery
- guaranteed minimum bandwidth to flow
- restrictions on changes in inter-packet spacing

# Network-layer service model

Network Architecture	Service Model	Quality of Service (QoS) Guarantees ?			
		Bandwidth	Loss	Order	Timing
Internet	best effort	none	no	no	no

Internet “best effort” service model

No guarantees on:

- i. successful datagram delivery to destination
- ii. timing or order of delivery
- iii. bandwidth available to end-end flow

# Network-layer service model

Network Architecture	Service Model	Quality of Service (QoS) Guarantees ?			
		Bandwidth	Loss	Order	Timing
Internet	best effort	none	no	no	no
ATM	Constant Bit Rate	Constant rate	yes	yes	yes
ATM	Available Bit Rate	Guaranteed min	no	yes	no
Internet	Intserv Guaranteed (RFC 1633)	yes	yes	yes	yes
Internet	Diffserv (RFC 2475)	possible	possibly	possibly	no

# Reflections on best-effort service:

- simplicity of mechanism has allowed Internet to be widely deployed adopted
- sufficient provisioning of bandwidth allows performance of real-time applications (e.g., interactive voice, video) to be “good enough” for “most of the time”
- replicated, application-layer distributed services (datacenters, content distribution networks) connecting close to clients’ networks, allow services to be provided from multiple locations
- congestion control of “elastic” services helps

*It's hard to argue with success of best-effort service model*

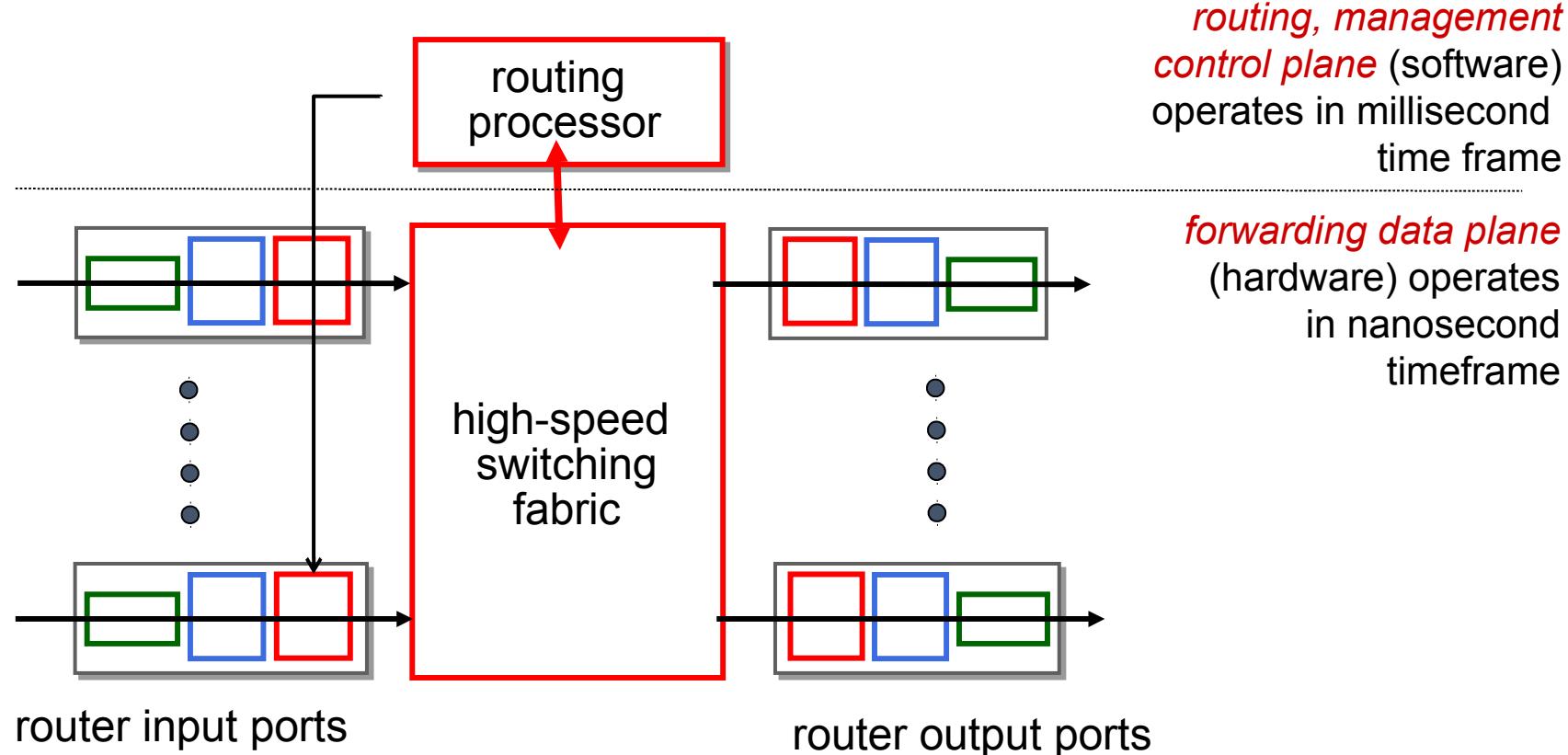
# Network layer: “data plane” roadmap

- Network layer: overview
  - data plane
  - control plane
- What's inside a router
  - input ports, switching, output ports
  - buffer management, scheduling
- IP: the Internet Protocol
  - datagram format
  - addressing
  - network address translation
  - IPv6
- Generalized Forwarding, SDN
  - Match+action
  - OpenFlow: match+action in action
- Middleboxes

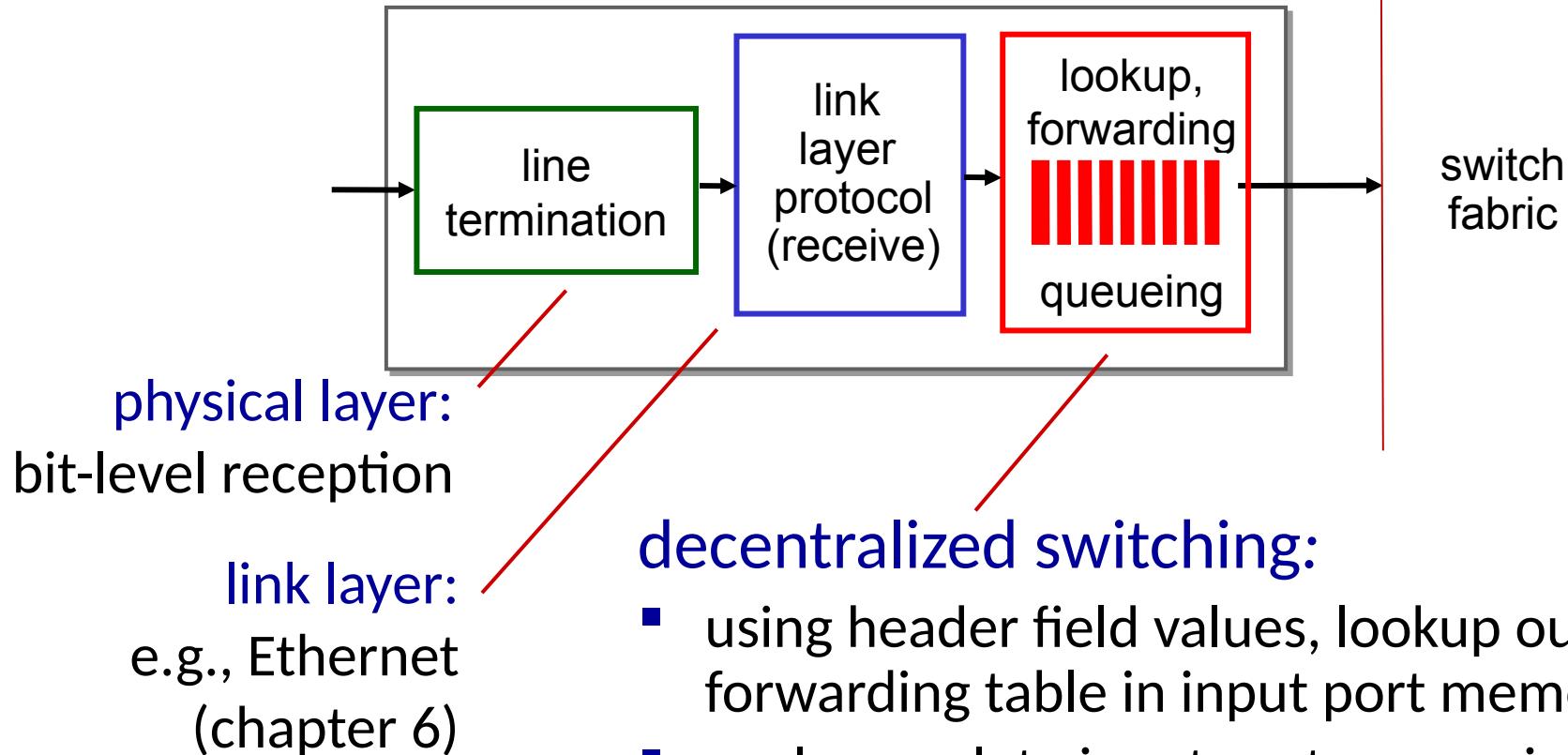


# Router architecture overview

high-level view of generic router architecture:



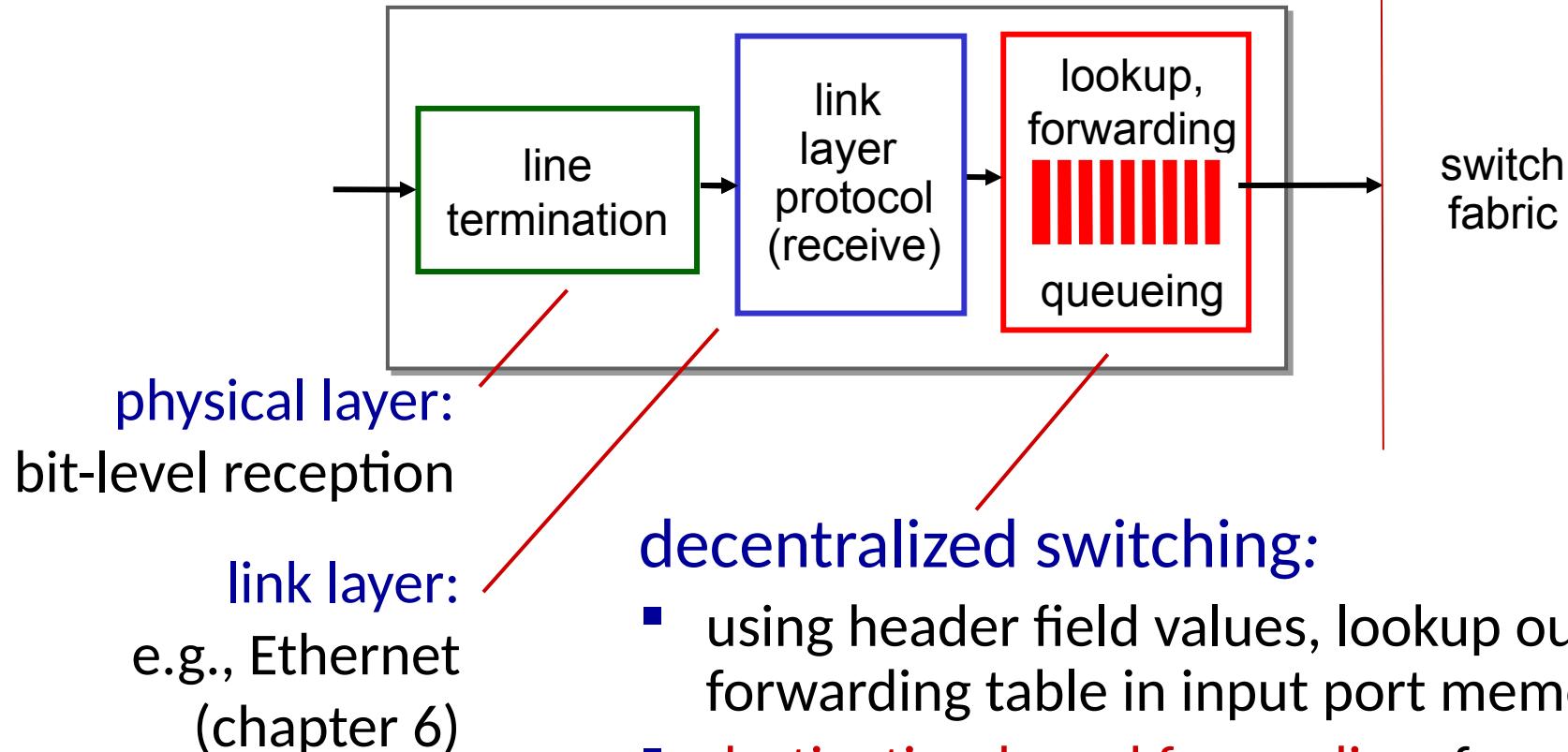
# Input port functions



## decentralized switching:

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

# Input port functions



## decentralized switching:

- using header field values, lookup output port using forwarding table in input port memory ("*match plus action*")
- **destination-based forwarding:** forward based only on destination IP address (traditional)
- **generalized forwarding:** forward based on any set of header field values

# Destination-based forwarding

<i>forwarding table</i>	
Destination Address Range	Link Interface
11001000 00010111 00010000 00000000 through	0
11001000 00010111 00010000 00000100 through	3
11001000 00010111 00010000 00000111	
11001000 00010111 00011000 11111111	
11001000 00010111 00011001 00000000 through	2
11001000 00010111 00011111 11111111	
otherwise	3

Q: but what happens if ranges don't divide up so nicely?

# Longest prefix matching

## longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range	Link interface
11001000 00010111 00010*** *****	0
11001000 00010111 00011000 *****	1
11001000 00010111 00011*** *****	2
otherwise	3

examples:

- |          |          |          |          |                  |
|----------|----------|----------|----------|------------------|
| 11001000 | 00010111 | 00010110 | 10100001 | which interface? |
| 11001000 | 00010111 | 00011000 | 10101010 | which interface? |

# Longest prefix matching

## longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range	Link interface
11001000 00010111 00010*****	0
11001000 00010111 00011000 *****	1
11001000 1 00011*** *****	2
otherwise	3

examples:

11001000 00010111 00010110 10100001 which interface?

11001000 00010111 00011000 10101010 which interface?

# Longest prefix matching

## longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range	Link interface
11001000 00010111 00010**** *****	0
11001000 00010111 000110000 *****	1
11001000 00010111 00011***** *****	2
otherwise	3

match!

examples:

11001000 00010111 00010110 10100001	which interface?
11001000 00010111 00011000 10101010	which interface?

# Longest prefix matching

## longest prefix match

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

Destination Address Range	Link interface
11001000 00010111 00010*** *****	0
11001000 00010111 00011000 *****	1
11001000 00010111 00011*** *****	2
otherwise	3

examples:

↑  
**match!**

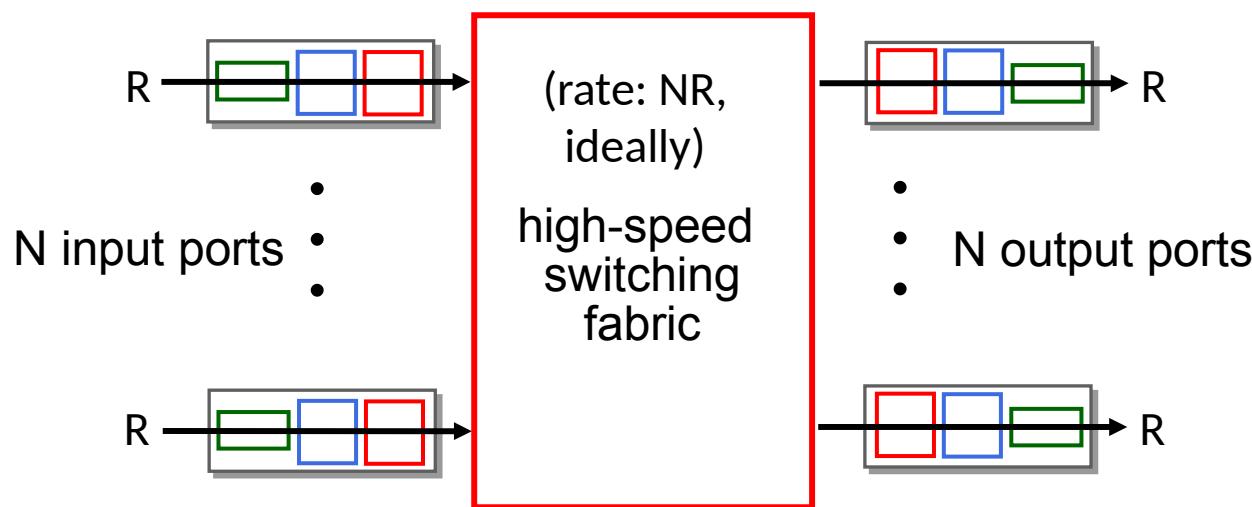
11001000	00010111	00010110	10100001	which interface?
11001000	00010111	00011000	10101010	which interface?

# Longest prefix matching

- we'll see *why* longest prefix matching is used shortly, when we study addressing
- longest prefix matching: often performed using ternary content addressable memories (TCAMs)
  - *content addressable*: present address to TCAM: retrieve address in one clock cycle, regardless of table size
  - Cisco Catalyst: ~1M routing table entries in TCAM

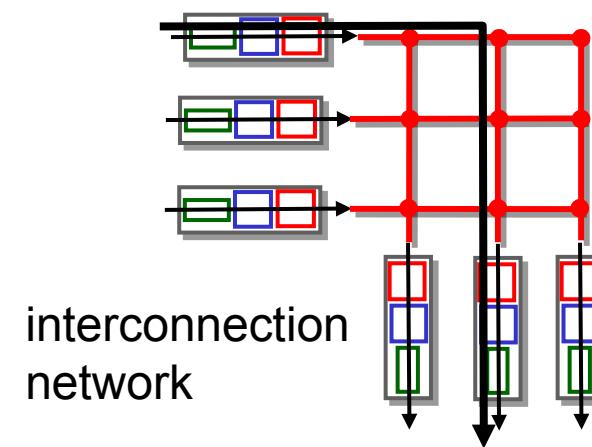
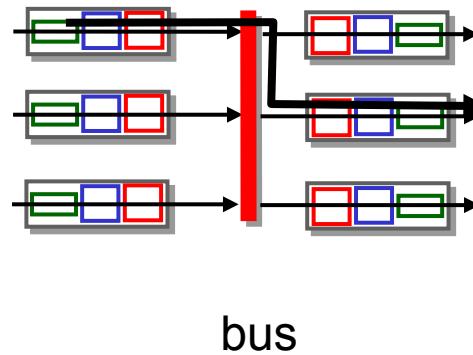
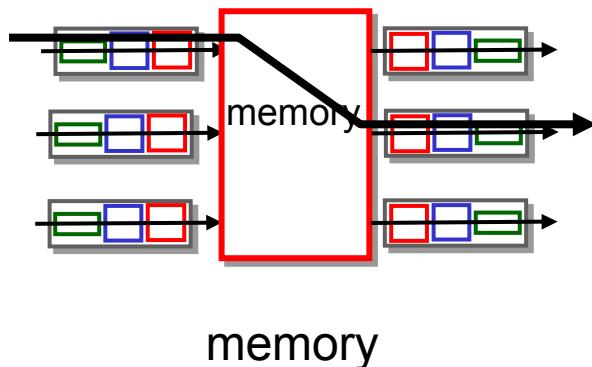
# Switching fabrics

- transfer packet from input link to appropriate output link
- **switching rate:** rate at which packets can be transferred from inputs to outputs
  - often measured as multiple of input/output line rate
  - $N$  inputs: switching rate  $N$  times line rate desirable



# Switching fabrics

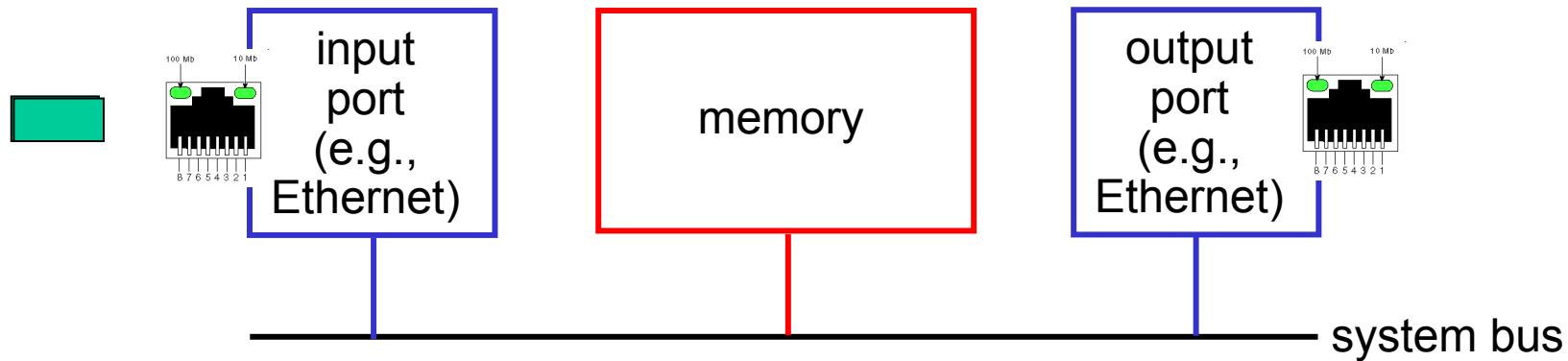
- transfer packet from input link to appropriate output link
- **switching rate:** rate at which packets can be transferred from inputs to outputs
  - often measured as multiple of input/output line rate
  - $N$  inputs: switching rate  $N$  times line rate desirable
- three major types of switching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



# Switching via memory

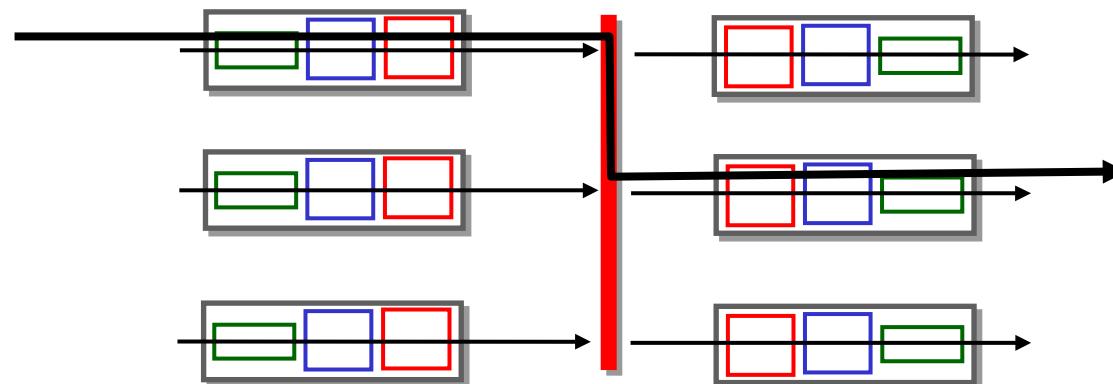
first generation routers:

- 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)



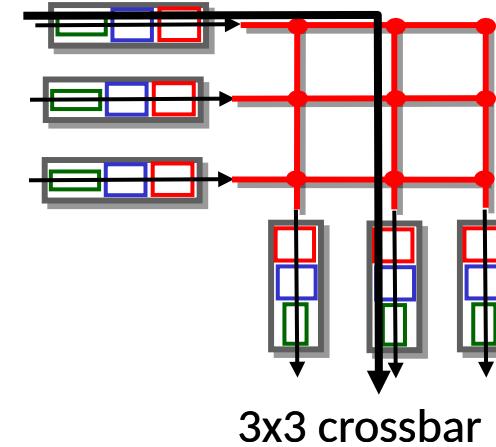
# Switching via a bus

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

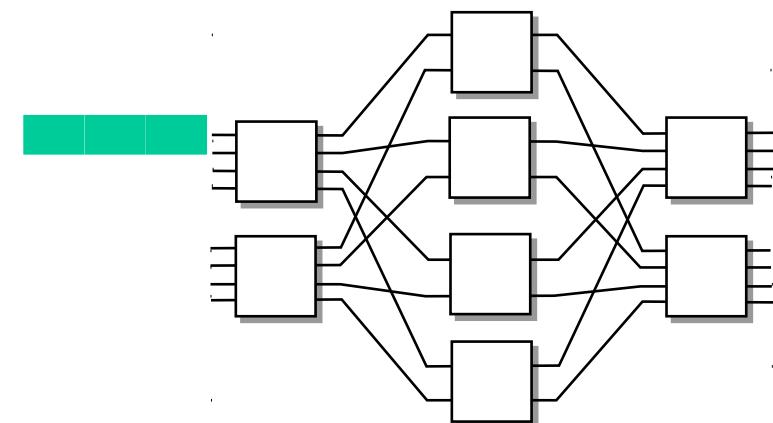


# Switching via interconnection network

- Crossbar, Clos networks, other interconnection nets initially developed to connect processors in multiprocessor
- multistage switch:  $n \times n$  switch from multiple stages of smaller switches
- exploiting parallelism:
  - fragment datagram into fixed length cells on entry
  - switch cells through the fabric, reassemble datagram at exit



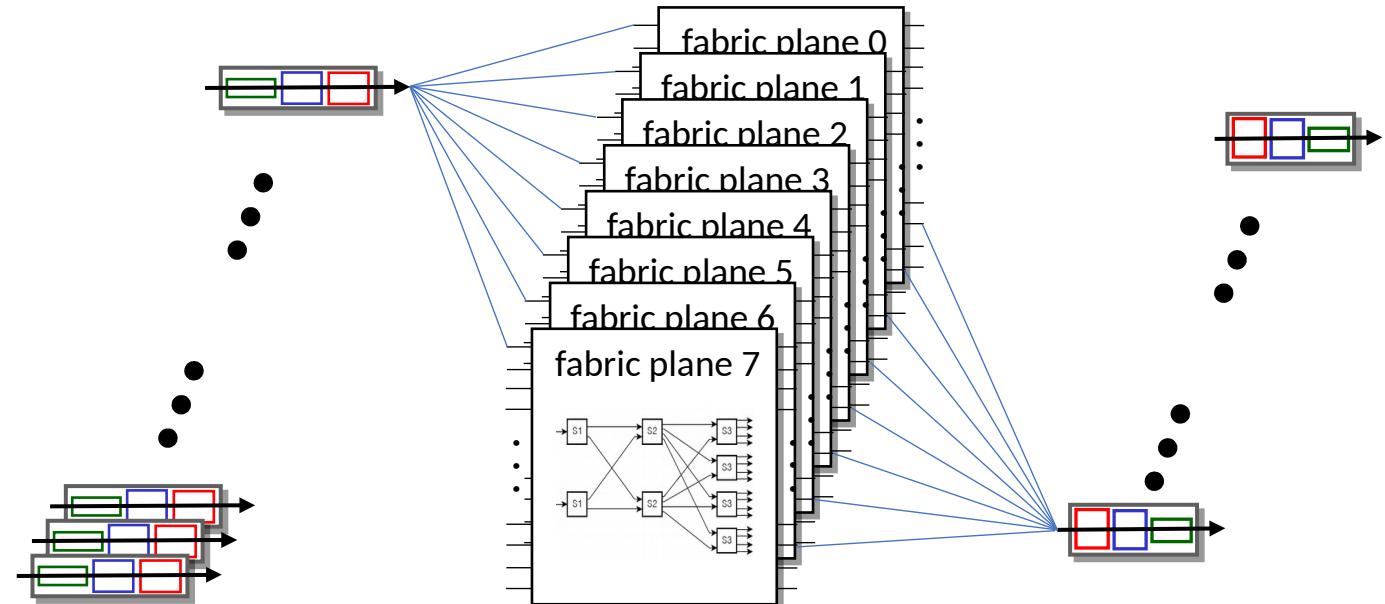
3x3 crossbar



8x8 multistage switch  
built from smaller-sized switches

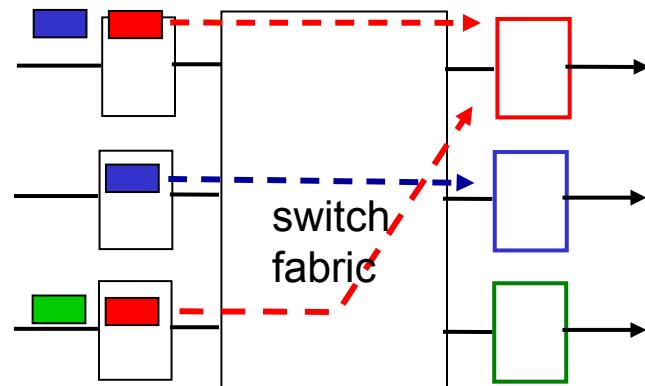
# Switching via interconnection network

- scaling, using multiple switching “planes” in parallel:
  - speedup, scaleup via parallelism
- Cisco CRS router:
  - basic unit: 8 switching planes
  - each plane: 3-stage interconnection network
  - up to 100's Tbps switching capacity

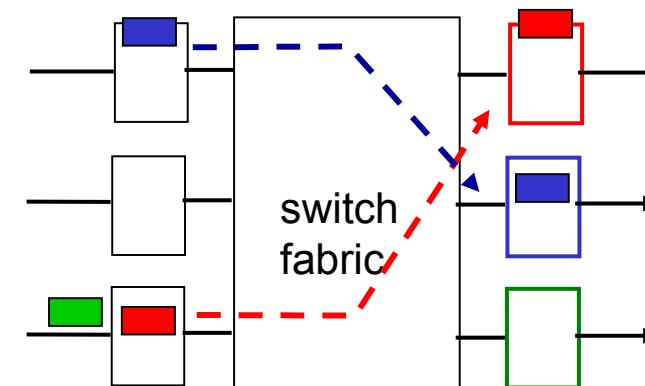


# Input port queuing

- If switch fabric slower than input ports combined -> queueing may occur at input queues
  - queueing 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 datagram can be transferred. lower red packet is *blocked*

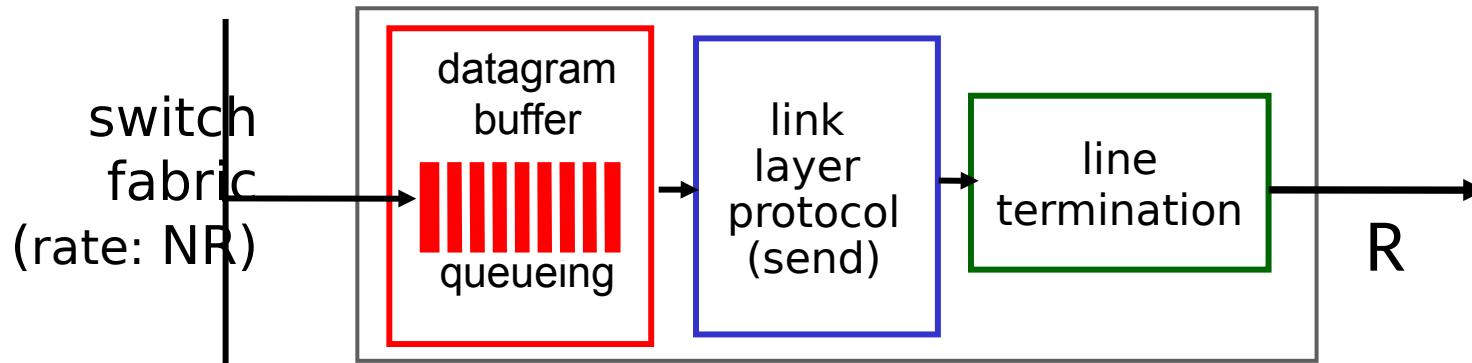


one packet time later: green packet experiences HOL blocking

# Output port queuing



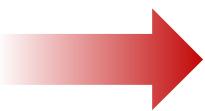
This is a really important slide



- *Buffering* required when datagrams arrive from fabric faster than link transmission rate. *Drop policy*: which datagrams to drop if no free buffers?
- *Scheduling discipline* chooses among queued datagrams for transmission

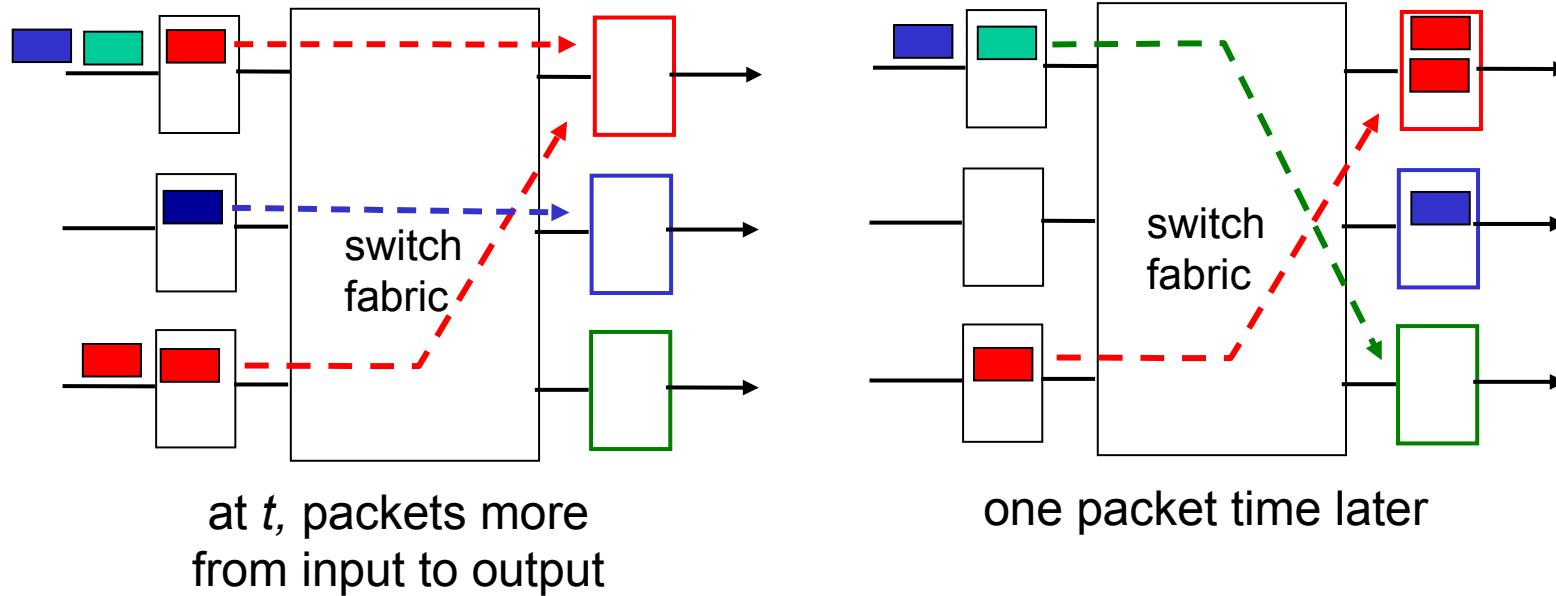


Datagrams can be lost due to congestion, lack of buffers



Priority scheduling – who gets best performance, network neutrality

# Output port queuing



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

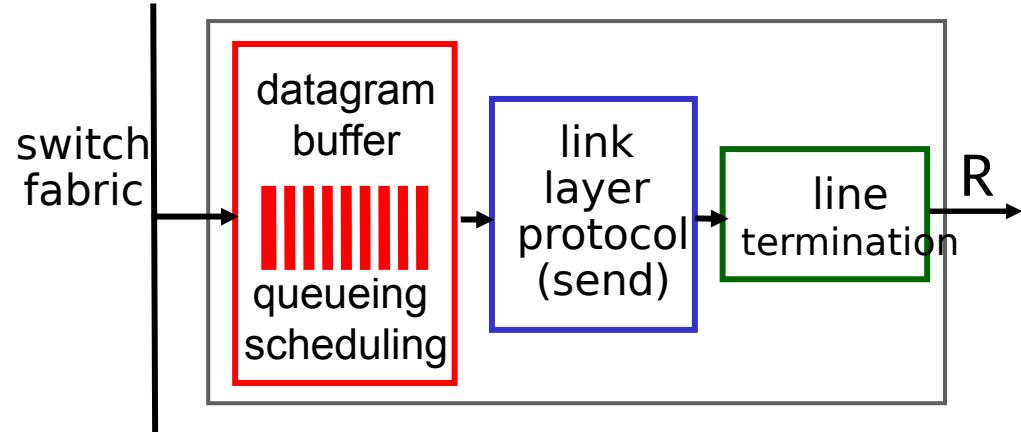
# How much buffering?

- RFC 3439 rule of thumb: average buffering equal to “typical” RTT (say 250 msec) times link capacity C
  - e.g., C = 10 Gbps link: 2.5 Gbit buffer
- more recent recommendation: with  $N$  flows, buffering equal to

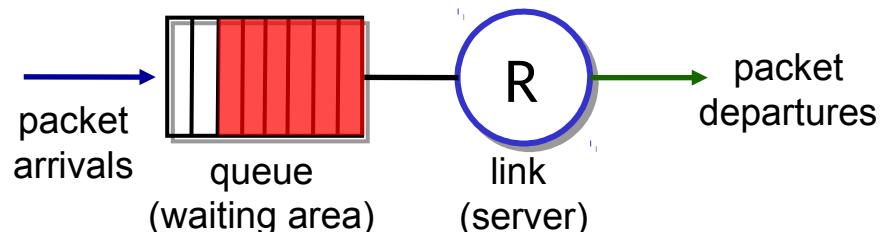
$$\frac{RTT \cdot C}{\sqrt{N}}$$

- but *too* much buffering can increase delays (particularly in home routers)
  - long RTTs: poor performance for realtime apps, sluggish TCP response
  - recall delay-based congestion control: “keep bottleneck link just full enough (busy) but no fuller”

# Buffer Management



Abstraction: queue



buffer management:

- **drop:** which packet to add, drop when buffers are full
  - **tail drop:** drop arriving packet
  - **priority:** drop/remove on priority basis
- **marking:** which packets to mark to signal congestion (ECN, RED)

# Packet Scheduling: FCFS

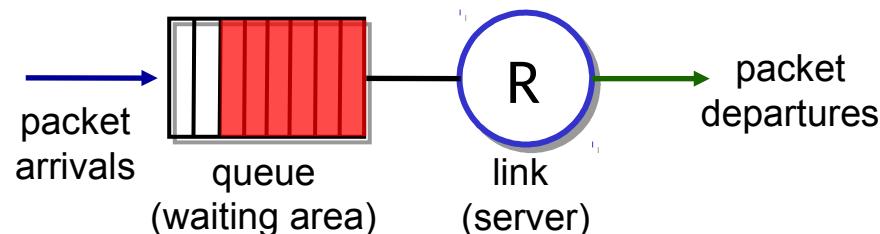
**packet scheduling:** deciding which packet to send next on link

- first come, first served
- priority
- round robin
- weighted fair queueing

**FCFS:** packets transmitted in order of arrival to output port

- also known as: First-in-first-out (FIFO)
- real world examples?

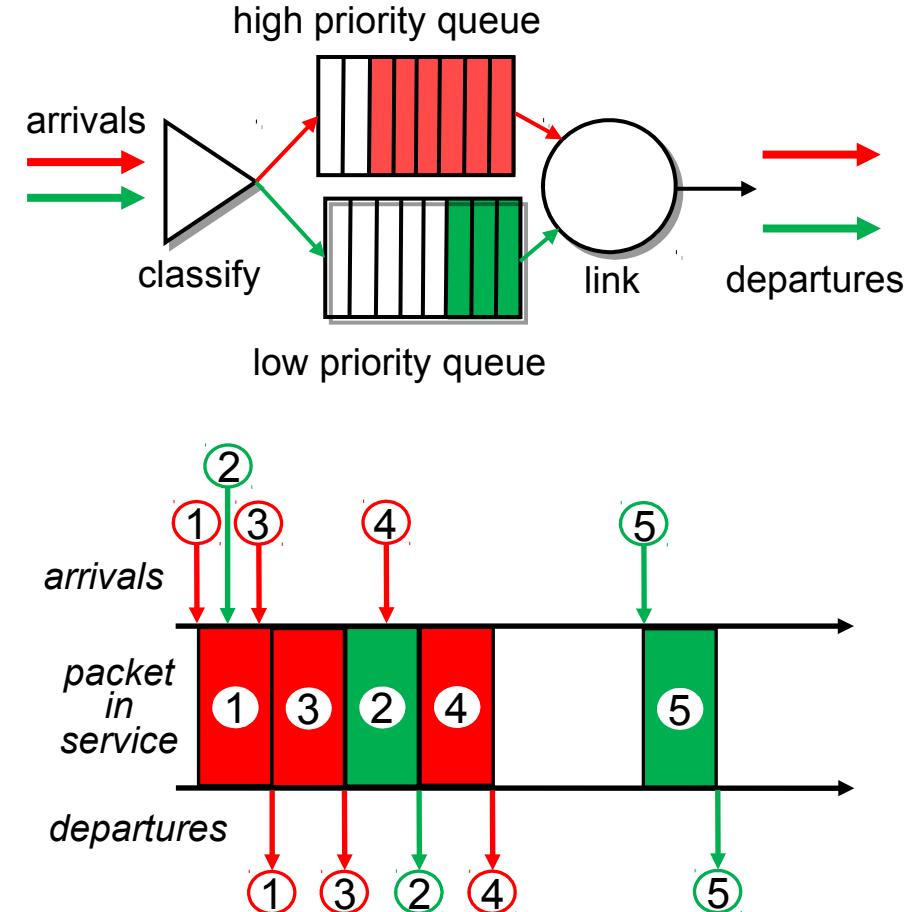
Abstraction: queue



# Scheduling policies: priority

## *Priority scheduling:*

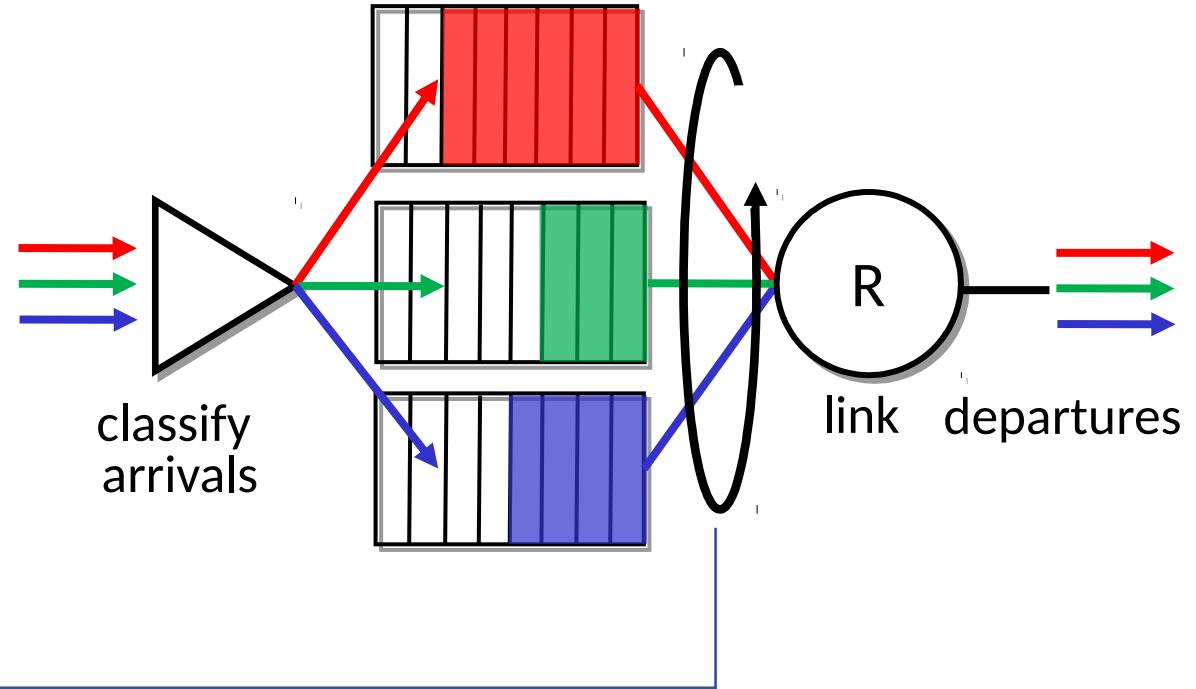
- arriving traffic classified, queued by class
  - any header fields can be used for classification
- send packet from highest priority queue that has buffered packets
  - FCFS within priority class



# Scheduling policies: round robin

*Round Robin (RR) scheduling:*

- arriving traffic classified, queued by class
  - any header fields can be used for classification
- server cyclically, repeatedly scans class queues, sending one complete packet from each class (if available) in turn



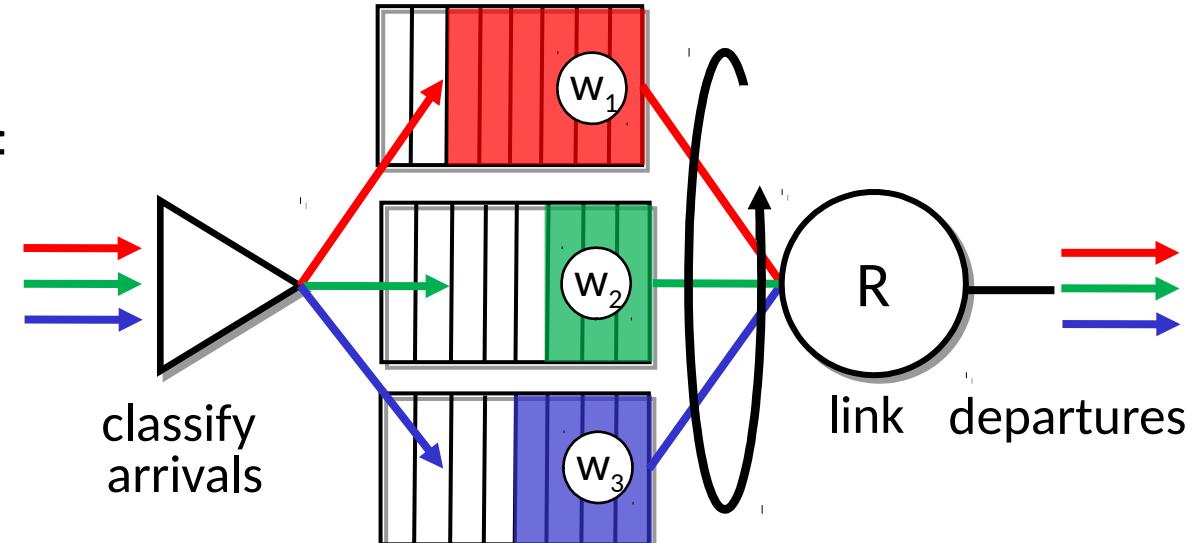
# Scheduling policies: weighted fair queueing

*Weighted Fair Queueing (WFQ):*

- generalized Round Robin
- each class,  $i$ , has weight,  $w_i$ , and gets weighted amount of service in each cycle:

$$\frac{w_i}{\sum_j w_j}$$

- minimum bandwidth guarantee (per-traffic-class)



# Sidebar: Network Neutrality

What is network neutrality?

- *technical*: how an ISP should share/allocation its resources
  - packet scheduling, buffer management are the *mechanisms*
- *social, economic* principles
  - protecting free speech
  - encouraging innovation, competition
- enforced *legal* rules and policies

*Different countries have different “takes” on network neutrality*

# Sidebar: Network Neutrality

2015 US FCC Order on *Protecting and Promoting an Open Internet*: three “clear, bright line” rules:

- **no blocking** ... “shall not block lawful content, applications, services, or non-harmful devices, subject to reasonable network management.”
- **no throttling** ... “shall not impair or degrade lawful Internet traffic on the basis of Internet content, application, or service, or use of a non-harmful device, subject to reasonable network management.”
- **no paid prioritization.** ... “shall not engage in paid prioritization”

# ISP: telecommunications or information service?

Is an ISP a “telecommunications service” or an “information service” provider?

- the answer *really* matters from a regulatory standpoint!

US Telecommunication Act of 1934 and 1996:

- *Title II*: imposes “common carrier duties” on *telecommunications services*: reasonable rates, non-discrimination and requires regulation
- *Title I*: applies to *information services*:
  - no common carrier duties (*not regulated*)
  - but grants FCC authority “... as may be necessary in the execution of its functions”<sup>4</sup>

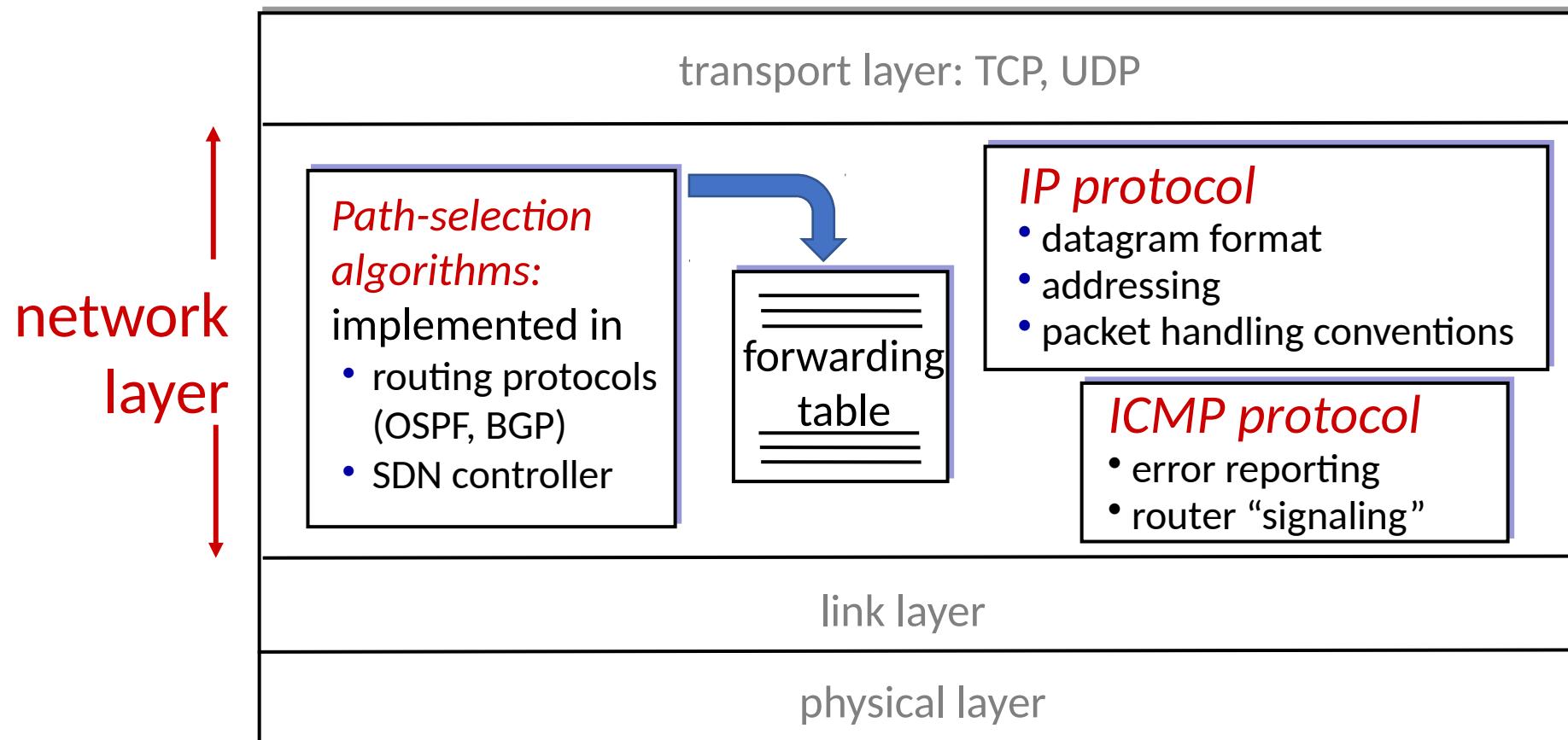
# Network layer: “data plane” roadmap

- Network layer: overview
  - data plane
  - control plane
- What's inside a router
  - input ports, switching, output ports
  - buffer management, scheduling
- IP: the Internet Protocol
  - datagram format
  - addressing
  - network address translation
  - IPv6
- Generalized Forwarding, SDN
  - match+action
  - OpenFlow: match+action in action
- Middleboxes

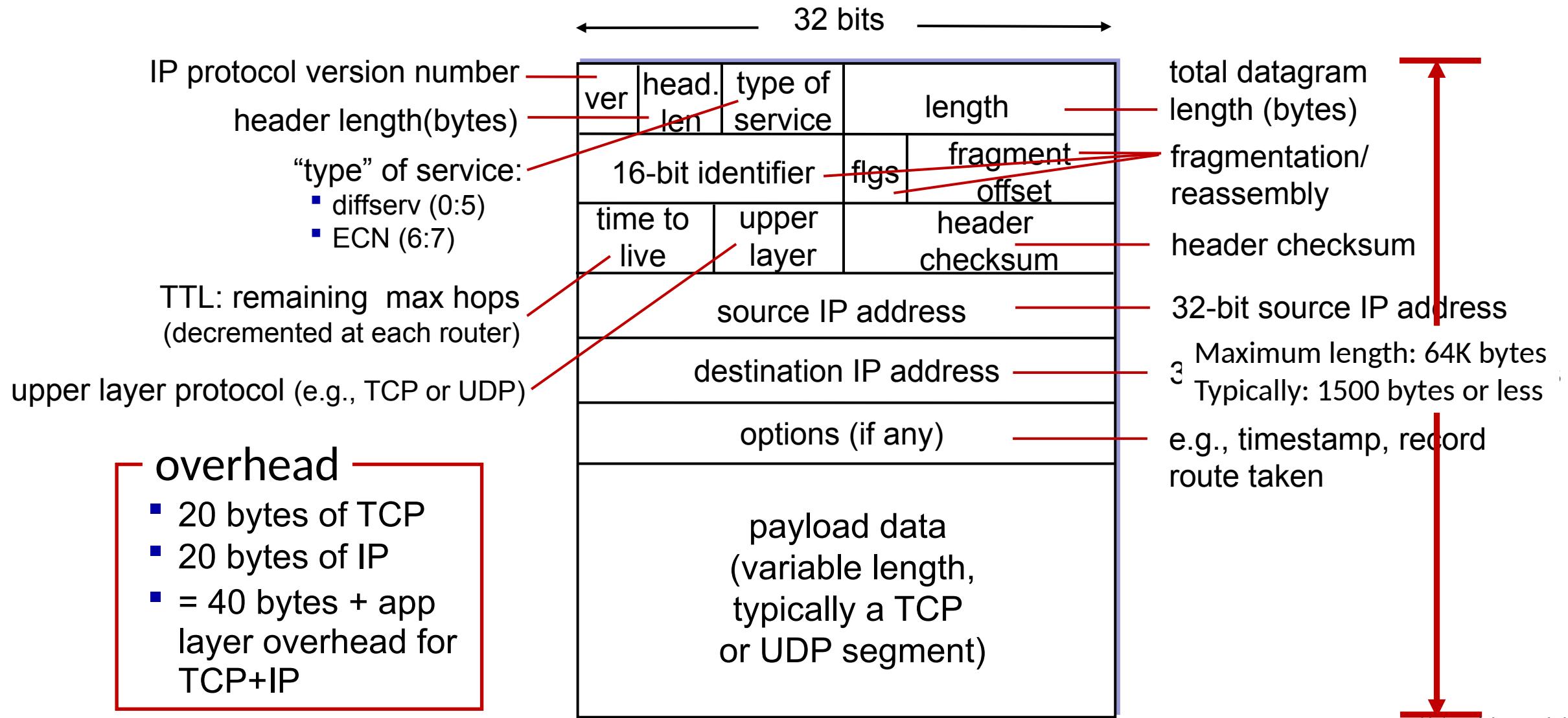


# Network Layer: Internet

host, router network layer functions:

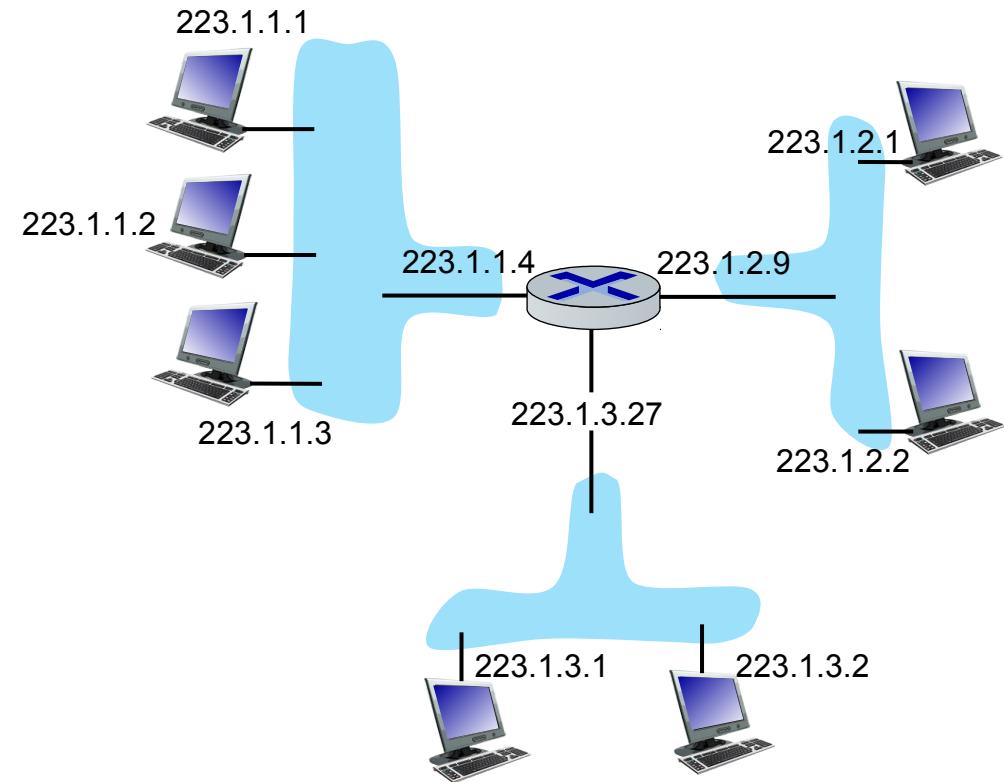


# IP Datagram format



# IP addressing: introduction

- **IP address:** 32-bit identifier associated with each host or router *interface*
- **interface:** connection between host/router and physical link
  - router's typically have multiple interfaces
  - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)

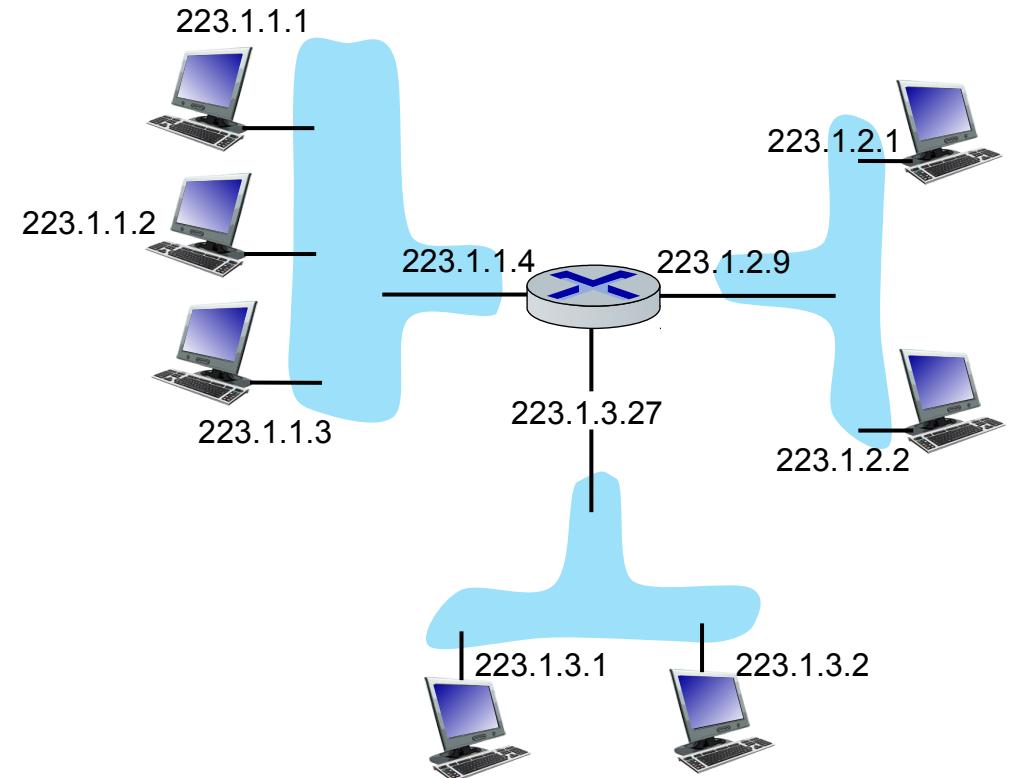


dotted-decimal IP address notation:

223.1.1.1 =  $\underbrace{11011111}_{223} \underbrace{00000001}_{1} \underbrace{00000001}_{1} \underbrace{00000001}_{1}$

# IP addressing: introduction

- **IP address:** 32-bit identifier associated with each host or router *interface*
- **interface:** connection between host/router and physical link
  - router's typically have multiple interfaces
  - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)



dotted-decimal IP address notation:

223.1.1.1 =  $\underbrace{11011111}_{223} \underbrace{00000001}_{1} \underbrace{00000001}_{1} \underbrace{00000001}_{1}$

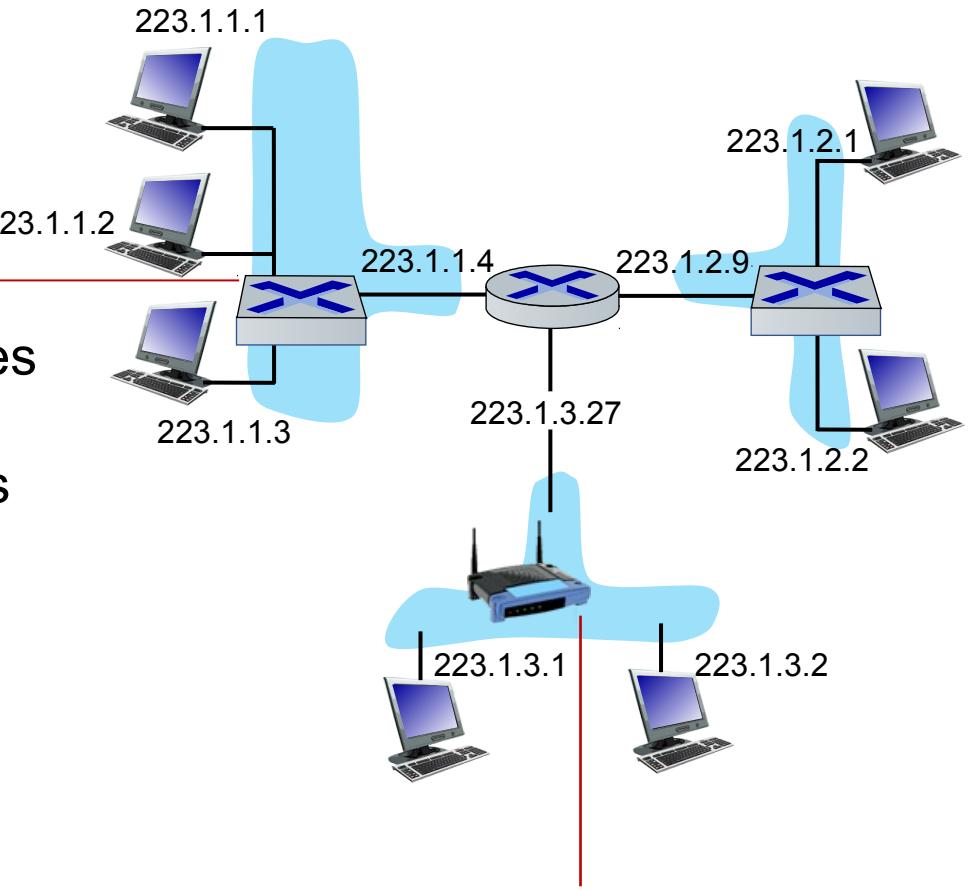
# IP addressing: introduction

**Q:** how are interfaces actually connected?

**A:** we'll learn about that in chapters 6, 7

*For now:* don't need to worry about how one interface is connected to another (with no intervening router)

**A:** wired Ethernet interfaces connected by Ethernet switches



**A:** wireless WiFi interfaces connected by WiFi base station

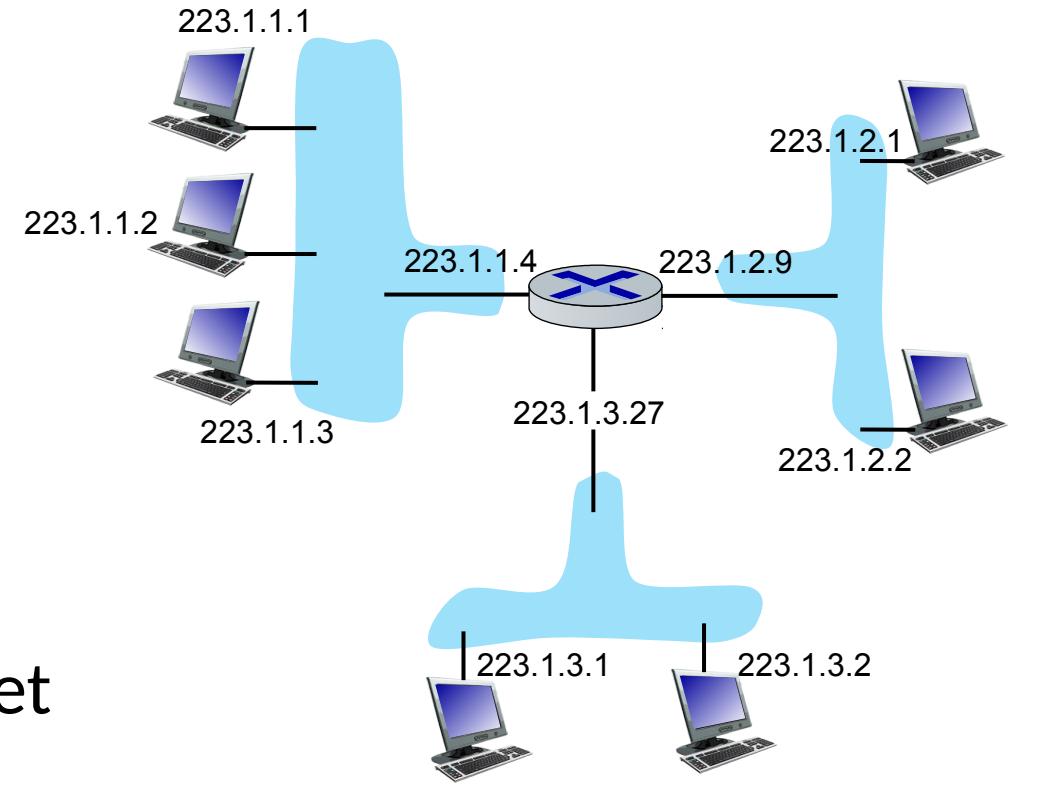
# Subnets

- *What's a subnet ?*

- device interfaces that can physically reach each other  
**without passing through an intervening router**

- **IP addresses have structure:**

- **subnet part:** devices in same subnet have common high order bits
- **host part:** remaining low order bits

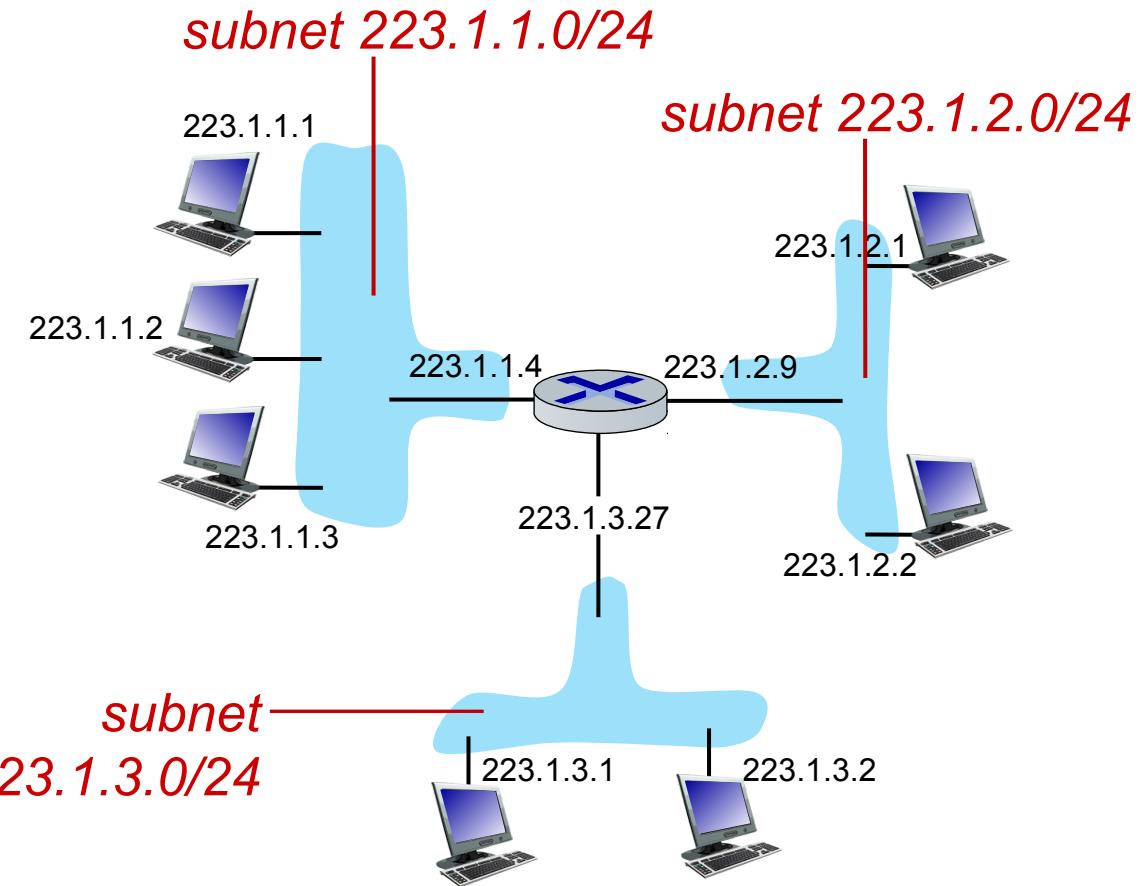


network consisting of 3 subnets

# Subnets

*Recipe for defining subnets:*

- detach each interface from its host or router, creating “islands” of isolated networks
- each isolated network is called a *subnet*

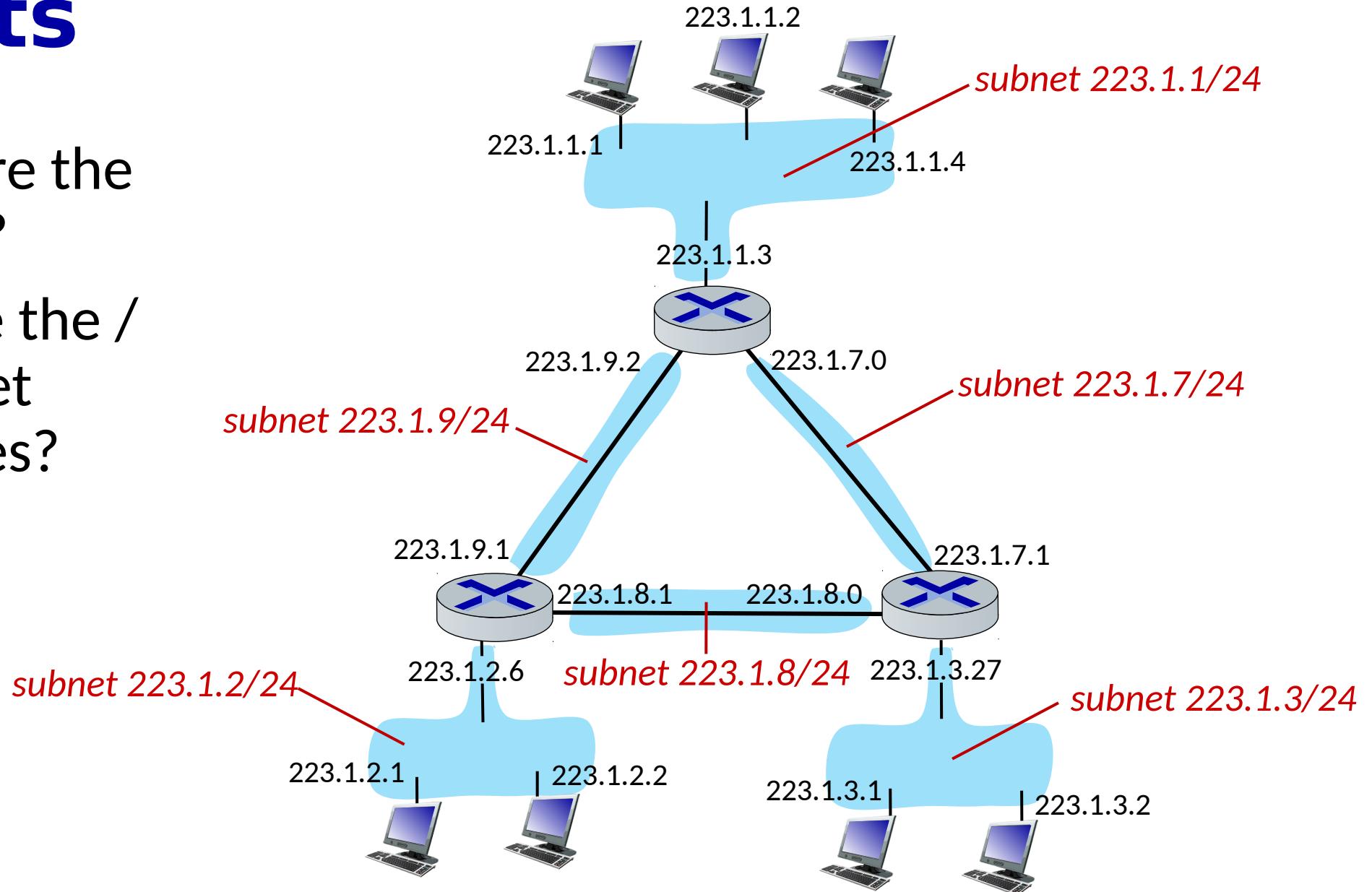


*subnet*  
223.1.3.0/24

subnet mask: /24  
(high-order 24 bits: subnet part of IP address)

# Subnets

- where are the subnets?
- what are the /24 subnet addresses?



# IP addressing: CIDR

CIDR: Classless InterDomain Routing (pronounced “cider”)

- subnet portion of address of arbitrary length
- address format:  $a.b.c.d/x$ , where  $x$  is # bits in subnet portion of address



# IP addresses: how to get one?

That's actually **two** questions:

1. Q: How does a *host* get IP address within its network (host part of address)?
2. Q: How does a *network* get IP address for itself (network part of address)

How does *host* get IP address?

- hard-coded by sysadmin in config file (e.g., /etc/rc.config in UNIX)
- **DHCP: Dynamic Host Configuration Protocol:** dynamically get address from server
  - “plug-and-play”

# DHCP: Dynamic Host Configuration Protocol

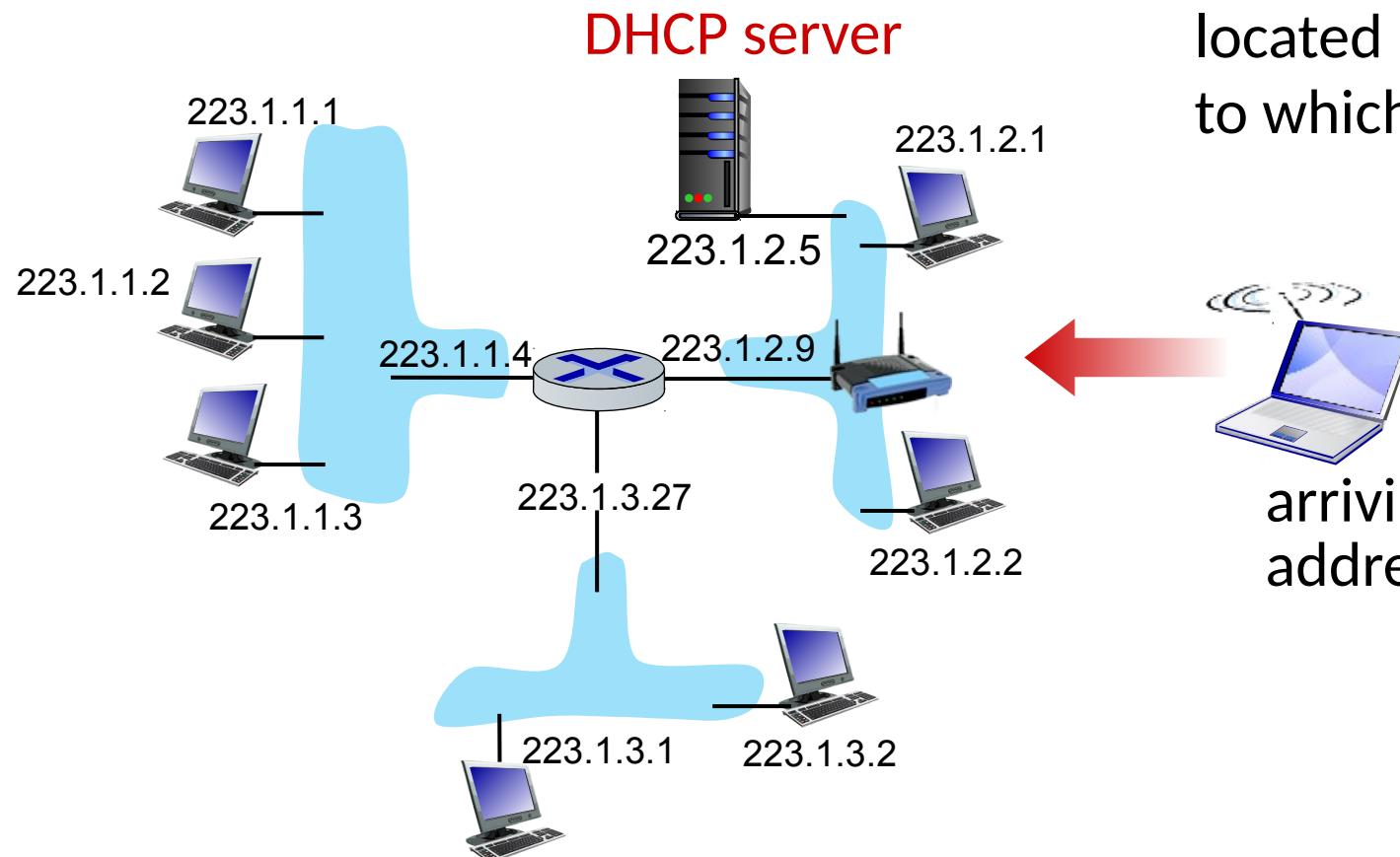
**goal:** host *dynamically* obtains IP address from network server when it “joins” network

- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/on)
- support for mobile users who join/leave network

## DHCP overview:

- host broadcasts **DHCP discover** msg [optional]
- DHCP server responds with **DHCP offer** msg [optional]
- host requests IP address: **DHCP request** msg
- DHCP server sends address: **DHCP ack** msg

# DHCP client-server scenario

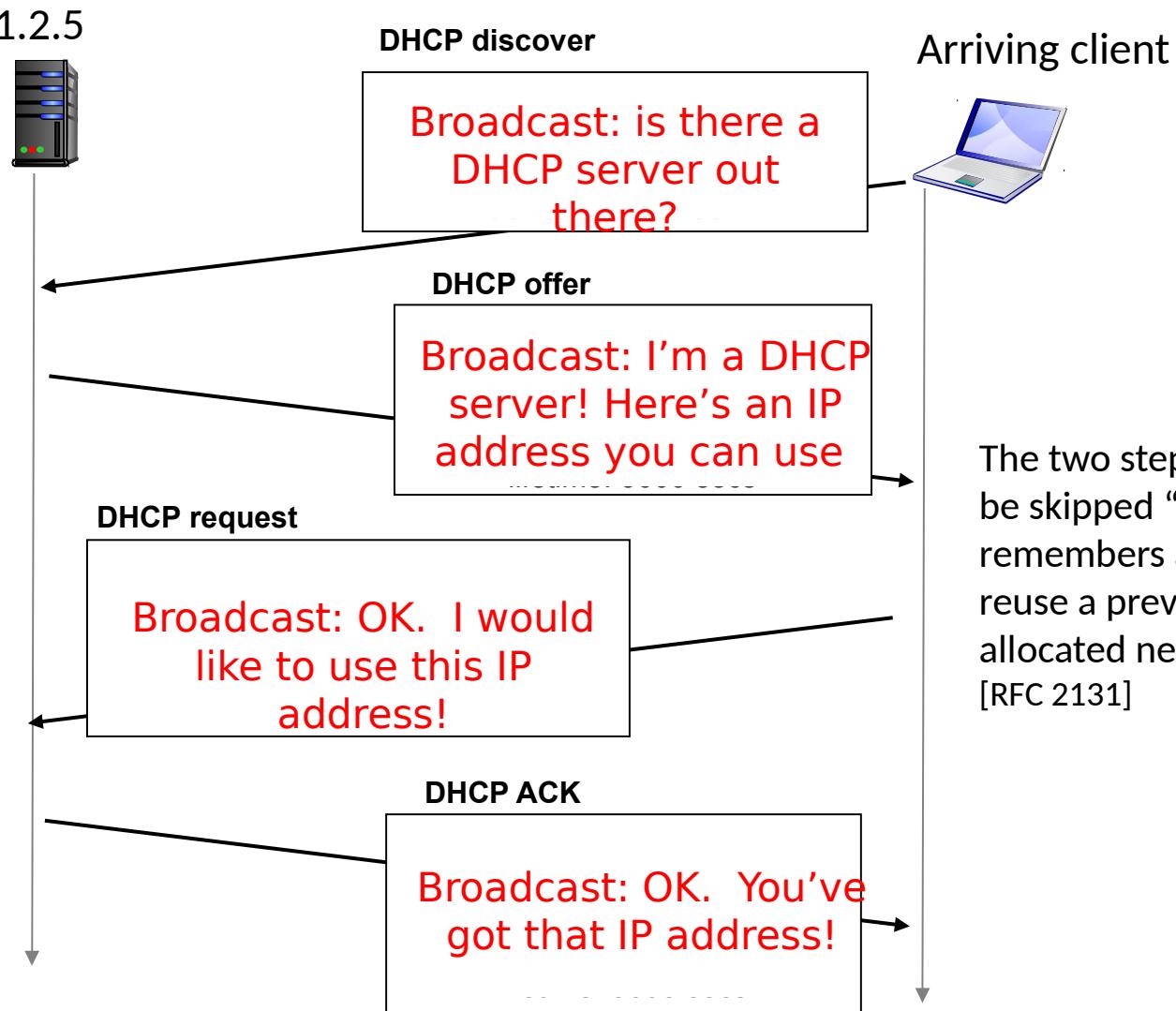


Typically, DHCP server will be co-located in router, serving all subnets to which router is attached

arriving **DHCP client** needs address in this network

# DHCP client-server scenario

DHCP server: 223.1.2.5

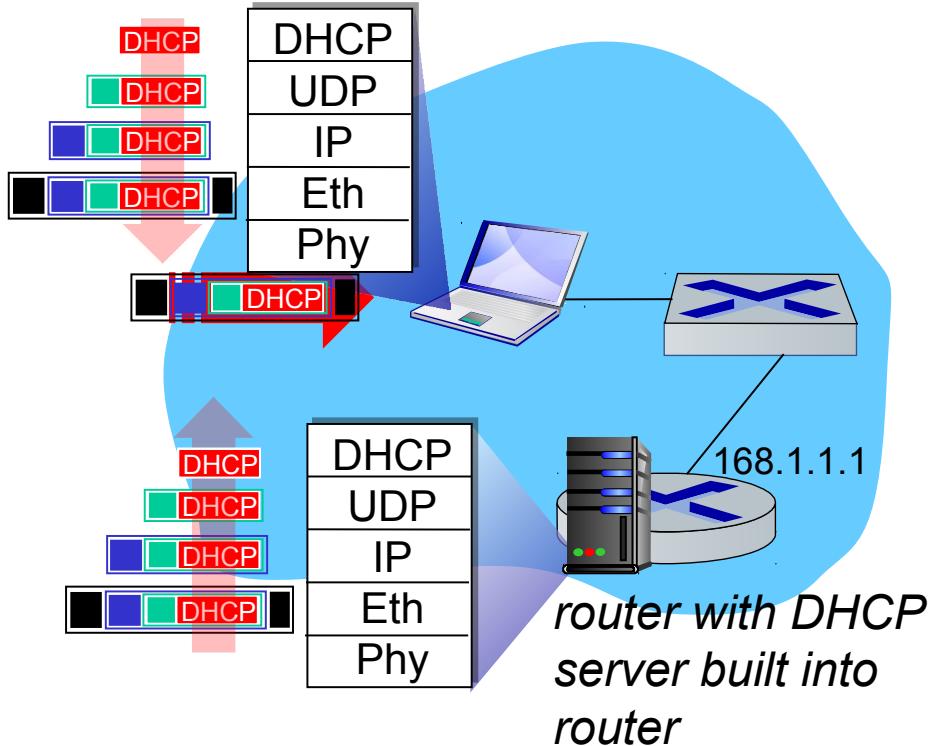


# DHCP: more than IP addresses

DHCP can return more than just allocated IP address on subnet:

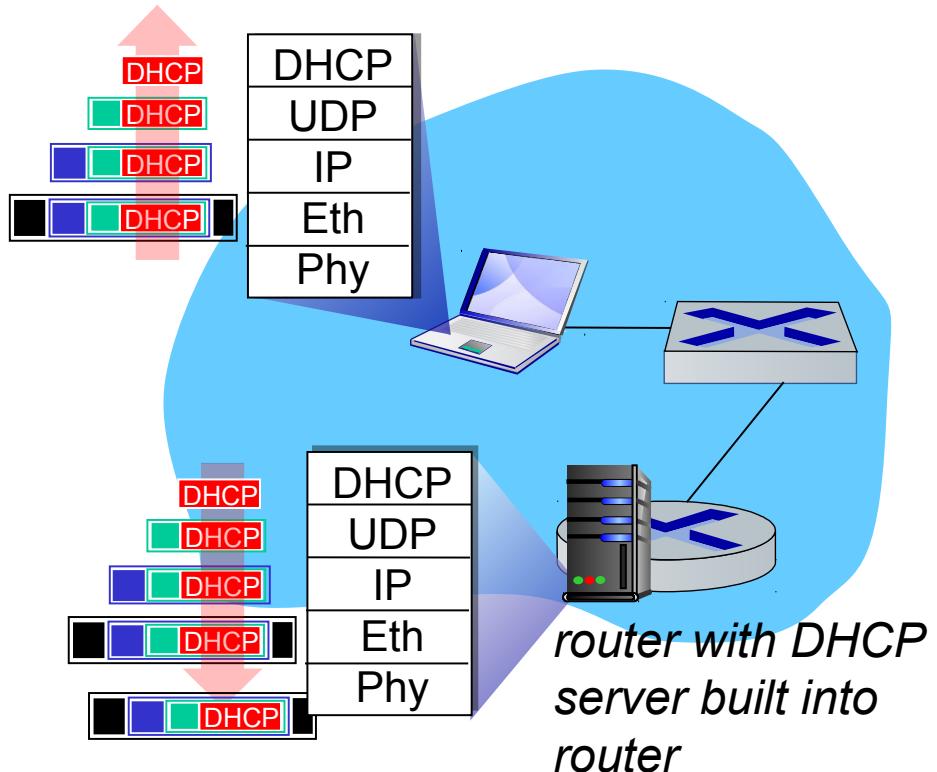
- address of first-hop router for client
- name and IP address of DNS sever
- network mask (indicating network versus host portion of address)

# DHCP: example



- Connecting laptop will use DHCP to get IP address, address of first-hop router, address of DNS server.
- DHCP REQUEST message encapsulated in UDP, encapsulated in IP, encapsulated in Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demux'ed to IP demux'ed, UDP demux'ed to DHCP

# DHCP: example



- DCP server formulates DHCP ACK containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulated DHCP server reply forwarded to client, demuxing up to DHCP at client
- client now knows its IP address, name and IP address of DNS server, IP address of its first-hop router

# IP addresses: how to get one?

**Q:** how does *network* get subnet part of IP address?

**A:** gets allocated portion of its provider ISP's address space

ISP's block      11001000 00010111 00010000 00000000    200.23.16.0/20

ISP can then allocate out its address space in 8 blocks:

Organization 0    11001000 00010111 00010000 00000000    200.23.16.0/23

Organization 1    11001000 00010111 00010010 00000000    200.23.18.0/23

Organization 2    11001000 00010111 00010100 00000000    200.23.20.0/23

...

.....

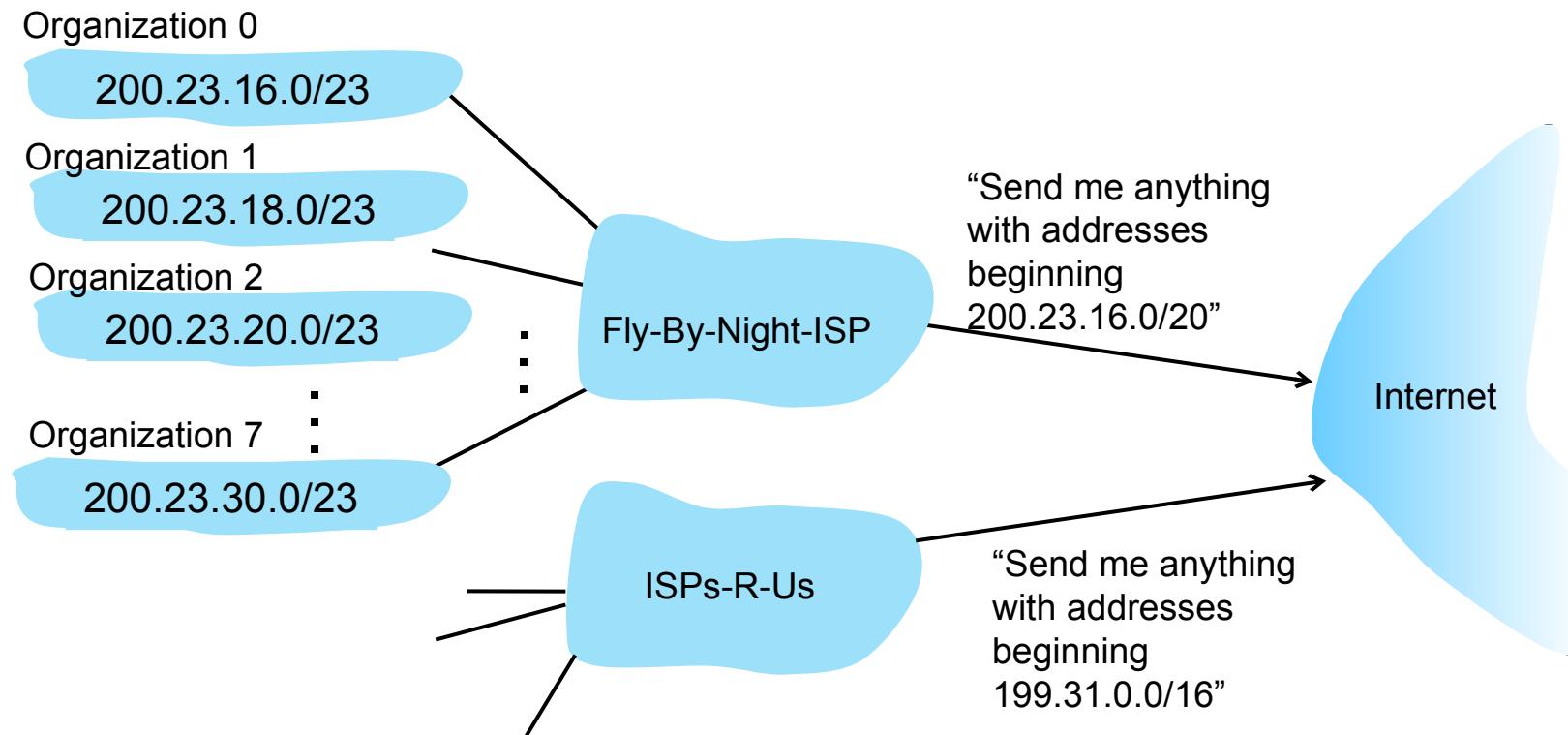
.....

.....

Organization 7    11001000 00010111 00011110 00000000    200.23.30.0/23

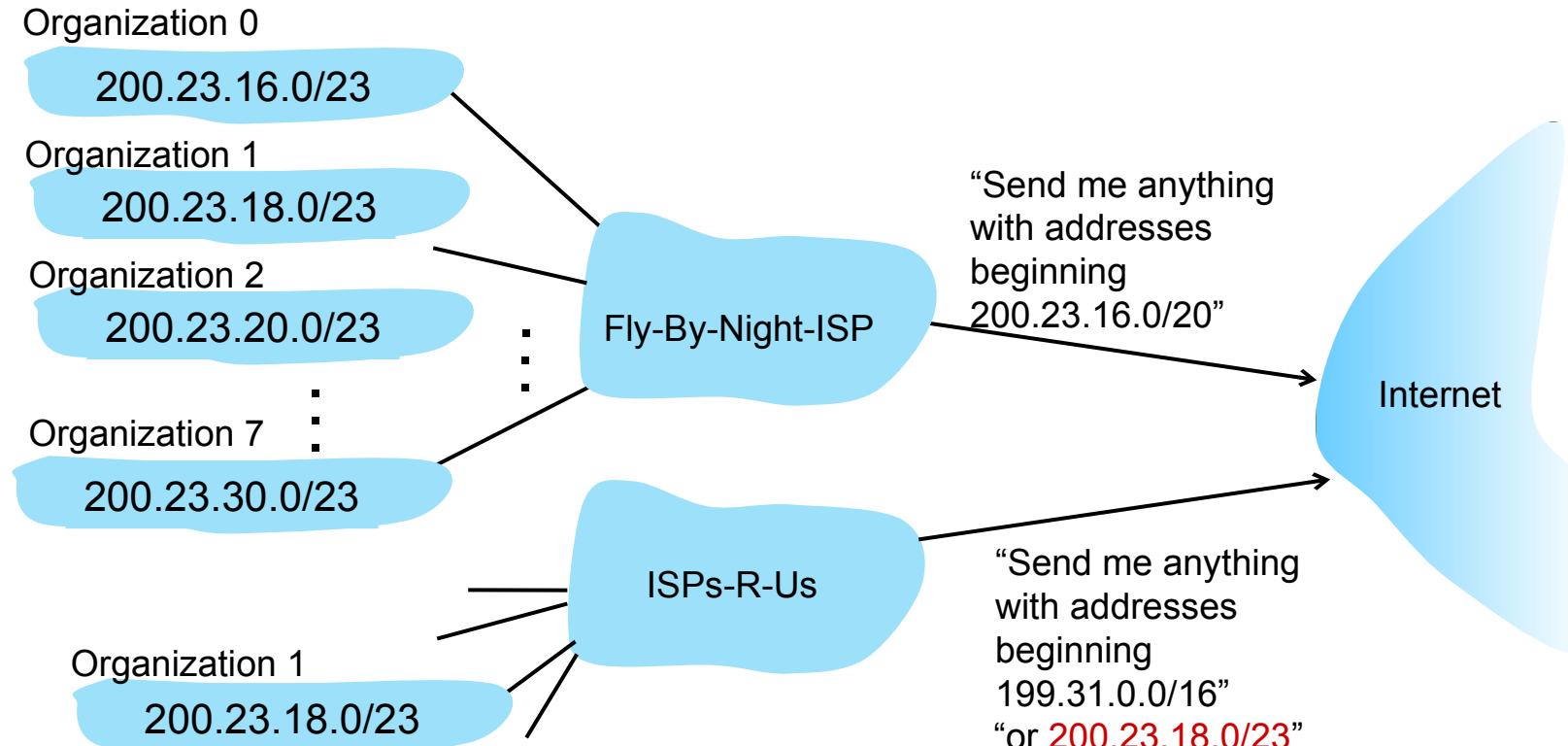
# Hierarchical addressing: route aggregation

hierarchical addressing allows efficient advertisement of routing information:



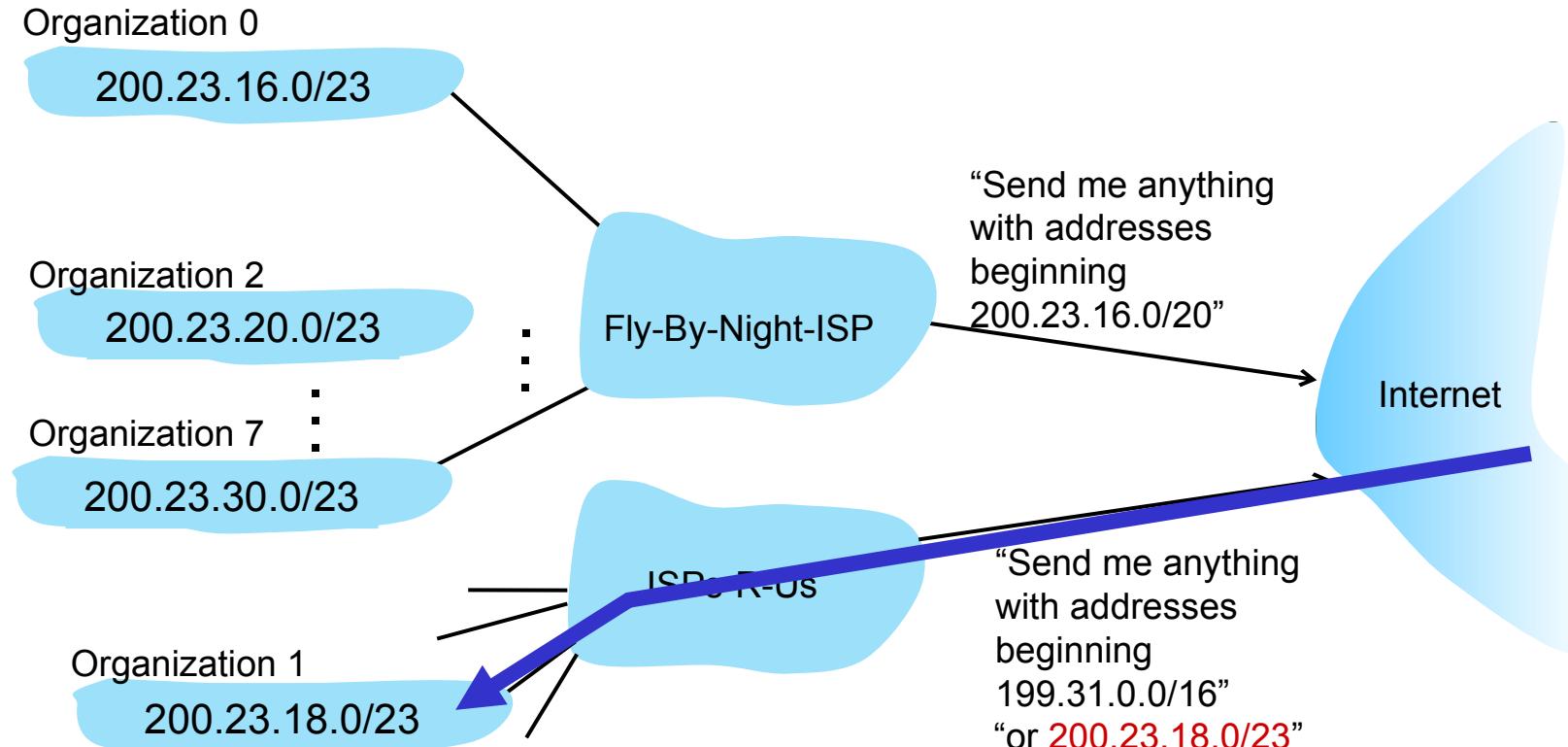
# Hierarchical addressing: more specific routes

- Organization 1 moves from Fly-By-Night-ISP to ISPs-R-Us
- ISPs-R-Us now advertises a more specific route to Organization 1



# Hierarchical addressing: more specific routes

- Organization 1 moves from Fly-By-Night-ISP to ISPs-R-Us
- ISPs-R-Us now advertises a more specific route to Organization 1



# IP addressing: last words ...

**Q:** how does an ISP get block of addresses?

**A:** ICANN: Internet Corporation for Assigned Names and Numbers

<http://www.icann.org/>

- allocates IP addresses, through 5 regional registries (RRs) (who may then allocate to local registries)
- manages DNS root zone, including delegation of individual TLD (.com, .edu , ...) management

**Q:** are there enough 32-bit IP addresses?

- ICANN allocated last chunk of IPv4 addresses to RRs in 2011
- NAT (next) helps IPv4 address space exhaustion
- IPv6 has 128-bit address space

"Who the hell knew how much address space we needed?" Vint Cerf (reflecting on decision to make IPv4 address 32 bits long)

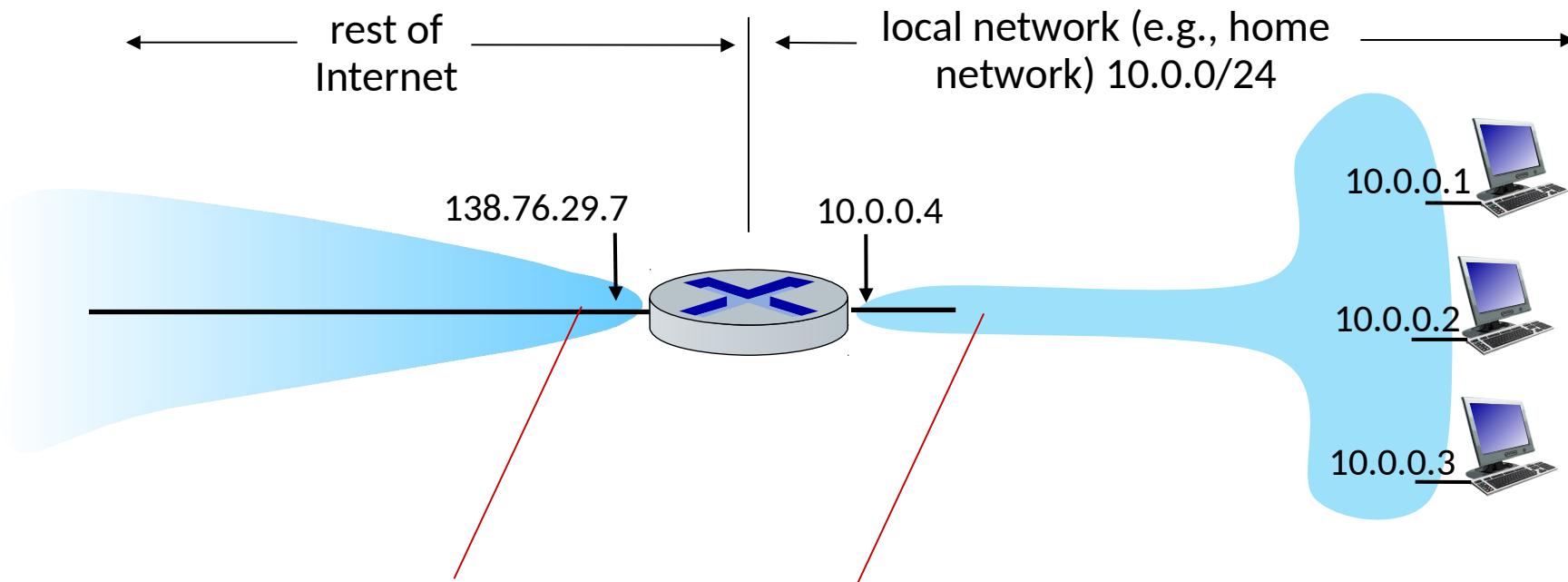
# Network layer: “data plane” roadmap

- Network layer: overview
  - data plane
  - control plane
- What's inside a router
  - input ports, switching, output ports
  - buffer management, scheduling
- IP: the Internet Protocol
  - datagram format
  - addressing
  - network address translation
  - IPv6
- Generalized Forwarding, SDN
  - match+action
  - OpenFlow: match+action in action
- Middleboxes



# NAT: network address translation

**NAT:** all devices in local network share just **one** IPv4 address as far as outside world is concerned



*all* datagrams **leaving** local network have **same** source NAT IP address: 138.76.29.7, but **different** source port numbers

datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

# NAT: network address translation

- all devices in local network have 32-bit addresses in a “private” IP address space (10/8, 172.16/12, 192.168/16 prefixes) that can only be used in local network
- advantages:
  - just **one** IP address needed from provider ISP for ***all*** devices
  - can change addresses of host in local network without notifying outside world
  - can change ISP without changing addresses of devices in local network
  - security: devices inside local net not directly addressable, visible by outside world

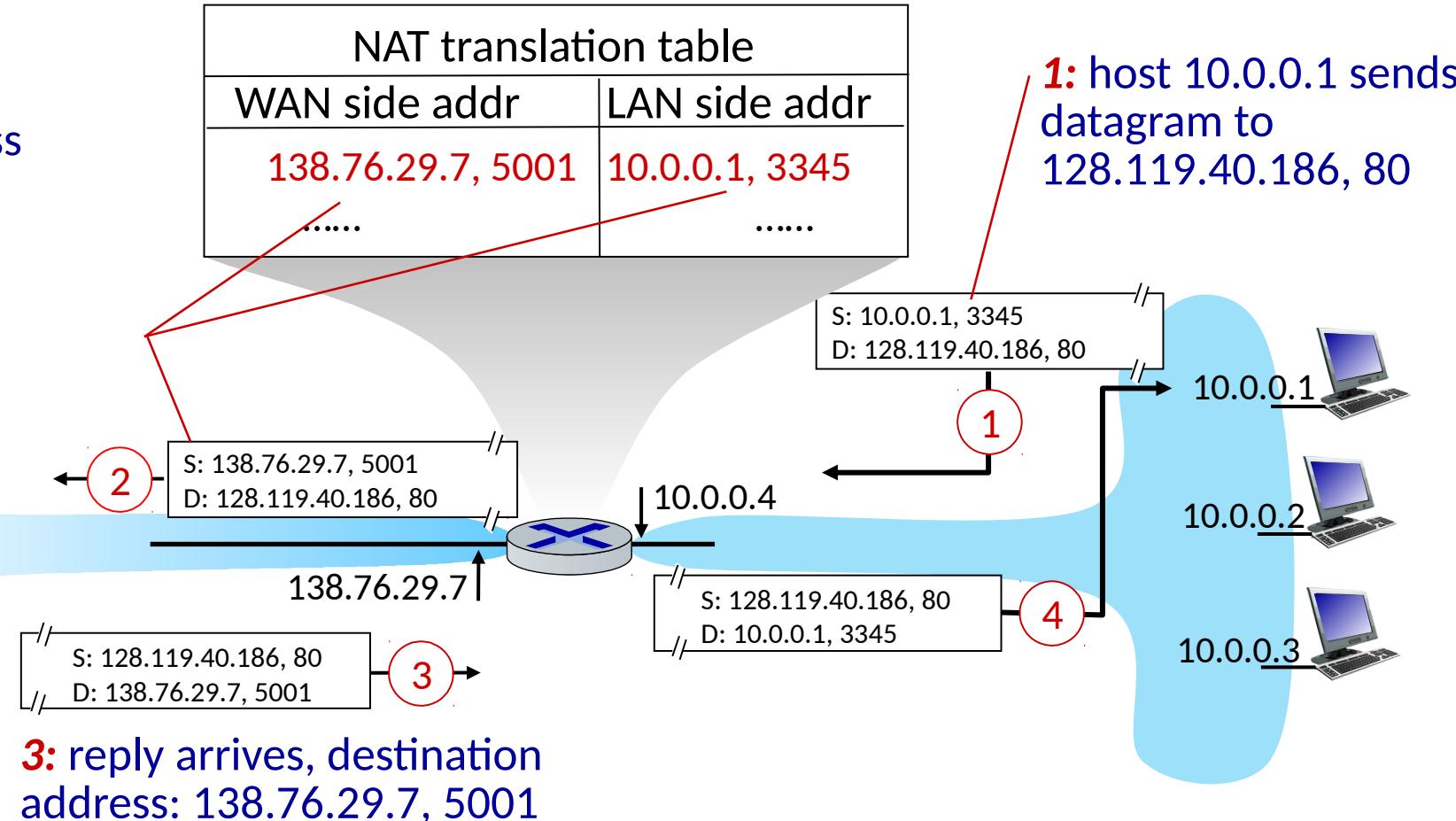
# NAT: network address translation

implementation: NAT router must (transparently):

- outgoing datagrams: replace (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
  - remote clients/servers will respond using (NAT IP address, new port #) as destination address
- remember (in NAT translation table) every (source IP address, port #) to (NAT IP address, new port #) translation pair
- incoming datagrams: replace (NAT IP address, new port #) in destination fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

# NAT: network address translation

2: NAT router changes datagram source address from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table



# NAT: network address translation

- NAT has been controversial:
  - routers “should” only process up to layer 3
  - address “shortage” should be solved by IPv6
  - violates end-to-end argument (port # manipulation by network-layer device)
  - NAT traversal: what if client wants to connect to server behind NAT?
- but NAT is here to stay:
  - extensively used in home and institutional nets, 4G/5G cellular nets

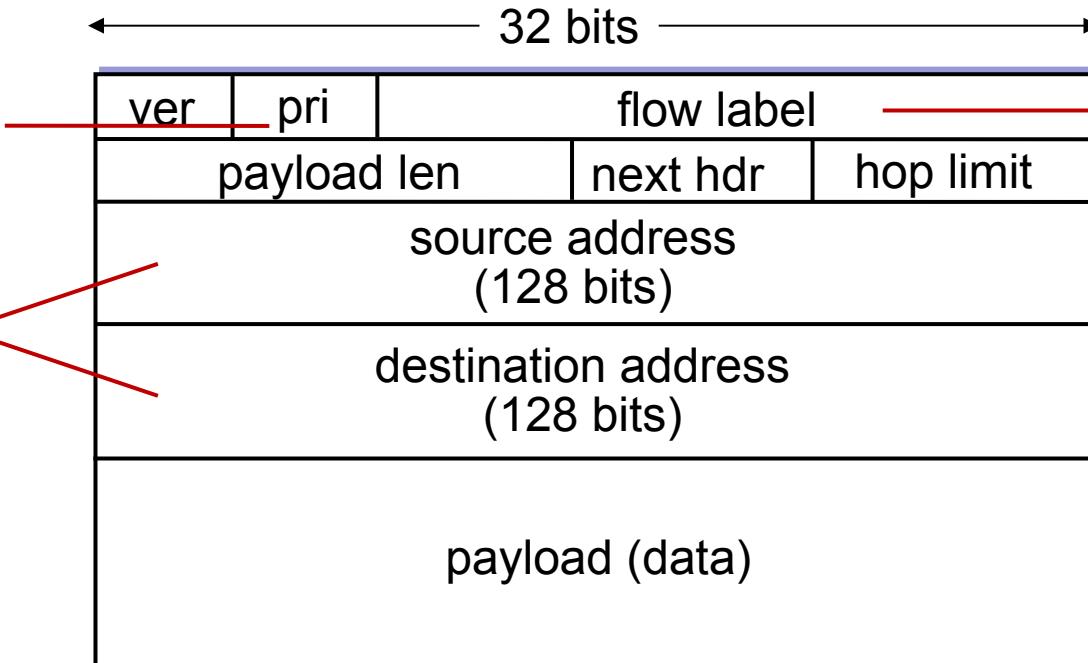
# IPv6: motivation

- **initial motivation:** 32-bit IPv4 address space would be completely allocated
- additional motivation:
  - speed processing/forwarding: 40-byte fixed length header
  - enable different network-layer treatment of “flows”

# IPv6 datagram format

**priority:** identify priority among datagrams in flow

**128-bit IPv6 addresses**



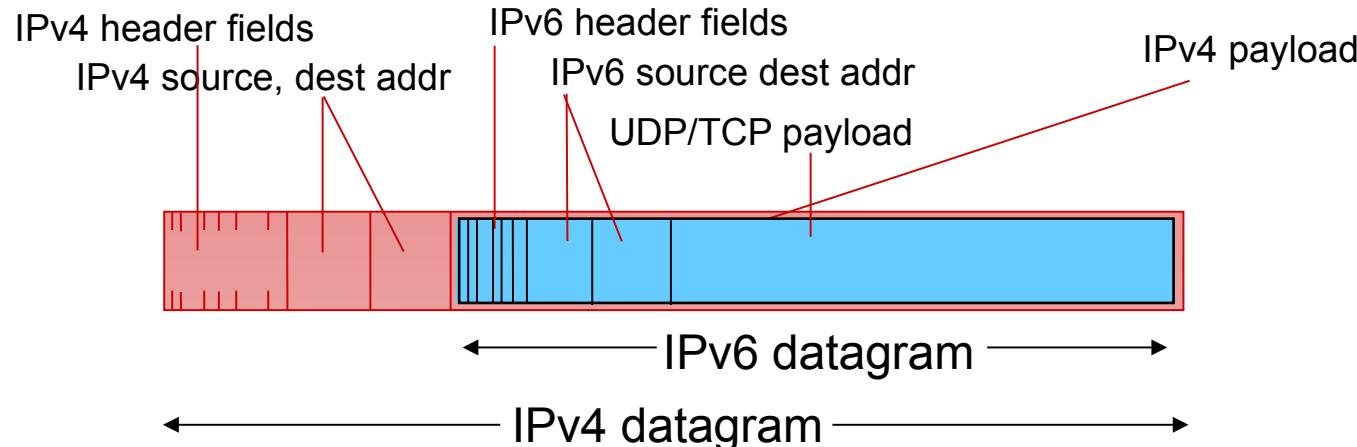
**flow label:** identify datagrams in same "flow." (concept of "flow" not well defined).

What's missing (compared with IPv4):

- no checksum (to speed processing at routers)
- no fragmentation/reassembly
- no options (available as upper-layer, next-header protocol at router)

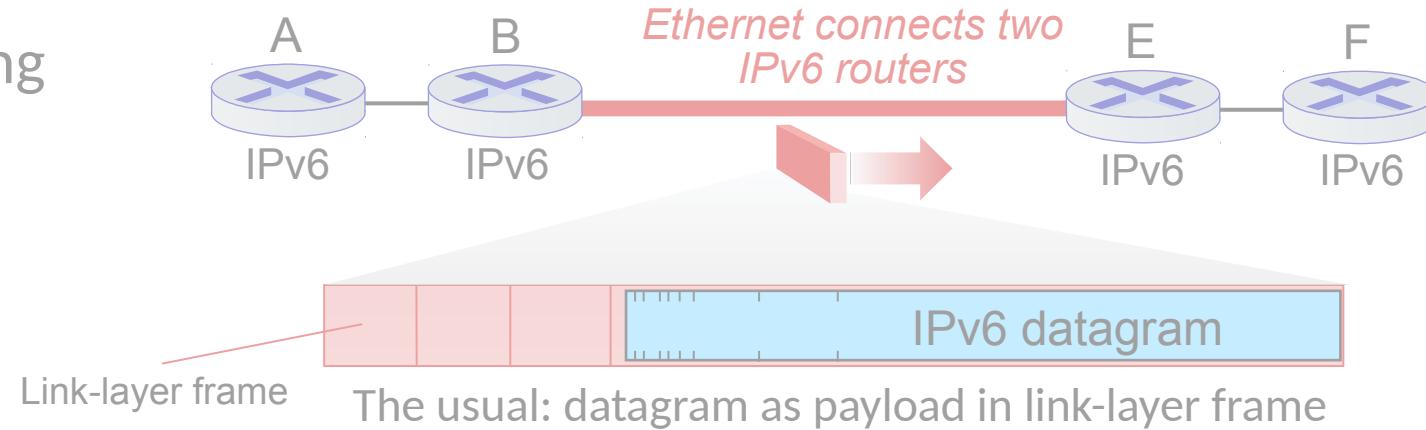
# Transition from IPv4 to IPv6

- not all routers can be upgraded simultaneously
  - no “flag days”
  - how will network operate with mixed IPv4 and IPv6 routers?
- tunneling:** IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers (“packet within a packet”)
  - tunneling used extensively in other contexts (4G/5G)

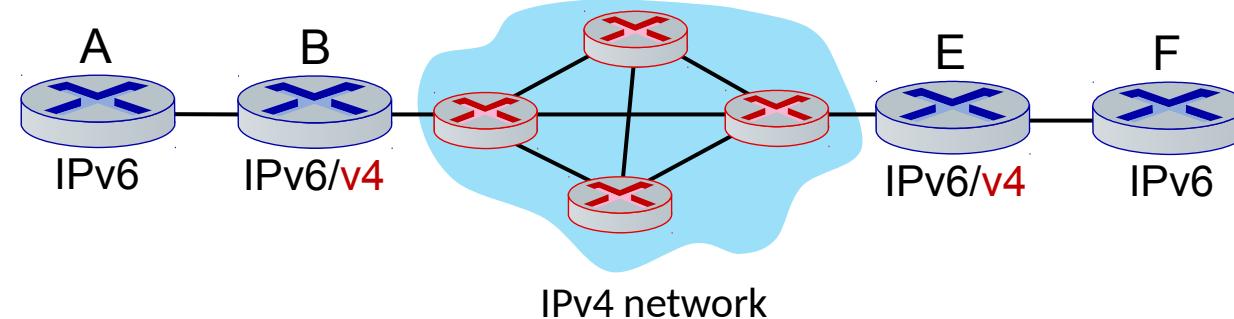


# Tunneling and encapsulation

Ethernet connecting  
two IPv6 routers:

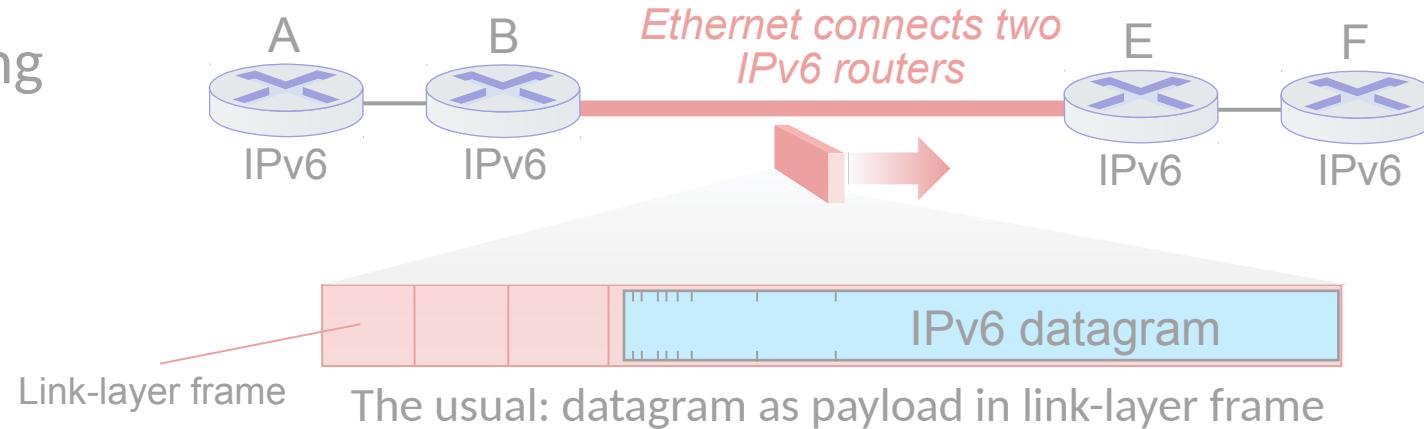


IPv4 network  
connecting two  
IPv6 routers

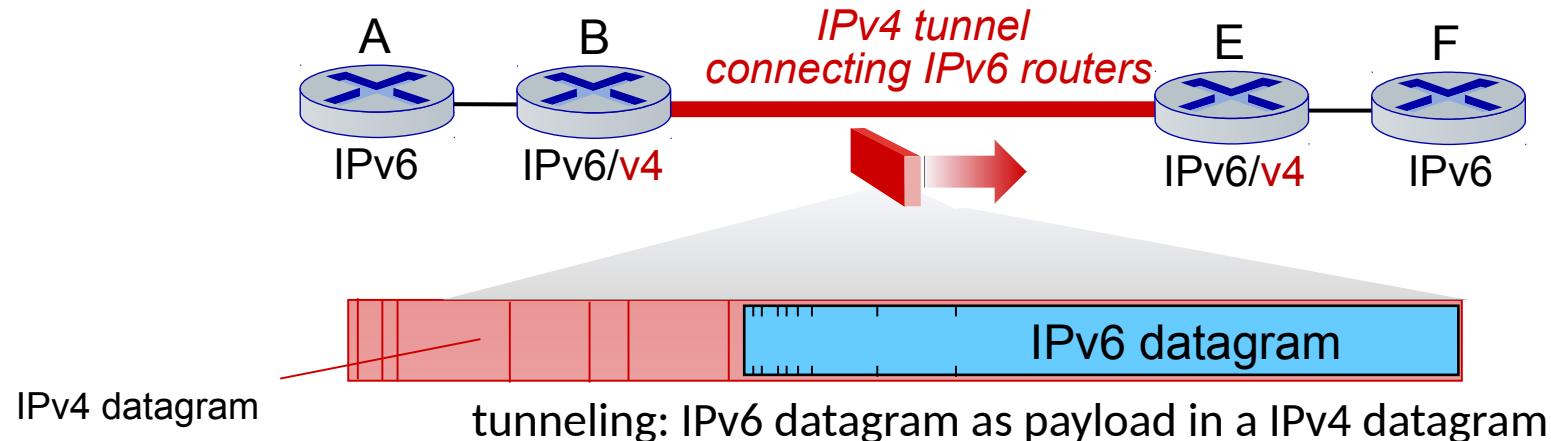


# Tunneling and encapsulation

Ethernet connecting  
two IPv6 routers:

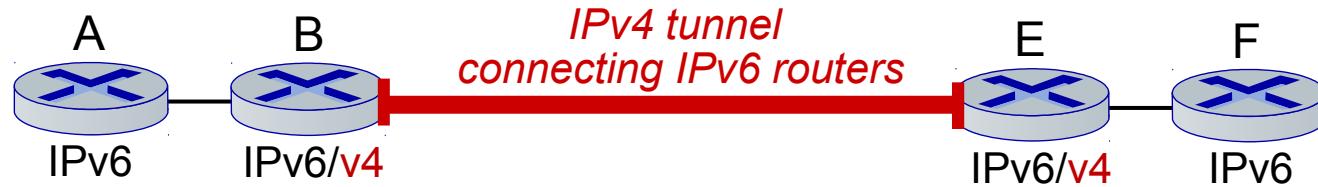


IPv4 tunnel  
connecting two  
IPv6 routers

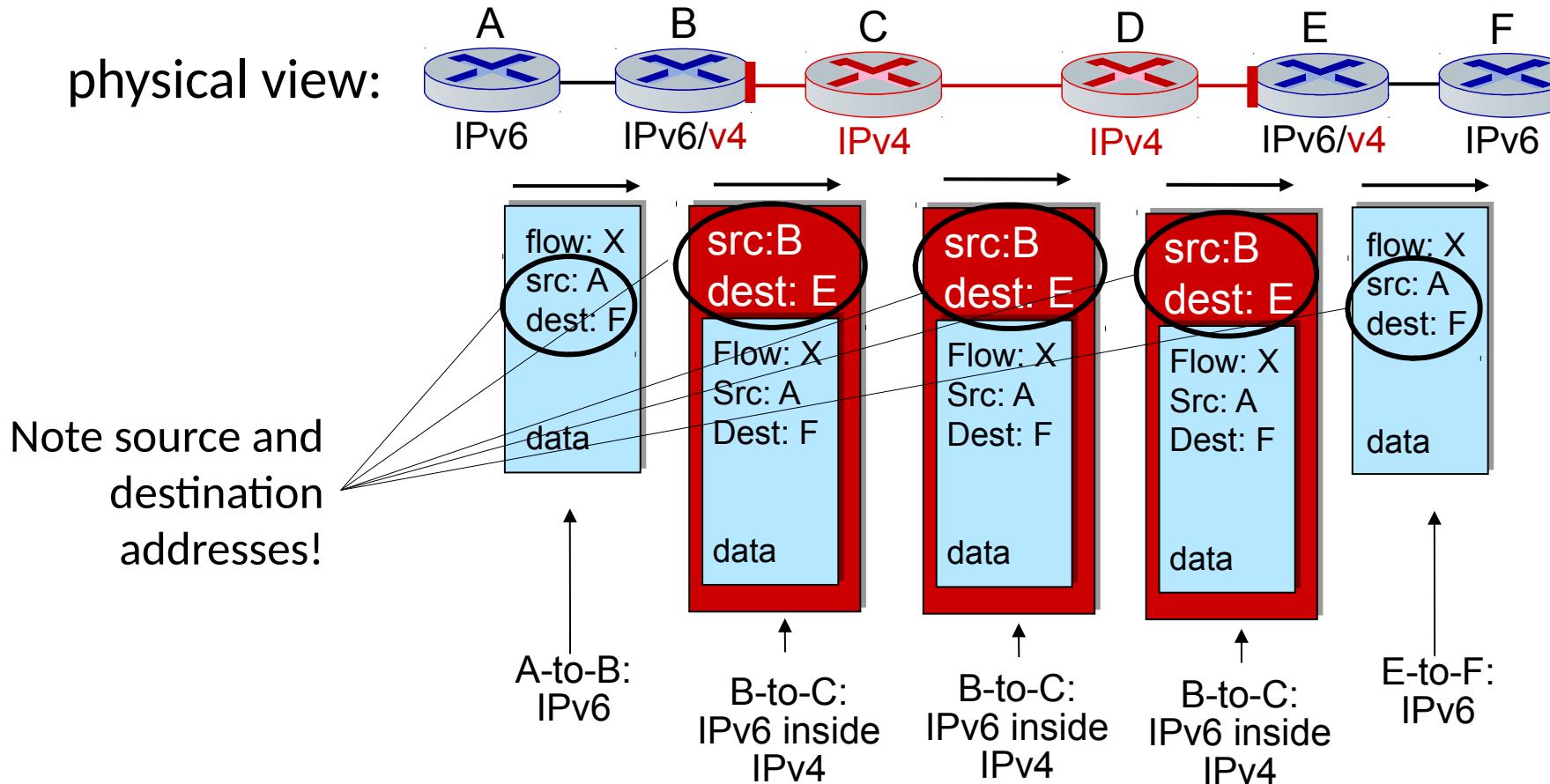


# Tunneling

logical view:



physical view:



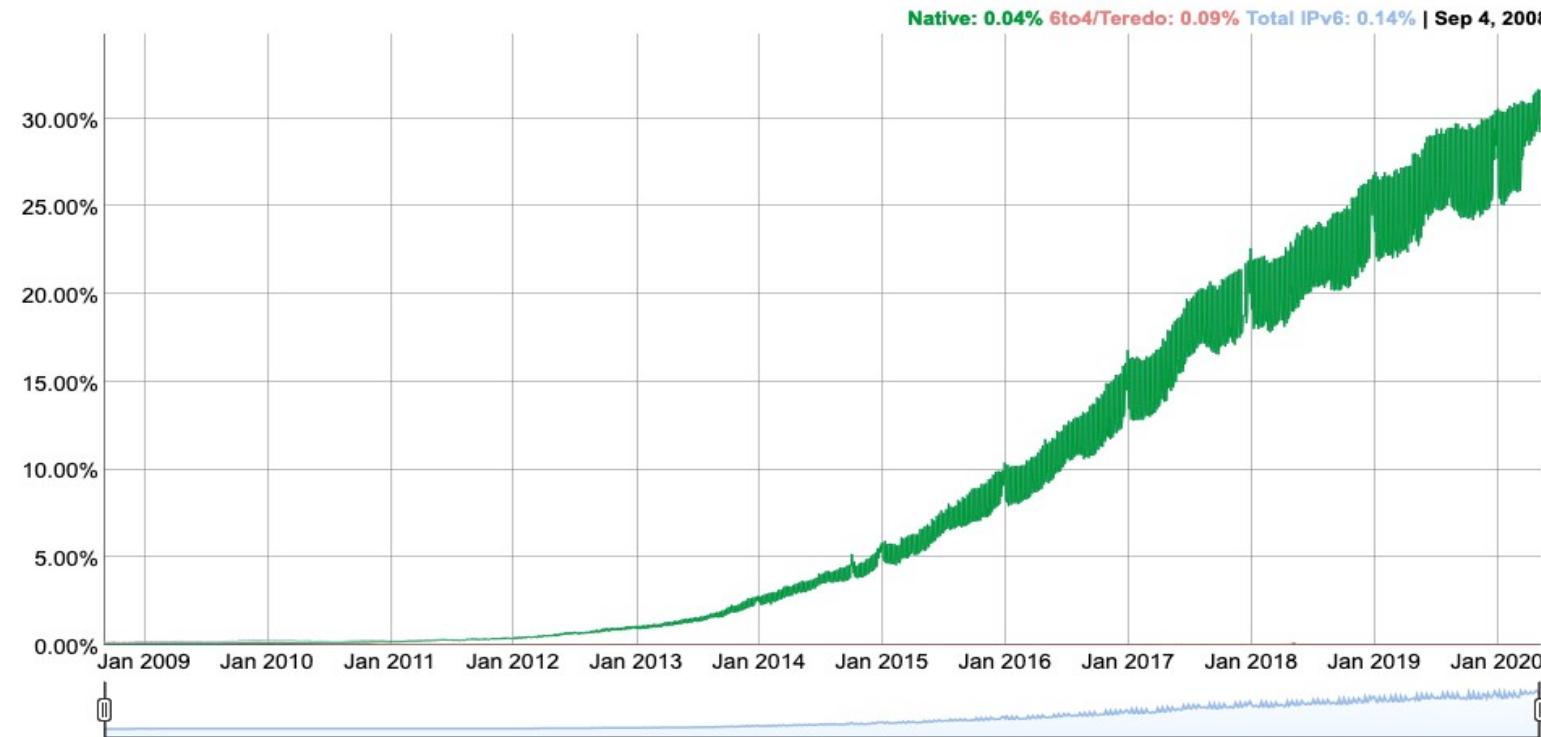
Note source and destination addresses!

# IPv6: adoption

- Google<sup>1</sup>: ~ 30% of clients access services via IPv6
- NIST: 1/3 of all US government domains are IPv6 capable

## IPv6 Adoption

We are continuously measuring the availability of IPv6 connectivity among Google users. The graph shows the percentage of users that access Google over IPv6.



1

<https://www.google.com/intl/en/ipv6/statistics.html>

# IPv6: adoption

- Google<sup>1</sup>: ~ 30% of clients access services via IPv6
- NIST: 1/3 of all US government domains are IPv6 capable
- Long (long!) time for deployment, use
  - 25 years and counting!
  - think of application-level changes in last 25 years: WWW, social media, streaming media, gaming, telepresence, ...
  - *Why?*

<sup>1</sup> <https://www.google.com/intl/en/ipv6/statistics.html>

# Network layer: “data plane” roadmap

- Network layer: overview
  - data plane
  - control plane
- What's inside a router
  - input ports, switching, output ports
  - buffer management, scheduling
- IP: the Internet Protocol
  - datagram format
  - addressing
  - network address translation
  - IPv6

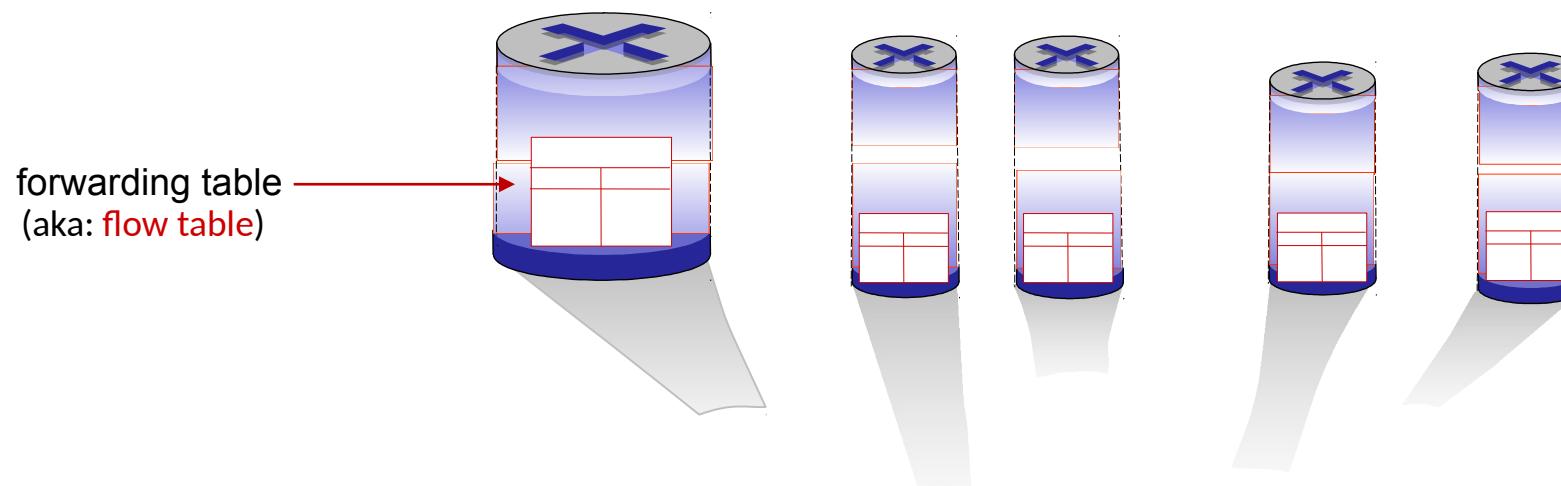
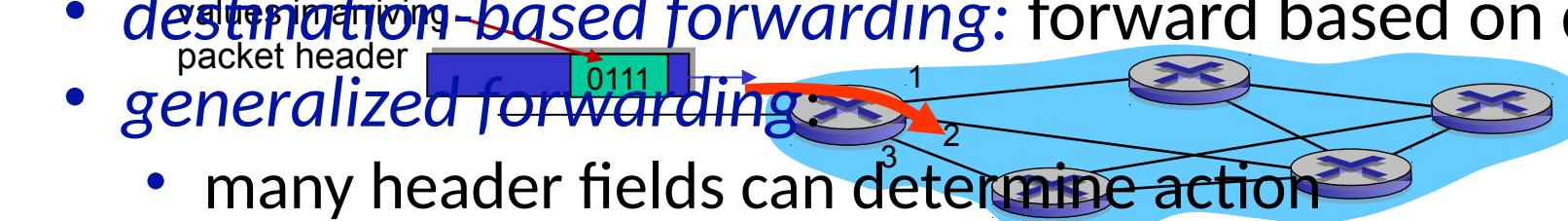


- Generalized Forwarding, SDN
  - Match+action
  - OpenFlow: match+action in action
- Middleboxes

# Generalized forwarding: match plus action

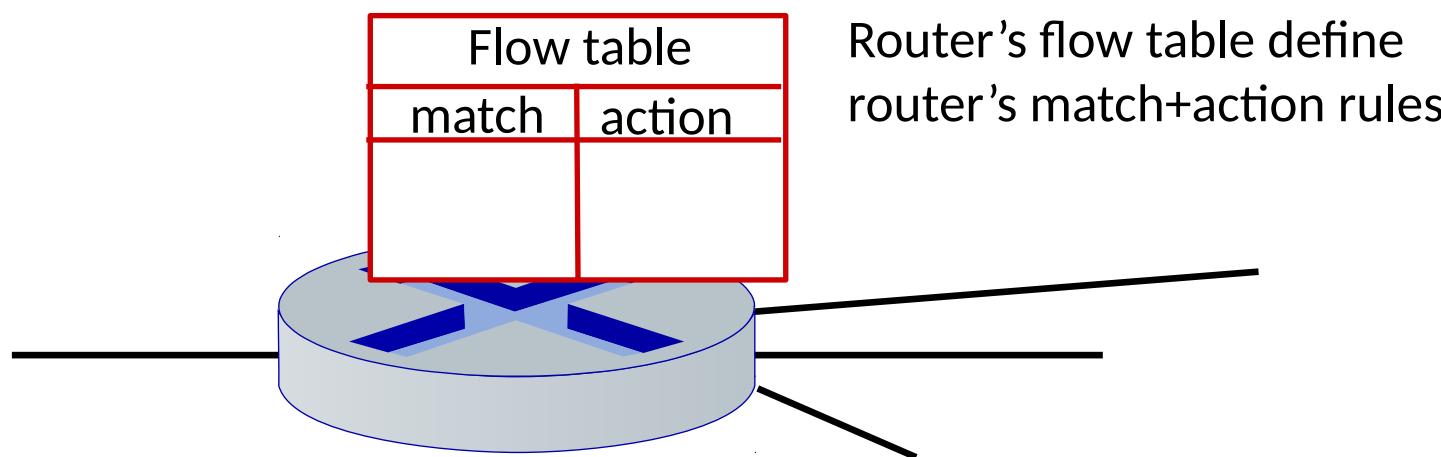
Review: each router contains a **forwarding table** (aka: **flow table**)

- “**match plus action**” abstraction: match bits in arriving packet, take action
  - *destination-based forwarding*: forward based on dest. IP address
  - *generalized forwarding*:
    - many header fields can determine action
    - many actions possible: drop/copy/modify/log packet



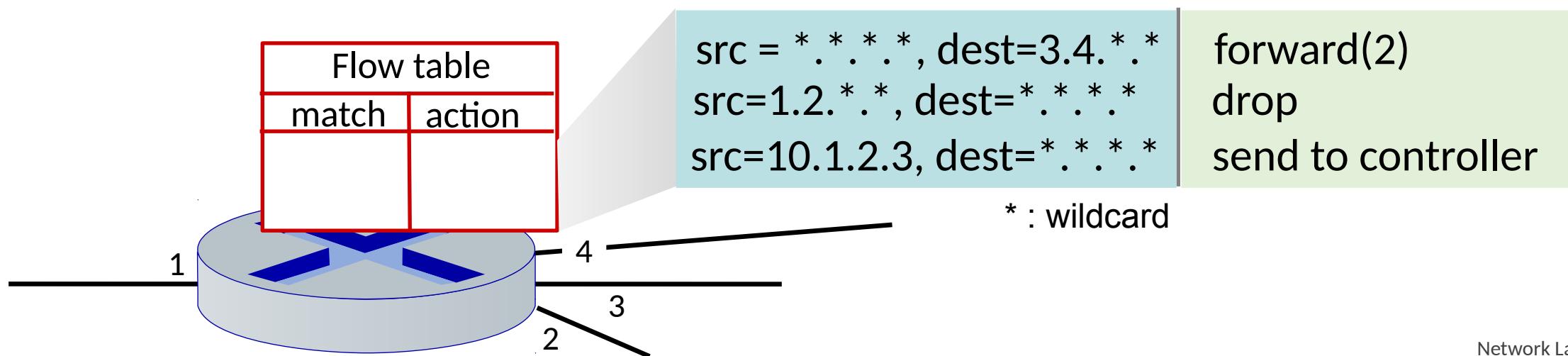
# Flow table abstraction

- **flow**: defined by header field values (in link-, network-, transport-layer fields)
- **generalized forwarding**: simple packet-handling rules
  - **match**: pattern values in packet header fields
  - **actions**: for matched packet: drop, forward, modify, matched packet or send matched packet to controller
  - **priority**: disambiguate overlapping patterns
  - **counters**: #bytes and #packets

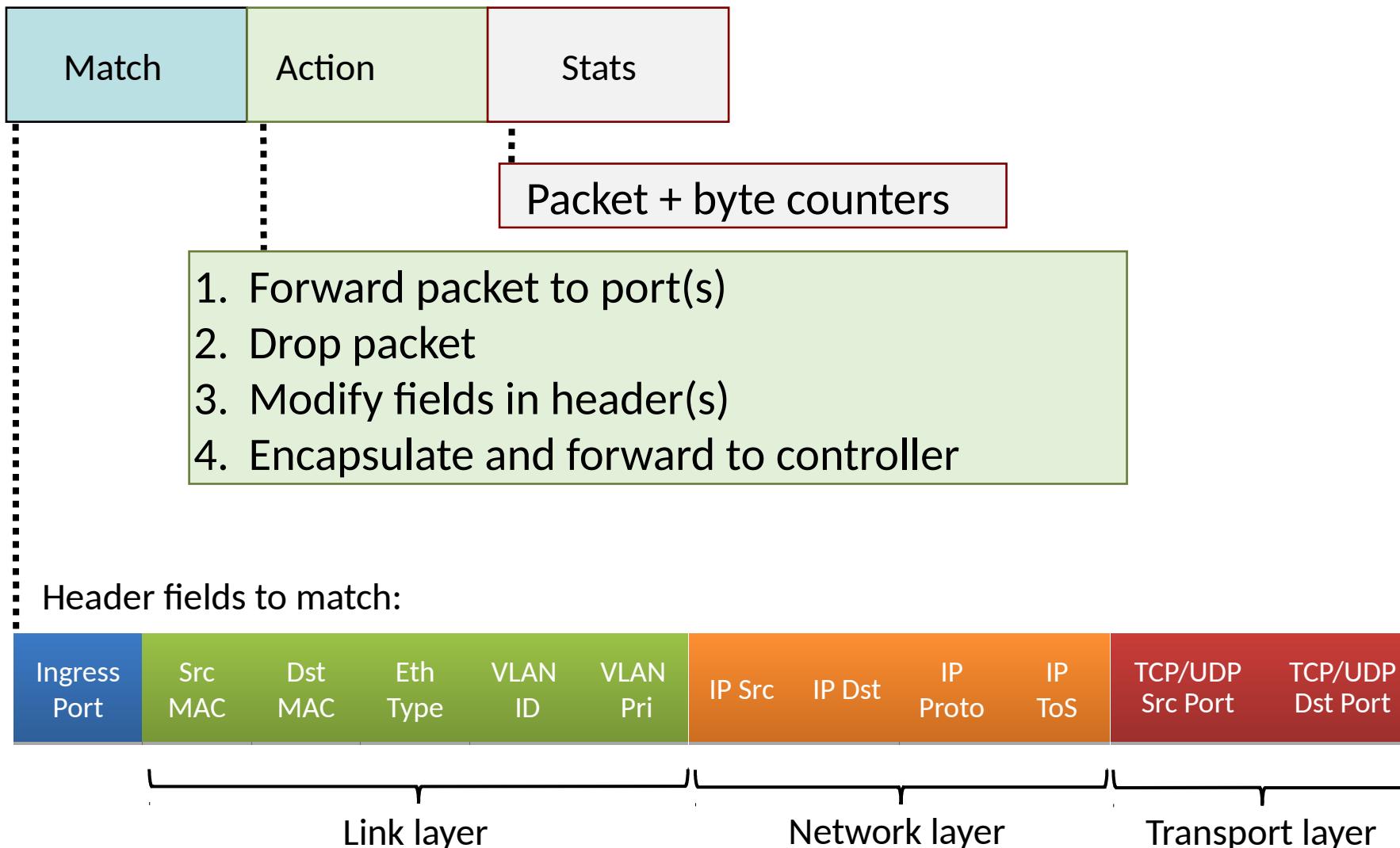


# Flow table abstraction

- **flow:** defined by header fields
- **generalized forwarding: simple** packet-handling rules
  - **match:** pattern values in packet header fields
  - **actions:** for matched packet: drop, forward, modify, matched packet or send matched packet to controller
  - **priority:** disambiguate overlapping patterns
  - **counters:** #bytes and #packets



# OpenFlow: flow table entries



# OpenFlow: examples

Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	51.6.0.8	*	*	*	*	port6

IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

Firewall:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	*	*	*	*	*	22 drop

Block (do not forward) all datagrams destined to TCP port 22 (ssh port #)

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	128.119.1.1	*	*	*	*	*	drop

Block (do not forward) all datagrams sent by host 128.119.1.1

# OpenFlow: examples

Layer 2 destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	22:A7:23: 11:E1:02	*	*	*	*	*	*	*	*	*	port3

layer 2 frames with destination MAC address 22:A7:23:11:E1:02 should be forwarded to output port 3

# OpenFlow abstraction

- **match+action:** abstraction unifies different kinds of devices

## Router

- *match:* longest destination IP prefix
- *action:* forward out a link

## Switch

- *match:* destination MAC address
- *action:* forward or flood

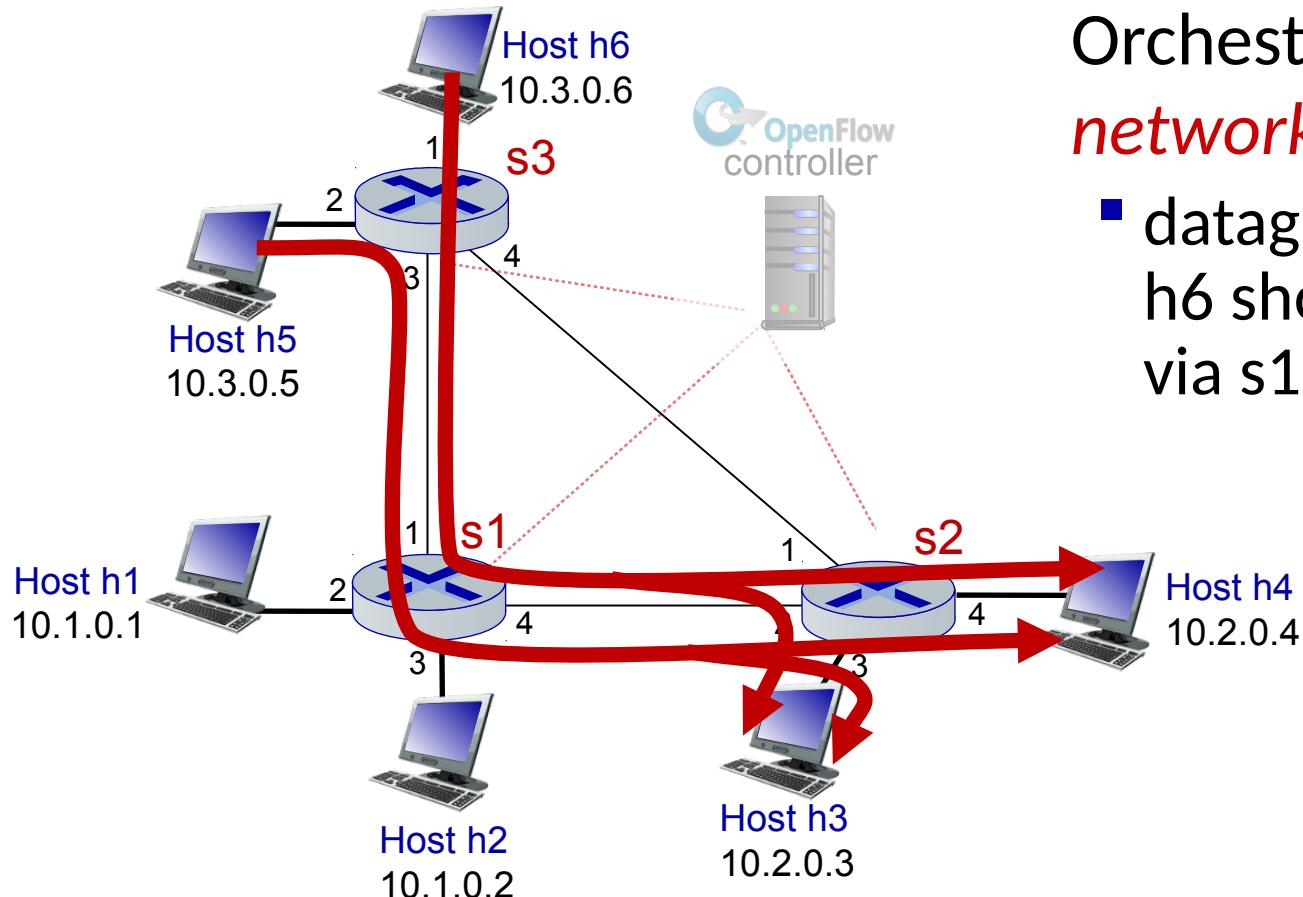
## Firewall

- *match:* IP addresses and TCP/UDP port numbers
- *action:* permit or deny

## NAT

- *match:* IP address and port
- *action:* rewrite address and port

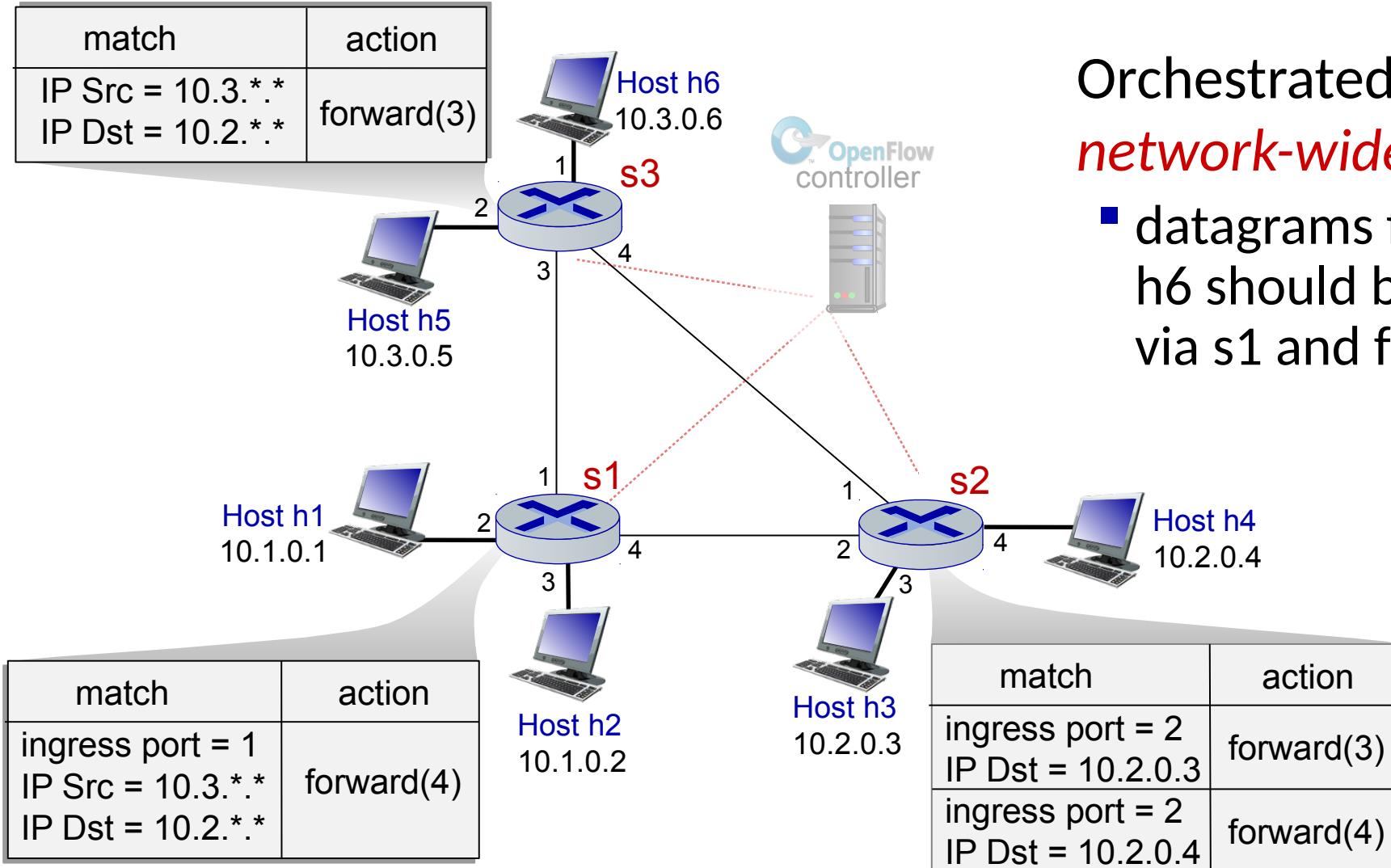
# OpenFlow example



Orchestrated tables can create ***network-wide*** behavior, e.g.,:

- datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

# OpenFlow example



Orchestrated tables can create **network-wide** behavior, e.g.:

- datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

# Generalized forwarding: summary

- “match plus action” abstraction: match bits in arriving packet header(s) in any layers, take action
  - matching over many fields (link-, network-, transport-layer)
  - local actions: drop, forward, modify, or send matched packet to controller
  - “program” network-wide behaviors
- simple form of “network programmability”
  - programmable, per-packet “processing”
  - *historical roots*: active networking
  - *today*: more generalized programming:  
P4 (see p4.org).

# Network layer: “data plane” roadmap

- Network layer: overview
- What's inside a router
- IP: the Internet Protocol
- Generalized Forwarding
- Middleboxes
  - middlebox functions
  - evolution, architectural principles of the Internet



# Middleboxes

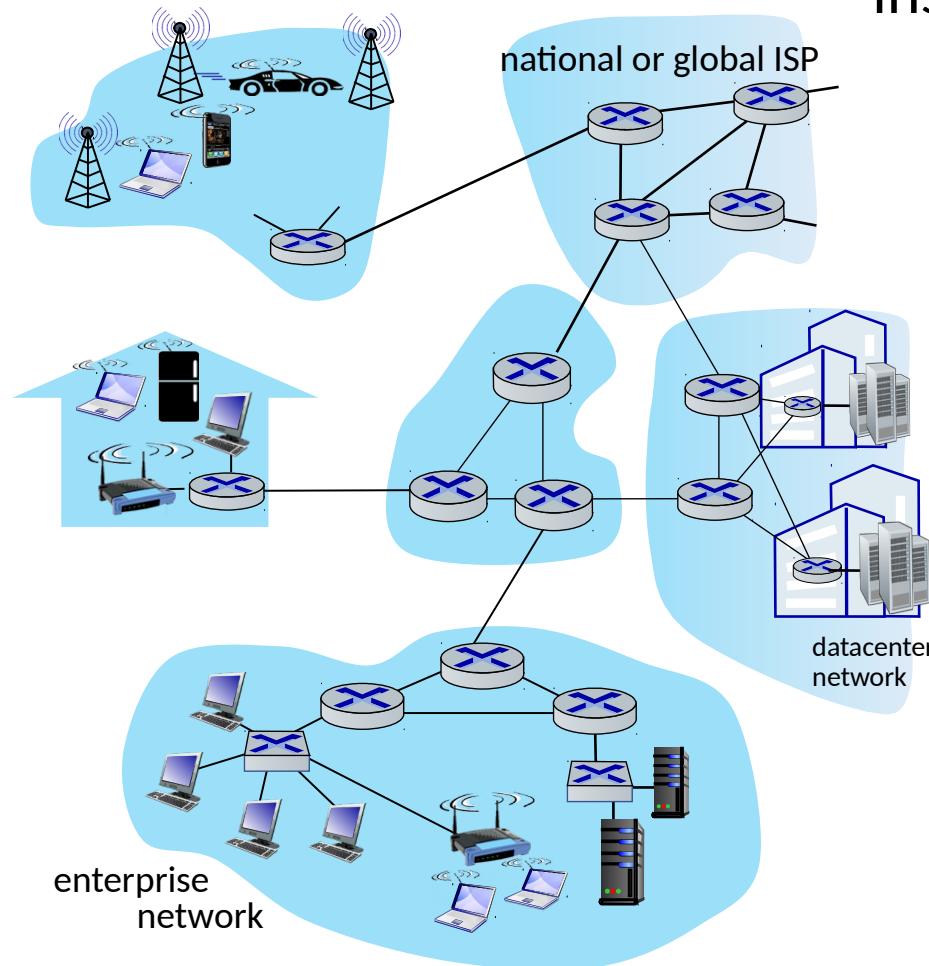
Middlebox (RFC 3234)

“any intermediary box performing functions apart from normal, standard functions of an IP router on the data path between a source host and destination host”

# Middleboxes everywhere!

NAT: home,  
cellular,  
institutional

Application-specific: service providers, institutional, CDN



Firewalls, IDS: corporate, institutional, service providers, ISPs

Load balancers: corporate, service provider, data center, mobile nets

Caches: service provider, mobile, CDNs

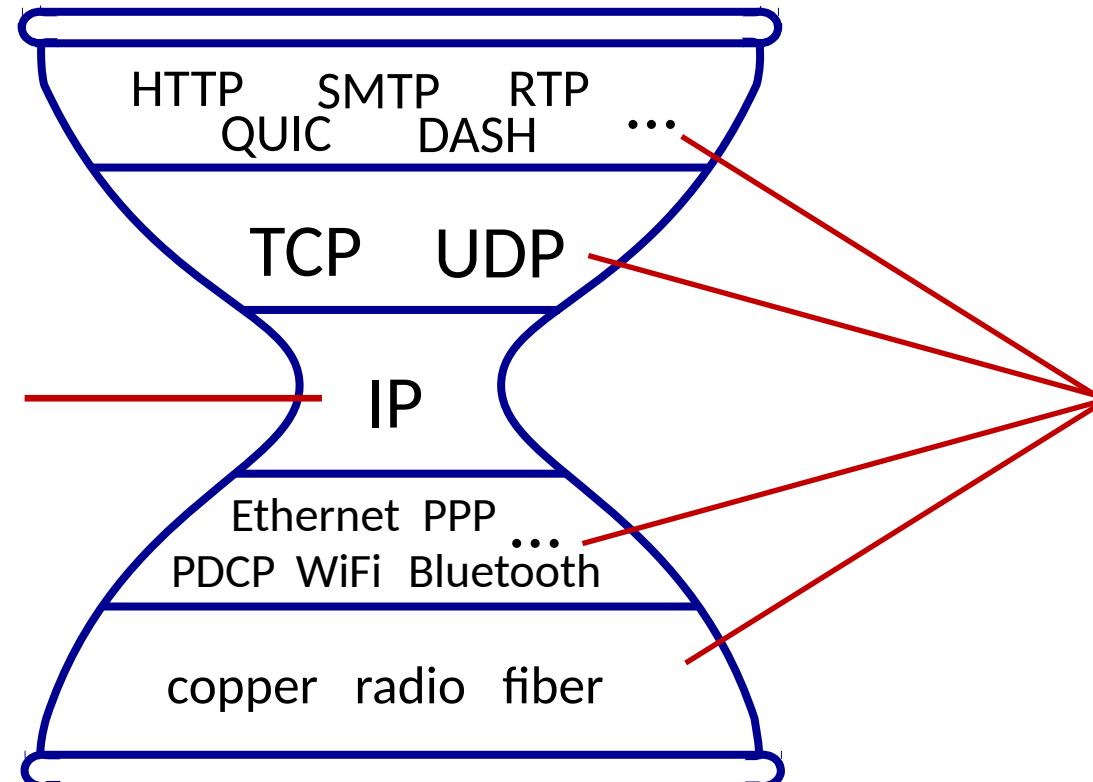
# Middleboxes

- initially: proprietary (closed) hardware solutions
- move towards “whitebox” hardware implementing open API
  - move away from proprietary hardware solutions
  - programmable local actions via match+action
  - move towards innovation/differentiation in software
- SDN: (logically) centralized control and configuration management often in private/public cloud
- network functions virtualization (NFV): programmable services over white box networking, computation, storage

# The IP hourglass

Internet's "thin waist":

- one network layer protocol: IP
- *must* be implemented by every (billions) of Internet-connected devices

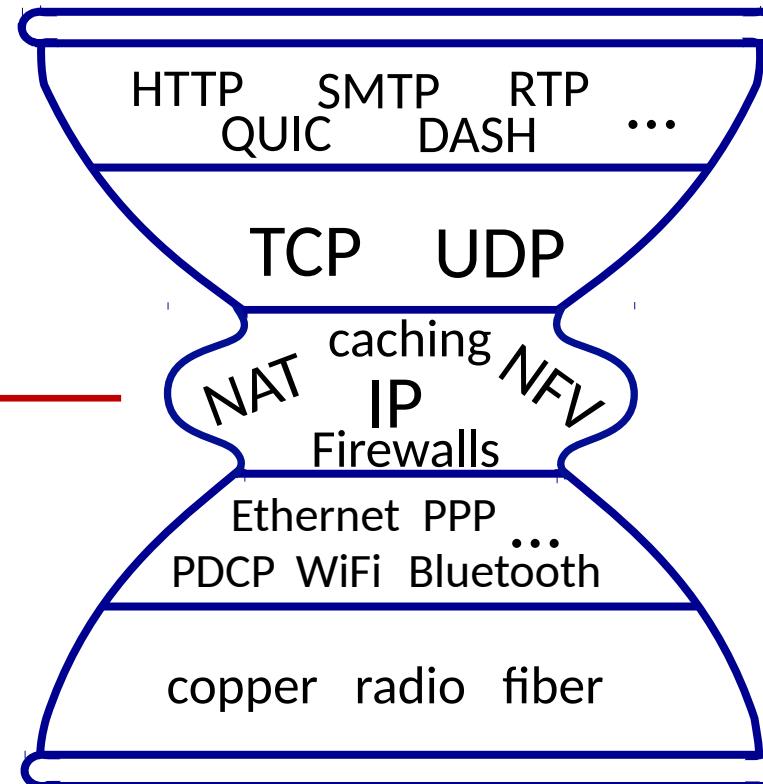


*many* protocols  
in physical, link,  
transport, and  
application  
layers

# The IP hourglass, at middle age

Internet's middle age  
“love handles”?

- middleboxes,  
operating inside the  
network



# Architectural Principles of the Internet

RFC 1958

“Many members of the Internet community would argue that there is no architecture, but only a tradition, which was not written down for the first 25 years (or at least not by the IAB). However, in very general terms, the community believes that

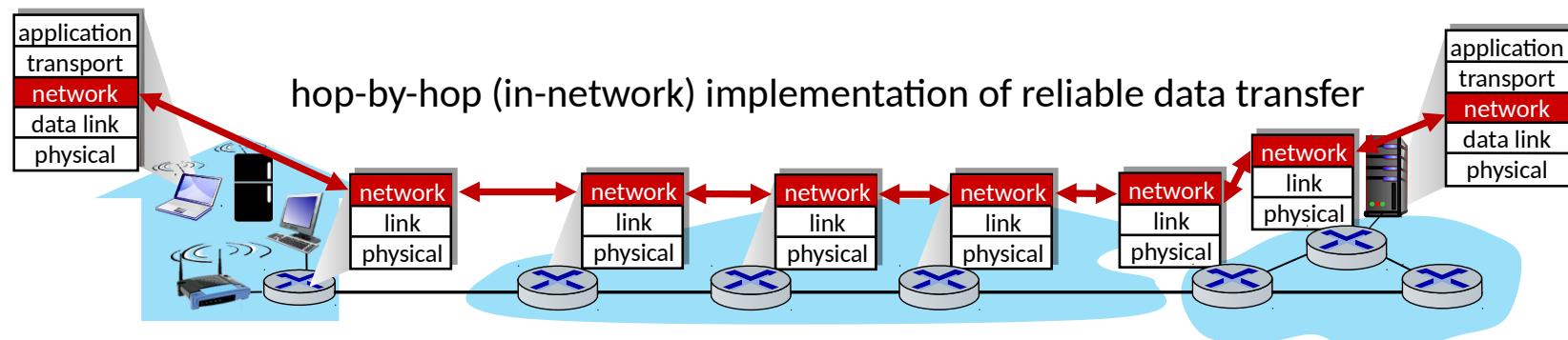
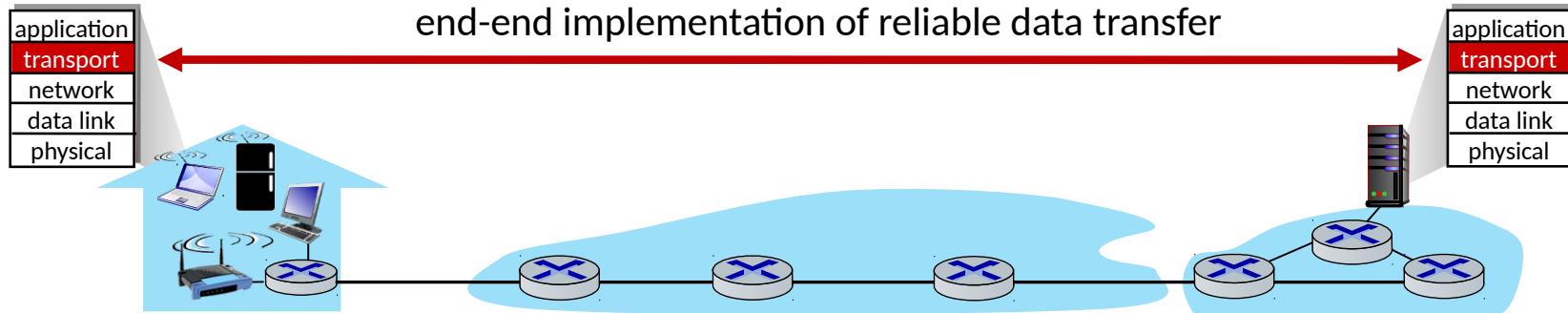
**the goal is connectivity, the tool is the Internet Protocol, and the intelligence is end to end rather than hidden in the network.”**

Three cornerstone beliefs:

- simple connectivity
- IP protocol: that narrow waist
- intelligence, complexity at network edge

# The end-end argument

- some network functionality (e.g., reliable data transfer, congestion) can be implemented in **network**, or at **network edge**



# The end-end argument

- some network functionality (e.g., reliable data transfer, congestion) can be implemented in network, or at network edge

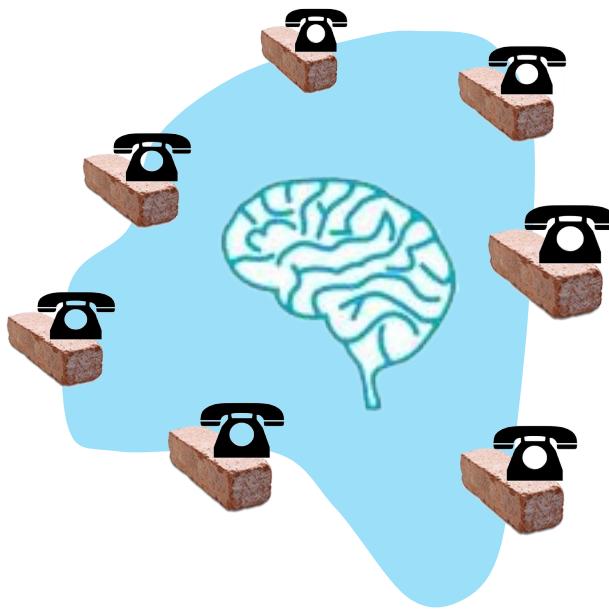
“The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)

We call this line of reasoning against low-level function implementation the “end-to-end argument.”

---

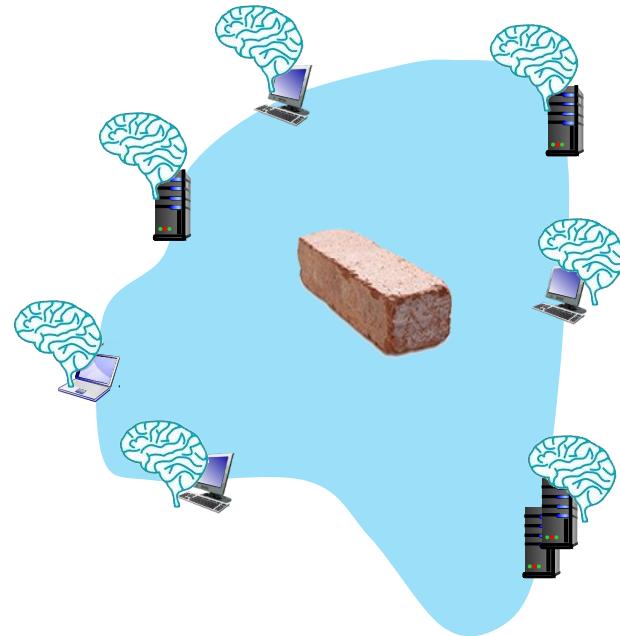
Saltzer, Reed, Clark 1981

# Where's the intelligence?



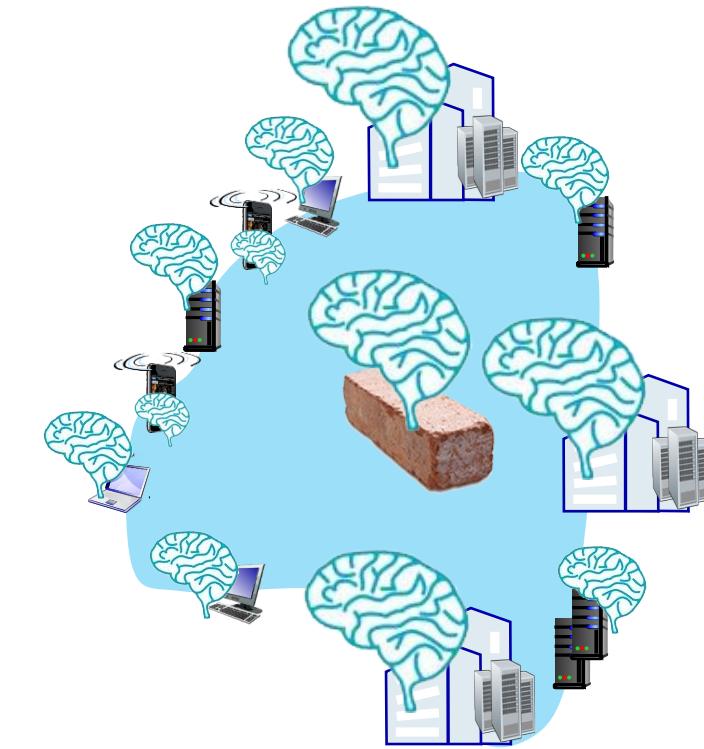
**20<sup>th</sup> century phone net:**

- intelligence/computing at network switches



**Internet (pre-2005)**

- intelligence, computing at edge



**Internet (post-2005)**

- programmable network devices
- intelligence, computing, massive application-level infrastructure at edge

# Chapter 4: done!

- Network layer: overview
- What's inside a router
- IP: the Internet Protocol
- Generalized Forwarding, SDN
- Middleboxes



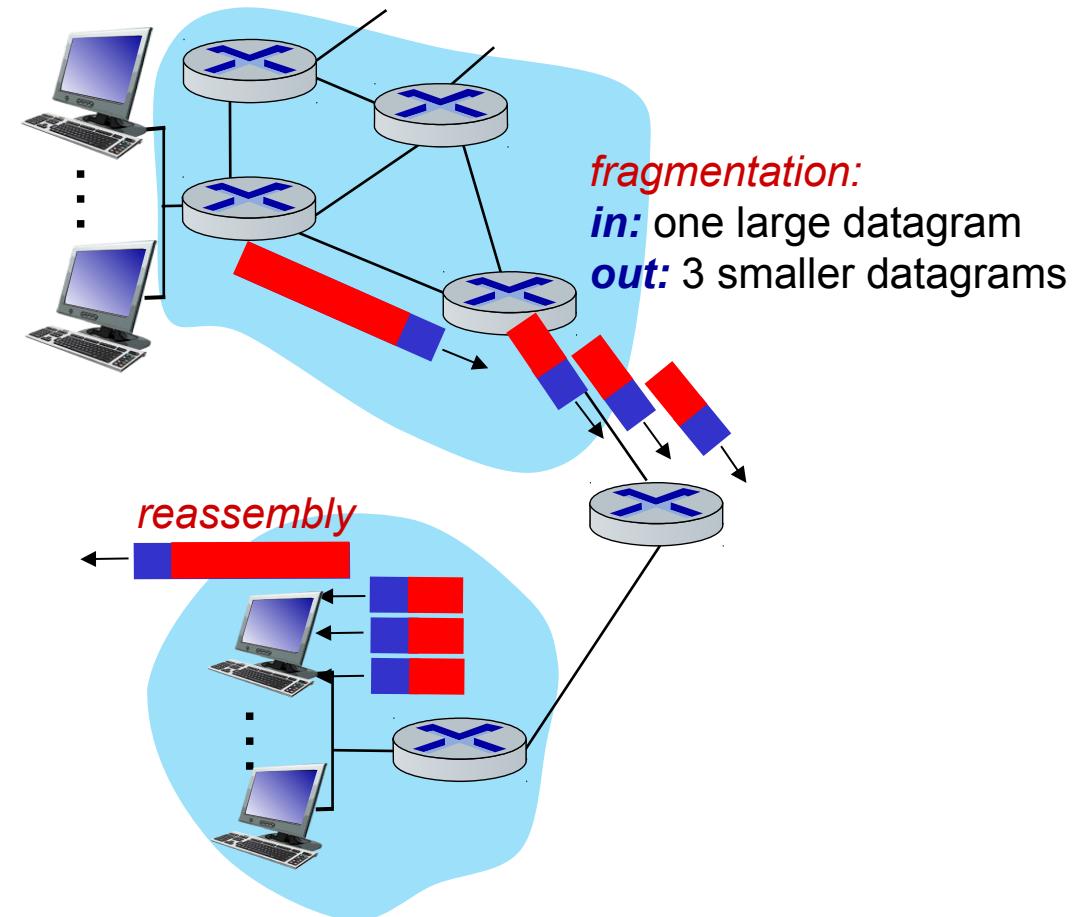
**Question:** how are forwarding tables (destination-based forwarding) or flow tables (generalized forwarding) computed?

**Answer:** by the control plane (next chapter)

# **Additional Chapter 4 slides**

# IP fragmentation/reassembly

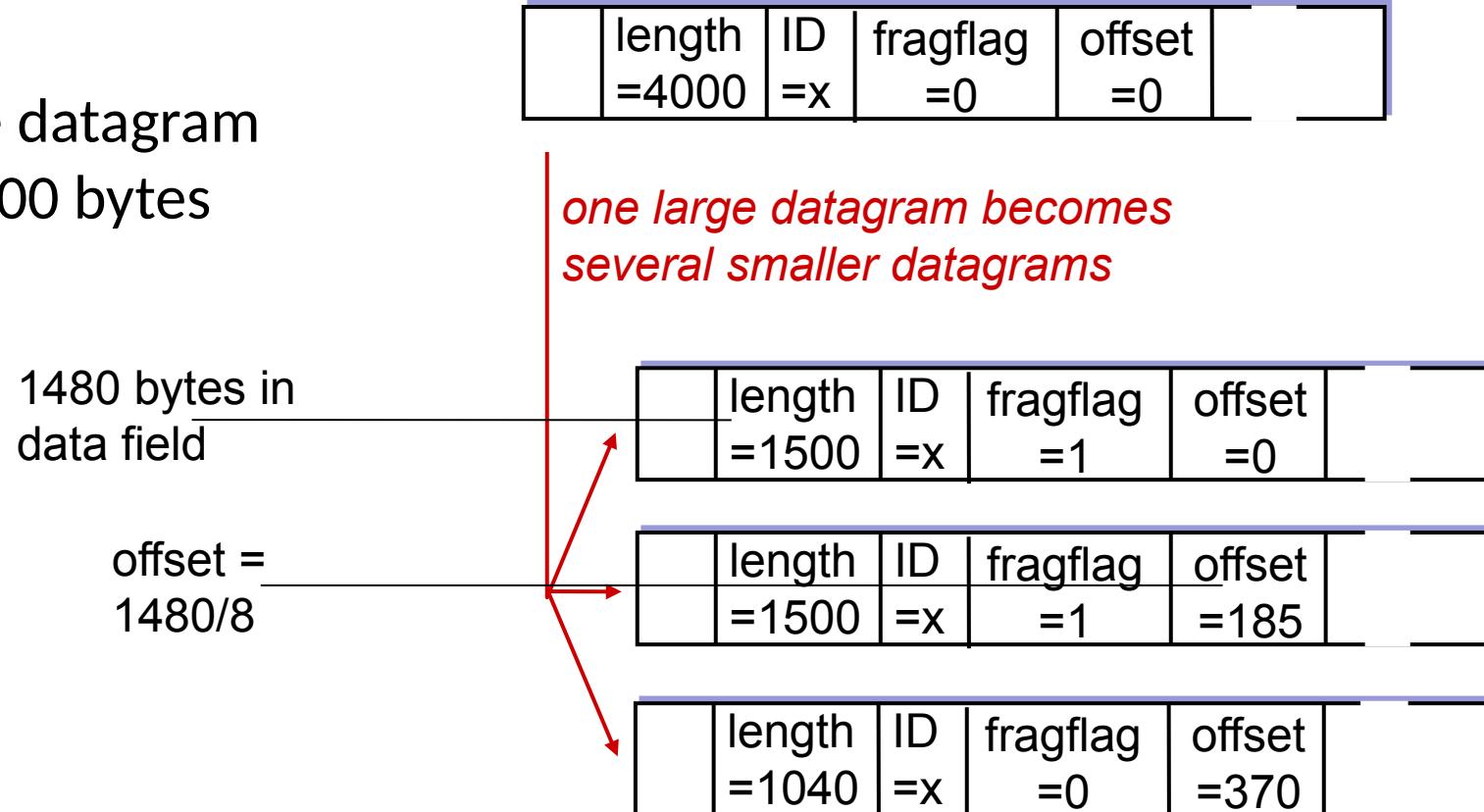
- network links have MTU (max. transfer size) - largest possible link-level frame
  - different link types, different MTUs
- large IP datagram divided (“fragmented”) within net
  - one datagram becomes several datagrams
  - “reassembled” only at *destination*
  - IP header bits used to identify, order related fragments



# IP fragmentation/reassembly

## example:

- 4000 byte datagram
- MTU = 1500 bytes



# DHCP: Wireshark output (home LAN)

Message type: **Boot Request (1)**

Hardware type: Ethernet

Hardware address length: 6

Hops: 0

**Transaction ID: 0x6b3a11b7**

Seconds elapsed: 0

Bootp flags: 0x0000 (Unicast)

Client IP address: 0.0.0.0 (0.0.0.0)

Your (client) IP address: 0.0.0.0 (0.0.0.0)

Next server IP address: 0.0.0.0 (0.0.0.0)

Relay agent IP address: 0.0.0.0 (0.0.0.0)

**Client MAC address: Wistron\_23:68:8a (00:16:d3:23:68:8a)**

Server host name not given

Boot file name not given

Magic cookie: (OK)

Option: (t=53,l=1) **DHCP Message Type = DHCP Request**

Option: (61) Client identifier

Length: 7; Value: 010016D323688A;

Hardware type: Ethernet

Client MAC address: Wistron\_23:68:8a (00:16:d3:23:68:8a)

Option: (t=50,l=4) Requested IP Address = 192.168.1.101

Option: (t=12,l=5) Host Name = "nomad"

**Option: (55) Parameter Request List**

Length: 11; Value: 010F03062C2E2F1F21F92B

**1 = Subnet Mask; 15 = Domain Name**

**3 = Router; 6 = Domain Name Server**

44 = NetBIOS over TCP/IP Name Server

.....

request

Message type: **Boot Reply (2)**

Hardware type: Ethernet

Hardware address length: 6

Hops: 0

**Transaction ID: 0x6b3a11b7**

Seconds elapsed: 0

Bootp flags: 0x0000 (Unicast)

**Client IP address: 192.168.1.101 (192.168.1.101)**

Your (client) IP address: 0.0.0.0 (0.0.0.0)

**Next server IP address: 192.168.1.1 (192.168.1.1)**

Relay agent IP address: 0.0.0.0 (0.0.0.0)

Client MAC address: Wistron\_23:68:8a (00:16:d3:23:68:8a)

Server host name not given

Boot file name not given

Magic cookie: (OK)

**Option: (t=53,l=1) DHCP Message Type = DHCP ACK**

**Option: (t=54,l=4) Server Identifier = 192.168.1.1**

**Option: (t=1,l=4) Subnet Mask = 255.255.255.0**

**Option: (t=3,l=4) Router = 192.168.1.1**

**Option: (6) Domain Name Server**

Length: 12; Value: 445747E2445749F244574092;

IP Address: 68.87.71.226;

IP Address: 68.87.73.242;

IP Address: 68.87.64.146

**Option: (t=15,l=20) Domain Name = "hsd1.ma.comcast.net."**

# Chapter 5

# Network

# Layer:

# Control

# Plane

A note on the use of these PowerPoint slides:

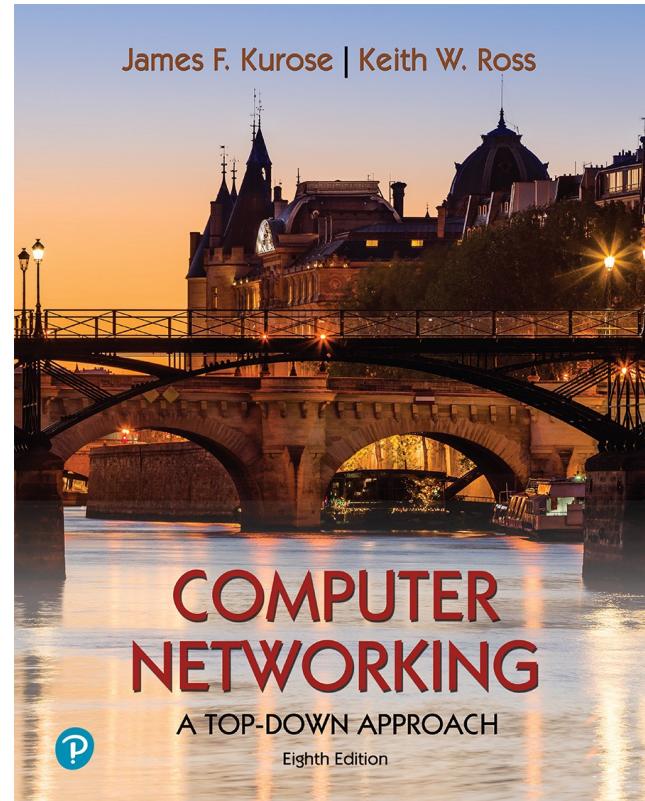
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2020  
J.F Kurose and K.W. Ross, All Rights Reserved



**Computer Networking: A  
Top-Down Approach**  
8<sup>th</sup> edition  
Jim Kurose, Keith Ross  
Pearson, 2020

# Network layer control plane: our goals

- understand principles behind network control plane:
  - traditional routing algorithms
  - SDN controllers
  - network management, configuration
- instantiation, implementation in the Internet:
  - OSPF, BGP
  - OpenFlow, ODL and ONOS controllers
  - Internet Control Message Protocol: ICMP
  - SNMP, YANG/NETCONF

# Network layer: “control plane” roadmap

- introduction
- routing protocols
  - link state
  - distance vector
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol



- network management, configuration
  - SNMP
  - NETCONF/YANG

# Network-layer functions

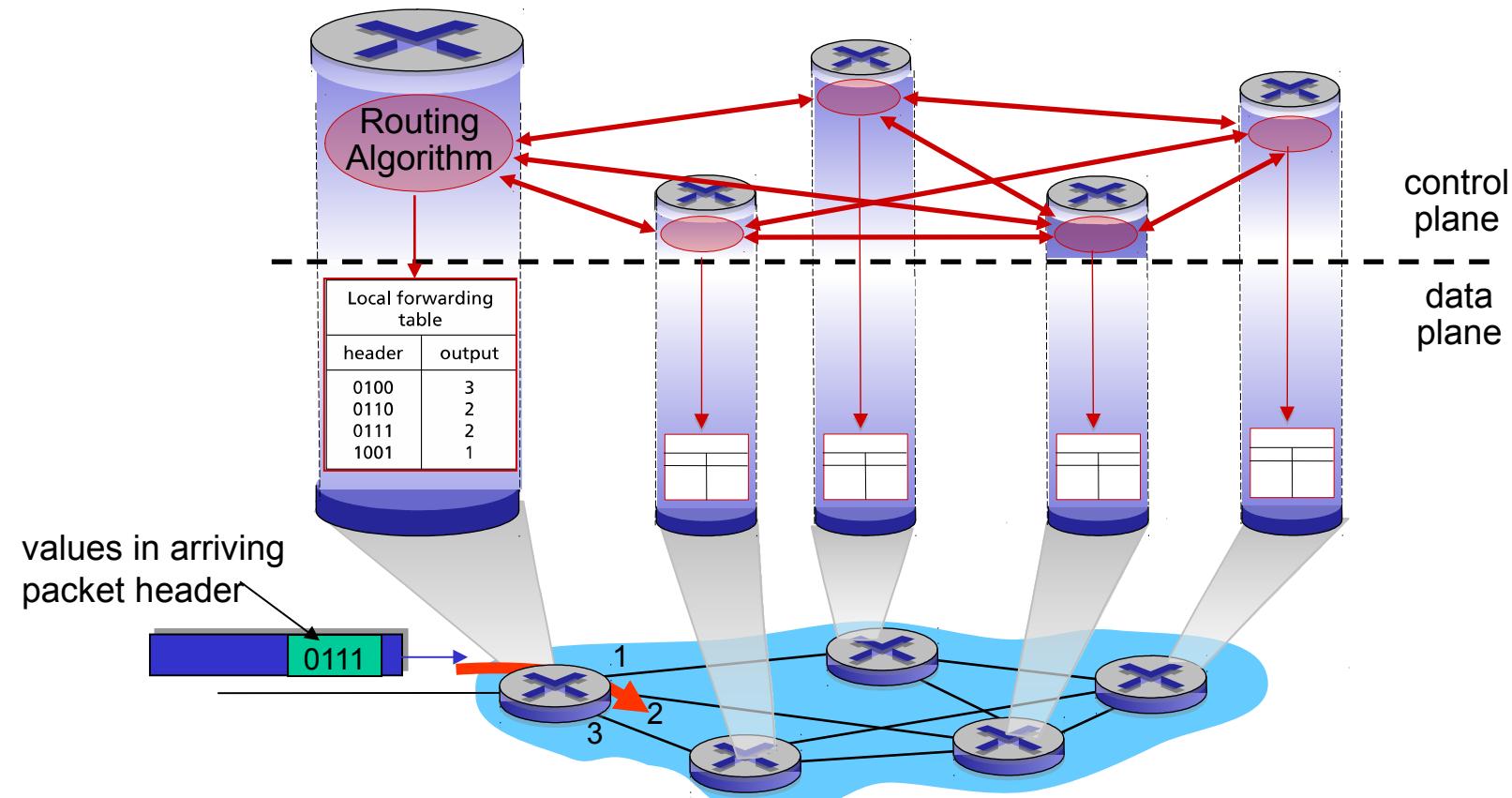
- **forwarding:** move packets from router's input to appropriate router output *data plane*
- **routing:** determine route taken by packets from source to destination *control plane*

Two approaches to structuring network control plane:

- per-router control (traditional)
- logically centralized control (software defined networking)

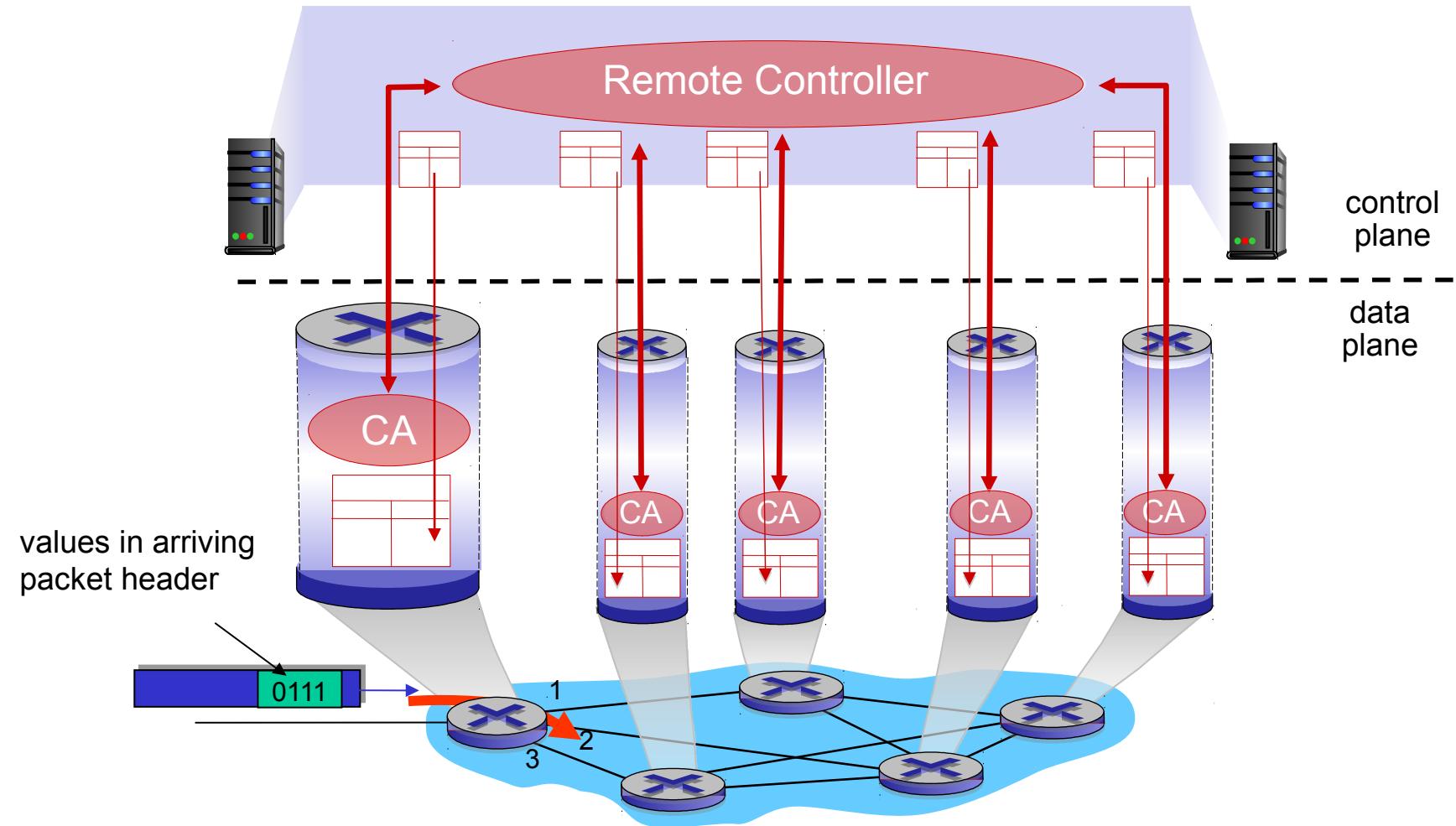
# Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane



# Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers



# Network layer: “control plane” roadmap

- introduction
- routing protocols
  - link state
  - distance vector
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol

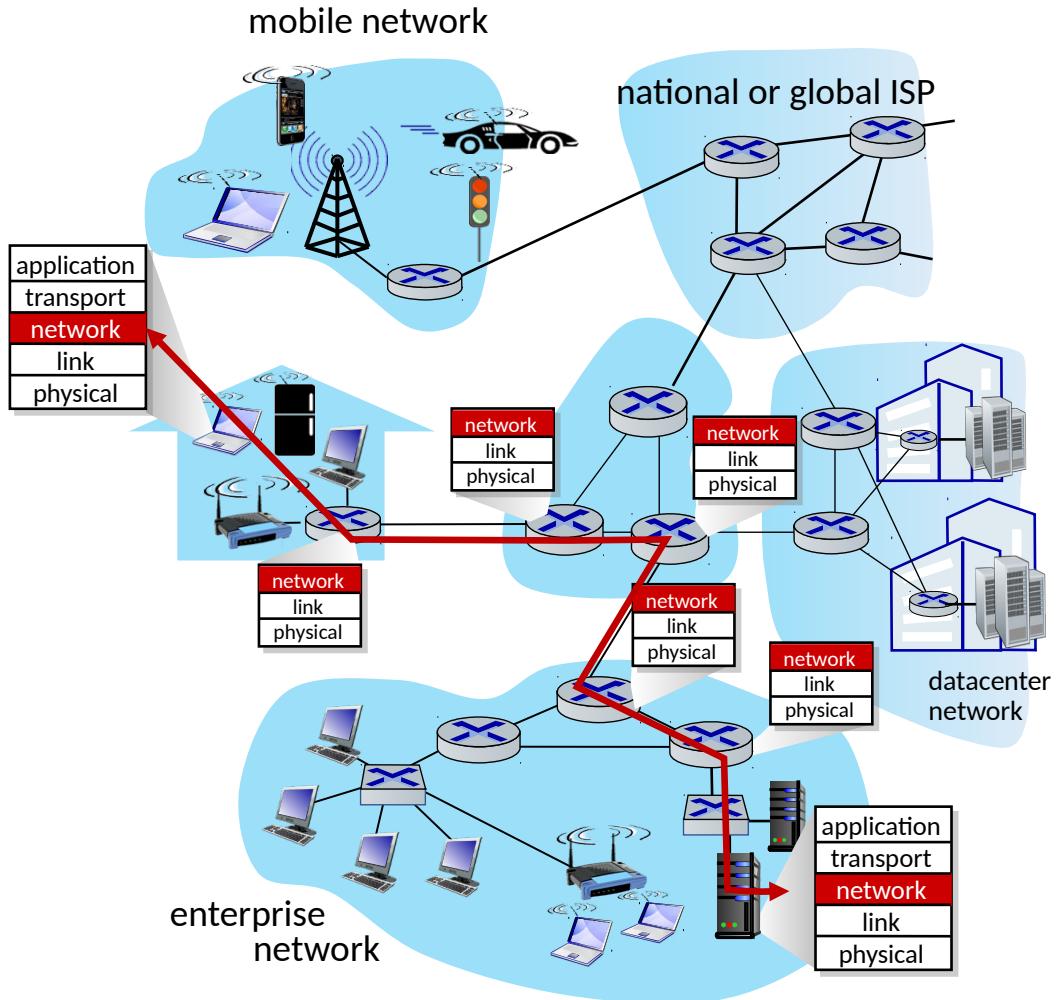


- network management, configuration
  - SNMP
  - NETCONF/YANG

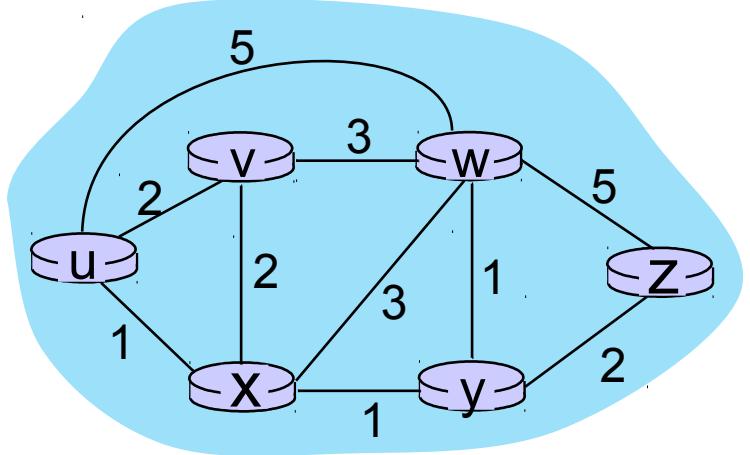
# Routing protocols

**Routing protocol goal:** determine “good” paths (equivalently, routes), from sending hosts to receiving host, through network of routers

- **path:** sequence of routers packets traverse from given initial source host to final destination host
- **“good”:** least “cost”, “fastest”, “least congested”
- routing: a “top-10” networking challenge!



# Graph abstraction: link costs



graph:  $G = (N, E)$

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

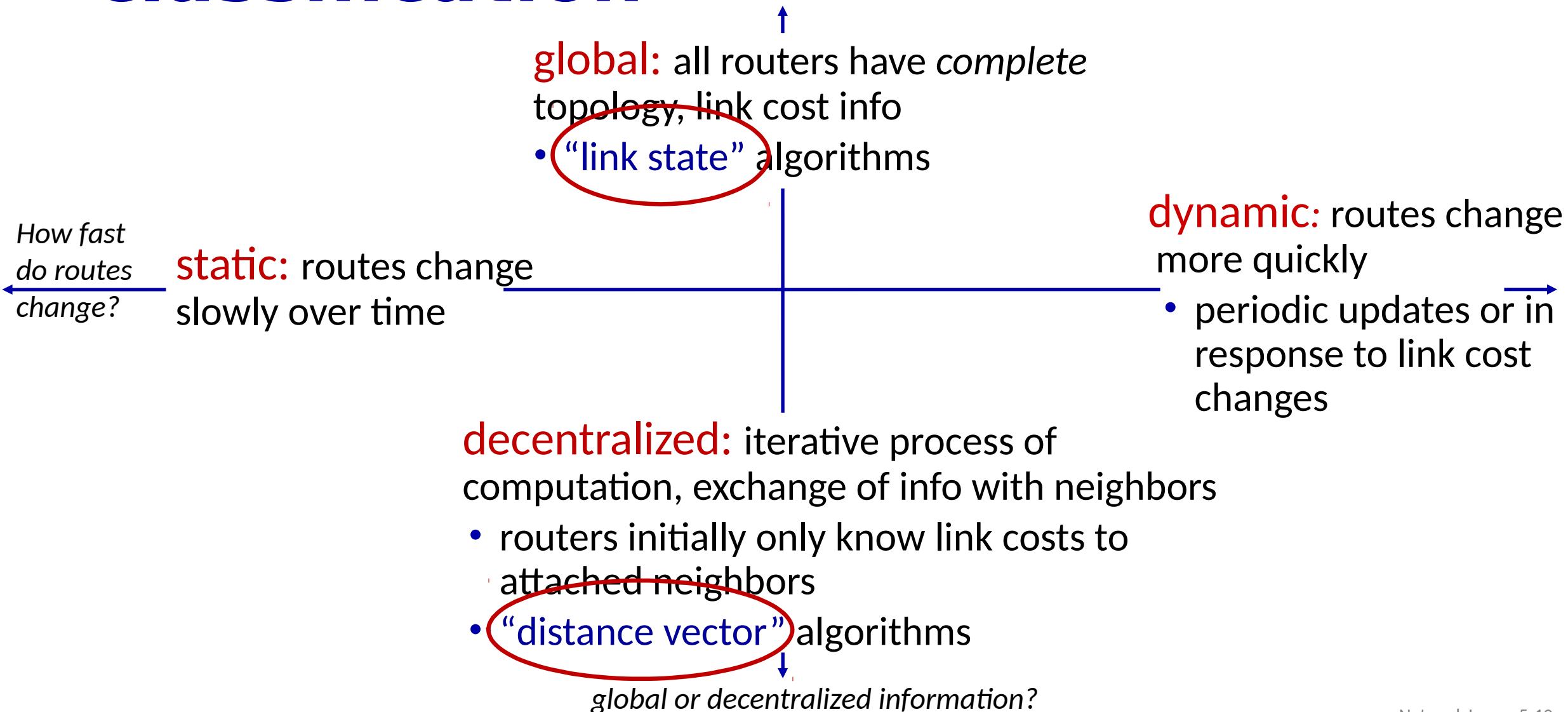
$E$ : set of links = {  $(u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z)$  }

$c_{a,b}$ : cost of *direct* link connecting  $a$  and  $b$

e.g.,  $c_{w,z} = 5, c_{u,z} = \infty$

cost defined by network operator:  
could always be 1, or inversely related  
to bandwidth, or inversely related to  
congestion

# Routing algorithm classification



# Network layer: “control plane” roadmap

- introduction
- routing protocols
  - link state
  - distance vector
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol



- network management, configuration
  - SNMP
  - NETCONF/YANG

# Dijkstra's link-state routing algorithm

- **centralized:** network 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 other nodes
  - gives *forwarding table* for that node
- **iterative:** after  $k$  iterations, know least cost path to  $k$  destinations

## notation

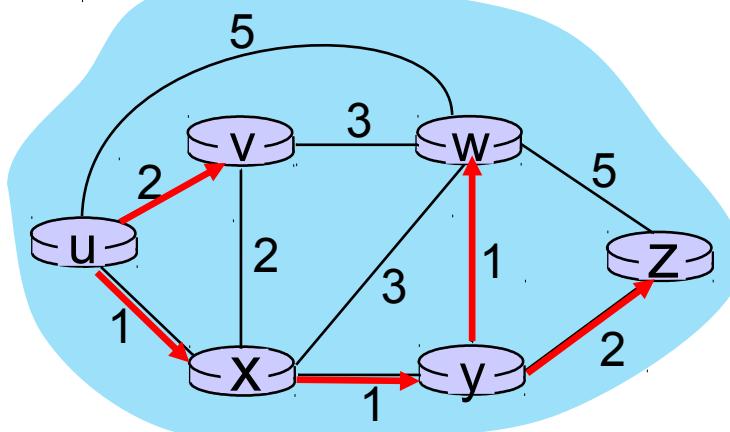
- $c_{x,y}$ : direct link cost from node  $x$  to  $y$ ;  $= \infty$  if not direct neighbors
- $D(v)$ : *current estimate* of cost of least-cost-path from source to destination  $v$
- $p(v)$ : predecessor node along path from source to  $v$
- $N'$ : set of nodes whose least-cost-path *definitively* known

# Dijkstra's link-state routing algorithm

```
1 Initialization:
2    $N' = \{u\}$                                 /* compute least cost path from u to all other nodes */
3   for all nodes v
4     if v adjacent to u                      /* u initially knows direct-path-cost only to direct neighbors */
5       then  $D(v) = c_{u,v}$                   /* but may not be minimum cost!
6     else  $D(v) = \infty$ 
7
8 Loop
9   find w not in  $N'$  such that  $D(w)$  is a minimum
10  add w to  $N'$ 
11  update  $D(v)$  for all v adjacent to w and not in  $N'$  :
12     $D(v) = \min(D(v), D(w) + c_{w,v})$ 
13  /* new least-path-cost to v is either old least-cost-path to v or known
14  least-cost-path to w plus direct-cost from w to v */
15 until all nodes in  $N'$ 
```

# Dijkstra's algorithm: an 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	2, u	5, u	1, u	$\infty$	$\infty$
1	ux	2, u	4, x	2, x	$\infty$	$\infty$
2	uxy	2, u	3, y	4, y	4, y	4, y
3	uxyv	3, y	3, y	4, y	4, y	4, y
4	uxyvw					
5	uxyvwz					

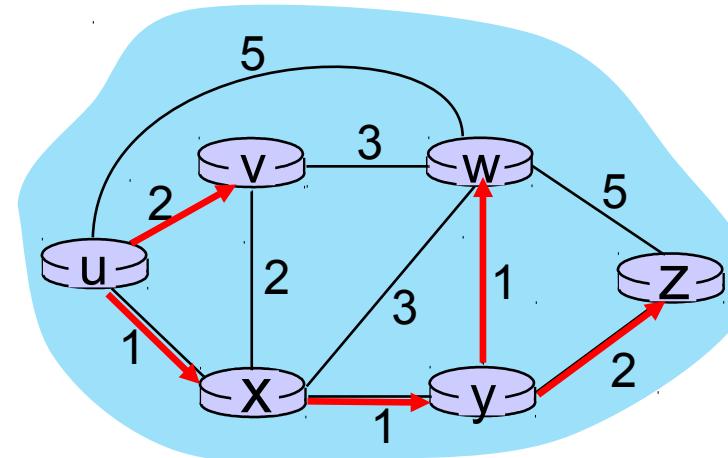


Initialization (step 0): For all  $a$ : if  $a$  adjacent to  $u$  then  $D(a) = c_{u,a}$

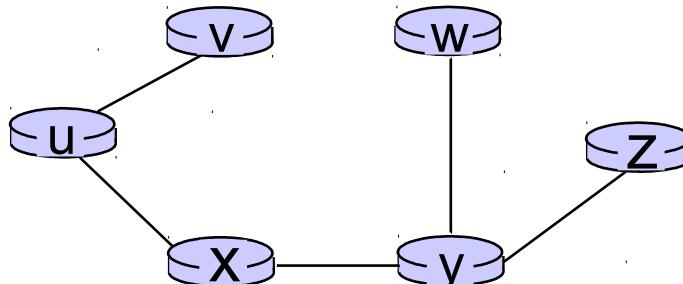
find  $a$  not in  $N'$  such that  $D(a)$  is a minimum  
add  $a$  to  $N'$   
update  $D(b)$  for all  $b$  adjacent to  $a$  and not in  $N'$  :  

$$D(b) = \min ( D(b), D(a) + c_{a,b} )$$

# Dijkstra's algorithm: an example



resulting least-cost-path tree from u:



resulting forwarding table in u:

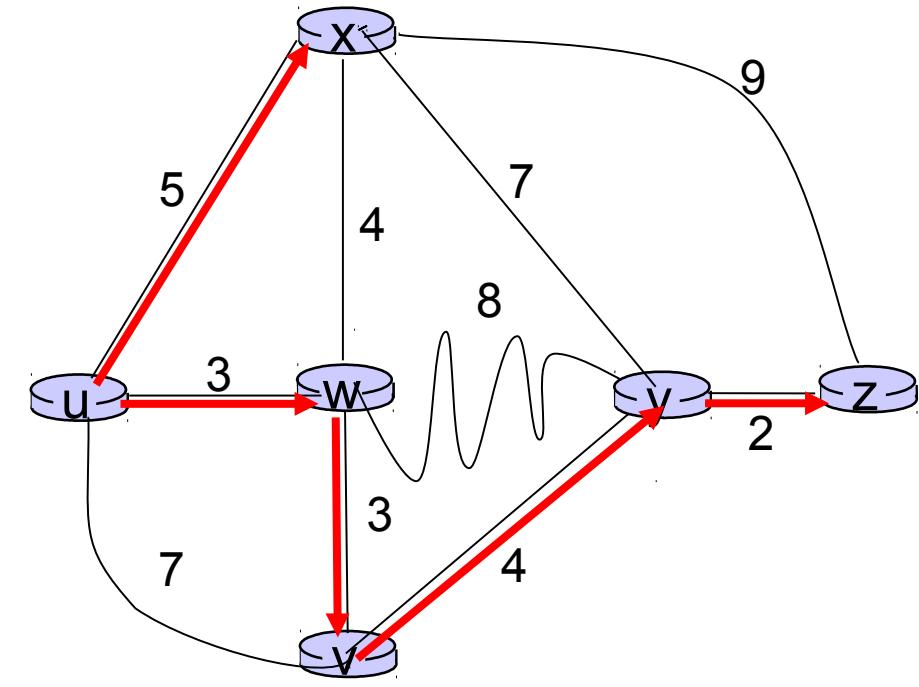
destination	outgoing link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
x	(u,x)

route from  $u$  to  $v$  directly

route from  $u$  to all other destinations via  $x$

# Dijkstra's algorithm: another example

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



notes:

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

# Dijkstra's algorithm: discussion

## algorithm complexity: $n$ nodes

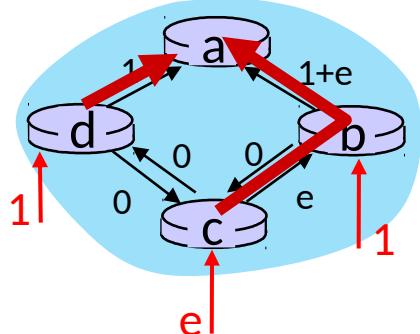
- each of  $n$  iteration: need to check all nodes,  $w$ , not in  $N$
- $n(n+1)/2$  comparisons:  $O(n^2)$  complexity
- more efficient implementations possible:  $O(n \log n)$

## message complexity:

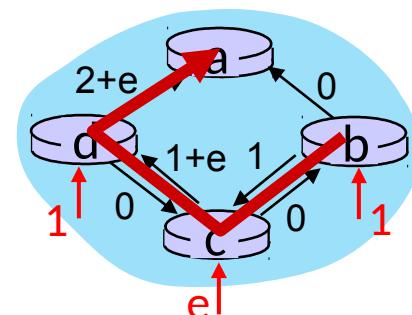
- each router must *broadcast* its link state information to other  $n$  routers
- efficient (and interesting!) broadcast algorithms:  $O(n)$  link crossings to disseminate a broadcast message from one source
- each router's message crosses  $O(n)$  links: overall message complexity:  $O(n^2)$

# Dijkstra's algorithm: oscillations possible

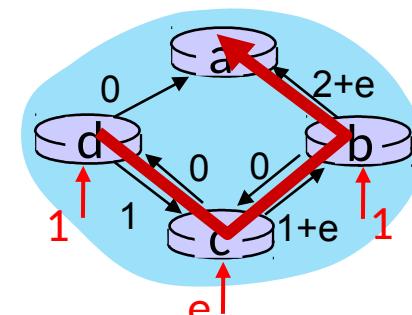
- when link costs depend on traffic volume, **route oscillations** possible
- sample scenario:
  - routing to destination a, traffic entering at d, c, e with rates 1,  $e$  ( $<1$ ), 1
  - link costs are directional, and volume-dependent



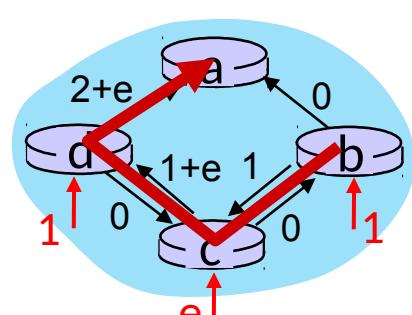
initially



given these costs,  
find new routing....  
resulting in new costs



given these costs,  
find new routing....  
resulting in new costs



given these costs,  
find new routing....  
resulting in new costs

# Network layer: “control plane” roadmap

- introduction
- routing protocols
  - link state
  - **distance vector**
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol



- network management, configuration
  - SNMP
  - NETCONF/YANG

# Distance vector algorithm

Based on *Bellman-Ford* (BF) equation (dynamic programming):

Bellman-Ford equation

Let  $D_x(y)$ : cost of least-cost path from  $x$  to  $y$ .

Then:

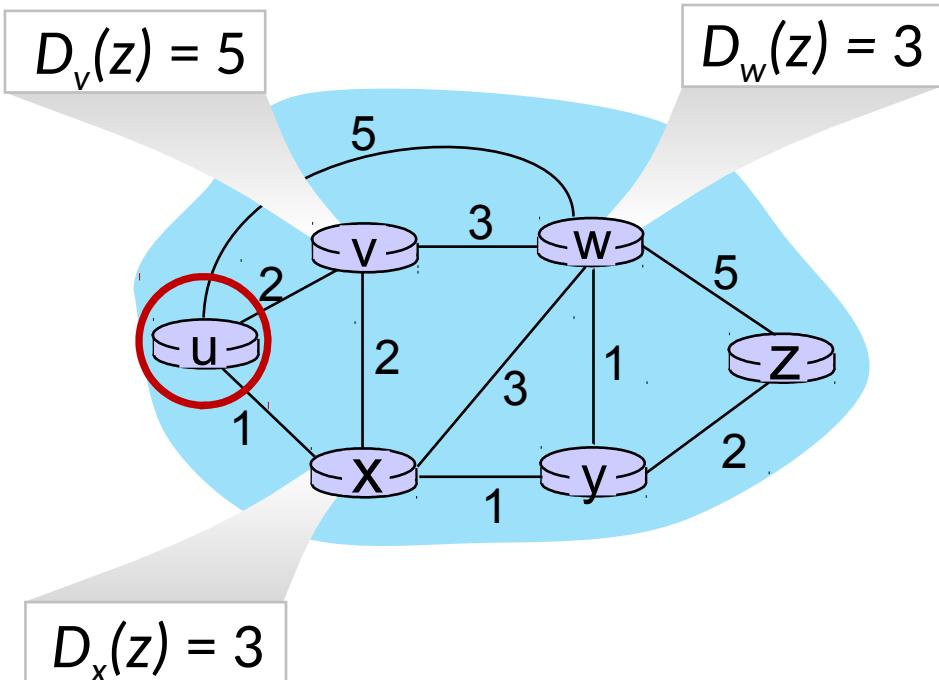
$$D_x(y) = \min_v \{ c_{x,v} + D_v(y) \}$$

*min* taken over all neighbors  $v$  of  $x$

$v$ 's estimated least-cost-path cost to  $y$   
direct cost of link from  $x$  to  $v$

# Bellman-Ford Example

Suppose that  $u$ 's neighboring nodes,  $x, v, w$ , know that for destination  $z$ :



Bellman-Ford equation says:

$$\begin{aligned} D_u(z) &= \min \{ c_{u,v} + D_v(z), \\ &\quad c_{u,w} + D_w(z), \\ &\quad c_{u,x} + D_x(z) \} \\ &= \min \{ 2 + 5, \\ &\quad 1 + 3, \\ &\quad 5 + 3 \} = 4 \end{aligned}$$

node achieving minimum (x) is next hop on estimated least-cost path to destination (z)

# 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 any neighbor, it updates its own DV using B-F equation:

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

- under minor, natural conditions, the estimate  $D_x(y)$  converge to the actual least cost  $d_x(y)$

# Distance vector algorithm:

each node:

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

*recompute* DV estimates using DV received from neighbor

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

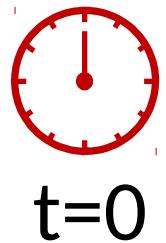
**iterative, asynchronous:** each local iteration caused by:

- local link cost change
- DV update message from neighbor

**distributed, self-stopping:** each node notifies neighbors *only* when its DV changes

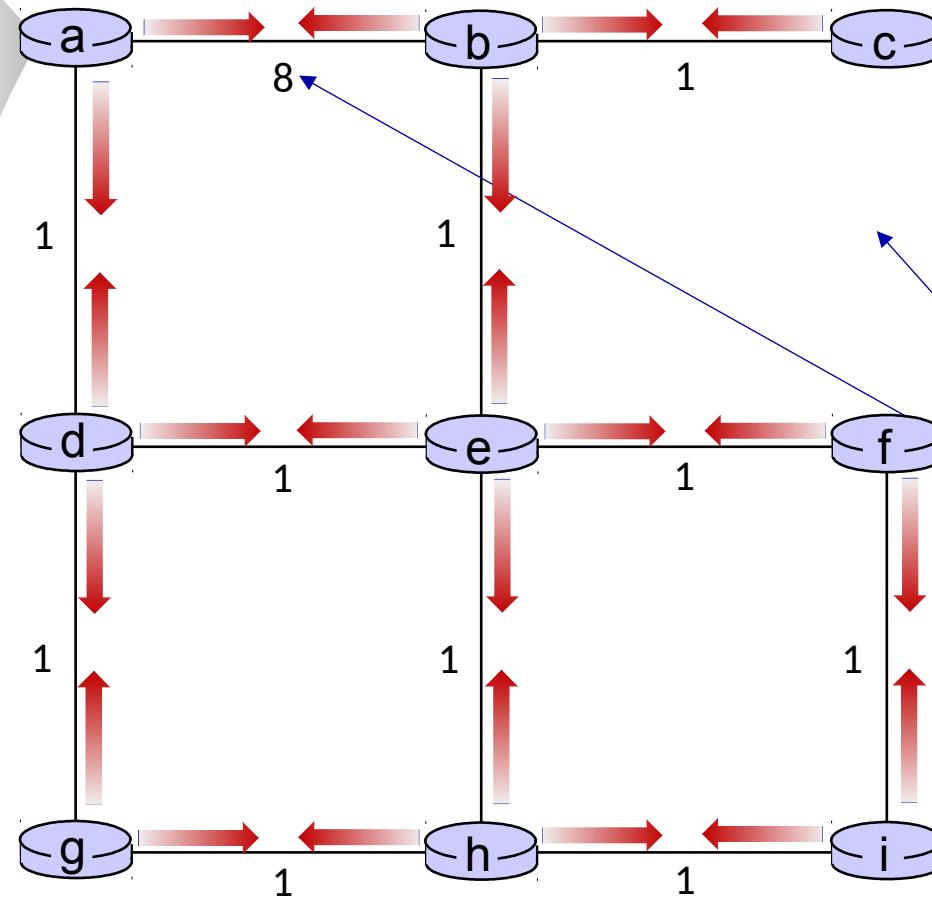
- neighbors then notify their neighbors – *only if necessary*
- no notification received, no actions taken!

# Distance vector: example



- All nodes have distance estimates to nearest neighbors (only)
- All nodes send their local distance vector to their neighbors

DV in a:
$D_a(a)=0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$



- A few asymmetries:
- missing link
  - larger cost

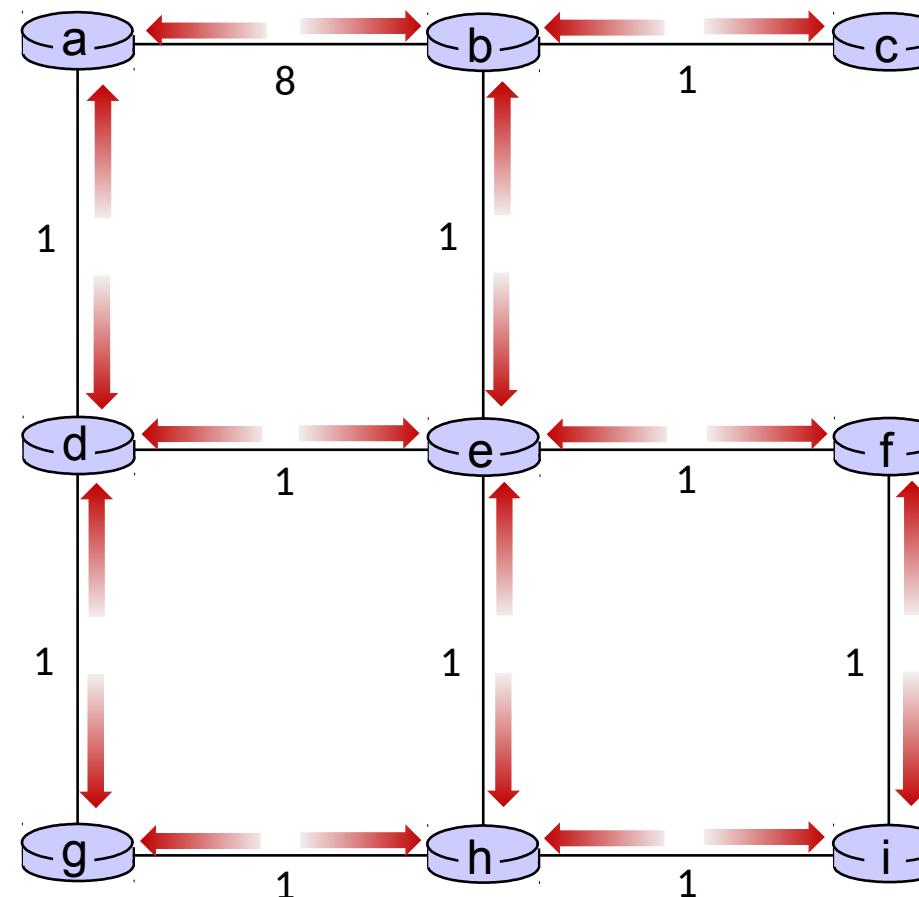
# Distance vector example: iteration



$t=1$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



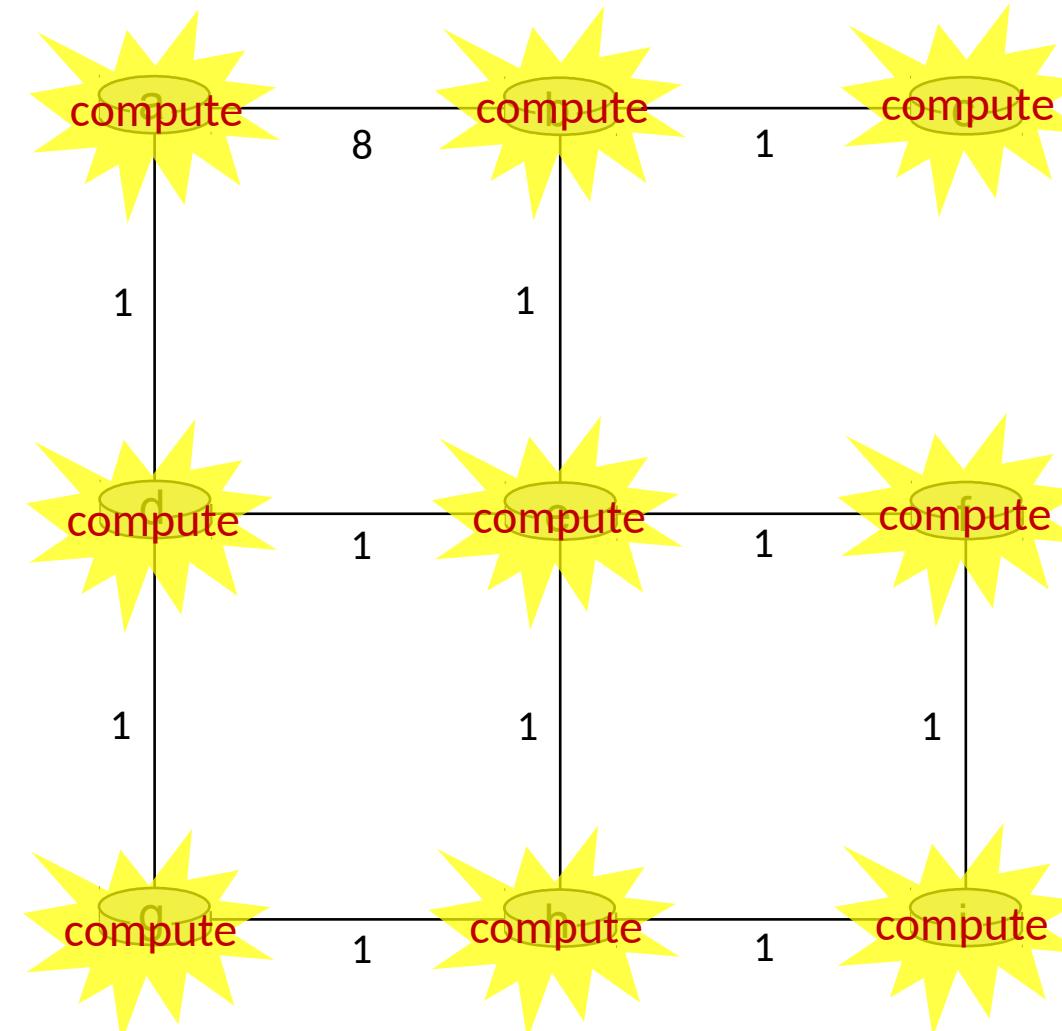
# Distance vector example: iteration



$t=1$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



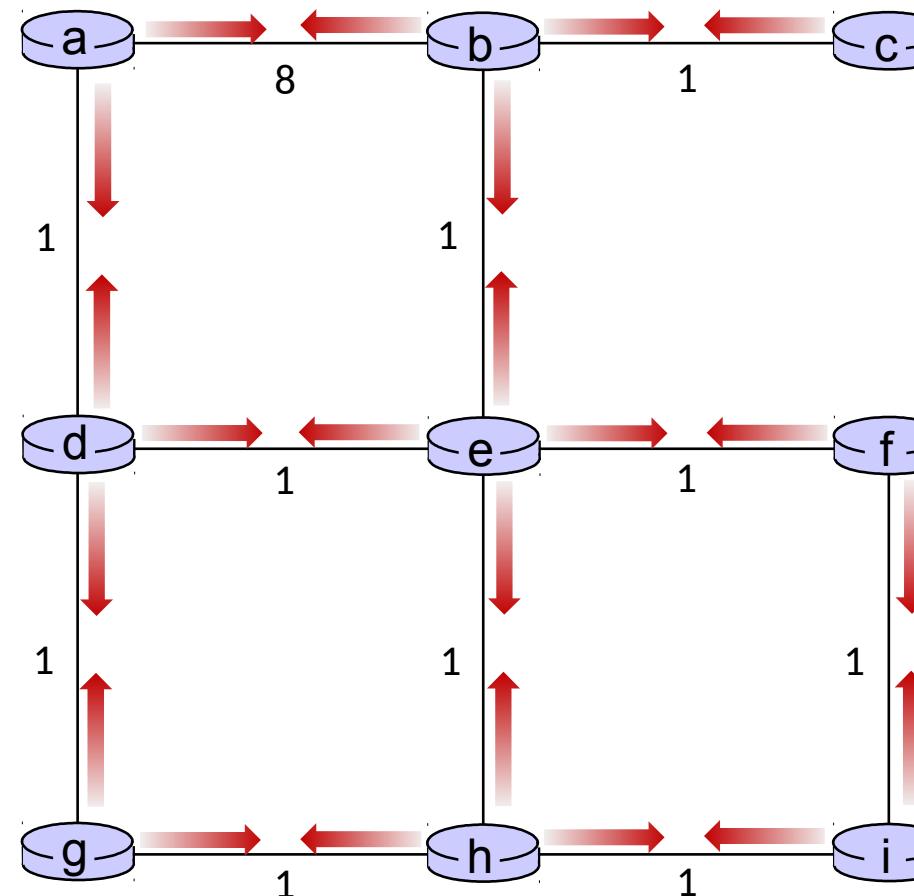
# Distance vector example: iteration



$t=1$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



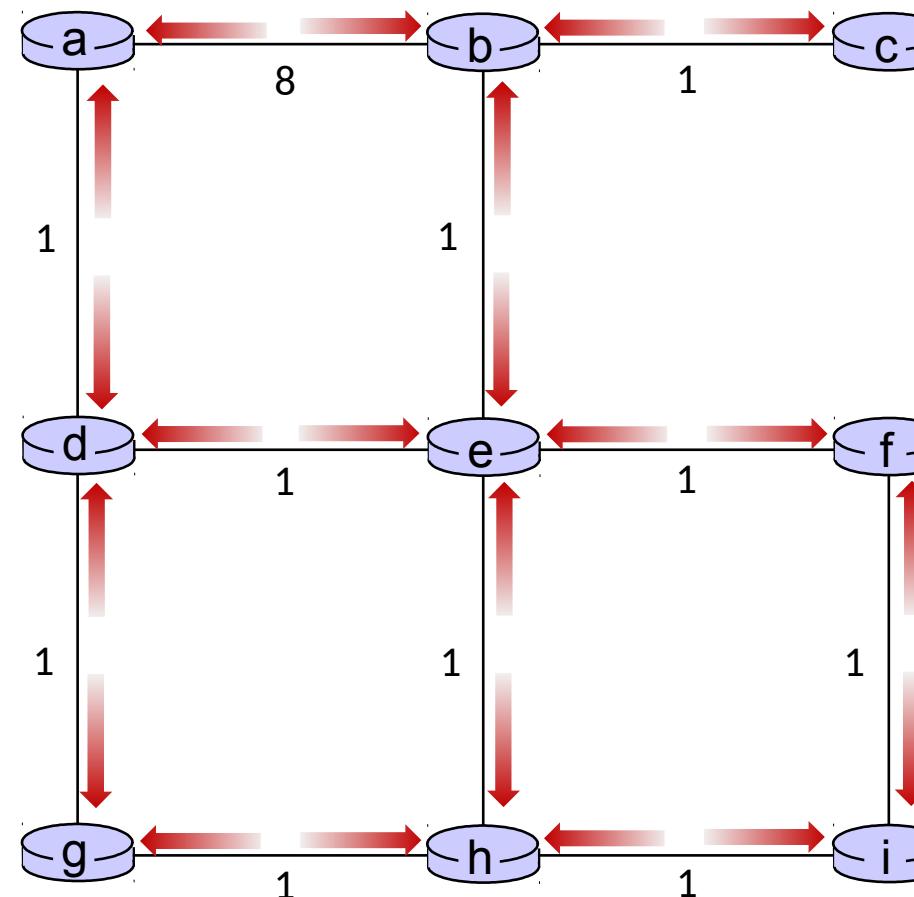
# Distance vector example: iteration



$t=2$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



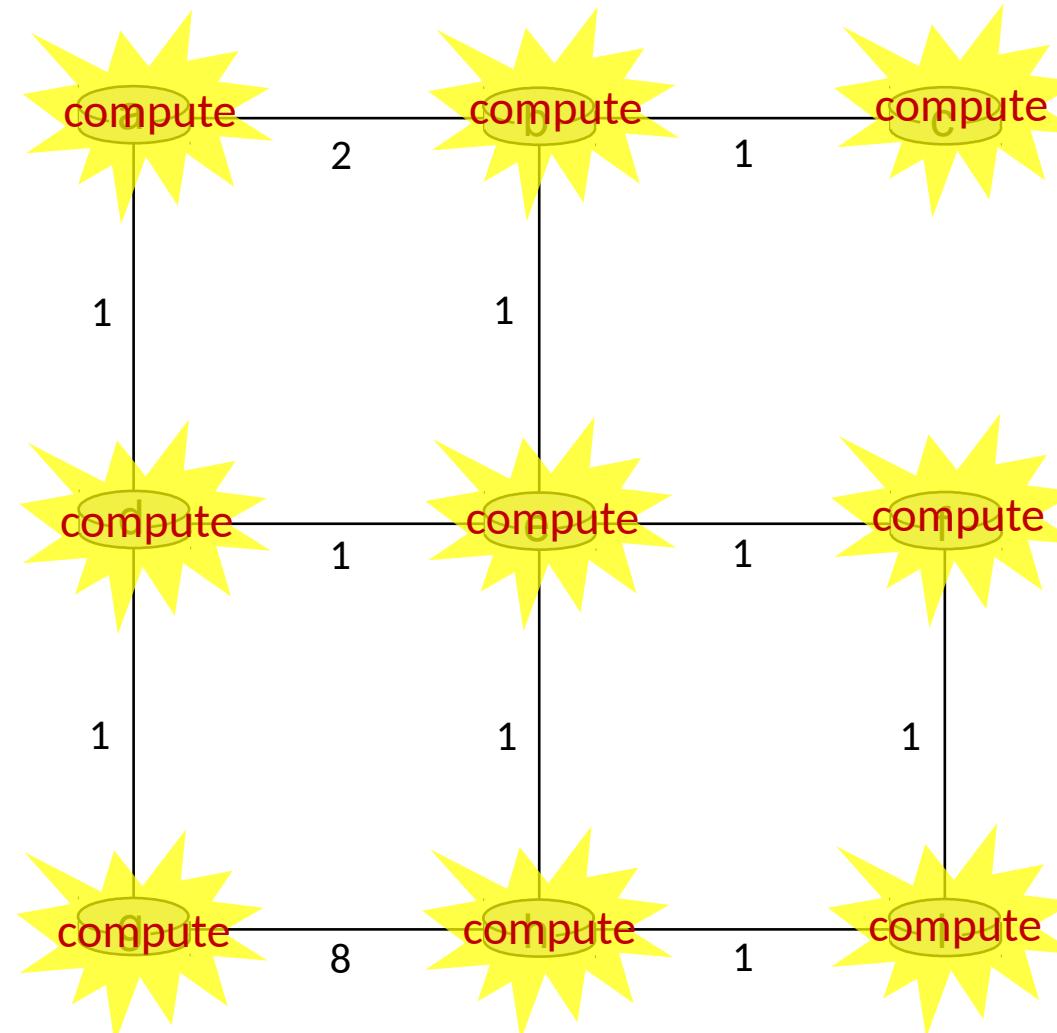
# Distance vector example: iteration



$t=2$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



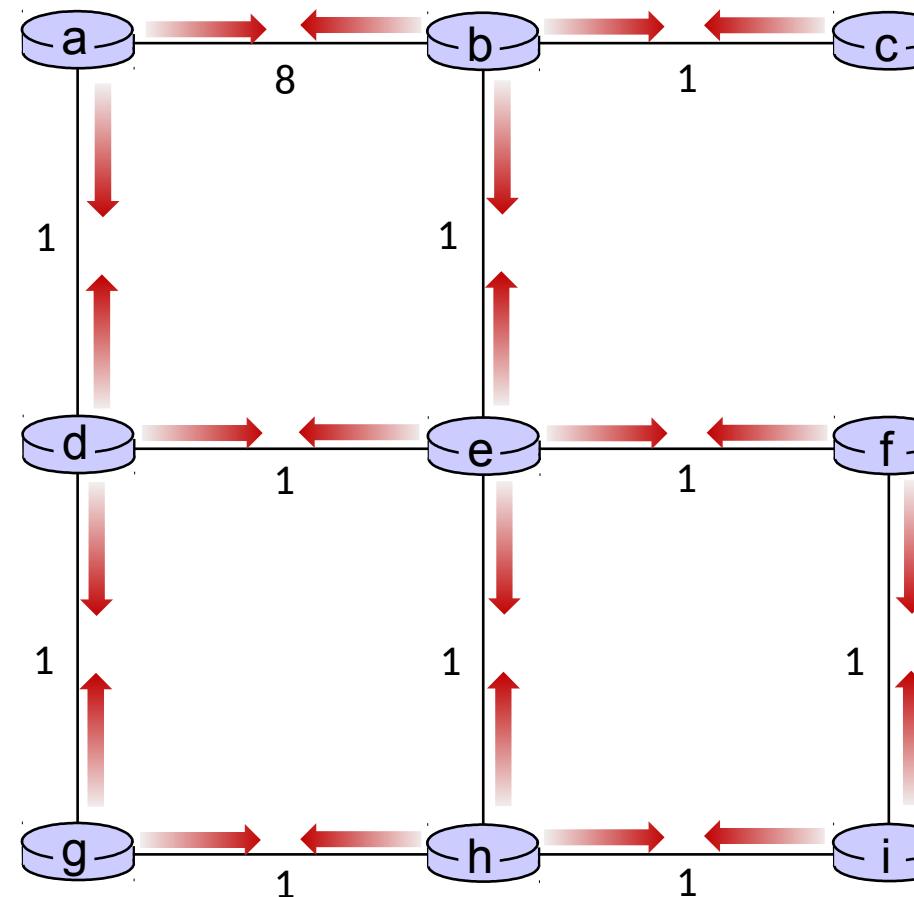
# Distance vector example: iteration



$t=2$

All nodes:

- receive distance vectors from neighbors
- compute their new local distance vector
- send their new local distance vector to neighbors



# Distance vector example: iteration

.... and so on

Let's next take a look at the iterative *computations* at nodes

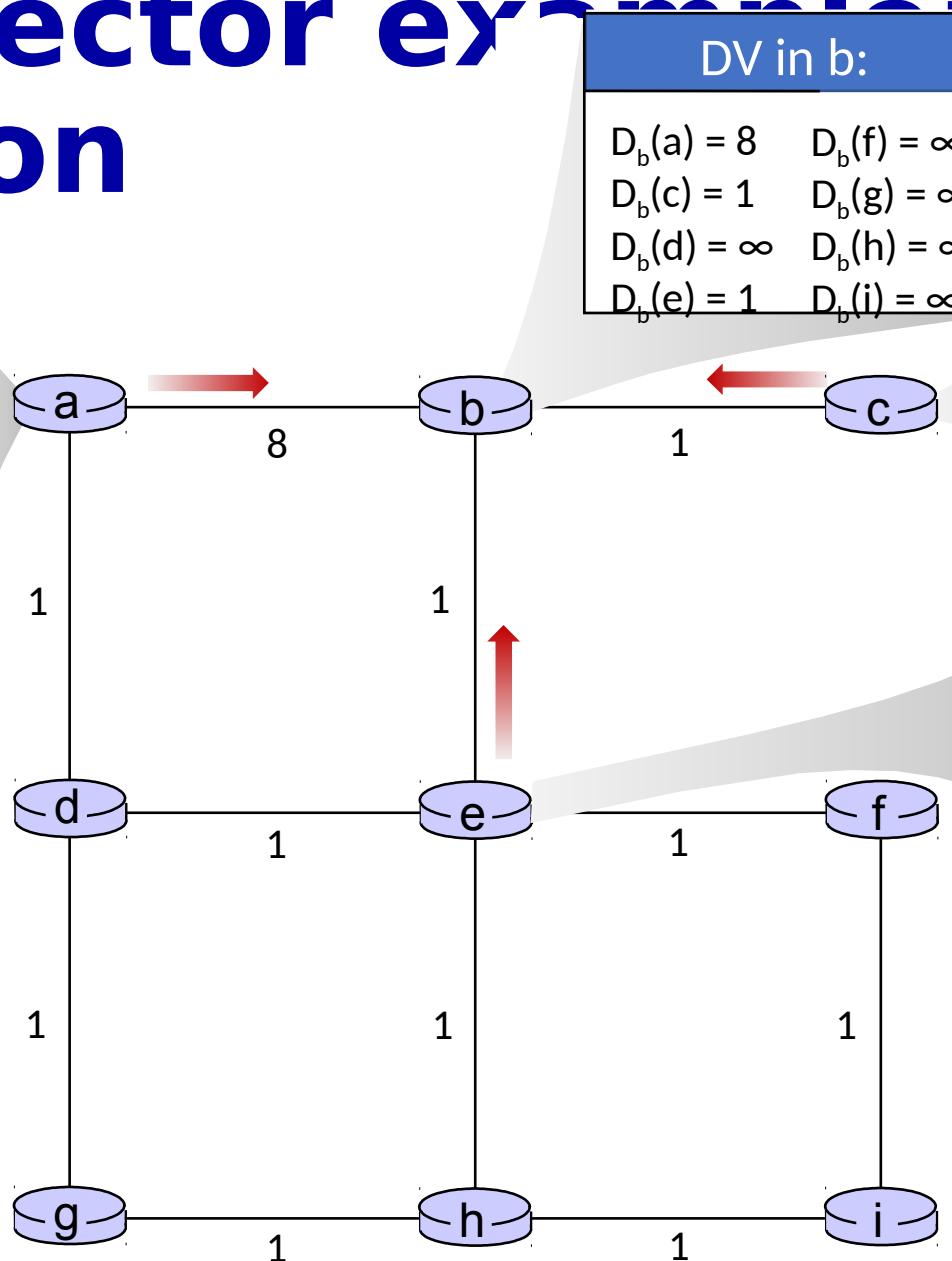
# Distance vector computation



$t=1$

- b receives DVs from a, c, e

DV in a:
$D_a(a) = 0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$



DV in b:
$D_b(a) = 8$
$D_b(f) = \infty$
$D_b(c) = 1$
$D_b(g) = \infty$
$D_b(d) = \infty$
$D_b(h) = \infty$
$D_b(e) = 1$
$D_b(i) = \infty$

DV in c:
$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

DV in e:
$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

# Distance vector computation



$t=1$

- b receives DVs from a, c, e, computes:

$$D_b(a) = \min\{c_{b,a} + D_a(a), c_{b,c} + D_c(a), c_{b,e} + D_e(a)\} = \min\{8, \infty, \infty\} = 8$$

$$D_b(c) = \min\{c_{b,a} + D_a(c), c_{b,c} + D_c(c), c_{b,e} + D_e(c)\} = \min\{\infty, 1, \infty\} = 1$$

$$D_b(d) = \min\{c_{b,a} + D_a(d), c_{b,c} + D_c(d), c_{b,e} + D_e(d)\} = \min\{9, 2, \infty\} = 2$$

$$D_b(e) = \min\{c_{b,a} + D_a(e), c_{b,c} + D_c(e), c_{b,e} + D_e(e)\} = \min\{\infty, \infty, 1\} = 1$$

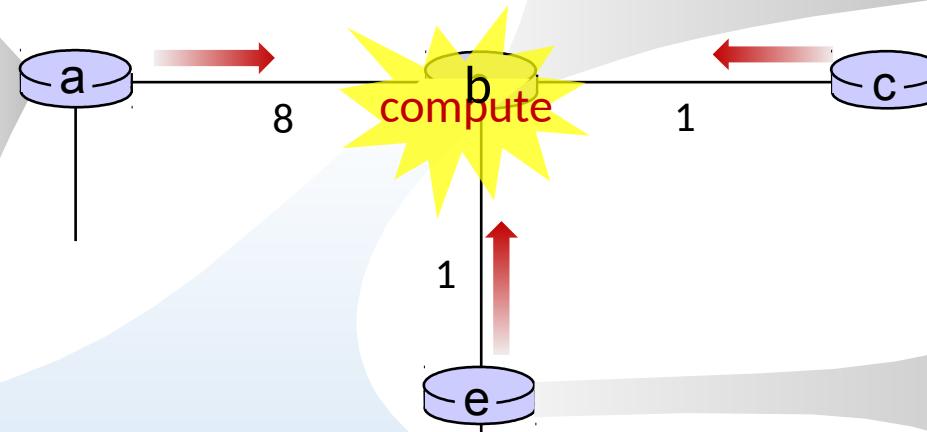
$$D_b(f) = \min\{c_{b,a} + D_a(f), c_{b,c} + D_c(f), c_{b,e} + D_e(f)\} = \min\{\infty, \infty, 2\} = 2$$

$$D_b(g) = \min\{c_{b,a} + D_a(g), c_{b,c} + D_c(g), c_{b,e} + D_e(g)\} = \min\{\infty, \infty, \infty\} = \infty$$

$$D_b(h) = \min\{c_{b,a} + D_a(h), c_{b,c} + D_c(h), c_{b,e} + D_e(h)\} = \min\{\infty, \infty, 2\} = 2$$

$$D_b(i) = \min\{c_{b,a} + D_a(i), c_{b,c} + D_c(i), c_{b,e} + D_e(i)\} = \min\{\infty, \infty, \infty\} = \infty$$

DV in a:
$D_a(a) = 0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$



DV in b:
$D_b(a) = 8$
$D_b(f) = \infty$
$D_b(c) = 1$
$D_b(g) = \infty$
$D_b(d) = \infty$
$D_b(h) = \infty$
$D_b(e) = 1$
$D_b(i) = \infty$

DV in c:
$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

DV in e:
$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

DV in b:
$D_b(a) = 8$
$D_b(f) = 2$
$D_b(c) = 1$
$D_b(g) = \infty$
$D_b(d) = 2$
$D_b(h) = 2$
$D_b(e) = 1$
$D_b(i) = \infty$

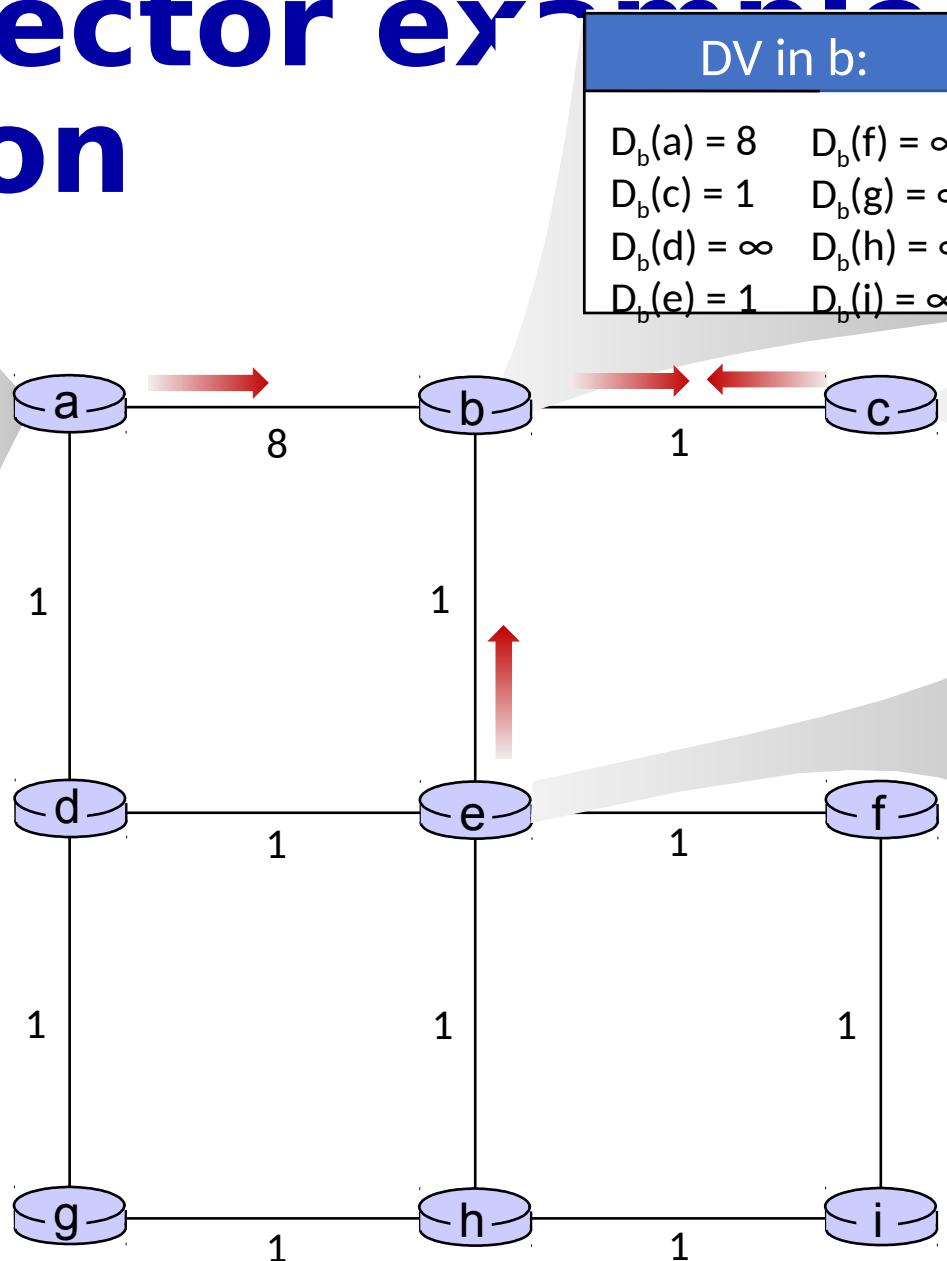
# Distance vector computation



$t=1$

- c receives DVs from b

DV in a:
$D_a(a) = 0$
$D_a(b) = 8$
$D_a(c) = \infty$
$D_a(d) = 1$
$D_a(e) = \infty$
$D_a(f) = \infty$
$D_a(g) = \infty$
$D_a(h) = \infty$
$D_a(i) = \infty$



DV in b:
$D_b(a) = 8$
$D_b(f) = \infty$
$D_b(c) = 1$
$D_b(g) = \infty$
$D_b(d) = \infty$
$D_b(h) = \infty$
$D_b(e) = 1$
$D_b(i) = \infty$

DV in c:
$D_c(a) = \infty$
$D_c(b) = 1$
$D_c(c) = 0$
$D_c(d) = \infty$
$D_c(e) = \infty$
$D_c(f) = \infty$
$D_c(g) = \infty$
$D_c(h) = \infty$
$D_c(i) = \infty$

DV in e:
$D_e(a) = \infty$
$D_e(b) = 1$
$D_e(c) = \infty$
$D_e(d) = 1$
$D_e(e) = 0$
$D_e(f) = 1$
$D_e(g) = \infty$
$D_e(h) = 1$
$D_e(i) = \infty$

# Distance vector computation



$t=1$

- c receives DVs from b computes:

$$D_c(a) = \min\{c_{c,b} + D_b(a)\} = 1 + 8 = 9$$

$$D_c(b) = \min\{c_{c,b} + D_b(b)\} = 1 + 0 = 1$$

$$D_c(d) = \min\{c_{c,b} + D_b(d)\} = 1 + \infty = \infty$$

$$D_c(e) = \min\{c_{c,b} + D_b(e)\} = 1 + 1 = 2$$

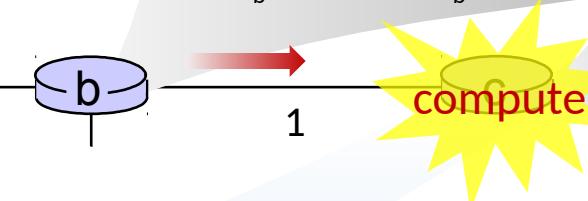
$$D_c(f) = \min\{c_{c,b} + D_b(f)\} = 1 + \infty = \infty$$

$$D_c(g) = \min\{c_{c,b} + D_b(g)\} = 1 + \infty = \infty$$

$$D_c(h) = \min\{c_{c,b} + D_b(h)\} = 1 + \infty = \infty$$

$$D_c(i) = \min\{c_{c,b} + D_b(i)\} = 1 + \infty = \infty$$

DV in b:	
$D_b(a) = 8$	$D_b(f) = \infty$
$D_b(c) = 1$	$D_b(g) = \infty$
$D_b(d) = \infty$	$D_b(h) = \infty$
$D_b(e) = 1$	$D_b(i) = \infty$



DV in c:	
$D_c(a) = \infty$	
$D_c(b) = 1$	
$D_c(c) = 0$	
$D_c(d) = \infty$	
$D_c(e) = \infty$	
$D_c(f) = \infty$	
$D_c(g) = \infty$	
$D_c(h) = \infty$	
$D_c(i) = \infty$	

DV in c:	
$D_c(a) = 9$	
$D_c(b) = 1$	
$D_c(c) = 0$	
$D_c(d) = 2$	
$D_c(e) = \infty$	
$D_c(f) = \infty$	
$D_c(g) = \infty$	
$D_c(h) = \infty$	
$D_c(i) = \infty$	

\* Check out the online interactive exercises for more examples:  
[http://gaia.cs.umass.edu/kurose\\_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

# Distance vector computation



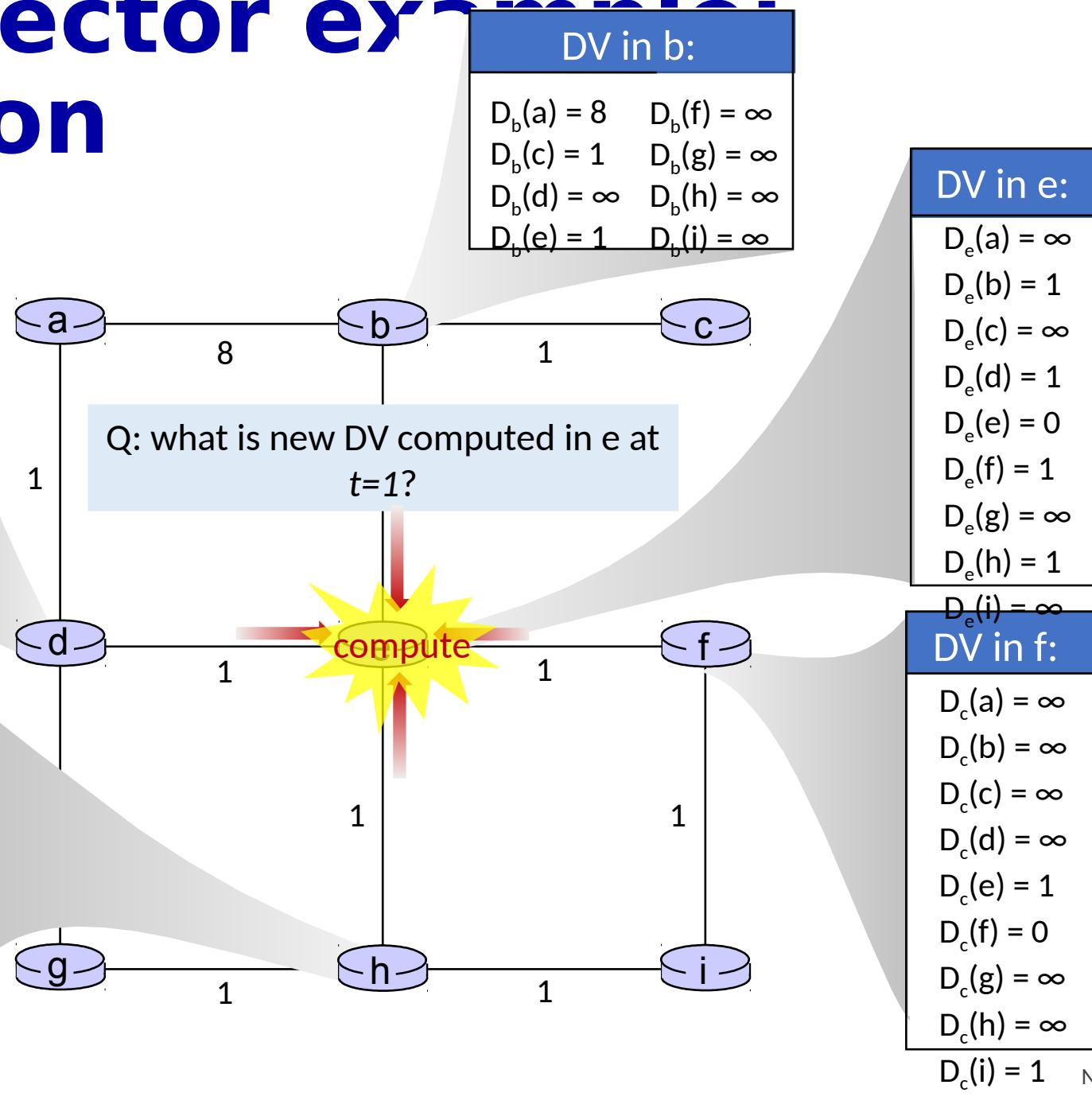
$t=1$

- e receives DVs from b, d, f, h

DV in d:
$D_c(a) = 1$
$D_c(b) = \infty$
$D_c(c) = \infty$
$D_c(d) = 0$
$D_c(e) = 1$
$D_c(f) = \infty$
$D_c(g) = 1$
$D_c(h) = \infty$
$D_c(i) = \infty$

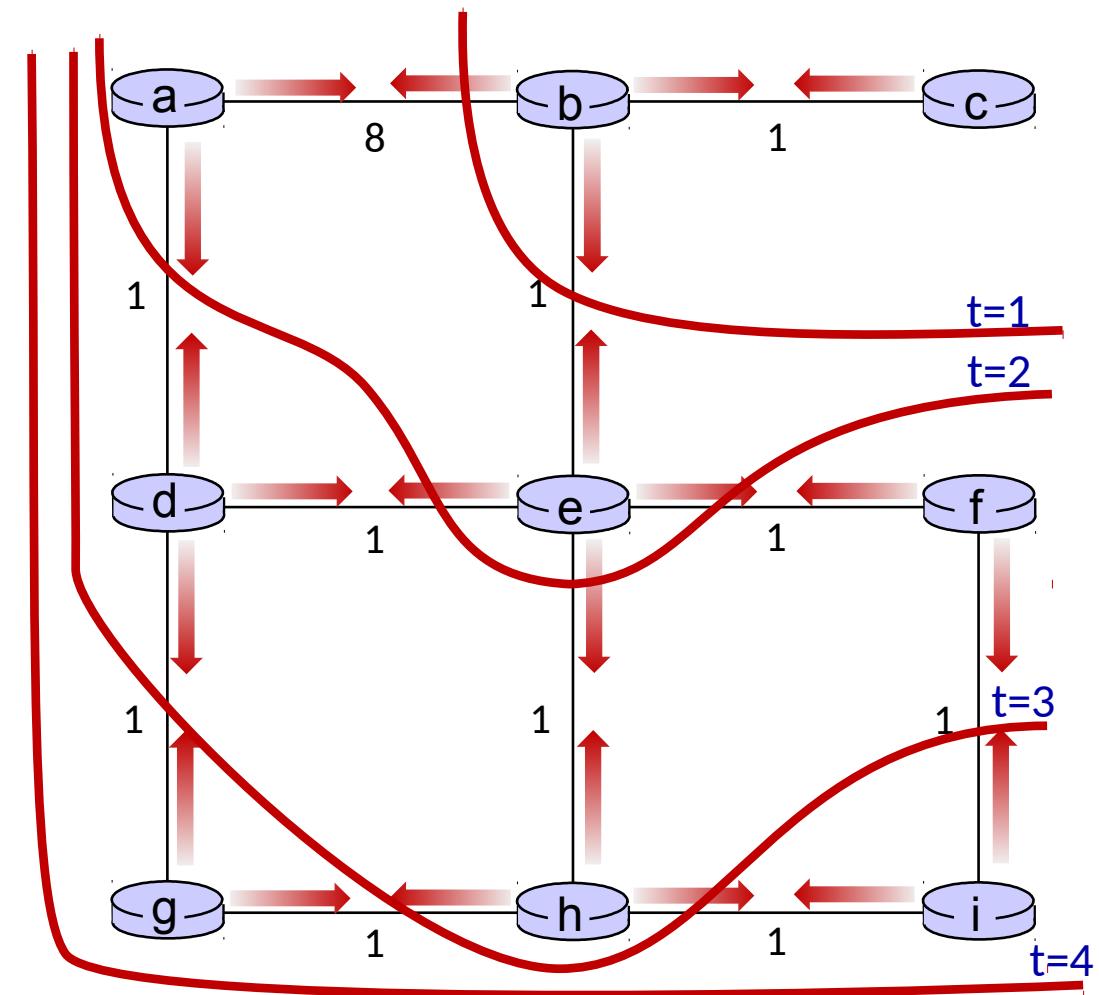
DV in h:
$D_c(a) = \infty$
$D_c(b) = \infty$
$D_c(c) = \infty$
$D_c(d) = \infty$
$D_c(e) = 1$
$D_c(f) = \infty$
$D_c(g) = 1$
$D_c(h) = 0$
$D_c(i) = 1$



# Distance vector: state information diffusion

Iterative communication, computation steps diffuses information through network:

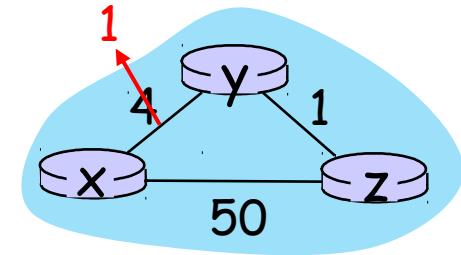
-  t=0 c's state at t=0 is at c only
-  t=1 c's state at t=0 has propagated to b, and may influence distance vector computations up to **1** hop away, i.e., at b
-  t=2 c's state at t=0 may now influence distance vector computations up to **2** hops away, i.e., at b and now at a, e as well
-  t=3 c's state at t=0 may influence distance vector computations up to **3** hops away, i.e., at b,a,e and now at c,f,h as well
-  t=4 c's state at t=0 may influence distance vector computations up to **4** hops away, i.e., at b,a,e, c, f, h and now at g,i as well



# Distance vector: link cost changes

## link cost changes:

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



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

“good news travels fast”

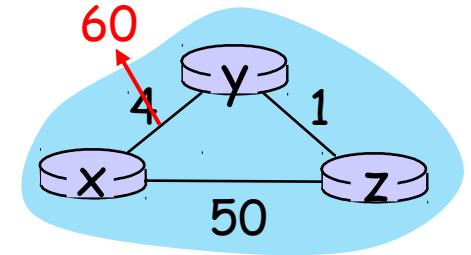
$t_1$ : z receives update from y, updates its table, computes new least cost to x , sends its neighbors its DV.

$t_2$ : 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 change
- “bad news travels slow” – count-to-infinity problem:
  - y sees direct link to x has new cost 60, but z has said it has a path at cost of 5. So y computes “my new cost to x will be 6, via z); notifies z of new cost of 6 to x.
  - z learns that path to x via y has new cost 6, so z computes “my new cost to x will be 7 via y), notifies y of new cost of 7 to x.
  - y learns that path to x via z has new cost 7, so y computes “my new cost to x will be 8 via y), notifies z of new cost of 8 to x.
  - z learns that path to x via y has new cost 8, so z computes “my new cost to x will be 9 via y), notifies y of new cost of 9 to x.
  - ...
- see text for solutions. *Distributed algorithms are tricky!*



# Comparison of LS and DV algorithms

## message complexity

LS:  $n$  routers,  $O(n^2)$  messages sent

DV: exchange between neighbors;  
convergence time varies

## speed of convergence

LS:  $O(n^2)$  algorithm,  $O(n^2)$  messages  
• may have oscillations

DV: convergence time varies  
• may have routing loops  
• count-to-infinity problem

robustness: what happens if router malfunctions, or is compromised?

### LS:

- router can advertise incorrect *link* cost
- each router computes only its own table

### DV:

- DV router can advertise incorrect *path* cost (“I have a *really* low cost path to everywhere”): black-holing
- each router’s table used by others: error propagate thru network

# Network layer: “control plane” roadmap

- introduction
- routing protocols
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol



- network management, configuration
  - SNMP
  - NETCONF/YANG

# Making routing scalable

our routing study thus far - idealized

- all routers identical
- network “flat”

... not true in practice

**scale:** billions of destinations:

- can't store all destinations in routing tables!
- routing table exchange would swamp links!

**administrative autonomy:**

- Internet: a network of networks
- each network admin may want to control routing in its own network

# Internet approach to scalable routing

aggregate routers into regions known as “autonomous systems” (AS) (a.k.a. “domains”)

**intra-AS (aka “intra-domain”):**

routing among *within same AS (“network”)*

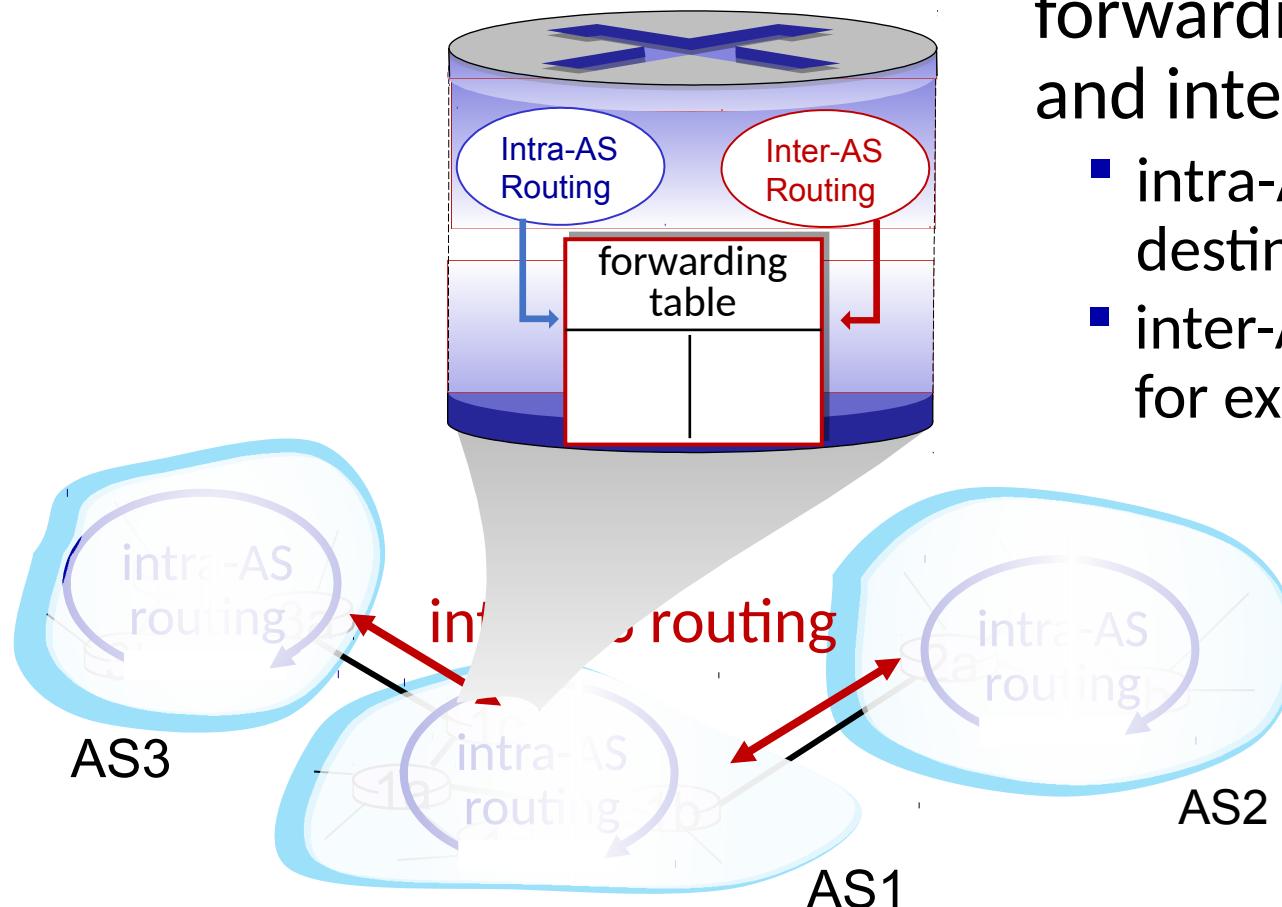
- all routers in AS must run same intra-domain protocol
- routers in different AS can run different intra-domain routing protocols
- **gateway router:** at “edge” of its own AS, has link(s) to router(s) in other AS'es

**inter-AS (aka “inter-domain”):**

routing *among* AS'es

- gateways perform inter-domain routing (as well as intra-domain routing)

# Interconnected ASes



forwarding table configured by intra-  
and inter-AS routing algorithms

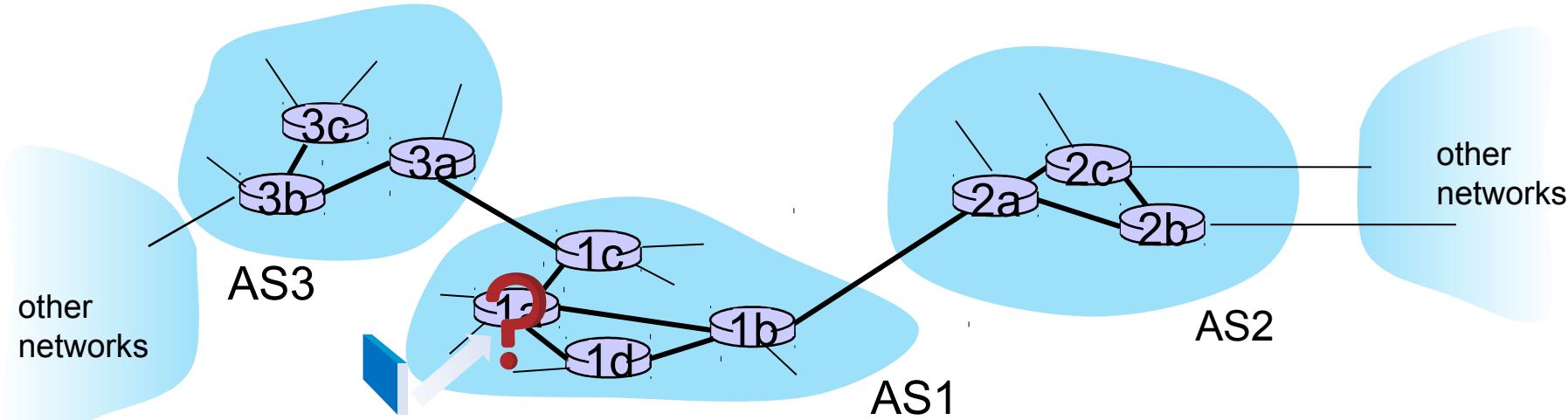
- intra-AS routing determine entries for destinations within AS
- inter-AS & intra-AS determine entries for external destinations

# Inter-AS routing: a role in intradomain forwarding

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

**AS1 inter-domain routing must:**

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



# Inter-AS routing: routing within an AS

most common intra-AS routing protocols:

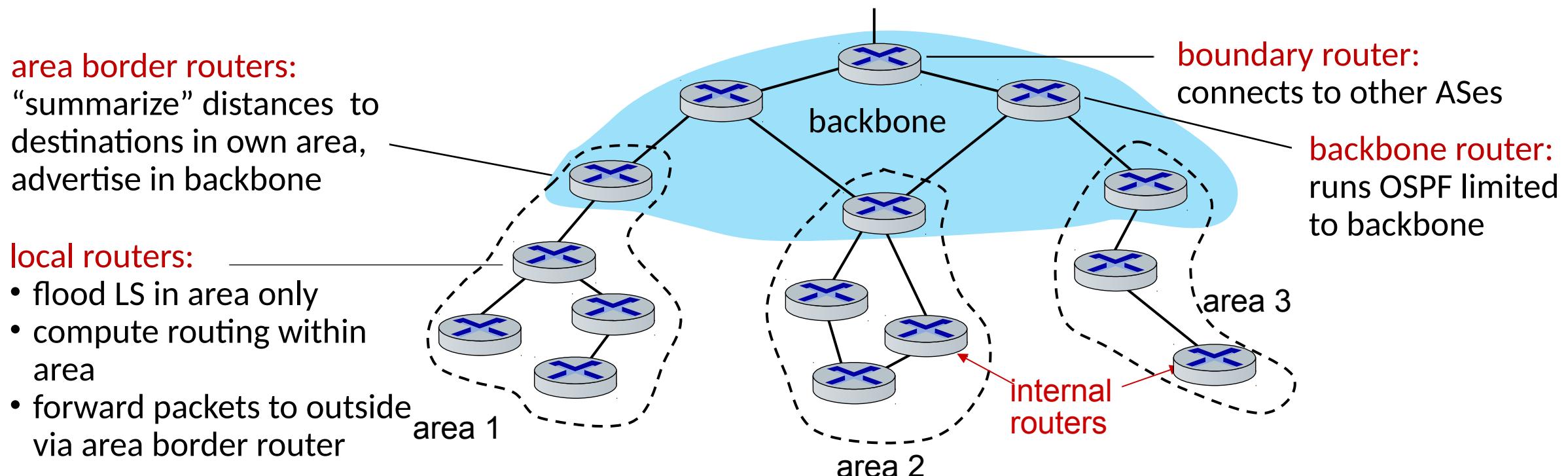
- **RIP: Routing Information Protocol [RFC 1723]**
  - classic DV: DVs exchanged every 30 secs
  - no longer widely used
- **EIGRP: Enhanced Interior Gateway Routing Protocol**
  - DV based
  - formerly Cisco-proprietary for decades (became open in 2013 [RFC 7868])
- **OSPF: Open Shortest Path First [RFC 2328]**
  - link-state routing
  - IS-IS protocol (ISO standard, not RFC standard) essentially same as OSPF

# OSPF (Open Shortest Path First) routing

- “open”: publicly available
- classic link-state
  - each router floods OSPF link-state advertisements (directly over IP rather than using TCP/UDP) to all other routers in entire AS
  - multiple link costs metrics possible: bandwidth, delay
  - each router has full topology, uses Dijkstra’s algorithm to compute forwarding table
- *security*: all OSPF messages authenticated (to prevent malicious intrusion)

# Hierarchical OSPF

- two-level hierarchy: local area, backbone.
  - link-state advertisements flooded only in area, or backbone
  - each node has detailed area topology; only knows direction to reach other destinations



# Network layer: “control plane” roadmap

- introduction
- routing protocols
- intra-ISP routing: OSPF
- **routing among ISPs: BGP**
- SDN control plane
- Internet Control Message Protocol

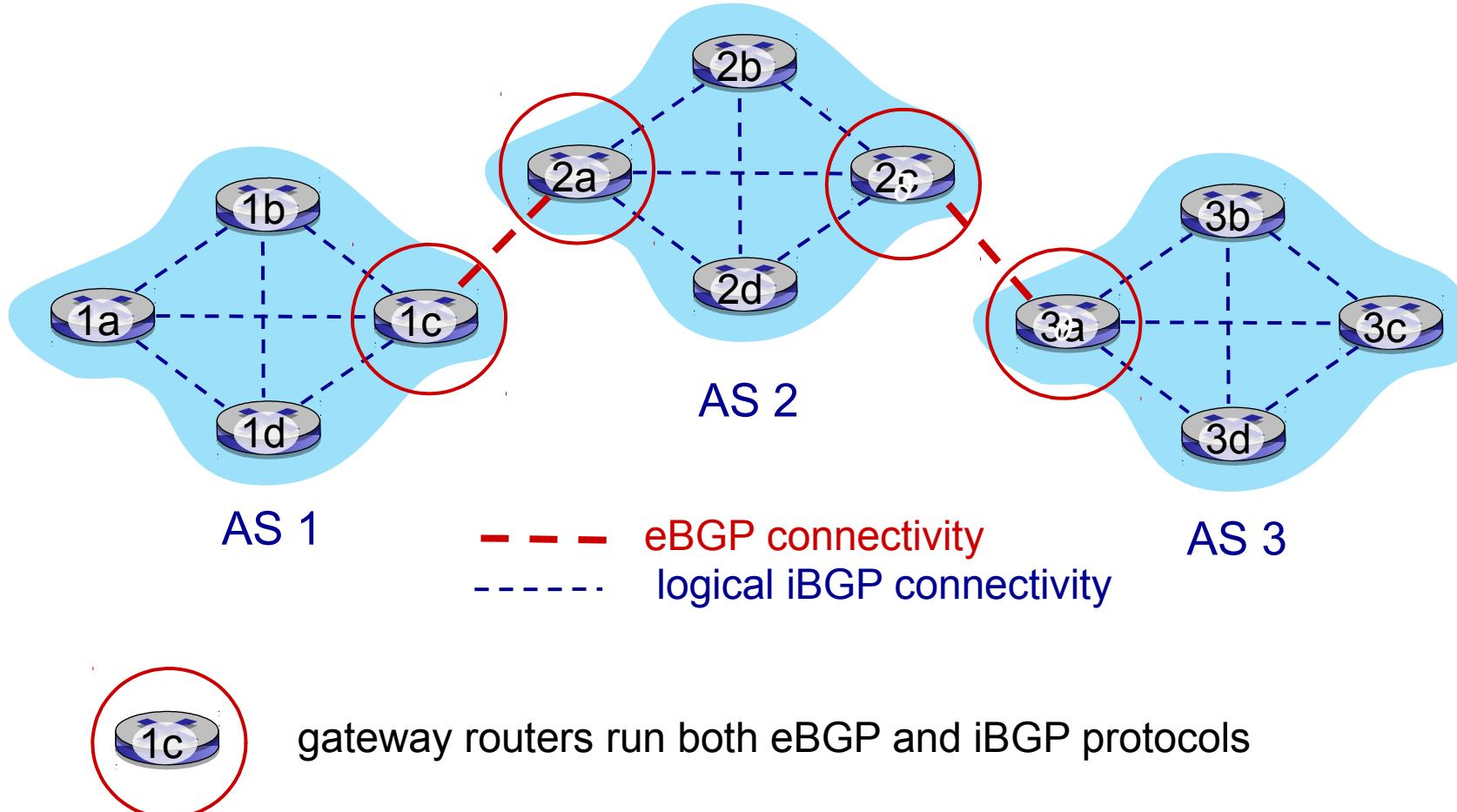


- network management, configuration
  - SNMP
  - NETCONF/YANG

# Internet inter-AS routing: BGP

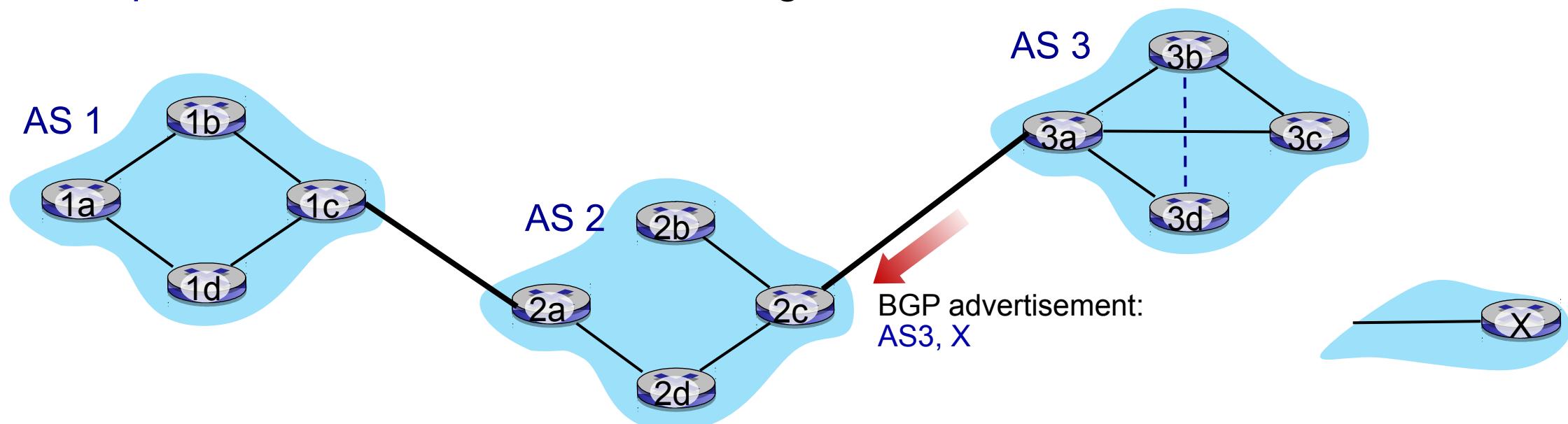
- BGP (Border Gateway Protocol): *the de facto inter-domain routing protocol*
  - “glue that holds the Internet together”
- allows subnet to advertise its existence, and the destinations it can reach, to rest of Internet: *“I am here, here is who I can reach, and how”*
- BGP provides each AS a means to:
  - eBGP: obtain subnet reachability information from neighboring ASes
  - iBGP: propagate reachability information to all AS-internal routers.
  - determine “good” routes to other networks based on reachability information and *policy*

# eBGP, iBGP connections



# BGP basics

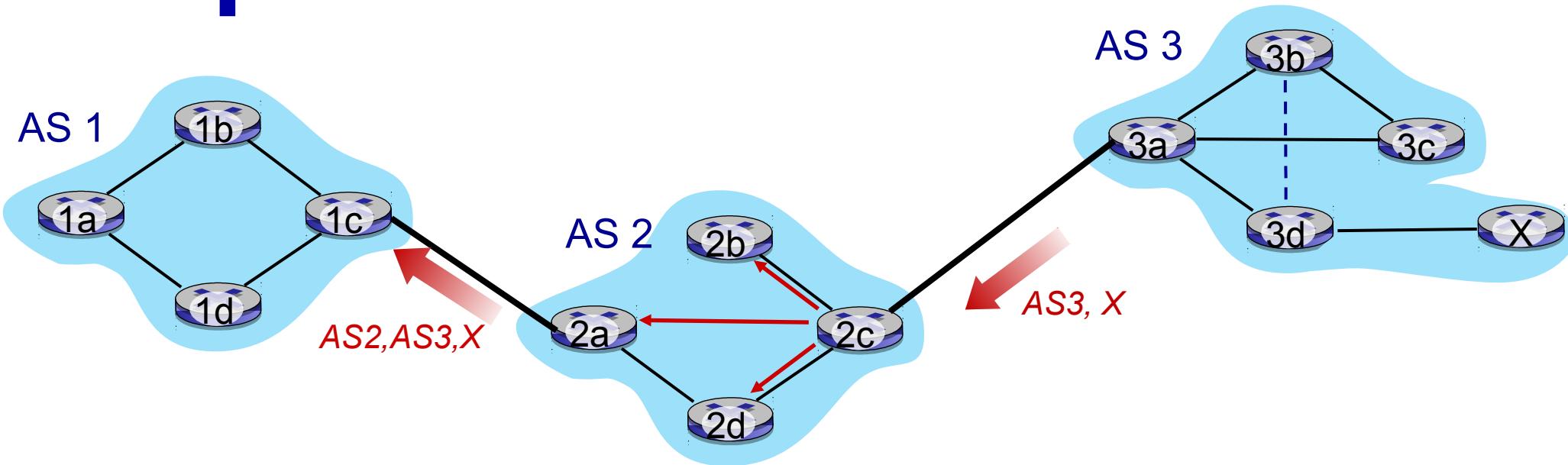
- **BGP session:** two BGP routers (“peers”) exchange BGP messages over semi-permanent TCP connection:
  - advertising *paths* to different destination network prefixes (BGP is a “path vector” protocol)
- when AS3 gateway 3a advertises **path AS3,X** to AS2 gateway 2c:
  - AS3 *promises* to AS2 it will forward datagrams towards X



# Path attributes and BGP routes

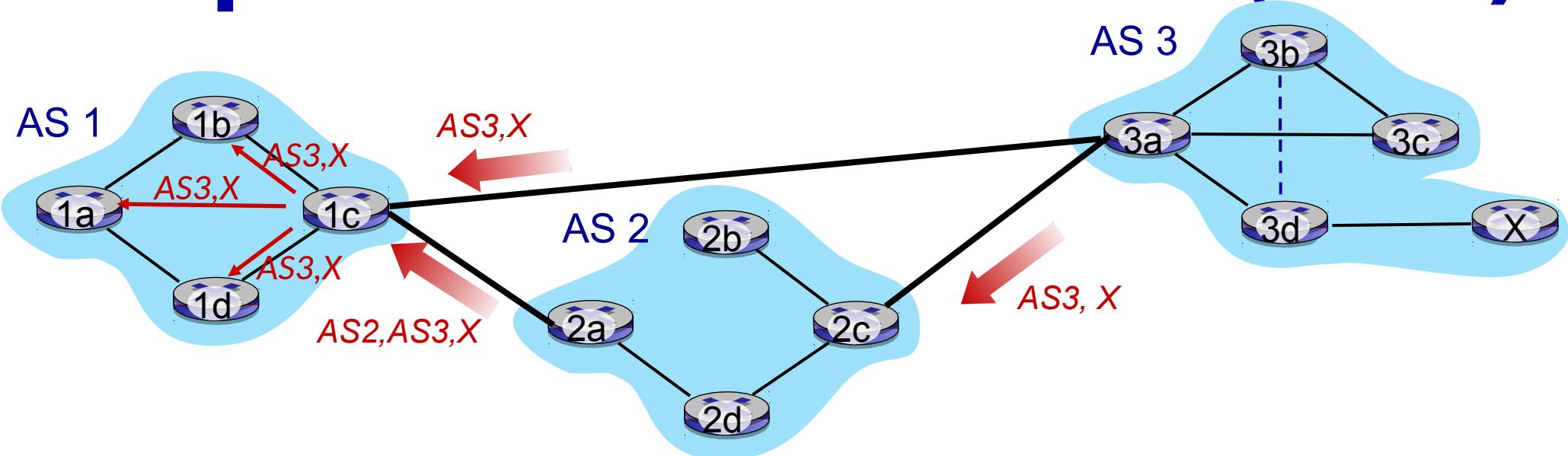
- BGP advertised route: prefix + attributes
  - prefix: destination being advertised
  - two important attributes:
    - AS-PATH: list of ASes through which prefix advertisement has passed
    - NEXT-HOP: indicates specific internal-AS router to next-hop AS
- policy-based routing:
  - gateway receiving route advertisement uses *import policy* to accept/decline path (e.g., never route through AS Y).
  - AS policy also determines whether to *advertise* path to other other neighboring ASes

# BGP path advertisement



- AS2 router 2c receives path advertisement **AS3,X** (via eBGP) from AS3 router 3a
- based on AS2 policy, AS2 router 2c accepts path AS3,X, propagates (via iBGP) to all AS2 routers
- based on AS2 policy, AS2 router 2a advertises (via eBGP) path **AS2, AS3, X** to AS1 router 1c

# BGP path advertisement (more)



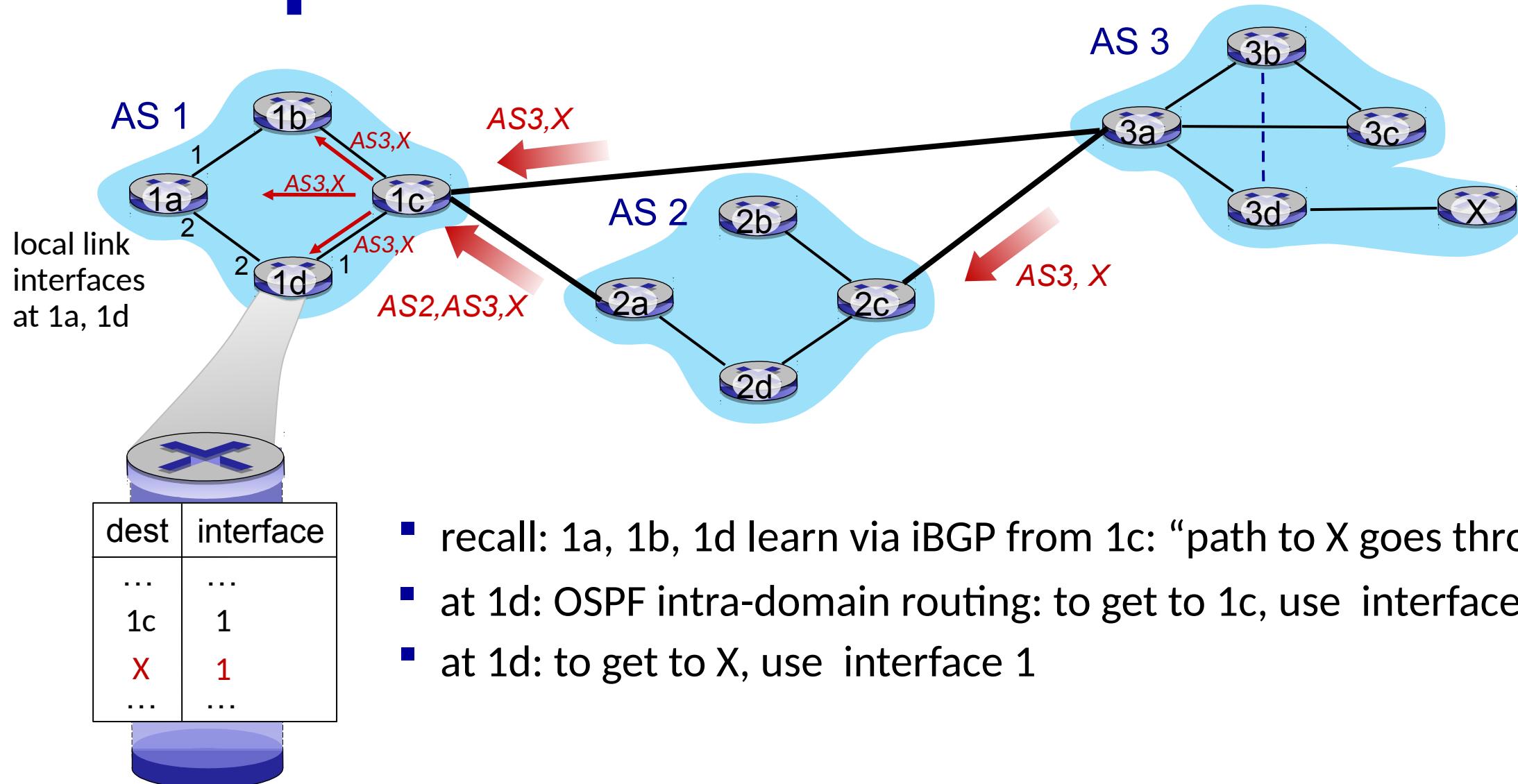
gateway router may learn about multiple paths to destination:

- AS1 gateway router 1c learns path **AS2,AS3,X** from 2a
- AS1 gateway router 1c learns path **AS3,X** from 3a
- based on *policy*, AS1 gateway router 1c chooses path **AS3,X** and advertises path within AS1 via iBGP

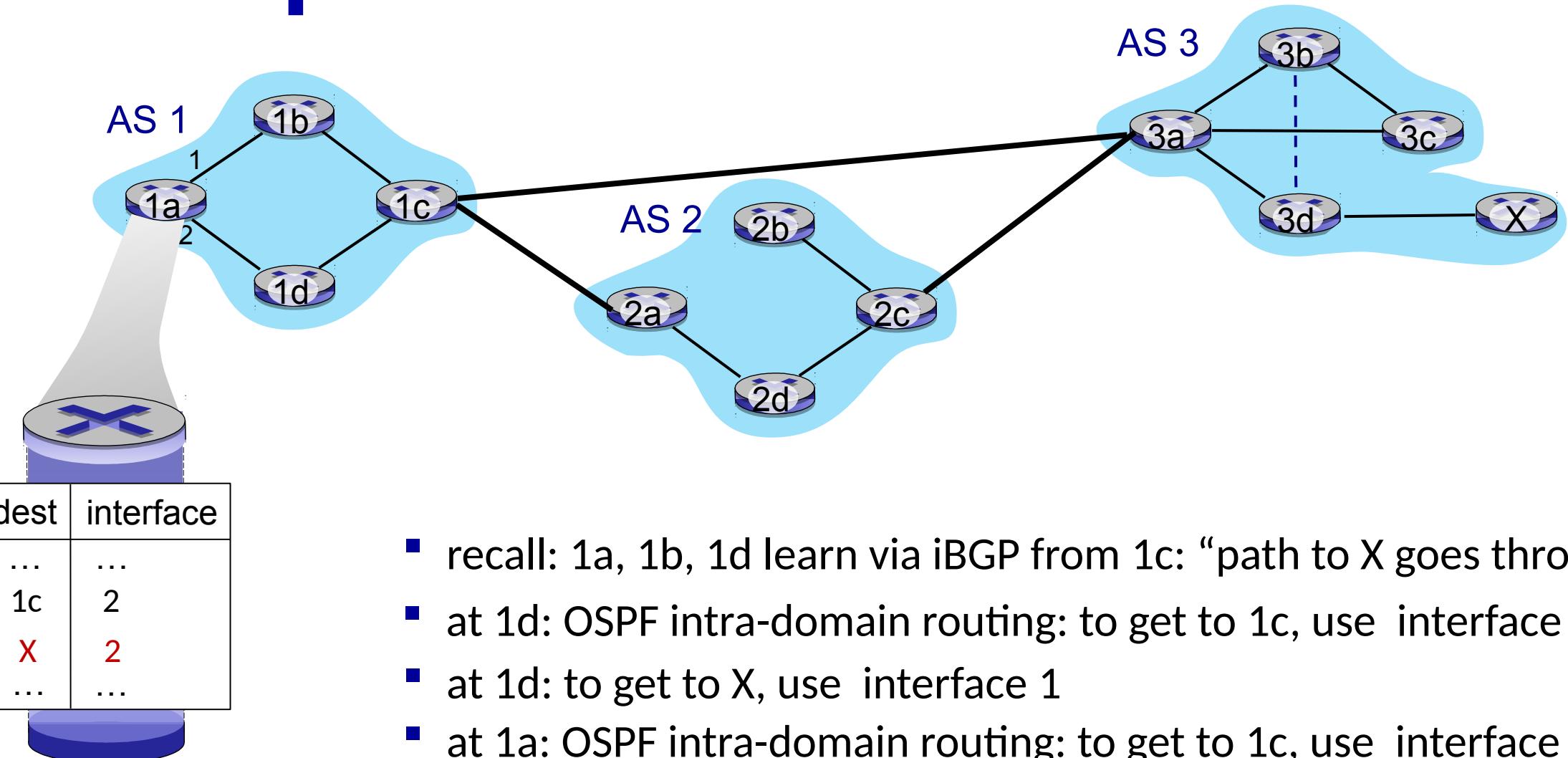
# BGP messages

- BGP messages exchanged between peers over TCP connection
- BGP messages:
  - **OPEN**: opens TCP connection to remote BGP peer and authenticates sending BGP peer
  - **UPDATE**: advertises new path (or withdraws old)
  - **KEEPALIVE**: keeps connection alive in absence of UPDATES; also ACKs OPEN request
  - **NOTIFICATION**: reports errors in previous msg; also used to close connection

# BGP path advertisement



# BGP path advertisement



- recall: 1a, 1b, 1d learn via iBGP from 1c: “path to X goes through 1c”
- at 1d: OSPF intra-domain routing: to get to 1c, use interface 1
- at 1d: to get to X, use interface 1
- at 1a: OSPF intra-domain routing: to get to 1c, use interface 2
- at 1a: to get to X, use interface 2

# Why different Intra-, Inter-AS routing ?

policy:

- inter-AS: admin wants control over how its traffic routed, who routes through its network
- intra-AS: single admin, so policy less of an issue

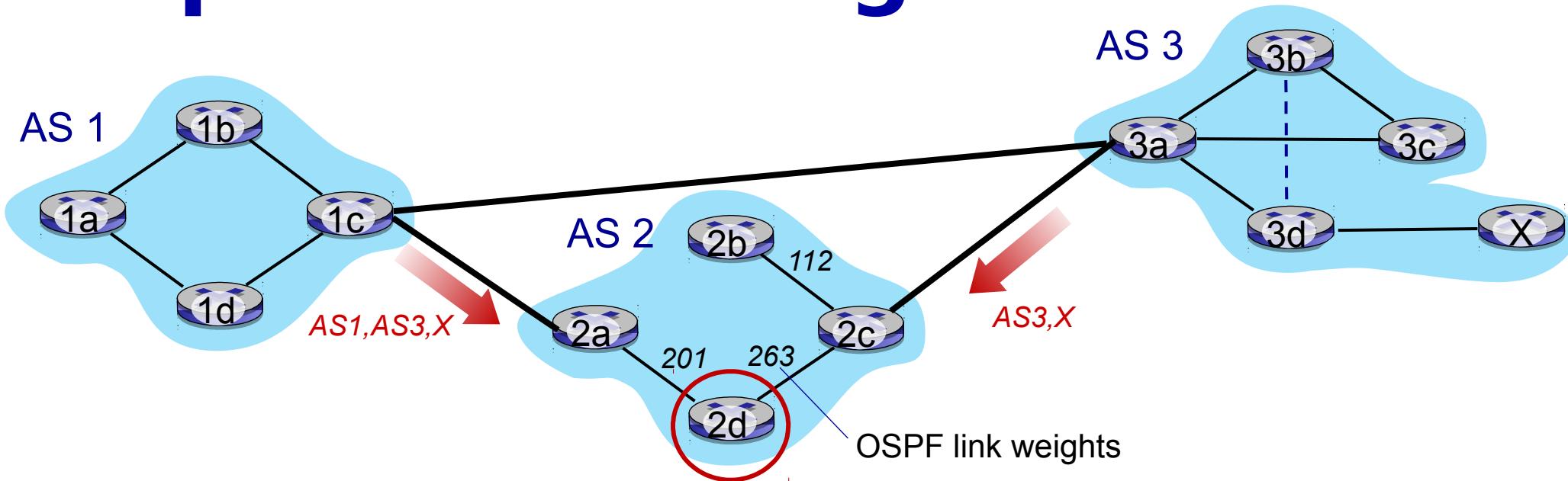
scale:

- hierarchical routing saves table size, reduced update traffic

performance:

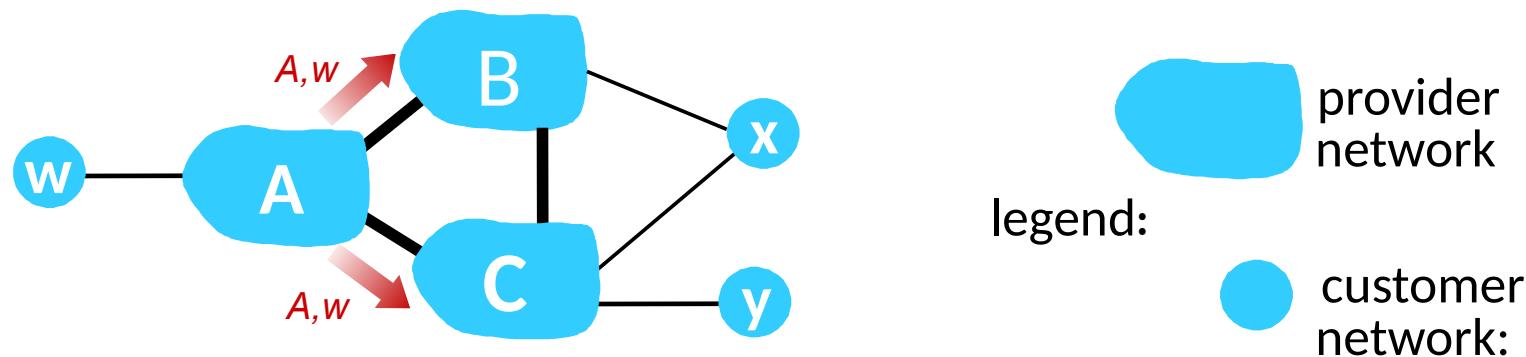
- intra-AS: can focus on performance
- inter-AS: policy dominates over performance

# Hot potato routing



- 2d learns (via iBGP) it can route to X via 2a or 2c
- **hot potato routing:** choose local gateway that has least *intra-domain* cost (e.g., 2d chooses 2a, even though more AS hops to X): don't worry about inter-domain cost!

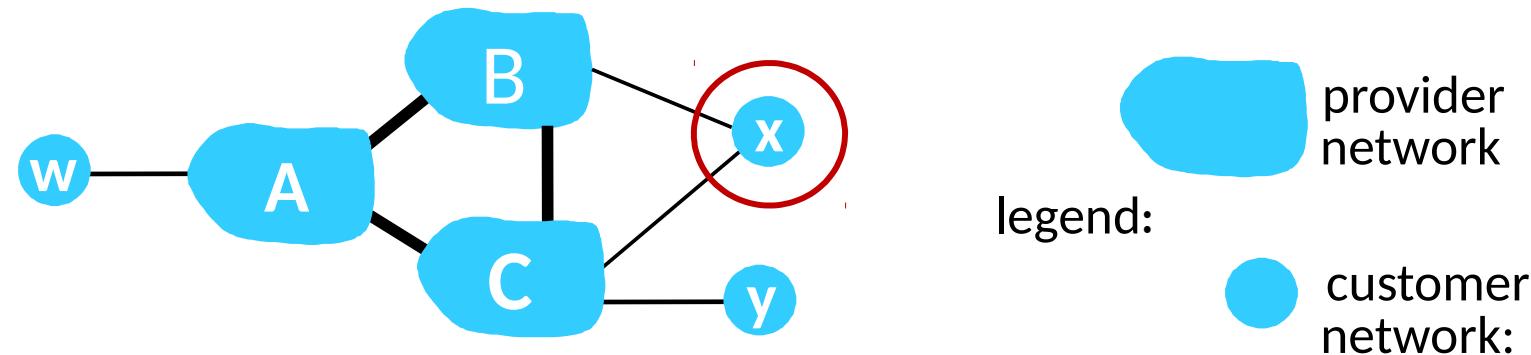
# BGP: achieving policy via advertisements



ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs – a typical “real world” policy)

- A advertises path Aw to B and to C
- B *chooses not to advertise BAw to C!*
  - B gets no “revenue” for routing CBAw, since none of C, A, w are B's customers
  - C does not learn about CBAw path
- C will route CAw (not using B) to get to w

# BGP: achieving policy via advertisements (more)



ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs – a typical “real world” policy)

- A,B,C are **provider networks**
- x,w,y are **customer** (of provider networks)
- x is **dual-homed**: attached to two networks
- **policy to enforce**: x does not want to route from B to C via x
  - .. so x will not advertise to B a route to C

# BGP route selection

- router may learn about more than one route to destination AS, selects route based on:
  1. local preference value attribute: policy decision
  2. shortest AS-PATH
  3. closest NEXT-HOP router: hot potato routing
  4. additional criteria

# Network layer: “control plane” roadmap

- introduction
- routing protocols
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- **SDN control plane**
- Internet Control Message Protocol



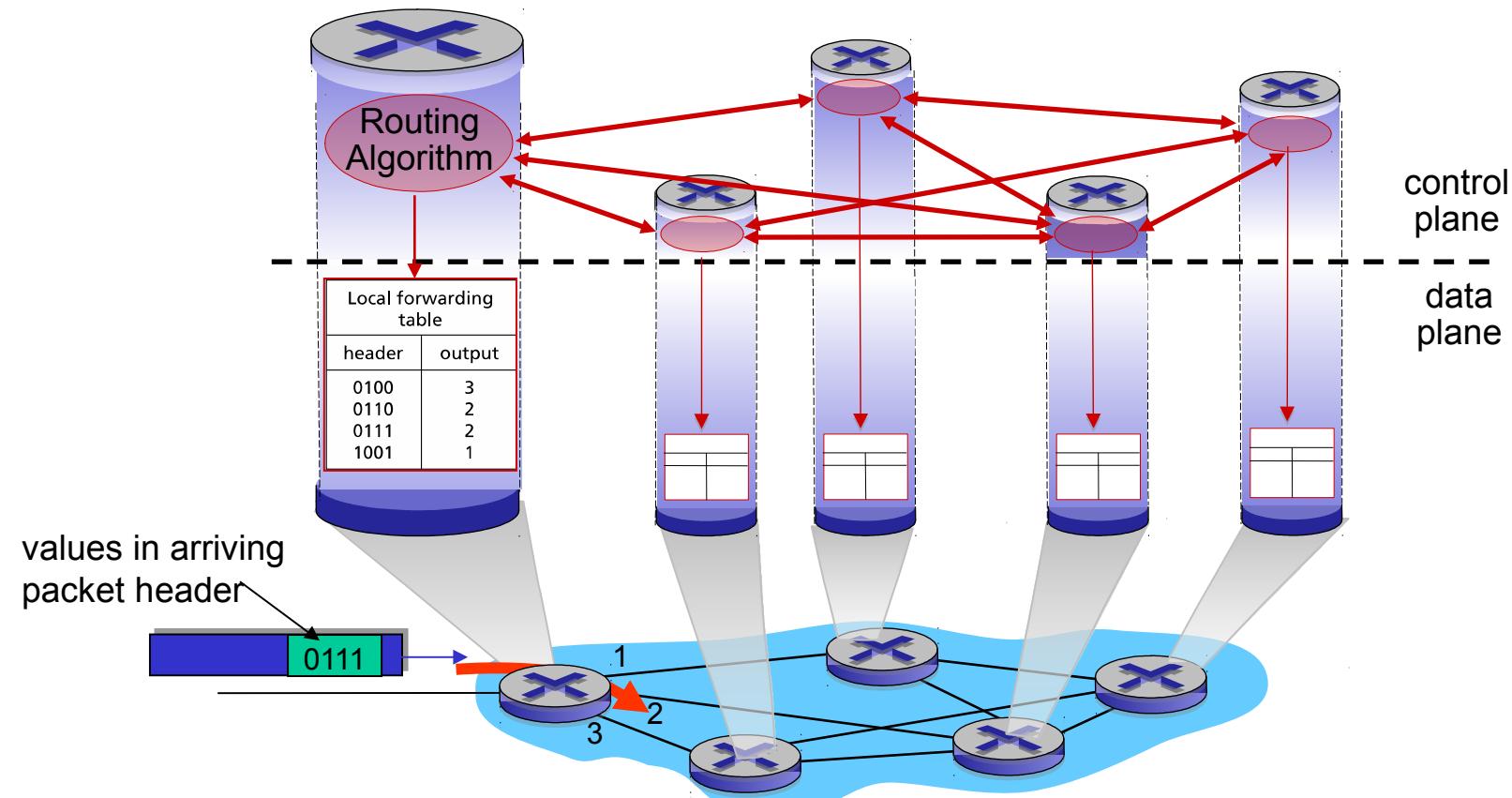
- network management, configuration
  - SNMP
  - NETCONF/YANG

# Software defined networking (SDN)

- Internet network layer: historically implemented via distributed, per-router control approach:
  - *monolithic* router contains switching hardware, runs proprietary implementation of Internet standard protocols (IP, RIP, IS-IS, OSPF, BGP) in proprietary router OS (e.g., Cisco IOS)
  - different “middleboxes” for different network layer functions: firewalls, load balancers, NAT boxes, ..
- ~2005: renewed interest in rethinking network control plane

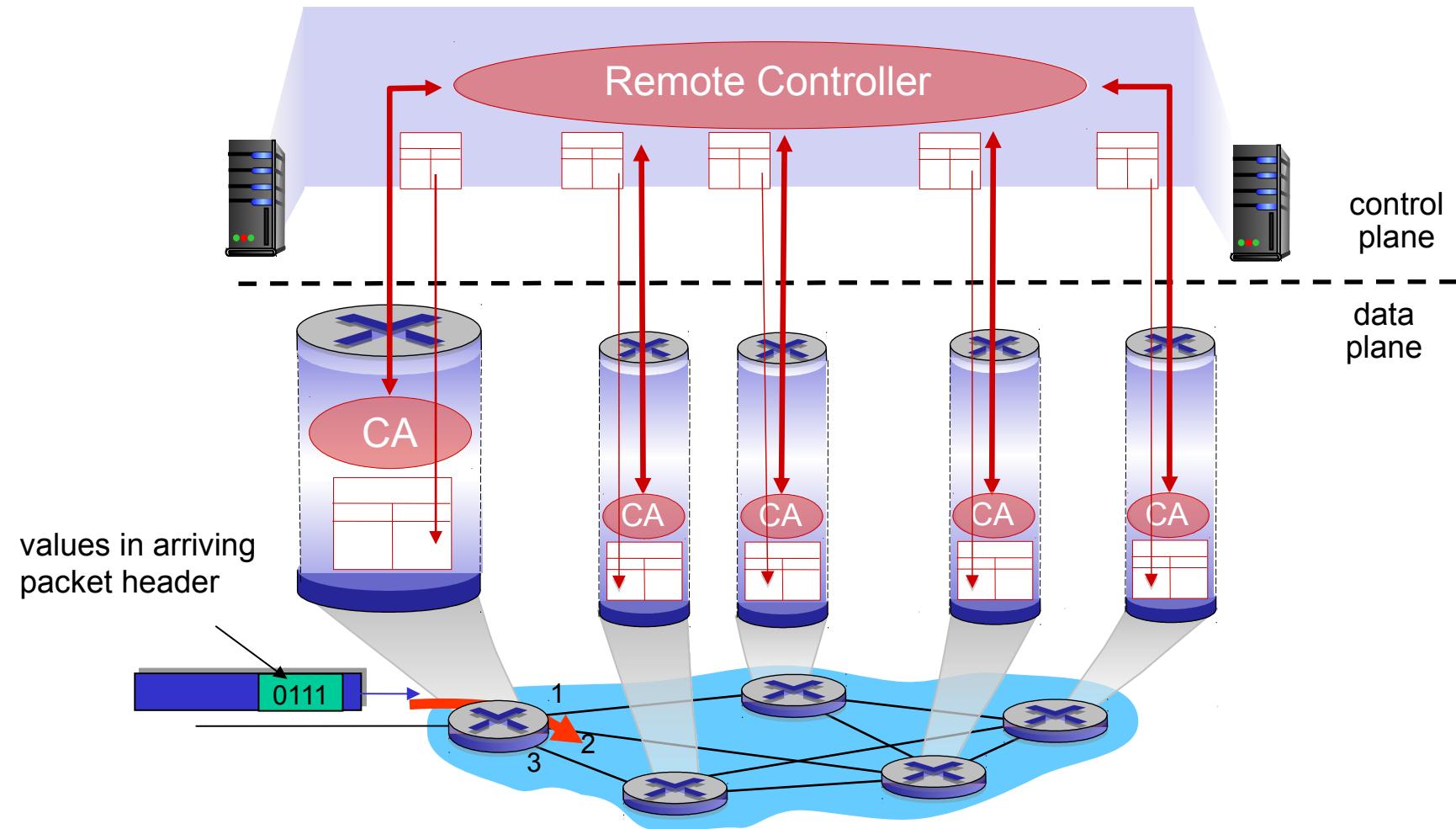
# Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane to compute forwarding tables



# Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers



# Software defined networking (SDN)

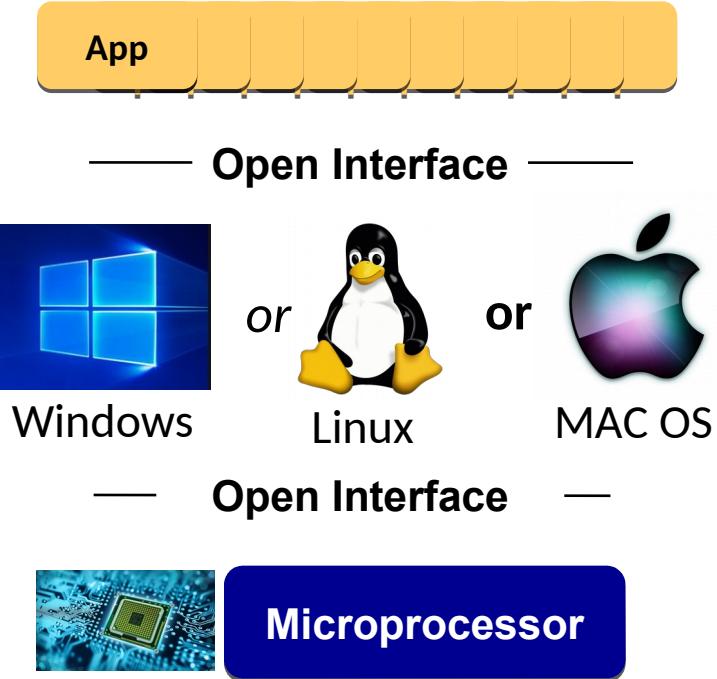
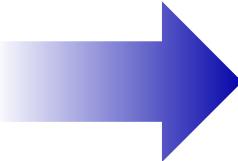
*Why a logically centralized control plane?*

- easier network management: avoid router misconfigurations, greater flexibility of traffic flows
- table-based forwarding (recall OpenFlow API) allows “programming” routers
  - centralized “programming” easier: compute tables centrally and distribute
  - distributed “programming” more difficult: compute tables as result of distributed algorithm (protocol) implemented in each-and-every router
- open (non-proprietary) implementation of control plane
  - foster innovation: let 1000 flowers bloom

# SDN analogy: mainframe to PC revolution

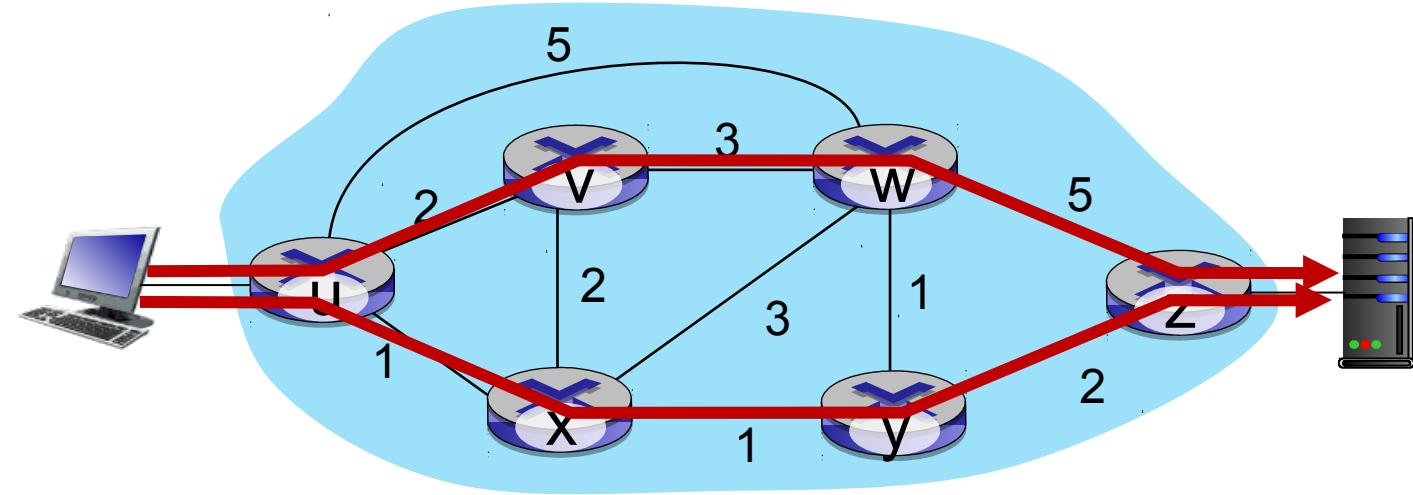


Vertically integrated  
Closed, proprietary  
Slow innovation  
Small industry



Horizontal  
Open interfaces  
Rapid innovation  
Huge industry

# Traffic engineering: difficult with traditional routing

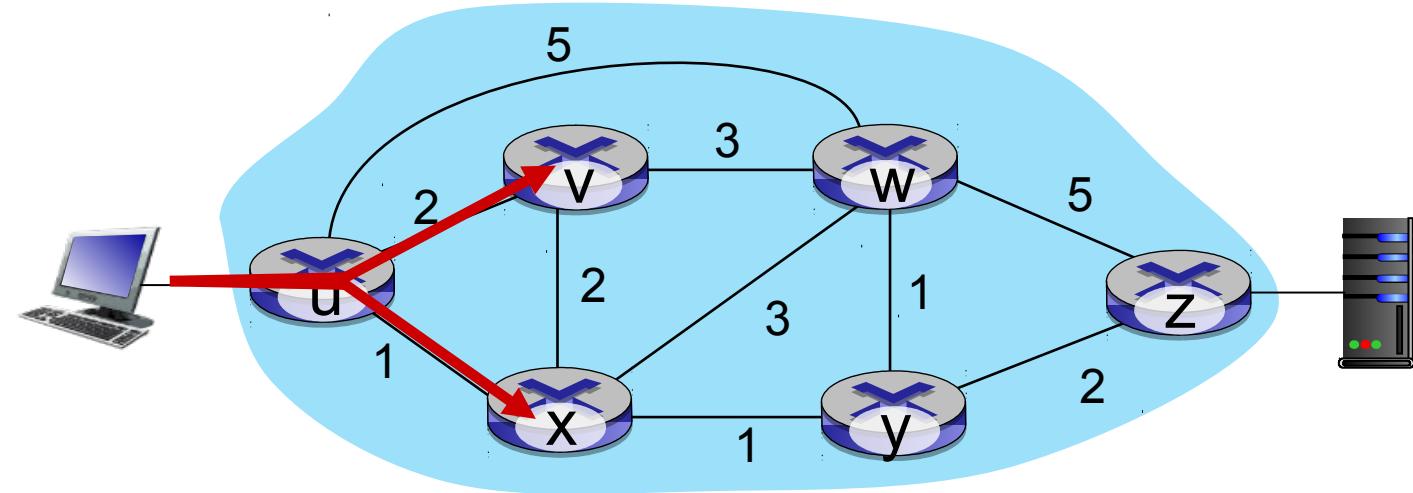


Q: what if network operator wants u-to-z traffic to flow along uvwz, rather than uxyz?

A: need to re-define link weights so traffic routing algorithm computes routes accordingly (or need a new routing algorithm)!

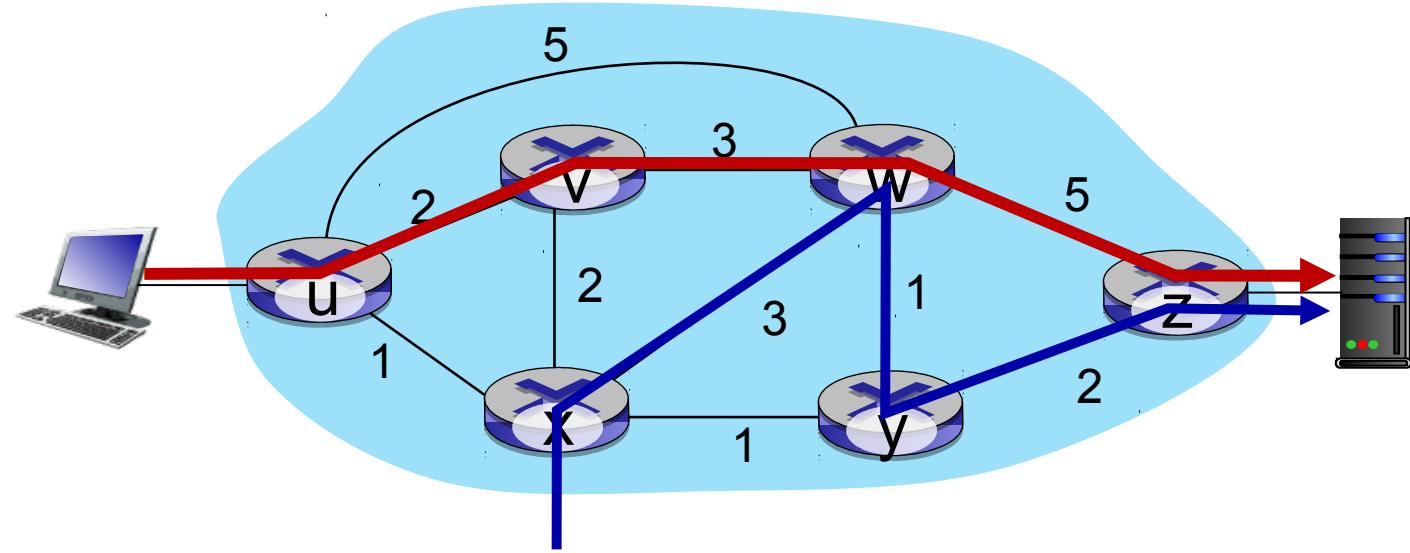
*link weights are only control “knobs”: not much control!*

# Traffic engineering: difficult with traditional routing



Q: what if network operator wants to split u-to-z traffic along uvwz *and* uxyz (load balancing)?  
A: can't do it (or need a new routing algorithm)

# Traffic engineering: difficult with traditional routing



Q: what if w wants to route blue and red traffic differently from w to z?

A: can't do it (with destination-based forwarding, and LS, DV routing)

We learned in Chapter 4 that generalized forwarding and SDN can be used to achieve *any* routing desired

# Software defined networking (SDN)

4. programmable control applications

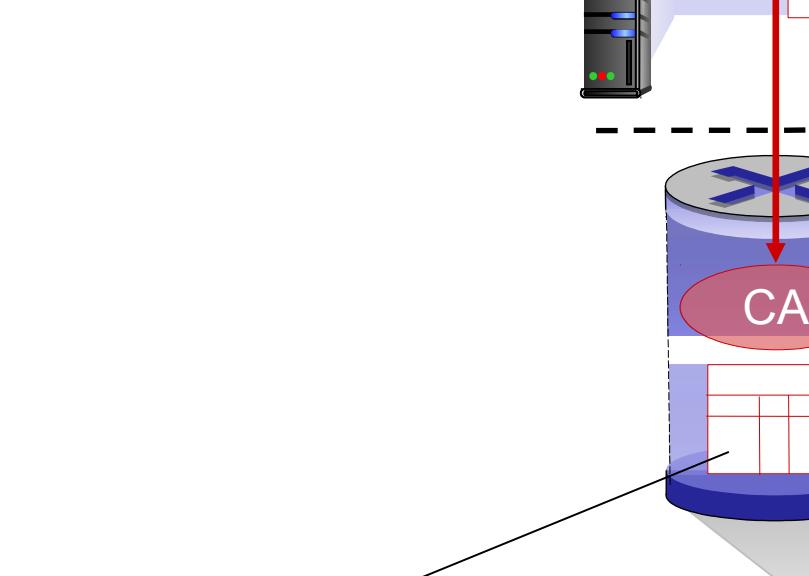
routing

access control

...

load balance

3. control plane functions external to data-plane switches



1: generalized “flow-based” forwarding (e.g., OpenFlow)

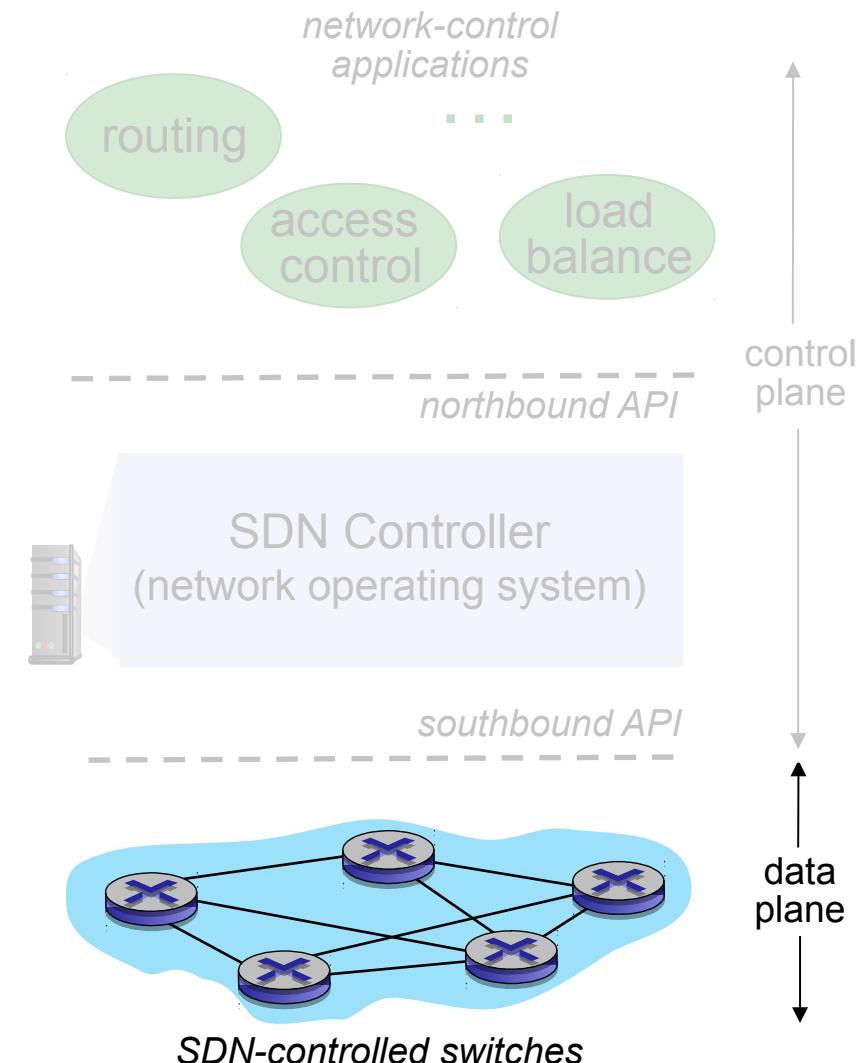
2. control, data plane separation

control plane  
data plane

# Software defined networking (SDN)

## Data-plane switches:

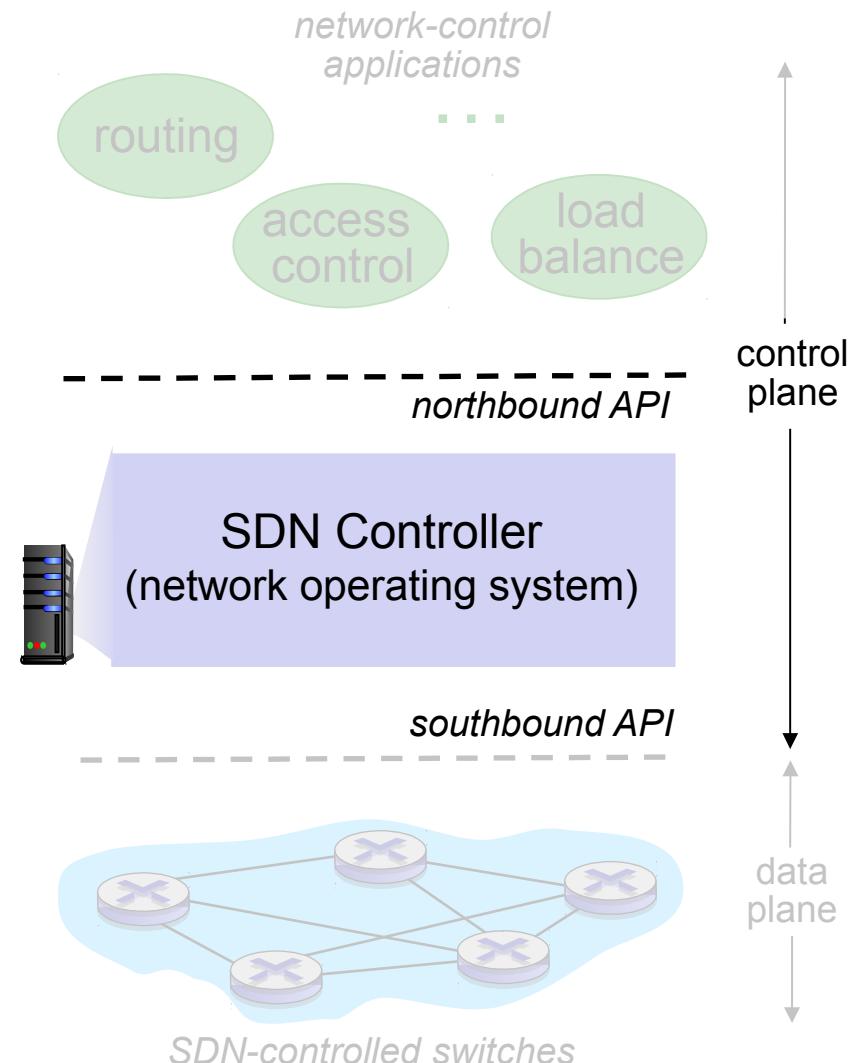
- fast, simple, commodity switches implementing generalized data-plane forwarding (Section 4.4) in hardware
- flow (forwarding) table computed, installed under controller supervision
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable, what is not
- protocol for communicating with controller (e.g., OpenFlow)



# Software defined networking (SDN)

## SDN controller (network OS):

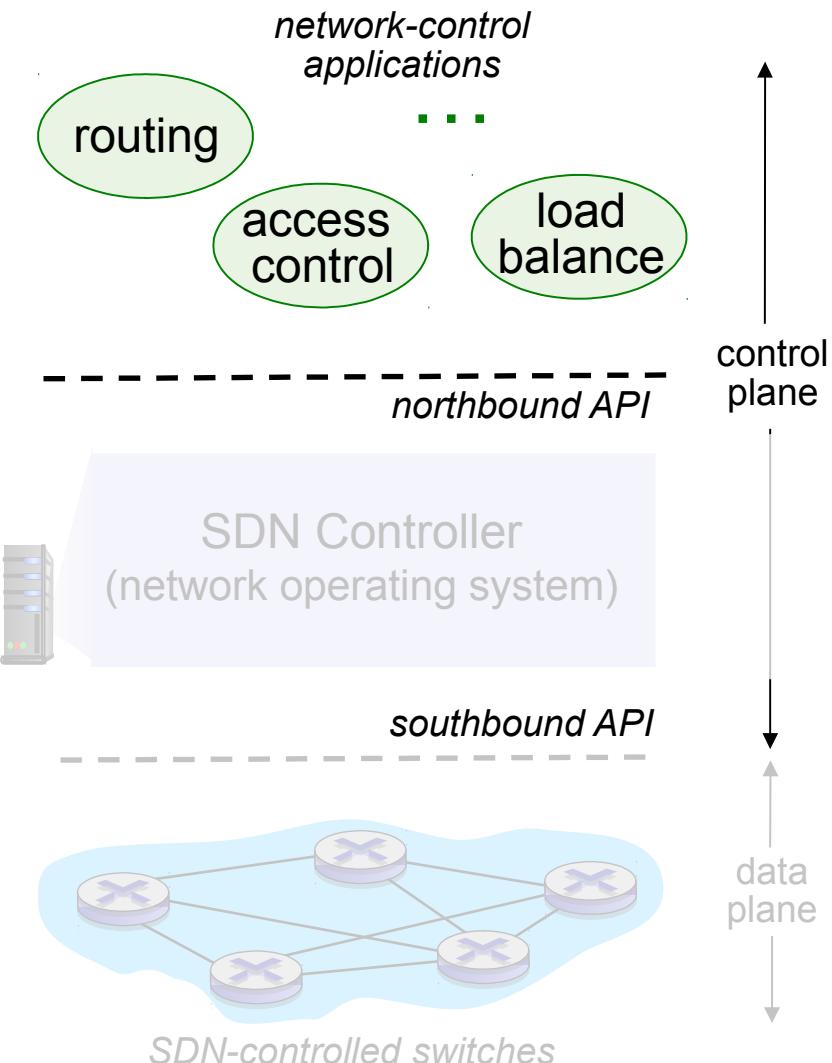
- maintain network state information
- interacts with network control applications “above” via northbound API
- interacts with network switches “below” via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



# Software defined networking (SDN)

## network-control apps:

- “brains” of control: implement control functions using lower-level services, API provided by SDN controller
- *unbundled*: can be provided by 3<sup>rd</sup> party: distinct from routing vendor, or SDN controller

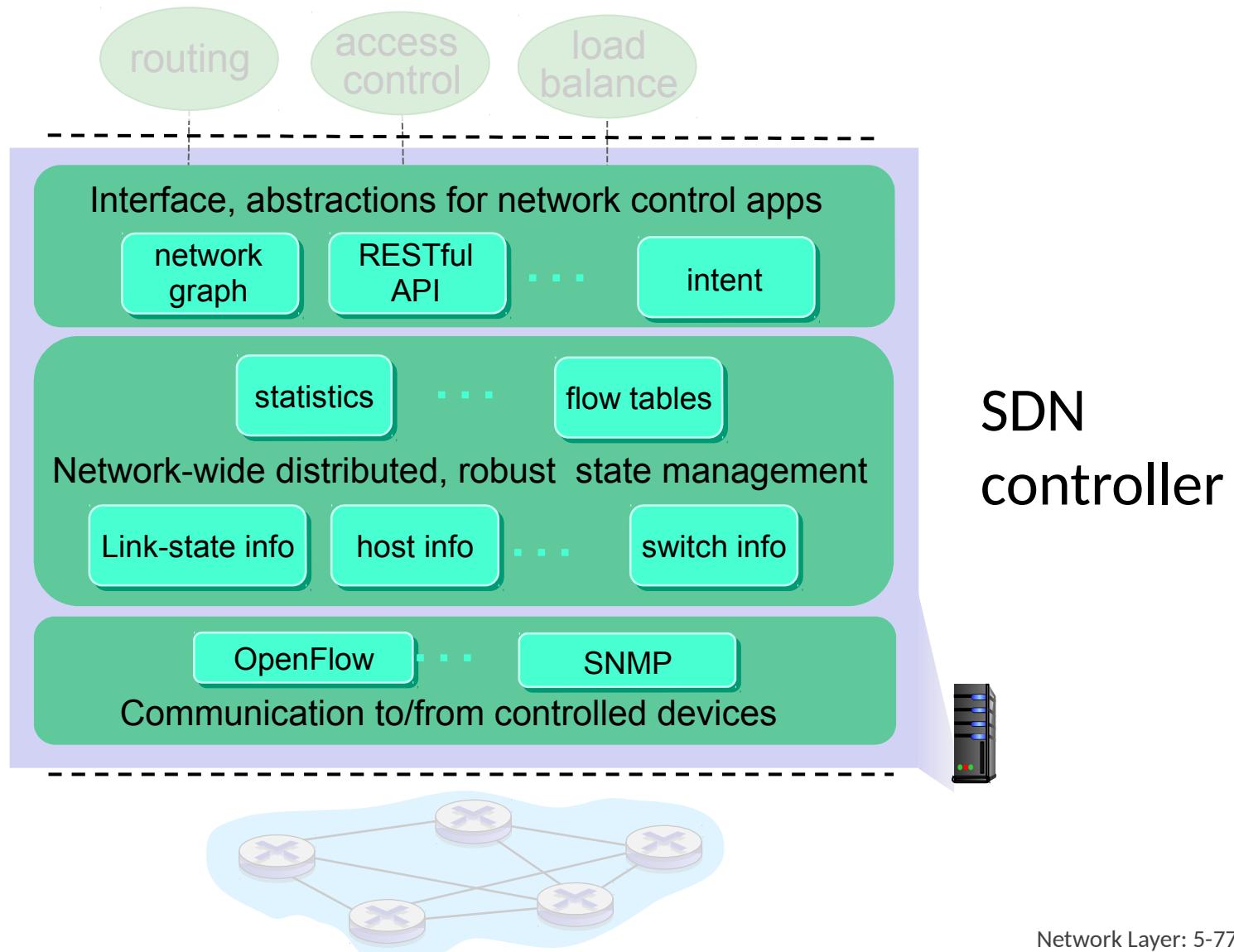


# Components of SDN controller

interface layer to network control apps: abstractions API

network-wide state management : state of networks links, switches, services: a *distributed database*

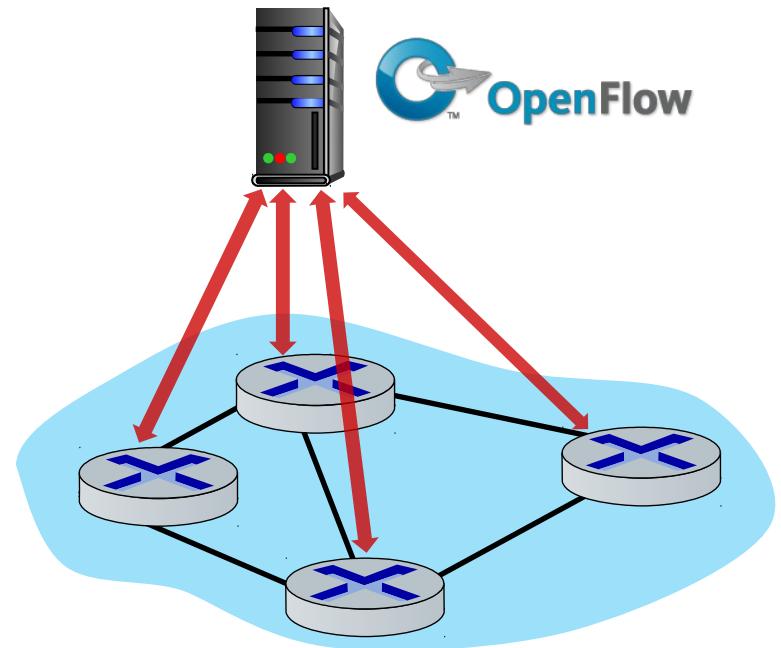
communication: communicate between SDN controller and controlled switches



# OpenFlow protocol

- operates between controller, switch
- TCP used to exchange messages
  - optional encryption
- three classes of OpenFlow messages:
  - controller-to-switch
  - asynchronous (switch to controller)
  - symmetric (misc.)
- distinct from OpenFlow API
  - API used to specify generalized forwarding actions

OpenFlow Controller

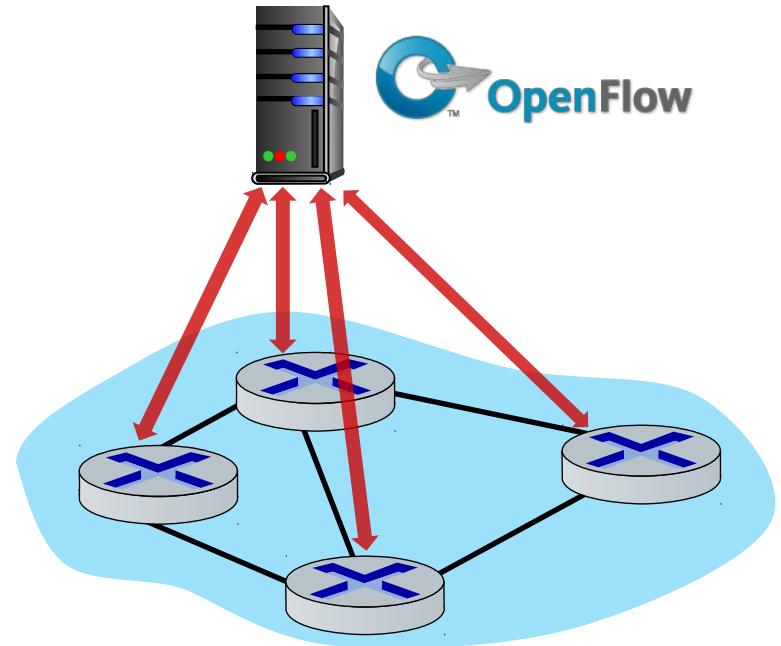


# OpenFlow: controller-to-switch messages

## Key controller-to-switch messages

- *features*: controller queries switch features, switch replies
- *configure*: controller queries/sets switch configuration parameters
- *modify-state*: add, delete, modify flow entries in the OpenFlow tables
- *packet-out*: controller can send this packet out of specific switch port

## OpenFlow Controller

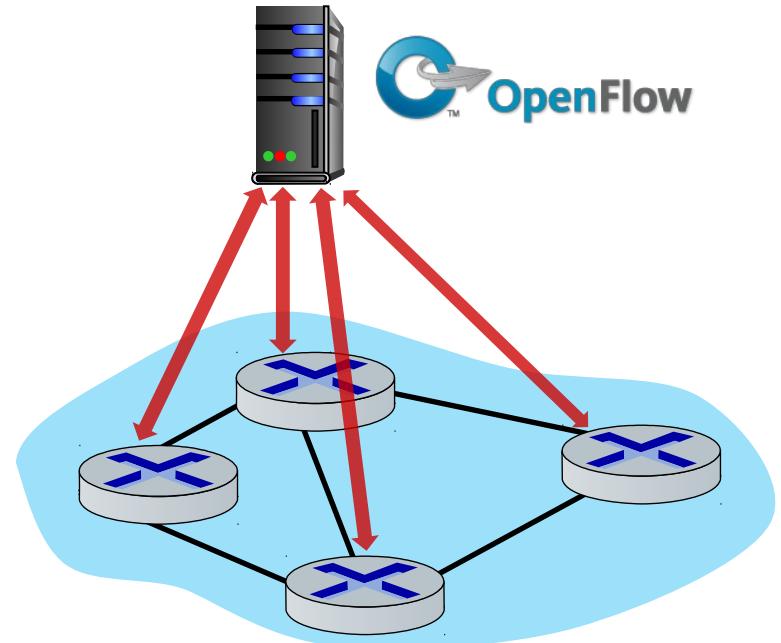


# OpenFlow: switch-to-controller messages

## Key switch-to-controller messages

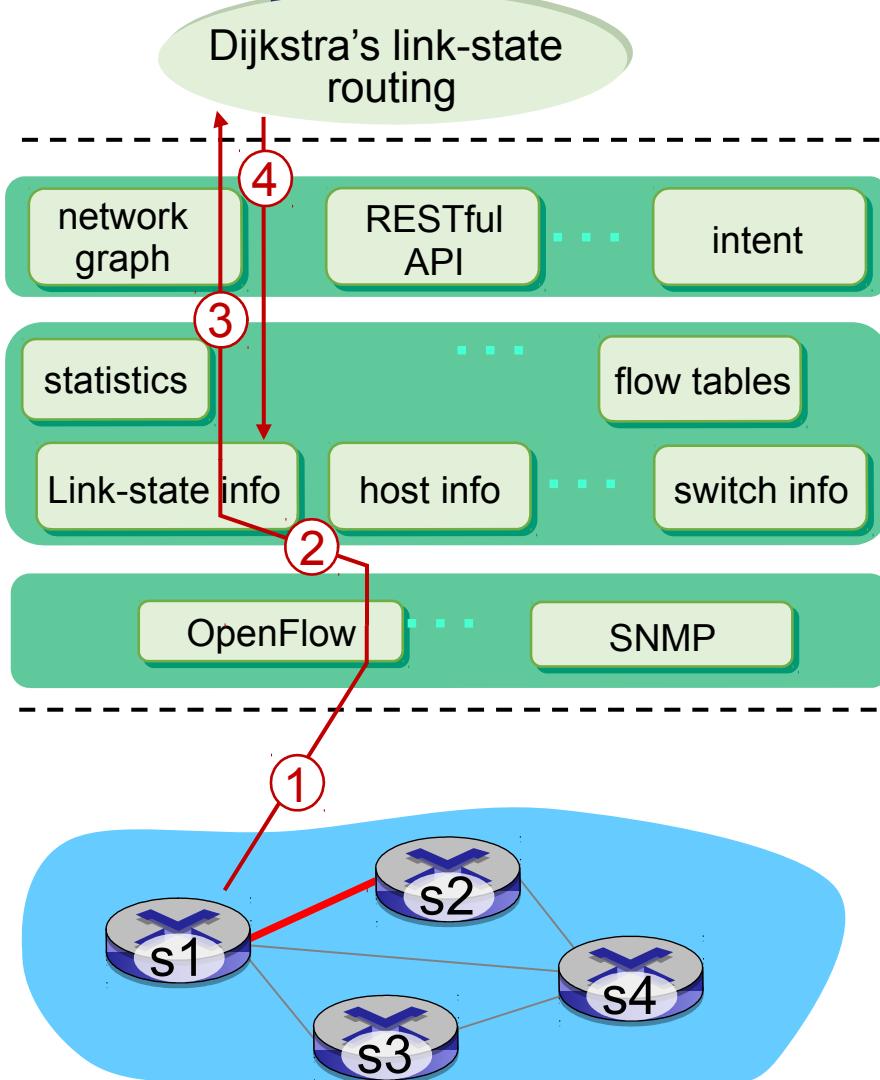
- ***packet-in***: transfer packet (and its control) to controller. See packet-out message from controller
- ***flow-removed***: flow table entry deleted at switch
- ***port status***: inform controller of a change on a port.

## OpenFlow Controller



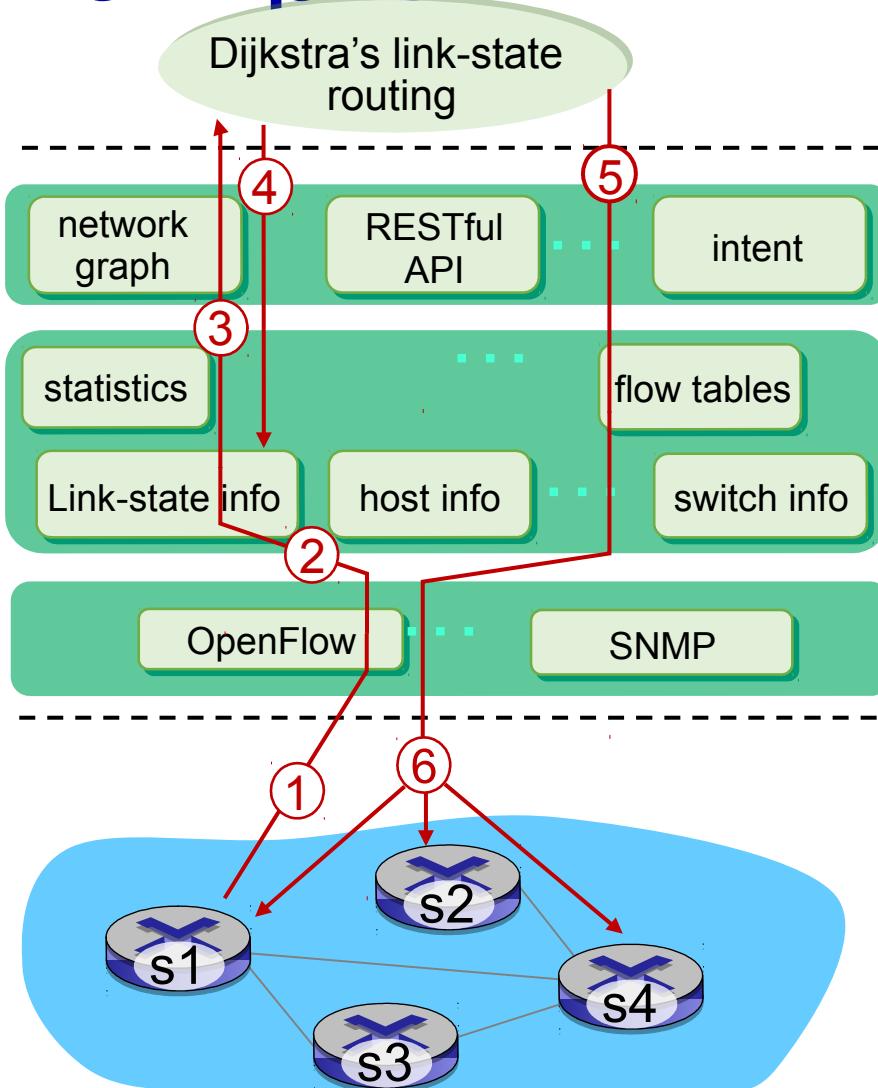
Fortunately, network operators don't "program" switches by creating/sending OpenFlow messages directly. Instead use higher-level abstraction at controller

# SDN: control/data plane interaction example



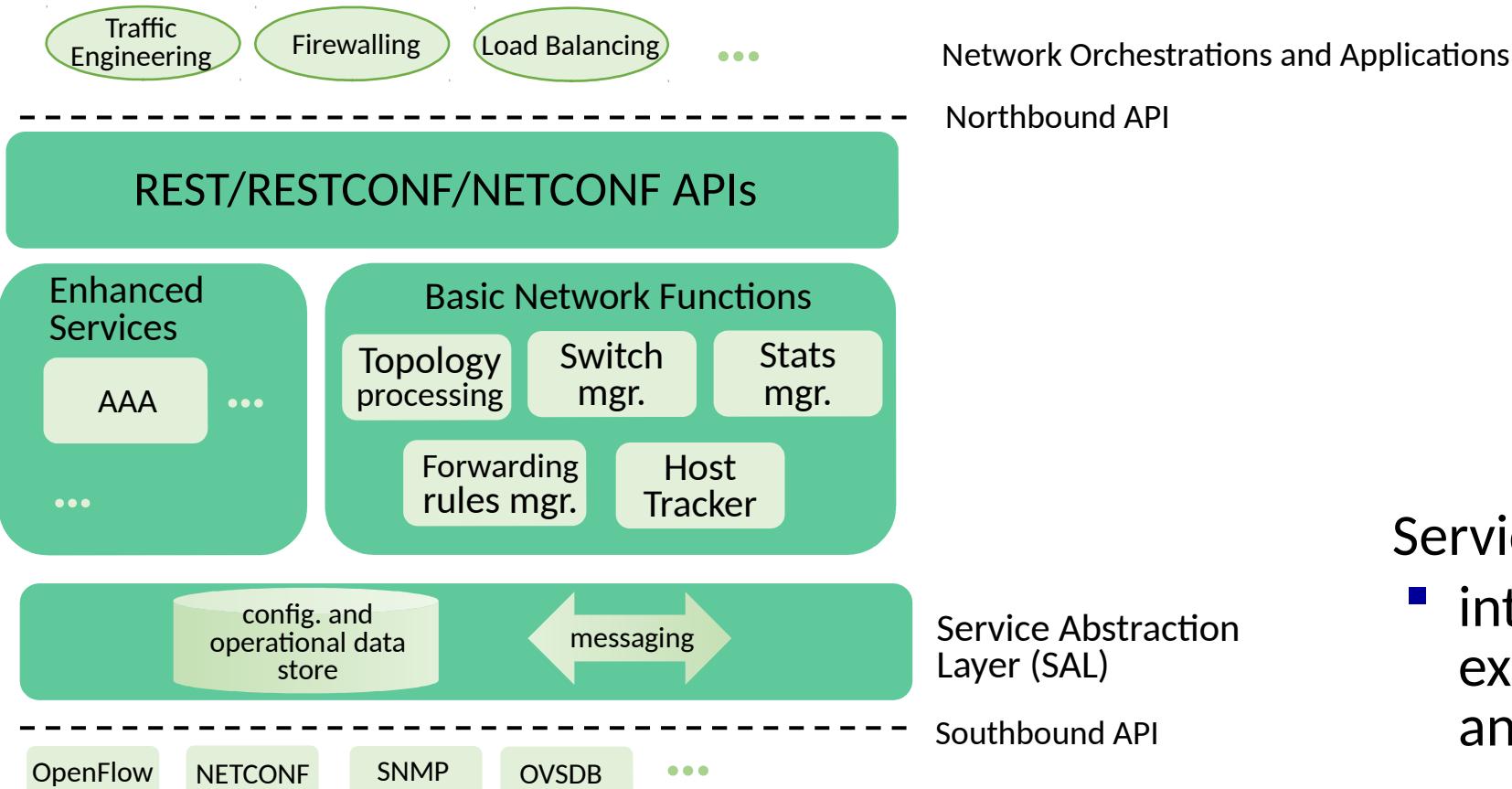
- ① S1, experiencing link failure uses OpenFlow port status message to notify controller
- ② SDN controller receives OpenFlow message, updates link status info
- ③ Dijkstra's routing algorithm application has previously registered to be called when ever link status changes. It is called.
- ④ Dijkstra's routing algorithm access network graph info, link state info in controller, computes new routes

# SDN: control/data plane interaction example



- ⑤ link state routing app interacts with flow-table-computation component in SDN controller, which computes new flow tables needed
- ⑥ controller uses OpenFlow to install new tables in switches that need updating

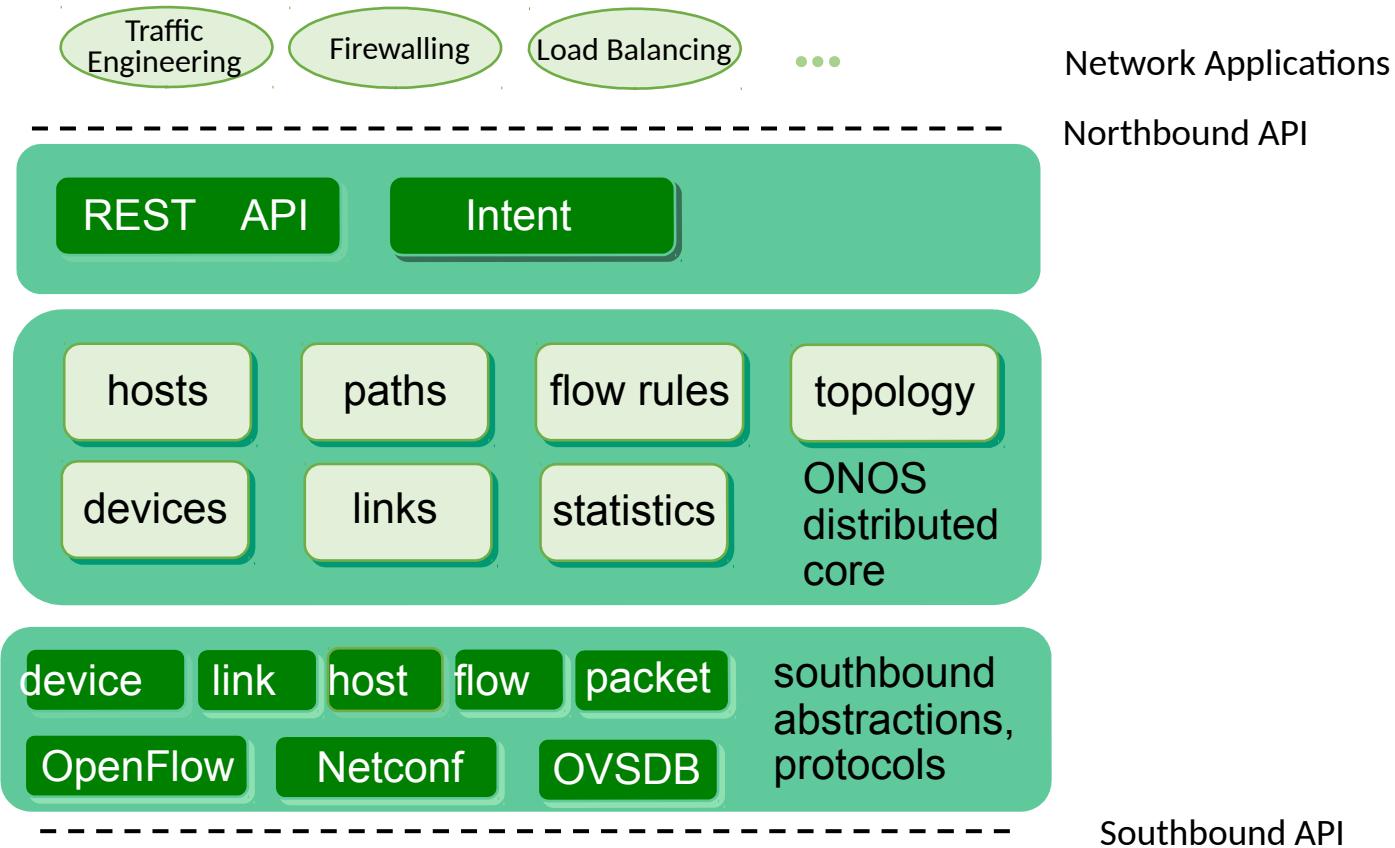
# OpenDaylight (ODL) controller



**Service Abstraction Layer:**

- interconnects internal, external applications and services

# ONOS controller



- control apps separate from controller
- intent framework: high-level specification of service: what rather than how
- considerable emphasis on distributed core: service reliability, replication performance scaling

# SDN: selected challenges

- hardening the control plane: dependable, reliable, performance-scalable, secure distributed system
  - robustness to failures: leverage strong theory of reliable distributed system for control plane
  - dependability, security: “baked in” from day one?
- networks, protocols meeting mission-specific requirements
  - e.g., real-time, ultra-reliable, ultra-secure
- Internet-scaling: beyond a single AS
- SDN critical in 5G cellular networks

# **SDN and the future of traditional network protocols**

- SDN-computed versus router-computer forwarding tables:
  - just one example of logically-centralized-computed versus protocol computed
- one could imagine SDN-computed congestion control:
  - controller sets sender rates based on router-reported (to controller) congestion levels



How will implementation of network functionality (SDN versus protocols) evolve?



# Network layer: “control plane” roadmap

- introduction
- routing protocols
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol



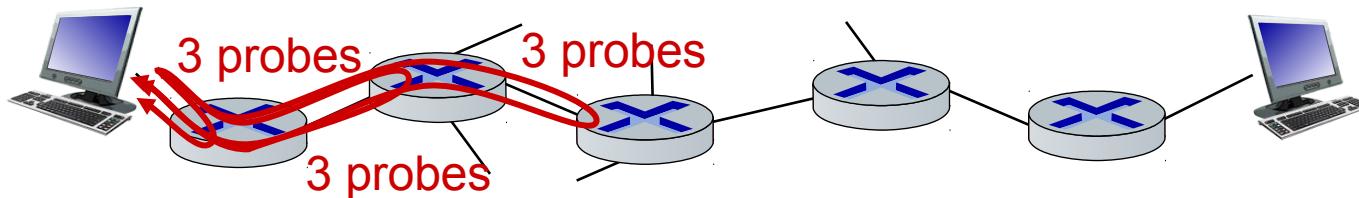
- network management, configuration
  - SNMP
  - NETCONF/YANG

# ICMP: internet control message protocol

- used by hosts and routers to communicate network-level information
  - error reporting: unreachable host, network, port, protocol
  - echo request/reply (used by ping)
- network-layer “above” IP:
  - ICMP messages carried in IP datagrams
- *ICMP message*: type, code plus first 8 bytes of IP datagram causing error

Type	Code	description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

# Traceroute and ICMP



- source sends sets of UDP segments to destination
  - 1<sup>st</sup> set has TTL =1, 2<sup>nd</sup> set has TTL=2, etc.
- datagram in  $n$ th set arrives to  $n$ th router:
  - router discards datagram and sends source ICMP message (type 11, code 0)
  - ICMP message possibly includes name of router & IP address
- when ICMP message arrives at source: record RTTs

## stopping criteria:

- UDP segment eventually arrives at destination host
- destination returns ICMP “port unreachable” message (type 3, code 3)
- source stops

# Network layer: “control plane” roadmap

- introduction
- routing protocols
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol



- network management, configuration
  - SNMP
  - NETCONF/YANG

# What is network management?

- autonomous systems (aka “network”): 1000s of interacting hardware/software components
- other complex systems requiring monitoring, configuration, control:
  - jet airplane, nuclear power plant, others?

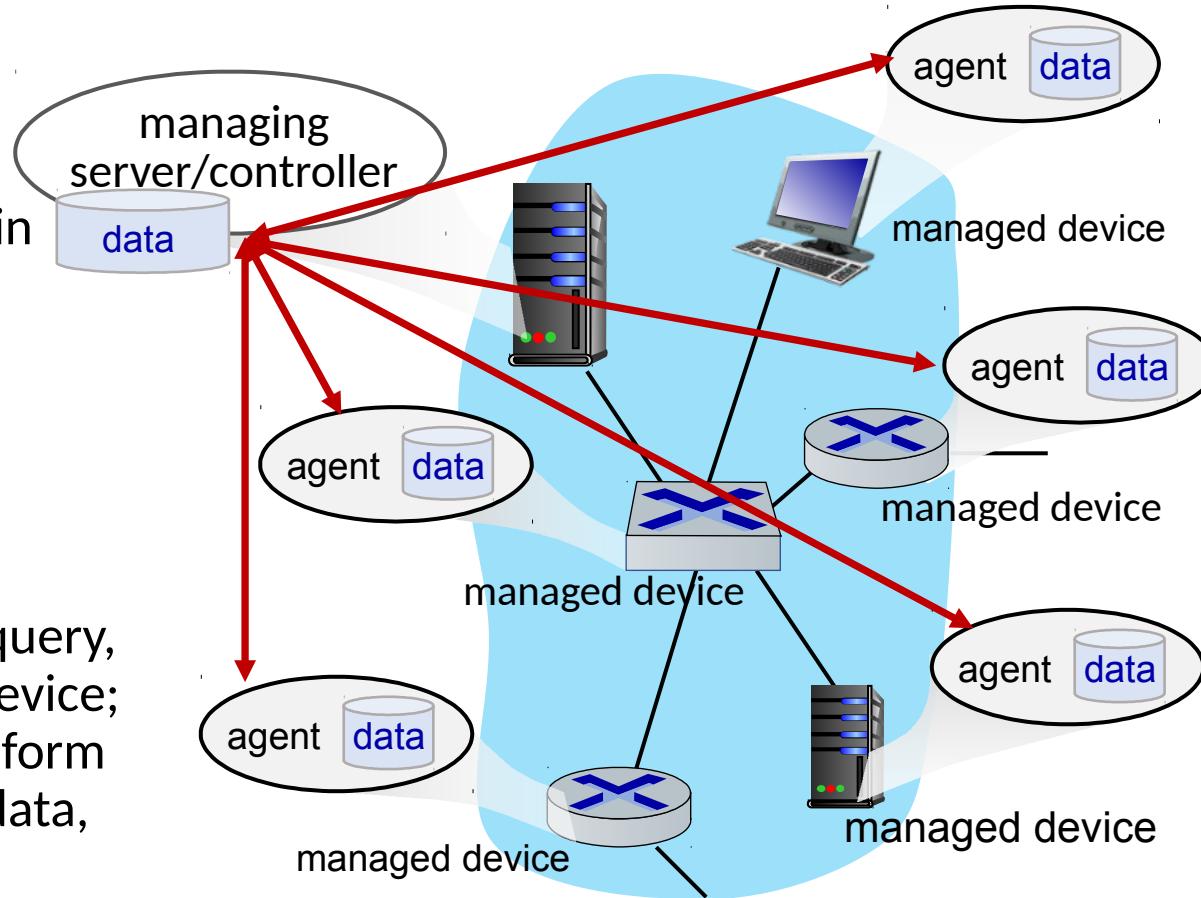


"Network management includes the deployment, integration and coordination of the hardware, software, and human elements to monitor, test, poll, configure, analyze, evaluate, and control the network and element resources to meet the real-time, operational performance, and Quality of Service requirements at a reasonable cost."

# Components of network management

**Managing server:**  
application, typically  
with network  
managers (humans) in  
the loop

**Network  
management  
protocol:** used by  
managing server to query,  
configure, manage device;  
used by devices to inform  
managing server of data,  
events.



**Managed device:**  
equipment with manageable,  
configurable hardware,  
software components

**Data:** device “state”  
configuration data,  
operational data,  
device statistics

# Network operator approaches to management

## CLI (Command Line Interface)

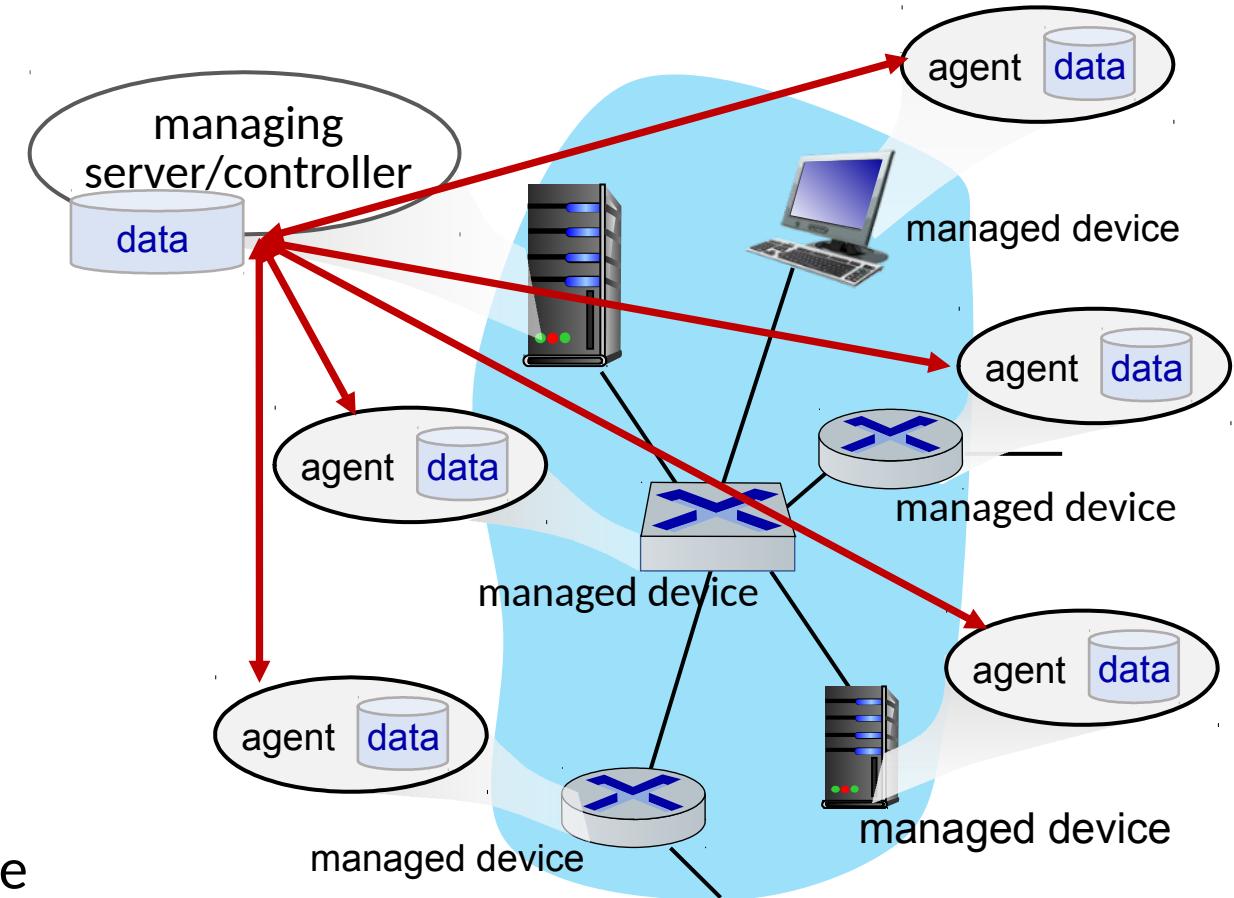
- operator issues (types, scripts) direct to individual devices (e.g., via ssh)

## SNMP/MIB

- operator queries/sets devices data (MIB) using Simple Network Management Protocol (SNMP)

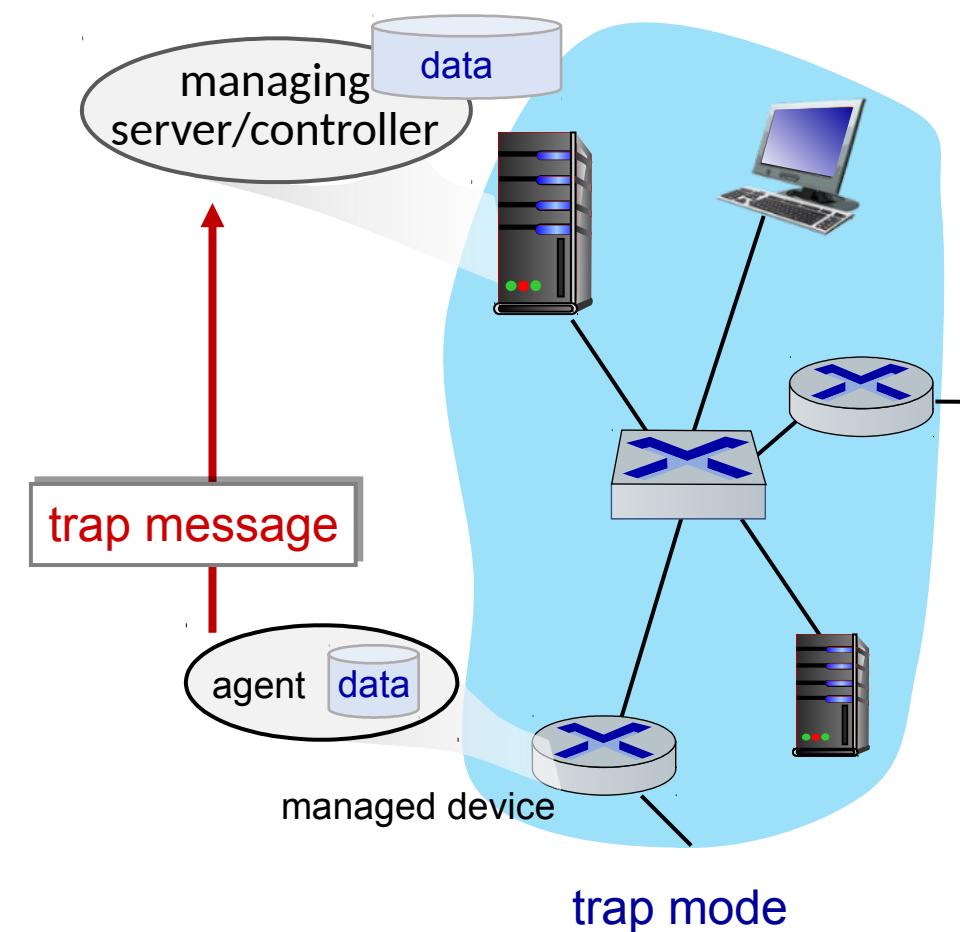
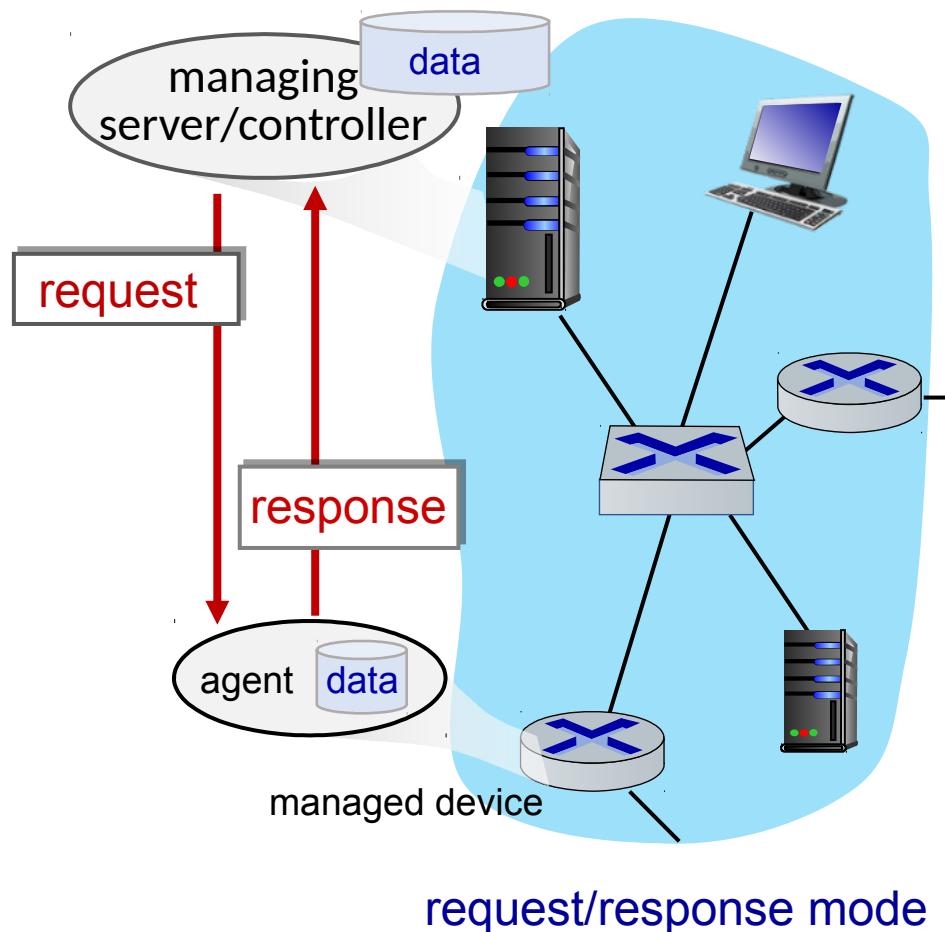
## NETCONF/YANG

- more abstract, network-wide, holistic
- emphasis on multi-device configuration management.
- YANG: data modeling language
- NETCONF: communicate YANG-compatible actions/data to/from/among remote devices



# SNMP protocol

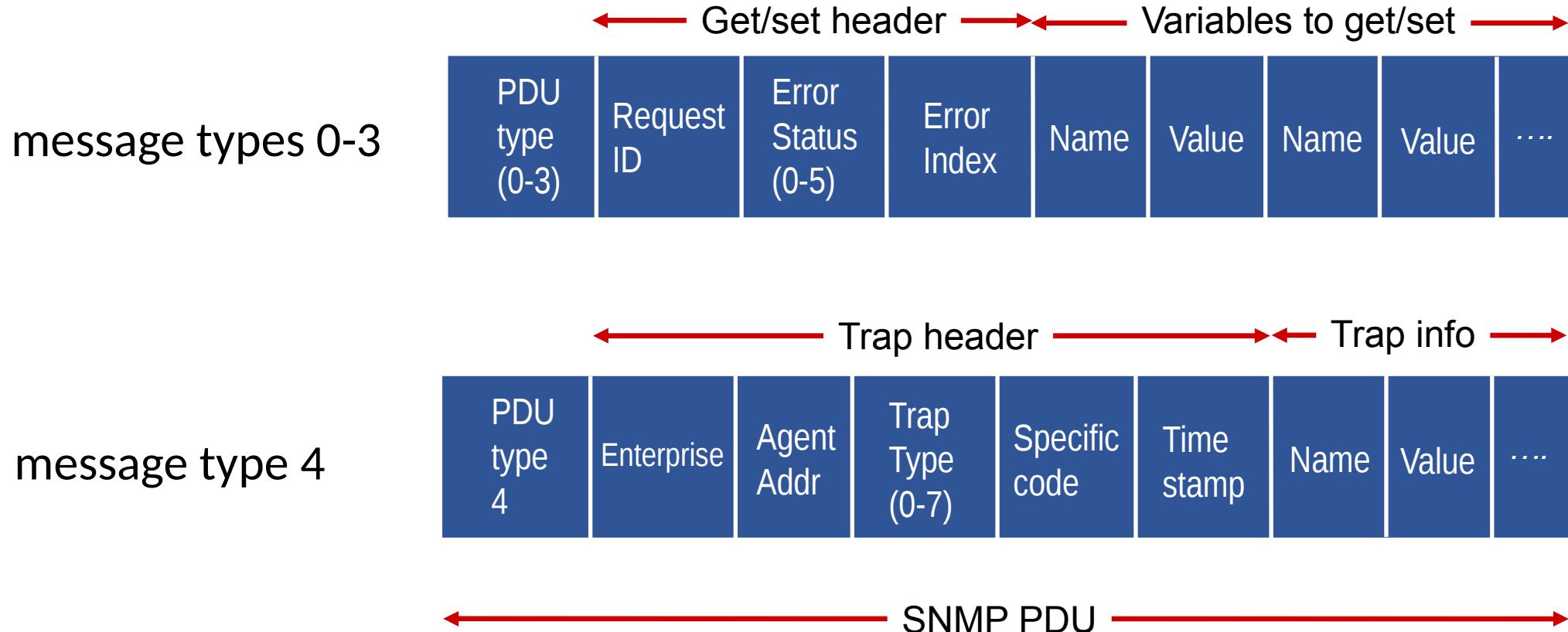
Two ways to convey MIB info, commands:



# SNMP protocol: message types

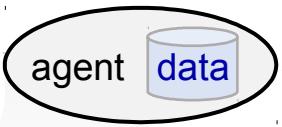
Message type	Function
GetRequest GetNextRequest GetBulkRequest	manager-to-agent: “get me data” (data instance, next data in list, block of data).
SetRequest	manager-to-agent: set MIB value
Response	Agent-to-manager: value, response to Request
Trap	Agent-to-manager: inform manager of exceptional event

# SNMP protocol: message formats



# SNMP: Management Information Base (MIB)

- managed device's operational (and some configuration) data
- gathered into device **MIB module**
  - 400 MIB modules defined in RFC's; many more vendor-specific MIBs
- **Structure of Management Information (SMI):** data definition language
- example MIB variables for UDP protocol:

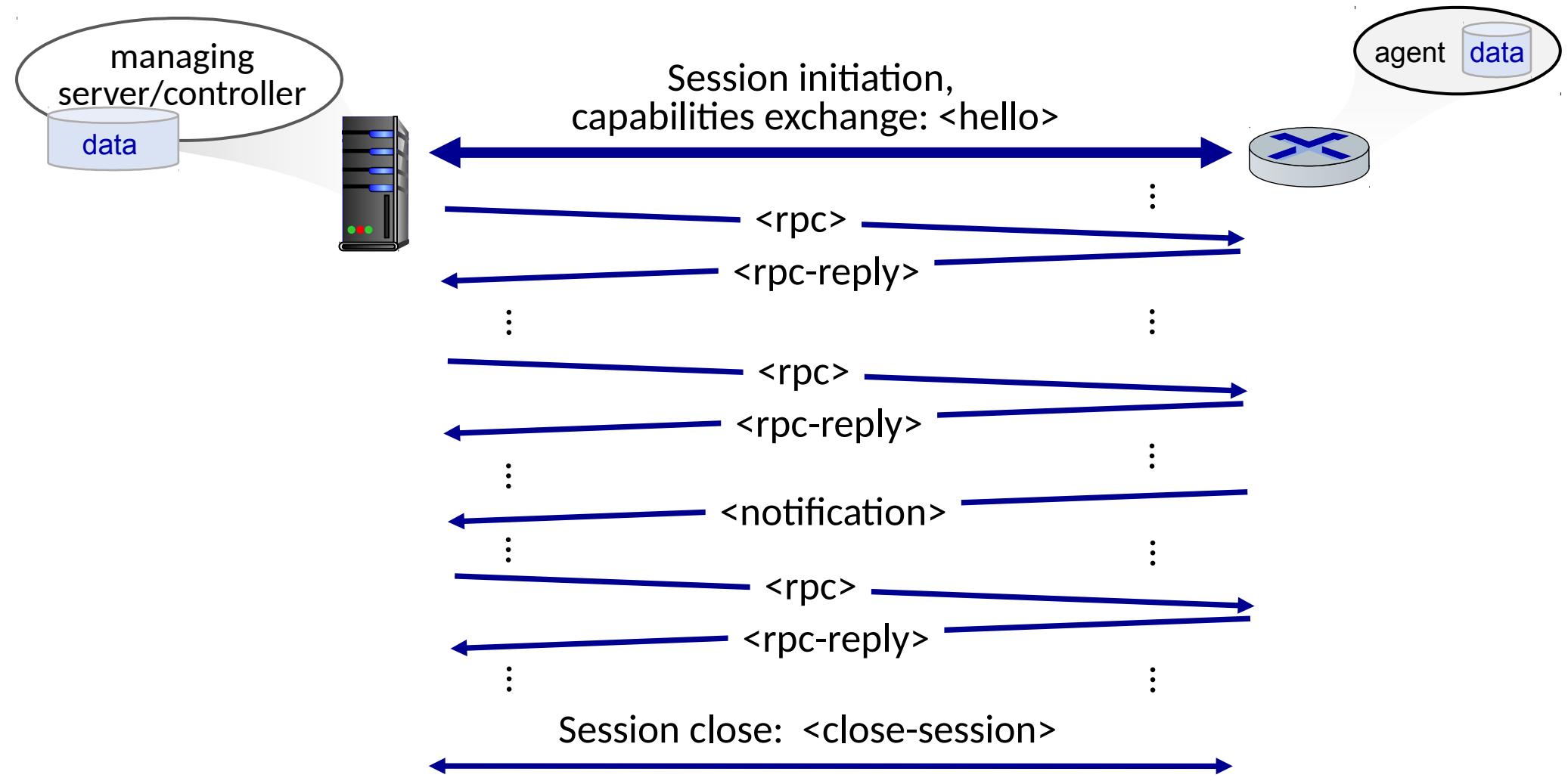


Object ID	Name	Type	Comments
1.3.6.1.2.1.7.1	UDPIInDatagrams	32-bit counter	total # datagrams delivered
1.3.6.1.2.1.7.2	UDPNoPorts	32-bit counter	# undeliverable datagrams (no application at port)
1.3.6.1.2.1.7.3	UDInErrors	32-bit counter	# undeliverable datagrams (all other reasons)
1.3.6.1.2.1.7.4	UDPOutDatagrams	32-bit counter	total # datagrams sent
1.3.6.1.2.1.7.5	udpTable	SEQUENCE	one entry for each port currently in use

# NETCONF overview

- **goal:** actively manage/configure devices network-wide
- operates between managing server and managed network devices
  - actions: retrieve, set, modify, activate configurations
  - **atomic-commit** actions over multiple devices
  - query operational data and statistics
  - subscribe to notifications from devices
- remote procedure call (RPC) paradigm
  - NETCONF protocol messages encoded in XML
  - exchanged over secure, reliable transport (e.g., TLS) protocol

# NETCONF initialization, exchange, close



# Selected NETCONF Operations

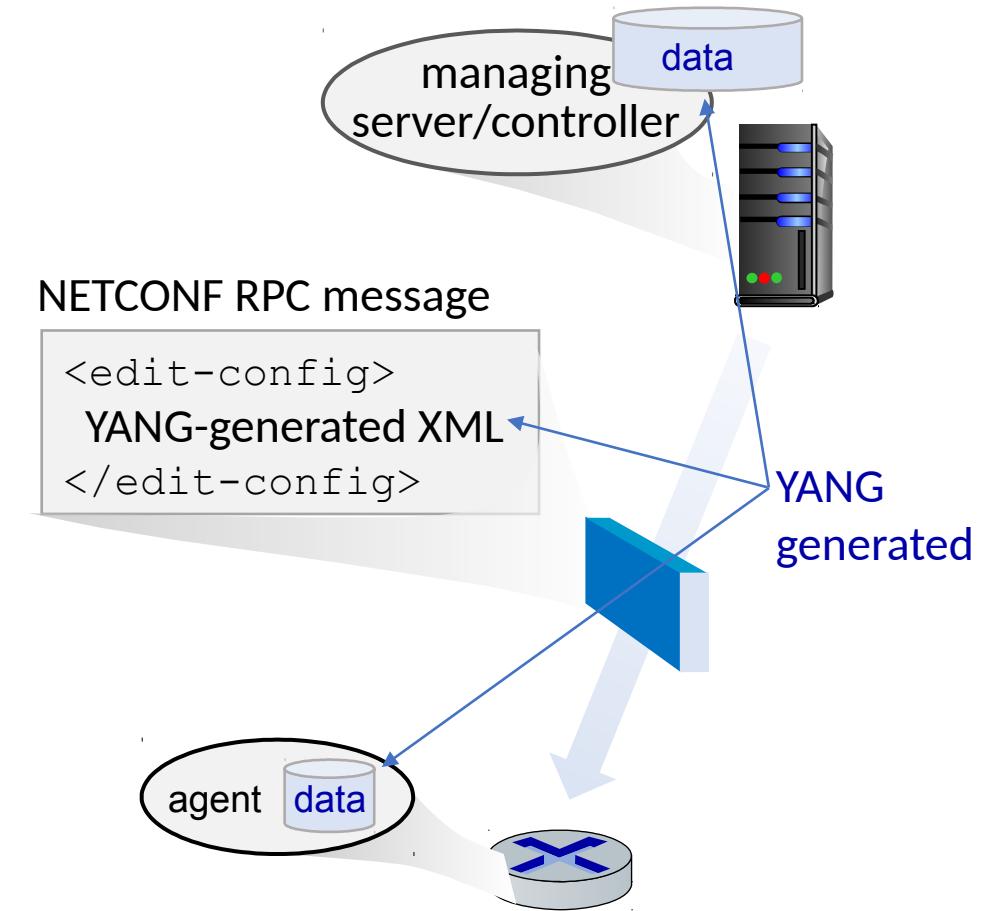
NETCONF	Operation Description
<get-config>	Retrieve all or part of a given configuration. A device may have multiple configurations.
<get>	Retrieve all or part of both configuration state and operational state data.
<edit-config>	Change specified (possibly running) configuration at managed device. Managed device <rpc-reply> contains <ok> or <rpcerror> with rollback.
<lock>, <unlock>	Lock (unlock) configuration datastore at managed device (to lock out NETCONF, SNMP, or CLIs commands from other sources).
<create-subscription>, <notification>	Enable event notification subscription from managed device

# Sample NETCONF RPC message

```
01 <?xml version="1.0" encoding="UTF-8"?>
02 <rpc message-id="101" note message id
03   xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
04     <edit-config>    change a configuration
05       <target>
06         <running/>  change the running configuration
07       </target>
08     <config>
09       <top xmlns="http://example.com/schema/
10         1.2/config">
11           <interface>
12             <name>Ethernet0/0</name>  change MTU of Ethernet 0/0 interface to 1500
13             <mtu>1500</mtu>
14           </interface>
15         </top>
16       </config>
17     </edit-config>
18   </rpc>
```

# YANG

- data modeling language used to specify structure, syntax, semantics of NETCONF network management data
  - built-in data types, like SMI
- XML document describing device, capabilities can be generated from YANG description
- can express constraints among data that must be satisfied by a valid NETCONF configuration
  - ensure NETCONF configurations satisfy correctness, consistency constraints



# Network layer: Summary

we've learned a lot!

- approaches to network control plane
  - per-router control (traditional)
  - logically centralized control (software defined networking)
- traditional routing algorithms
  - implementation in Internet: OSPF , BGP
- SDN controllers
  - implementation in practice: ODL, ONOS
- Internet Control Message Protocol
- network management

*next stop: link layer!*

# Network layer, control plane: Done!

- introduction
- routing protocols
  - link state
  - distance vector
- intra-ISP routing: OSPF
- routing among ISPs: BGP
- SDN control plane
- Internet Control Message Protocol



- network management, configuration
  - SNMP
  - NETCONF/YANG

# **Additional Chapter 5 slides**

# Distance vector: another example

$D_x()$

	x	y	z
x	0	2	7
y	$\infty$	$\infty$	$\infty$
z	$\infty$	$\infty$	$\infty$

$D_x()$

	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

$$D_x(z) = \min\{c_{x,y} + D_y(z), c_{x,z} + D_z(z)\}$$

$$= \min\{2+1, 7+0\} = 3$$

$D_y()$

	x	y	z
x	$\infty$	$\infty$	$\infty$
y	2	0	1
z	$\infty$	$\infty$	$\infty$

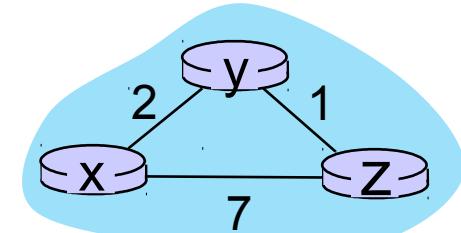
$$D_x(y) = \min\{c_{x,y} + D_y(y), c_{x,z} + D_z(y)\}$$

$$= \min\{2+0, 7+1\} = 2$$

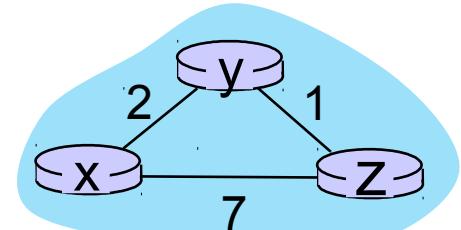
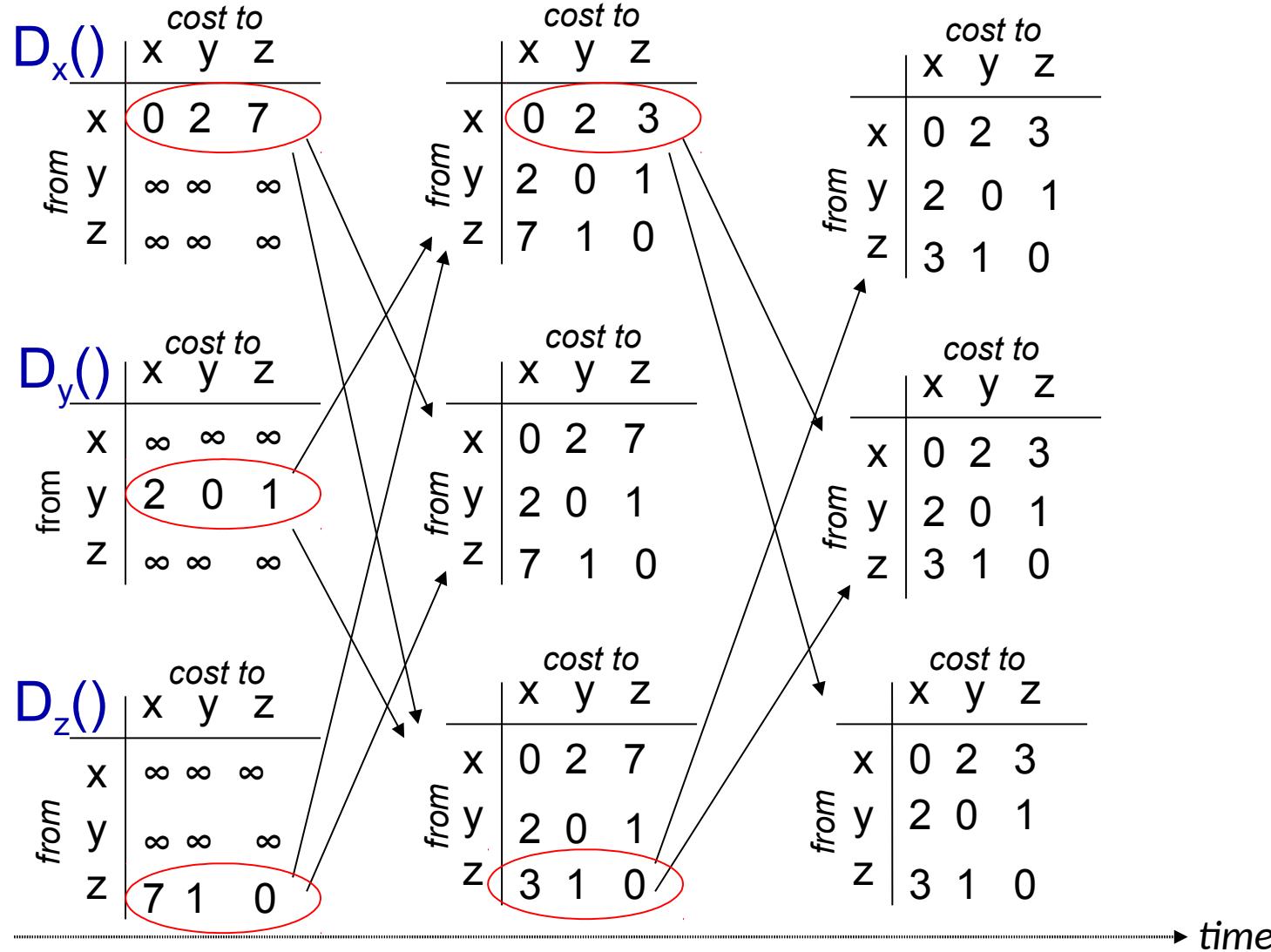
$D_z()$

	x	y	z
x	$\infty$	$\infty$	$\infty$
y	$\infty$	$\infty$	$\infty$
z	7	1	0

time



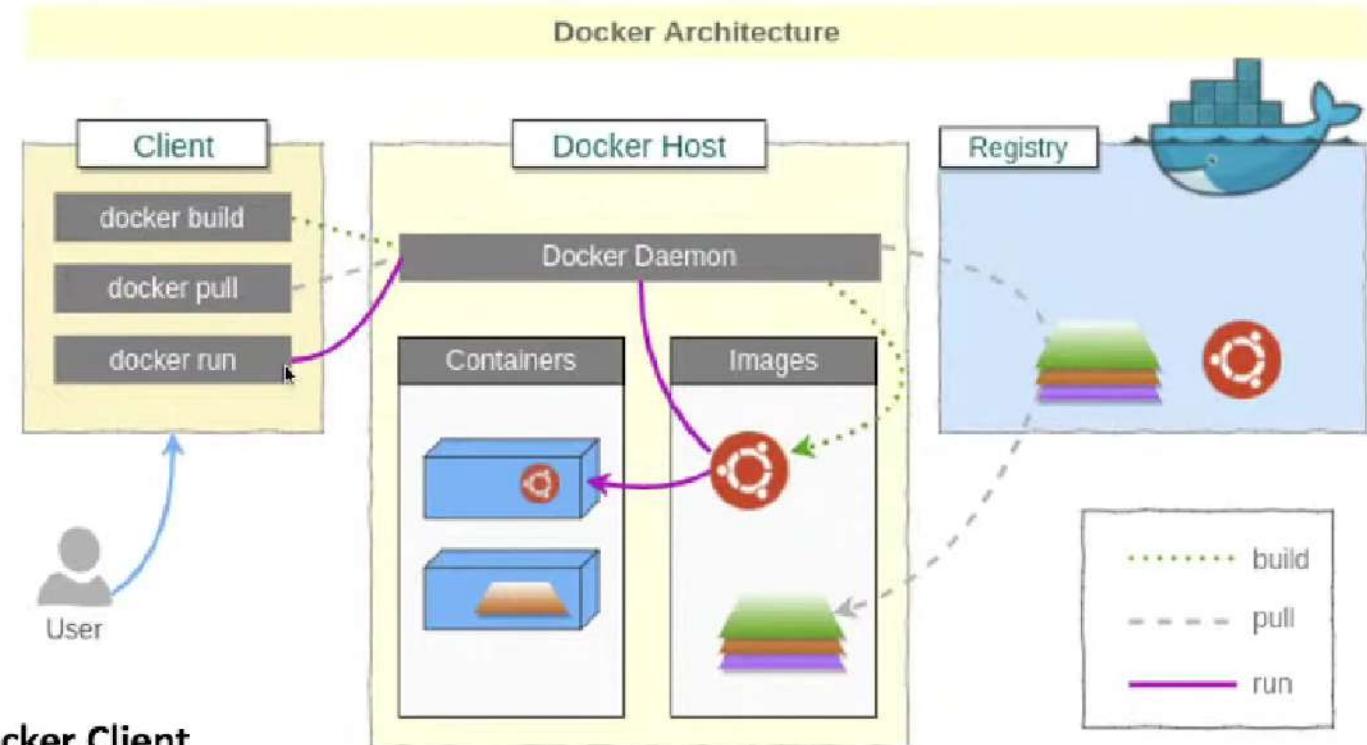
# Distance vector: another example





# Docker Architecture

- Docker uses a client-server architecture.
- The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon.
- The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.



- 1) Docker Client
- 2) Docker Daemon
- 3) Docker Registry
- 4) Docker Image

# Docker Image/Container and DockerFile Commands

- A Docker image is a read-only, inert template that comes with instructions for deploying containers. In Docker, everything basically revolves around images.
- An image consists of a collection of files (or layers) that pack together all the necessities—such as dependencies, source code, and libraries—needed to set up a completely functional container environment.
- Images are stored on a Docker registry, such as the Docker Hub, or on a local registry.

```
PS C:\WINDOWS\system32> docker pull bash:5.0
5.0: Pulling from library/bash
188c0c94c7c5: Pulling fs layer
94387ca39817: Pulling fs layer
efe7174943e6: Pulling fs layer
efe7174943e6: Verifying Checksum
efe7174943e6: Download complete
188c0c94c7c5: Verifying Checksum
188c0c94c7c5: Download complete
188c0c94c7c5: Pull complete
94387ca39817: Verifying Checksum
94387ca39817: Download complete
94387ca39817: Pull complete
efe7174943e6: Pull complete
Digest: sha256:01fad26fa8ba21bce6e8c47222acfdb54649957f1e86d53a0c8e03360271abf6
Status: Downloaded newer image for bash:5.0
docker.io/library/bash:5.0
```

```
PS C:\WINDOWS\system32> docker images
REPOSITORY          TAG      IMAGE ID
bash                5.0      39a95ac32011
PS C:\WINDOWS\system32>
```



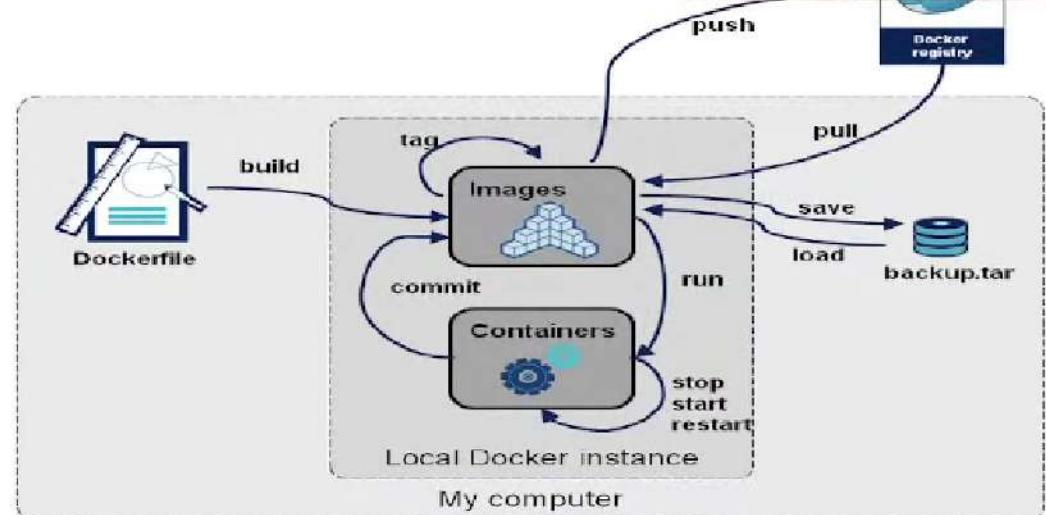
A Docker container is a virtualized runtime environment that provides isolation capabilities for separating the execution of applications from the underpinning system. It's an instance of a Docker image.

```
Image Version Pinning missing (DL3006,DL3007)
Base Image FROM ubuntu :12.04
Env. Variable MAINTAINER John Doe <joe@doe.org>
Comment # Add proxy settings
COPY ./setenv.sh /tmp/
RUN can execute any shell command
RUN sudo apt-get update
RUN sudo apt-get upgrade -y
Installing dependencies
RUN apt-get install -y wget[:1.12]
Dependency Version Pinning missing (DL3008,DL3013)
RUN sudo -E pip install scipy[:0.18.1]
Installing software (compiling, linking, etc.)
RUN cd /usr/src/vertica-sqlalchemy;
    sudo python ./setup.py install
Open Port EXPOSE 8888
# CMD ipython notebook --ip=...
ADD runcmd.sh /← Using ADD instead of COPY (DL3020)
RUN chmod u+x /runcmd.sh
```

# Docker Main Co



- **docker pull** ubuntu. (To pull image from hub/repo)
- **docker run -it -name c1 -d -p 82:80 ubuntu** (To run an image as container)
- **docker exec -it c1 bash** (To login into container)  
    Backup of container as Image
- **docker commit c1 apache-on-ubuntu:1.0**(To save the container data as new Image)
- **docker save apache-on-ubuntu:1.0 --output backup.tar** (To save the image as tar)
- **docker load -i backup.tar** (To unzip the image from tar)
- **docker start/stop/restart c1** (Container commands)
- **docker push image-name** (To push the image to container)
- **docker build Dockerfile**





## Docker Container States

- **Created** - Docker assigns the *created* state to the containers that were never started ever since they were created. Hence, no CPU or memory is used by the containers in this state.
- **Running** - When we start a container having created a state using the docker start command, it attains the running state. This state signifies that the processes are running in the isolated environment inside the container.
- **Restarting** - Docker supports four types of restart policies, namely – no, on-failure, always, unless-stopped. Restart policy decides the behaviour of the container when it exits. By default, the restart policy is set to no, which means that the container will not be started automatically after it exits.
- **Exited** - This state is achieved when the process inside the container terminates. **No CPU and memory are consumed** by the container.

The process inside the container was completed, and so it exited.

The process inside the container encountered an exception while running.

A container is intentionally stopped using the docker stop command.

No interactive terminal was set to a container running bash.

- **Pause** - A paused container consumes the same memory used while running the container, but the CPU is released completely.

```
$ docker create --name mycontainer httpd  
8d60cb560afc1397d6732672b2b4af16a08bf6289a5a00605125c5b35e8ee749  
$ docker inspect -f '{{.State.Status}}' mycontainer  
created  
$ docker start mycontainer  
mycontainer  
$ docker inspect -f '{{.State.Status}}' mycontainer  
running
```

```
$ docker run -itd --restart=always --name mycontainer centos:7 sleep 5  
f7d0e8becdac1ebf7aae25be2d02409f0f211fcc191aea000041d158f89be6f6
```

```
$ docker pause mycontainer  
mycontainer  
$ docker inspect -f '{{.State.Status}}' mycontainer  
paused
```

```
$ docker stats --no-stream  
CONTAINER ID  NAME          CPU %     MEM USAGE / LIMIT      MEM %     NET I/O      BLOCK I/O     PIDS  
1a44702ce17  mycontainer   0.00%    1.09MiB / 7.28GiB  0.01%    1.37KB / 0B    0B / 0B     1
```

# Docker Networking

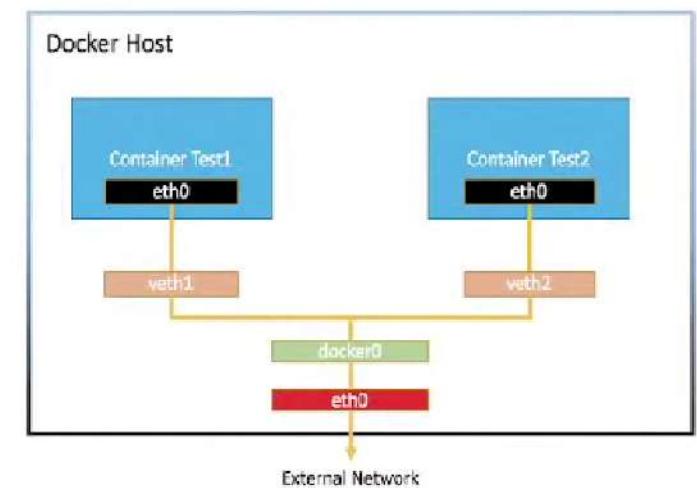


- 1) **Bridge:** The default network driver. Bridge networks apply to containers running on the same Docker daemon host
- 2) **Host:** For standalone containers, remove network isolation between the container and the Docker host, and use the host's networking directly
- 3) **Overlay:** Overlay networks connect multiple Docker daemons together and enable swarm services to communicate with each other. You can also use overlay networks to facilitate communication between a swarm service and a standalone container, or between two standalone containers on different Docker daemons.

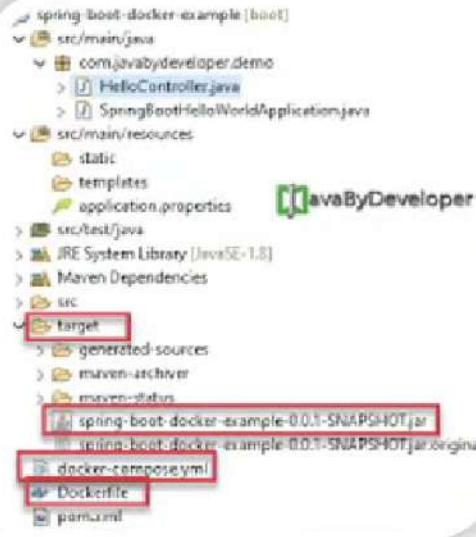
- **User-defined bridges** provide better isolation
- Containers can be attached and detached from user-defined networks on the fly.
- Each user-defined network creates a configurable bridge.

## Managing User defined bridge network

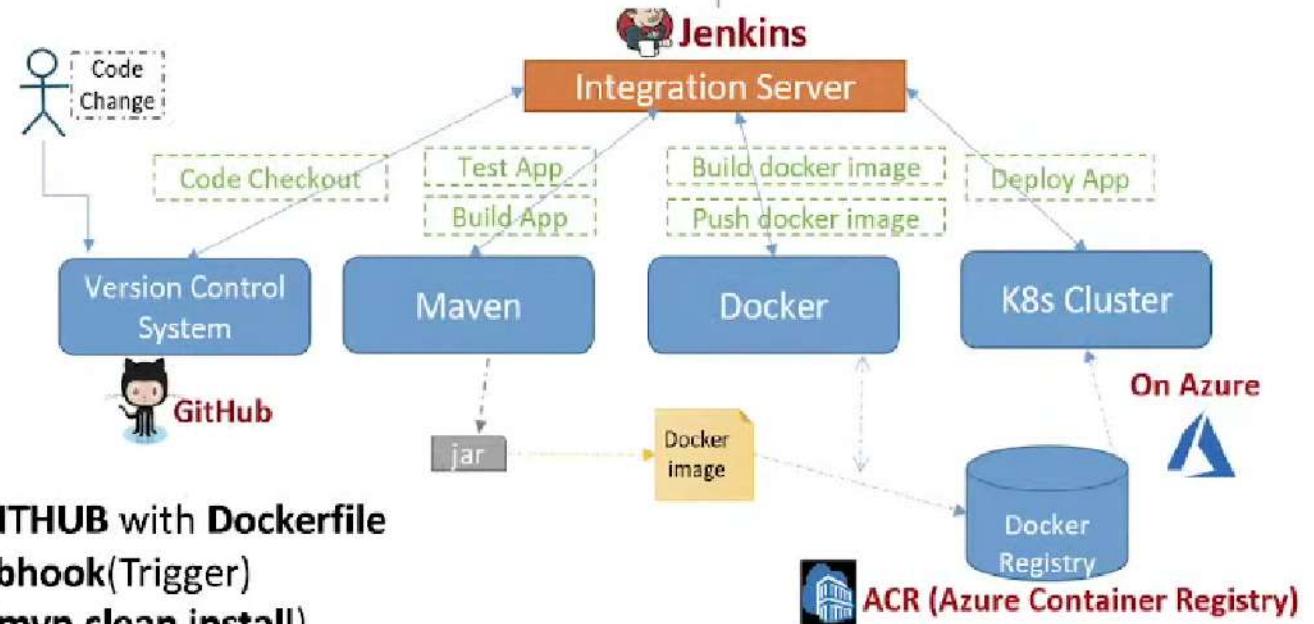
- `docker network create -d docker0` (Basic command)
- `docker network rm my-net` (To remove the network)
- `docker network create -d overlay docker0` (To connect to multiple daemons)
- `docker network create --driver=bridge --subnet=172.28.0.0/16 --ip-range=172.28.5.0/24 --gateway=172.28.5.254 docker0`
- `docker create --name my-nginx --network docker0 --publish 8080:80 nginx:latest` (To create a container without start state)
- `docker network disconnect docker0 my-nginx`



# DockerFile



```
FROM openjdk:8-jdk-alpine
RUN addgroup -S spring && adduser -S spring -G spring
USER spring:spring
ARG JAR_FILE=target/*.jar
COPY ${JAR_FILE} app.jar
ENTRYPOINT ["java","-jar","/app.jar"]
```



Stage 1 - Developer commits the JAVA code in **GITHUB** with **Dockerfile**

Stage 2 - **Jenkins** will check for changes with **webhook(Trigger)**

Stage 3 - **Maven** build happens with command (**mvn clean install**)

Stage 4 - **Docker build** command is given in Jenkins (docker build .)

Stage 4.1 -The **docker image** is created with the help of dockerfile

Stage 4.2 - **Docker push to dockerregistry or Jfrog**

Stage 5(Deploy App) - **ANSIBLE** (Which will pull the image from dockerregistry)



Press Esc to exit full screen

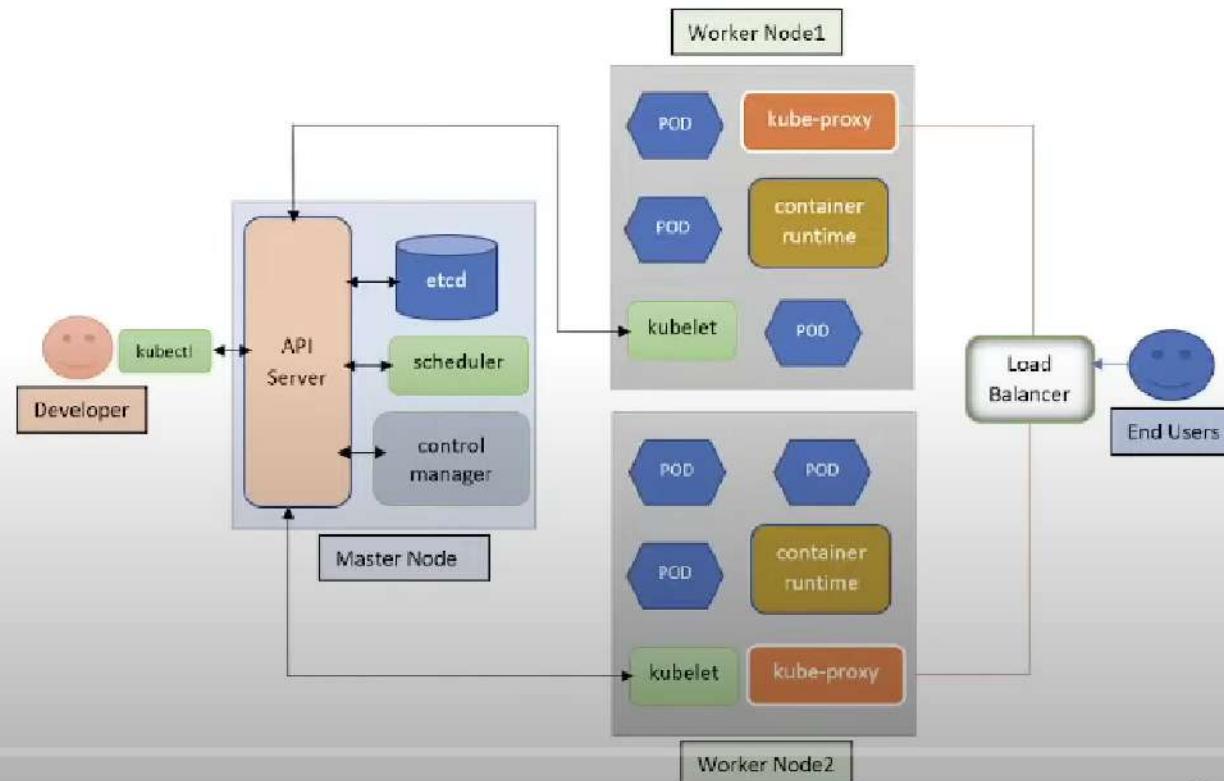


# Kubernetes Architecture:

A Kubernetes cluster consists of a set of worker machines, called nodes, that run containerized applications. Every cluster has at least one worker node.

The worker node(s) host the Pods that are the components of the application workload.

The control pane manages the worker nodes and the Pods in the cluster.



Press Esc to exit full screen



# Components of Kubernetes:

- **Master Node Components:**

1. API Server 2. Controller Manager 3. ETCD 4. Scheduler

- 1) **API Server:**

It is the front-end for the Kubernetes control plane.

- 2) **Controller Manager:** This is a component on the master that runs controllers.

- **Node Controller:** Responsible for noticing and responding when nodes go down.
  - **Replication Controller:** Responsible for maintaining the correct number of pods for every replication controller object in the system.
  - **Endpoints Controller:** Populates the Endpoints object (that is, it joins Services and Pods).
  - **Service Account and Token Controllers:** Create default accounts and API access tokens for new namespaces.
- 3) **ETCD:** Consistent and highly-available key value store used as Kubernetes' backing store for all cluster data.
  - 4) **kube-scheduler** - Control plane component that watches for newly created Pods with no assigned node, and selects a node for them to run on.



# Node Components

Node components run on every node, maintaining running pods and providing the Kubernetes runtime environment

1) **Kubelet** - An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod.

The kubelet takes a set of PodSpecs that are provided through various mechanisms and ensures that the containers described in those PodSpecs are running and healthy. The kubelet doesn't manage containers which were not created by Kubernetes.

2)**kube-proxy** - kube-proxy is a network proxy that runs on each node in your cluster, implementing part of the Kubernetes Service concept. Kube-proxy maintains network rules on nodes. These network rules allow network communication to your Pods from network sessions inside or outside of your cluster.

3)**Container runtime** - The container runtime is the software that is responsible for running containers.

# Manifest File Components



apiVersion

Kind

Metadata

Spec

File name : nginx-deployment.yaml

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-nginx
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
      ports:
      - containerPort: 80
```

Press Esc to exit full screen

# Service components:

Kubernetes ServiceTypes allow you to specify what kind of Service you want. The default is ClusterIP.

Type values and their behaviors are:

**ClusterIP:** Exposes the Service on a cluster-internal IP. Choosing this value makes the Service only reachable from within the cluster. This is the default ServiceType.(To talk to other nodes in the cluster)

**NodePort:** Exposes the Service on each Node's IP at a static port (the NodePort). A ClusterIP Service, to which the NodePort Service routes, is automatically created. You'll be able to contact the NodePort Service, from outside the cluster, by requesting <NodeIP>:<NodePort>. (The endpoint for node)

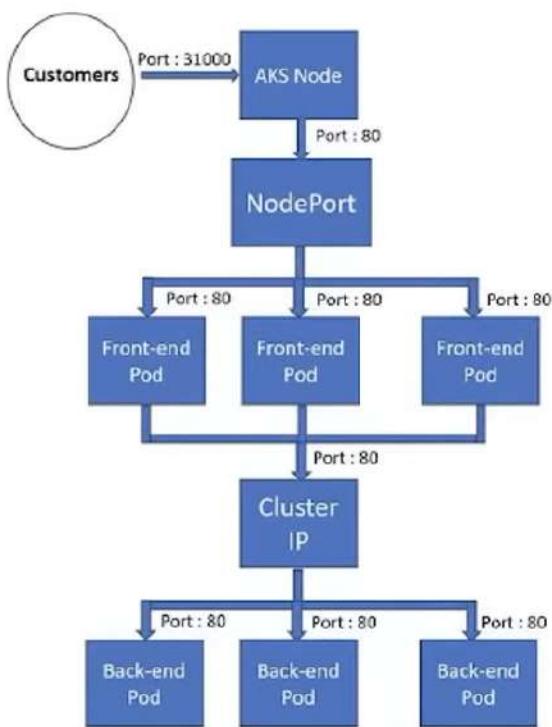
```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  type: NodePort
  selector:
    app: MyApp
  ports:
    # By default and for convenience,
    - port: 80
      targetPort: 80
      # Optional field
      # By default and for convenience,
      nodePort: 30007
```





- **LoadBalancer**: Exposes the Service externally using a cloud provider's load balancer. NodePort and ClusterIP Services, to which the external load balancer routes, are automatically created.
- **ExternalName**: Maps the Service to the contents of the externalName field (e.g. foo.bar.example.com), by returning a CNAME record with its value. No proxying of any kind is set up.

```
apiVersion: v1
kind: Service
metadata:
  name: my-nginx-svc
  labels:
    app: nginx
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: nginx
```



```
kind: Service
apiVersion: v1

metadata:
  name: hostname-service

spec:
  type: NodePort
  selector:
    app: echo-hostname
  ports:
    - nodePort: 30163
      port: 8080
      targetPort: 80
```

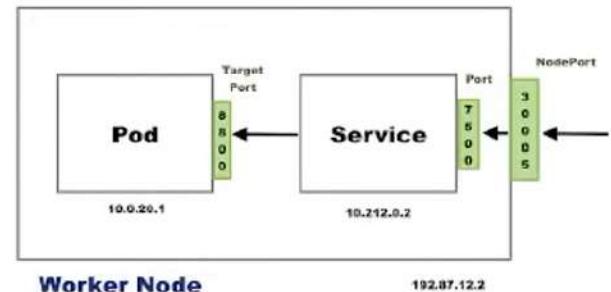
Make the service available to network requests from external clients

Forward requests to pods with label of this value

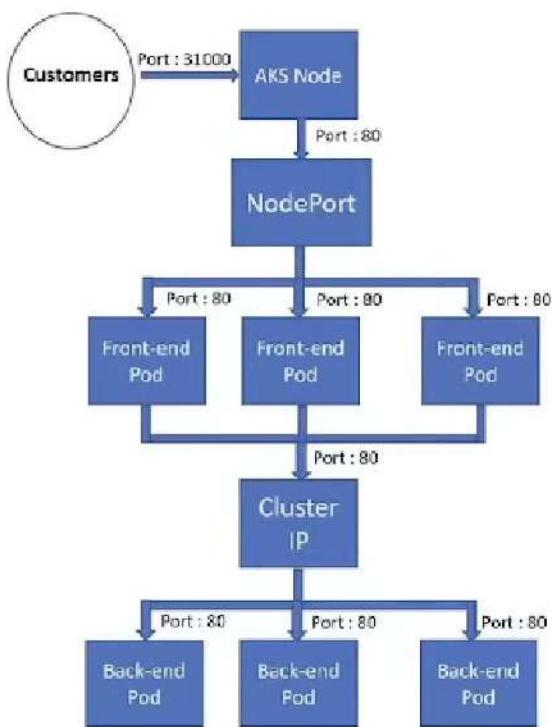
nodePort  
access service on this external port number

port  
port number exposed externally in cluster

targetPort  
port that containers are listening on



# Routing In kubernetes



```
kind: Service
apiVersion: v1

metadata:
  name: hostname-service

spec:
  type: NodePort
  selector:
    app: echo-hostname
  ports:
    - nodePort: 30163
      port: 8080
      targetPort: 80
```

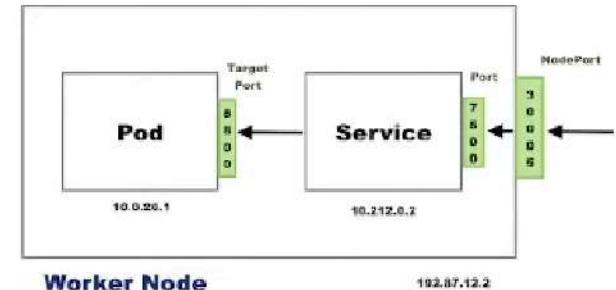
Make the service available to network requests from external clients.

Forward requests to pods with label of this value.

nodePort  
access service on this external port number

port  
port number exposed internally in cluster

targetPort  
pod port endpoints are referring to



# Routing In kubernetes



# Kubernetes Commands

- 1) Kubectl create deploy my-nginx –image=nginx –dry-run=client –o yaml > mynginx.yaml
- 2) Kubectl get pods
- 3) Kubectl get namespace
- 4) Kubectl create ns mydev
- 5) Kubectl create –f pod.yaml
- 6) Kubectl describe svc nginx
- 7) Kubectl exec -it pod1 bash



## Kubernetes IQ:

- 1) What is the architecture of kubernetes
- 2) What does control manager, etcd, scheduler, API server do
- 3) What is a manifest file and what are the components of it
- 4) What is node affinity, pod affinity , taint toleration
- 5) What is node port, cluster ip
- 6) What is persitant volumes and why we use it
- 7) Describe what is pod and what is pod lifecycle
- 8) What are the components on master and worked node
  - 9) What is ingress controller
  - 10) What are types of services in kuberntes
  - 11) How one pod talks with other pod
- 12) How the pod healthcheck is done(describe rediness, livesness)
- 13) How the monitoring is done(integration on Prometheus and grafana)
- 14) What is deamonset, replicaset, horizontal pod autoscaler
  - 15) Write a manifest file of your own choice
  - 16) What is namespace and why we use it
  - 17) What are helm charts and uses

Install Package and start service

```
# dnf install docker  
# systemctl restart docker  
# systemctl enable docker  
# docker ps  
# docker ps -a
```

```
# docker run hello-world  
# docker kill  
# docker ps -l #→ last exited  
# docker ps  
# docker images  
# docker rm  
# docker rmi
```

```
# docker run -it --rm --name ubuntu ubuntu
```

```
# cat /etc/lsb-release  
# uname -r
```

```
# cat /etc/fedora-release  
# uname -r
```

Modify container form within it

```
docker commit container-id  
Docker images  
Docker tag image-id my-image  
Docker commit container-id image-tag  
Docker commit container-id image-tag:v1.1  
Docker images
```

```
Docker save image-name -o image-name.tar  
Docker load -i image-name.tar  
Docker images
```

```
Tar xvf image-name.tar  
–or–  
Dnf instal archivemount  
Archivemount image-name.tar mountpoint  
Chroot mountpoint
```

```
Docker run httpd
docker run -it --rm -d httpd
Curl ip of container
Firefox ip of host
Docker run -p 80:80 httpd
docker run -it --rm -d -p 80:80 httpd
Firefox ip of host

Docker run -p 80 httpd
Docker ports httpd-container-id
```

```
Docker network ls
Docker network create my-net-1
Docker run -rm -ti --net my-net-1 --name my-server-1 ubuntu
Create containers in different network and see they can not talk to each other
```

```
docker run --rm -ti -v /root/data/:/data:z --name fedora fedora bash
```

```
Share volumes among two containers
docker run --rm -ti -v /data --name fedora-1 fedora bash
docker run --rm -ti --volumes-from fedora-1 --name fedora-2 fedora
```

## Dockerfile

```
From fedora
Run echo "Dockerfile demo"
CMD echo "Hello World"
```

```
Docker build -t new .
```

```
Docker run -rm -ti ubuntu sleep 5
```

```
Docker logs container-name
```