



CSC424 System Administration

Instructor: Dr. Hao Wu

Week 1 Introduction

How we see ourselves



The Job of a System Administrator

- What exactly does a System Administrator do?



The Job of a System Administrator

- What exactly does a System Administrator do?



The Job of a System Administrator

- What exactly does a System Administrator do?



The Job of a System Administrator

- What exactly does a System Administrator do?



The Job of a System Administrator

- What exactly does a System Administrator do?



The Job of a System Administrator

- What exactly does a System Administrator do?



The Job of a System Administrator

- What exactly does a System Administrator do?



The Job of a System Administrator

- What exactly does a System Administrator do?



The Job of a System Administrator

- What exactly does a System Administrator do?



The Job of a System Administrator

- What exactly does a System Administrator do?



The Job of a System Administrator

- What exactly does a System Administrator do?



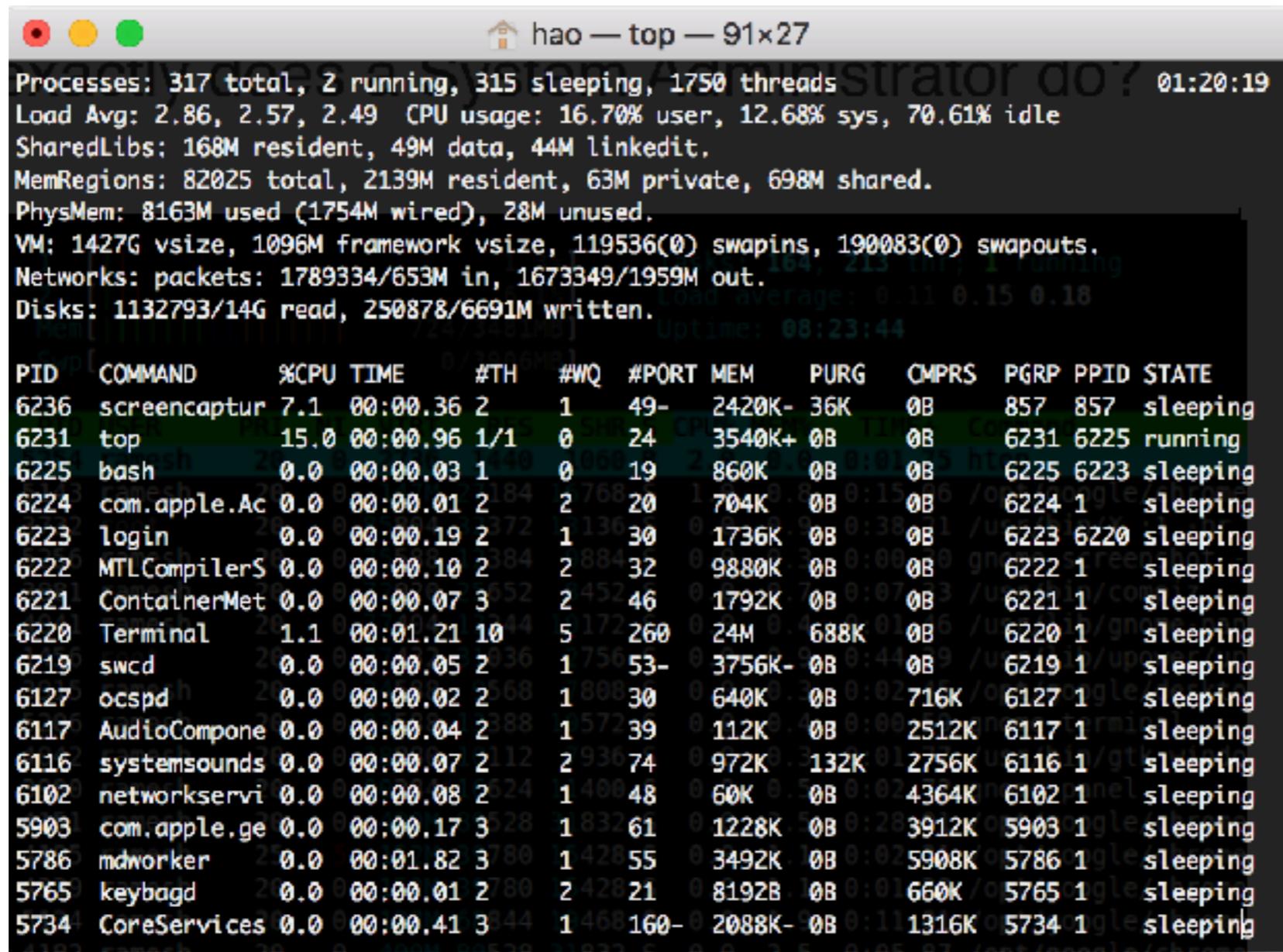
The Job of a System Administrator

- What exactly does a System Administrator do?



The Job of a System Administrator

- What exactly does a System Administrator do?



A screenshot of a terminal window titled "hao — top — 91x27". The window displays system statistics and a list of running processes. The statistics include:

- Processes: 317 total, 2 running, 315 sleeping, 1750 threads
- Load Avg: 2.86, 2.57, 2.49
- CPU usage: 16.70% user, 12.68% sys, 70.61% idle
- SharedLibs: 168M resident, 49M data, 44M linkedit.
- MemRegions: 82025 total, 2139M resident, 63M private, 698M shared.
- PhysMem: 8163M used (1754M wired), 28M unused.
- VM: 1427G vsize, 1096M framework vsize, 119536(0) swapins, 190083(0) swapouts.
- Networks: packets: 1789334/653M in, 1673349/1959M out.
- Disks: 1132793/14G read, 250878/6691M written.

The process list shows the following columns: PID, COMMAND, %CPU, TIME, #TH, #WQ, #PORT, MEM, PURG, CMPRS, PGRP, PPID, and STATE. The processes listed include screencaptur, top, bash, com.apple.Ac, login, MTLCompilerS, ContainerMet, Terminal, swcd, ocsprd, AudioCompone, systemsounds, networkservi, com.apple.ge, mdworker, keybagd, and CoreServices.

PID	COMMAND	%CPU	TIME	#TH	#WQ	#PORT	MEM	PURG	CMPRS	PGRP	PPID	STATE
6236	screencaptur	7.1	00:00.36	2	1	49-	2420K-	36K	0B	857	857	sleeping
6231	top	15.0	00:00.96	1/1	0	24	3540K+	0B	0B	6231	6225	running
6225	bash	0.0	00:00.03	1	0	19	860K	0B	0B	6225	6223	sleeping
6224	com.apple.Ac	0.0	00:00.01	2	2	20	704K	0B	0B	6224	1	sleeping
6223	login	0.0	00:00.19	2	1	30	1736K	0B	0B	6223	6220	sleeping
6222	MTLCompilerS	0.0	00:00.10	2	2	32	9880K	0B	0B	6222	1	sleeping
6221	ContainerMet	0.0	00:00.07	3	2	46	1792K	0B	0B	6221	1	sleeping
6220	Terminal	1.1	00:01.21	10	5	260	24M	688K	0B	6220	1	sleeping
6219	swcd	0.0	00:00.05	2	1	53-	3756K-	0B	0B	6219	1	sleeping
6127	ocsprd	0.0	00:00.02	2	1	30	640K	0B	716K	6127	1	sleeping
6117	AudioCompon	0.0	00:00.04	2	1	39	112K	0B	2512K	6117	1	sleeping
6116	systemsounds	0.0	00:00.07	2	2	74	972K	132K	2756K	6116	1	sleeping
6102	networkservi	0.0	00:00.08	2	1	48	60K	0B	4364K	6102	1	sleeping
5903	com.apple.ge	0.0	00:00.17	3	1	61	1228K	0B	3912K	5903	1	sleeping
5786	mdworker	0.0	00:01.82	3	1	55	3492K	0B	5908K	5786	1	sleeping
5765	keybagd	0.0	00:00.01	2	2	21	8192B	0B	660K	5765	1	sleeping
5734	CoreServices	0.0	00:00.41	3	1	160-	2088K-	0B	1316K	5734	1	sleeping

The Job of a System Administrator

- What exactly does a System Administrator do?
 - no precise job description
 - often learned by experience
 - "make things run"
 - work behind the scenes
 - often known as Operator, Network Administrator, System Programmer, System Manager, Service Engineer, Site Reliability Engineer etc.

Essential Duties of a System Administrator

- Controlling access
- Adding hardware
- Automating tasks
- Overseeing backups
- Installing and upgrading software
- Monitoring
- Troubleshooting
- Maintaining local documentation
- Vigilantly monitoring security
- Tuning performance
- Developing site policies
- Working with vendors
- Fire fighting

What is a System?

- "A group of interacting, interrelated, or interdependent elements that together form a complex whole."
- In the context of this class, we generally consider computer systems consisting of
 - the computers
 - the network
 - the users
 - the organization's goal and policies

Learning System Administration

- System Administration is a profession with no fixed career path
 - heavy reliance on practical experience
 - specializations in many different areas possible
 - breadth of expertise as necessary as depth in some areas
 - background knowledge and requirements vary

Learning System Administration

- Breadth of knowledge:
 - operating system concepts
 - TCP/IP networking
 - programming
 - security
 - ...
- Depth of knowledge:
 - certain OS flavor
 - specific service (DNS, E-Mail, Database, ...)
 - specific implementation/vendor (Oracle, Hadoop, Apache, Cisco, MS, AWS, ...)
 - specific area of expertise (security, storage, network, data center, ...)
 - ...

People think the internet looks like this



System Administrators know it looks like this



Content covered in this class

- We can only cover some of the system administration aspects:
 - Linux OS
 - Shell Script
 - Cloud

UNIX history

http://www.unix.org/what_is_unix/history_timeline.html

- Originally developed in 1969 at Bell Labs by Ken Thompson and Dennis Ritchie.
- 1973, Rewritten in C. This made it portable and changed the history of OS
- 1974: Thompson, Joy, Haley and students at Berkeley develop the Berkeley Software Distribution (BSD) of UNIX
- two main directions emerge: BSD and what was to become “System V”

Notable dates in UNIX history

- 1984 4.2BSD released (TCP/IP), 1986 4.3BSD released (NFS)
- 1991 Linus Torvalds starts working on the Linux kernel
- 1993 Settlement of USL vs. BSDi; NetBSD, then FreeBSD are created
- 1994 Single UNIX Specification introduced
- 1995 4.4BSD-Lite Release 2 (last CSRG release); OpenBSD forked off NetBSD
- 2000 Darwin created (derived from NeXT, FreeBSD, NetBSD)
- 2003 Xen; SELinux
- 2005 Hadoop; DTrace; ZFS; Solaris Containers
- 2006 AWS ("Cloud Computing" comes full circle)
- 2007 iOS; KVM appears in Linux
- 2008 Android; Solaris open sourced as OpenSolaris

Some UNIX versions

More UNIX (some generic, some trademark, some just unix-like):

1BSD	2BSD	3BSD	4BSD	4.4BSD Lite 1
4.4BSD Lite 2	386 BSD	A/UX	Acorn RISC iX	AIX
AIX PS/2	AIX/370	AIX/6000	AIX/ESA	AIX/RT
AMiX	AOS Lite	AOS Reno	ArchBSD	ASV
Atari Unix	BOS	BRL Unix	BSD Net/1	BSD Net/2
BSD/386	BSD/OS	CB Unix	Chorus	Chorus/MiX
Coherent	CTIX	Darwin	Debian GNU/Hurd	DEC OSF/1 ACP
Digital Unix	DragonFly BSD	Dynix	Dynix/ptx	ekkoBSD
FreeBSD	GNU	GNU-Darwin	HPBSD	HP-UX
HP-UX BLS	IBM AOS	IBM IX/370	Interactive 386/ix	Interactive IS
IRIX	Linux	Lites	LSX	Mac OS X
Mac OS X Server	Mach	MERT	MicroBSD	Mini Unix
Minix	Minix-VMD	MIPS OS	MirBSD	Mk Linux
Monterey	more/BSD	mt Xinu	MVS/ESA OpenEdition	NetBSD
NeXTSTEP	NonStop-UX	Open Desktop	Open UNIX	OpenBSD
OpenServer	OPENSTEP	OS/390 OpenEdition	OS/390 Unix	OSF/1
PC/IX	Plan 9	PWB	PWB/UNIX	QNX
QNX RTOS	QNX/Neutrino	QUNIX	ReliantUnix	Rhapsody
RISC iX	RT	SCO UNIX	SCO UnixWare	SCO Xenix
SCO Xenix System V/386	Security-Enhanced Linux	Sinix	Sinix ReliantUnix	Solaris
SPIX	SunOS	Tru64 Unix	Trusted IRIX/B	Trusted Solaris
Trusted Xenix	TS	UCLA Locus	UCLA Secure Unix	Ultrix
Ultrix 32M	Ultrix-11	Unicos	Unicos/mk	Unicox-max
UNICS	UNIX 32V	UNIX Interactive	UNIX System III	UNIX System IV
UNIX System V	UNIX System V Release 2	UNIX System V Release 3	UNIX System V Release 4	UNIX System V/286
UNIX System V/386	UNIX Time-Sharing System	UnixWare	UNSW	USG
Venix	Wollongong	Xenix OS	Xinu	xMach

Popular general purpose Linux distributions

Distribution	Web site	Comments
Arch	archlinux.org	For those who fear not the command line
CentOS	centos.org	Free analog of Red Hat Enterprise
CoreOS	coreos.com	Containers, containers everywhere
Debian	debian.org	Free as in freedom, most GNUish distro
Fedora	fedoraproject.org	Test bed for Red Hat Linux
Kali	kali.org	For penetration testers
Linux Mint	linuxmint.com	Ubuntu-based, desktop-friendly
openSUSE	opensuse.org	Free analog of SUSE Linux Enterprise
openWRT	openwrt.org	Linux for routers and embedded devices
Oracle Linux	oracle.com	Oracle-supported version of RHEL
RancherOS	rancher.com	20MiB, everything in containers
Red Hat Enterprise	redhat.com	Reliable, slow-changing, commercial
Slackware	slackware.com	Grizzled, long-surviving distro
SUSE Linux Enterprise	suse.com	Strong in Europe, multilingual
Ubuntu	ubuntu.com	Cleaned-up version of Debian

UNIX Everywhere

- Today, your desktop, server, cloud, TV, phone, watch, stereo, car navigation system, thermostat, door lock, etc. all run a Unix-like OS...
... with all the risks that entails.

Setup your Linux environment

- We will use VirtualBox to install Linux systems on your machine:

<https://www.virtualbox.org/wiki/Downloads>

- We will use CentOS as our Linux OS:

<https://www.centos.org/download/>

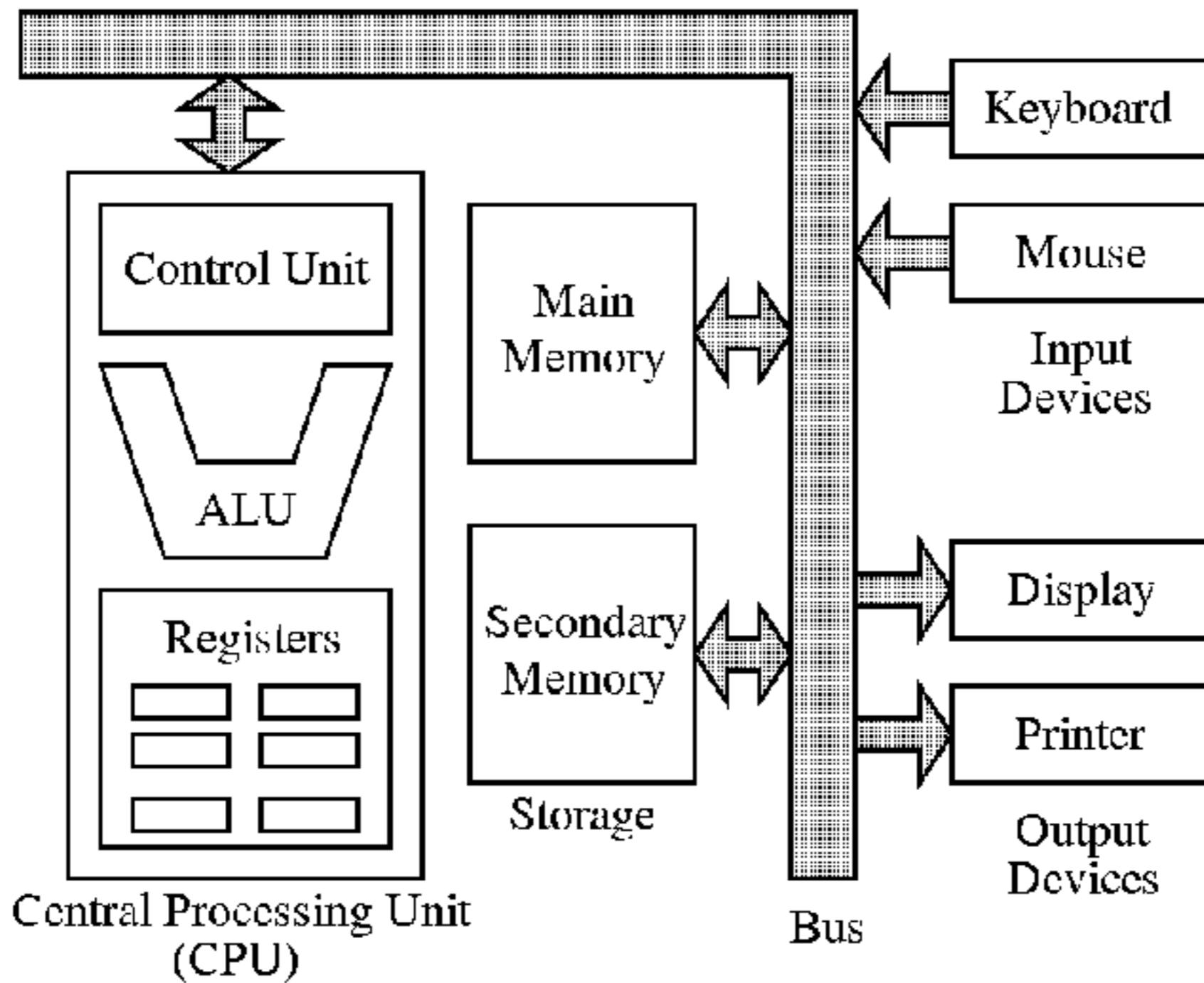


CSC424 System Administration

Instructor: Dr. Hao Wu

Week 2 Linux OS

Computer Architecture



Microprocessor

- Invention that brought about desktop and handheld computing
- Contains a processor on a single chip
- Fastest general purpose processors
- Multiprocessors
- Each chip (socket) contains multiple processors (cores)

Multicore CPUs and Multithreading Technologies

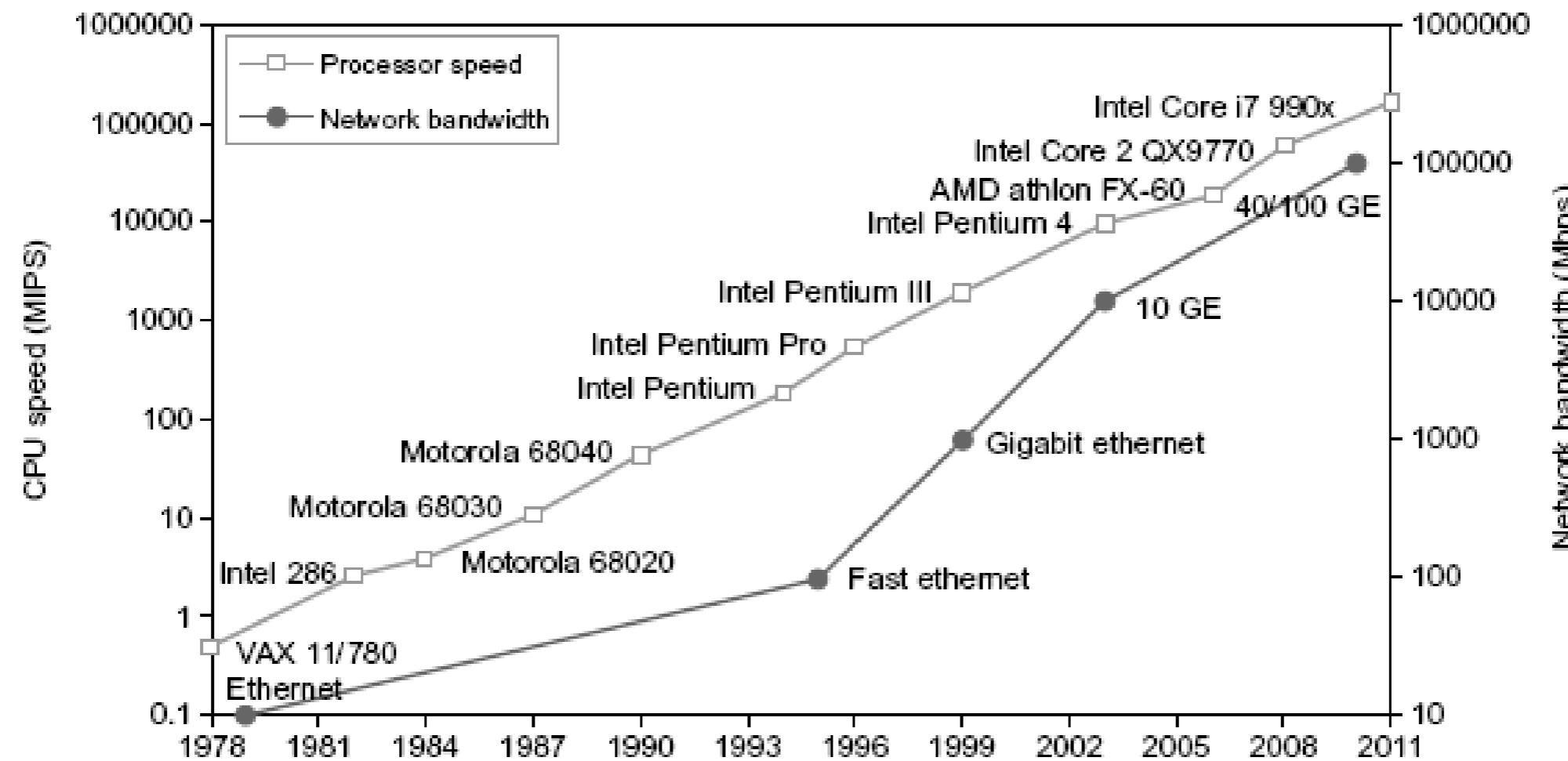


FIGURE 1.4

Improvement in processor and network technologies over 33 years.

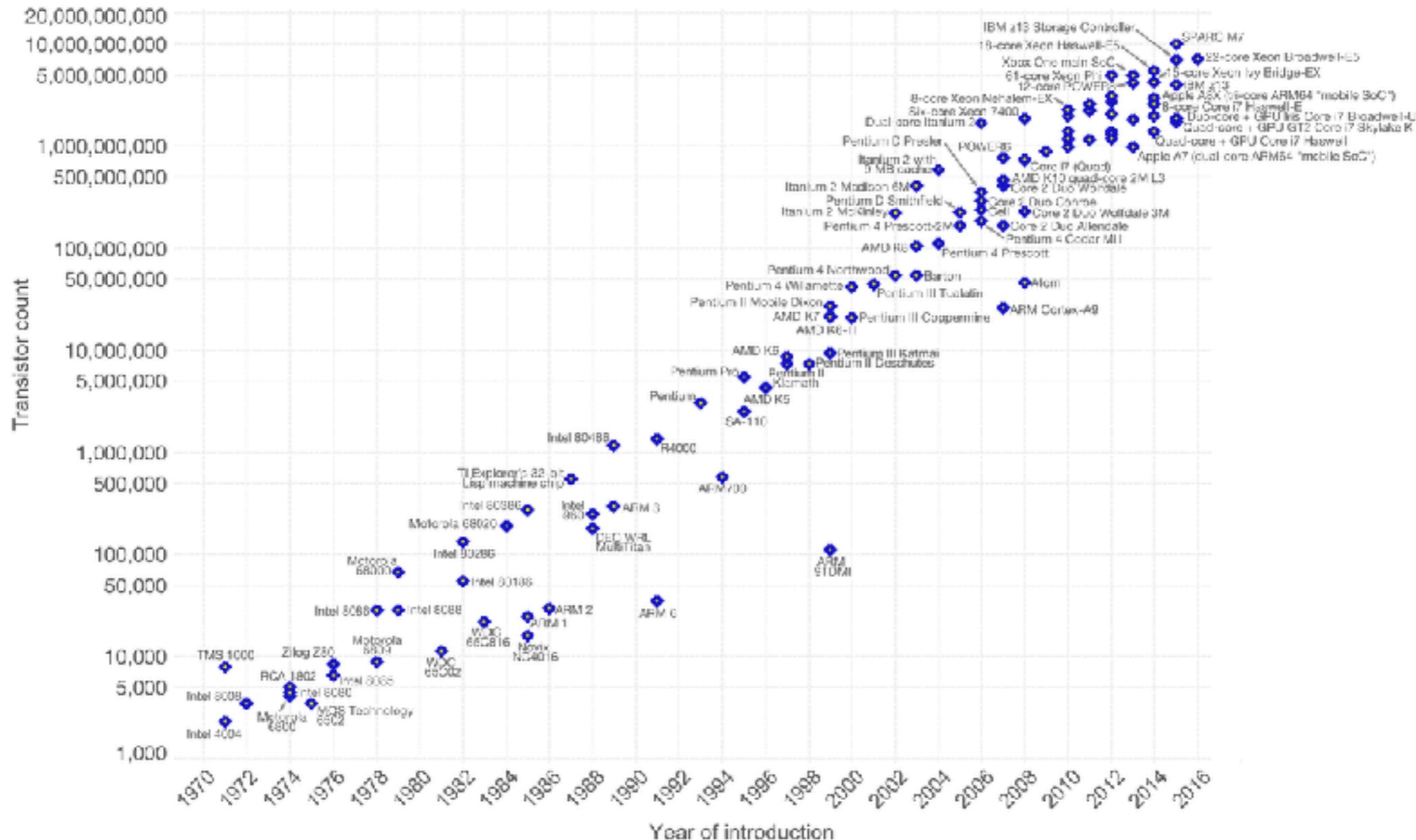
- 2017:
 - Intel Core i7 6950x: 317900 MIPS

Moore's Law

Moore's Law – The number of transistors on integrated circuit chips (1971-2016)

Our World
in Data

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are strongly linked to Moore's law.



Data source: Wikipedia (https://en.wikipedia.org/wiki/Transistor_count)

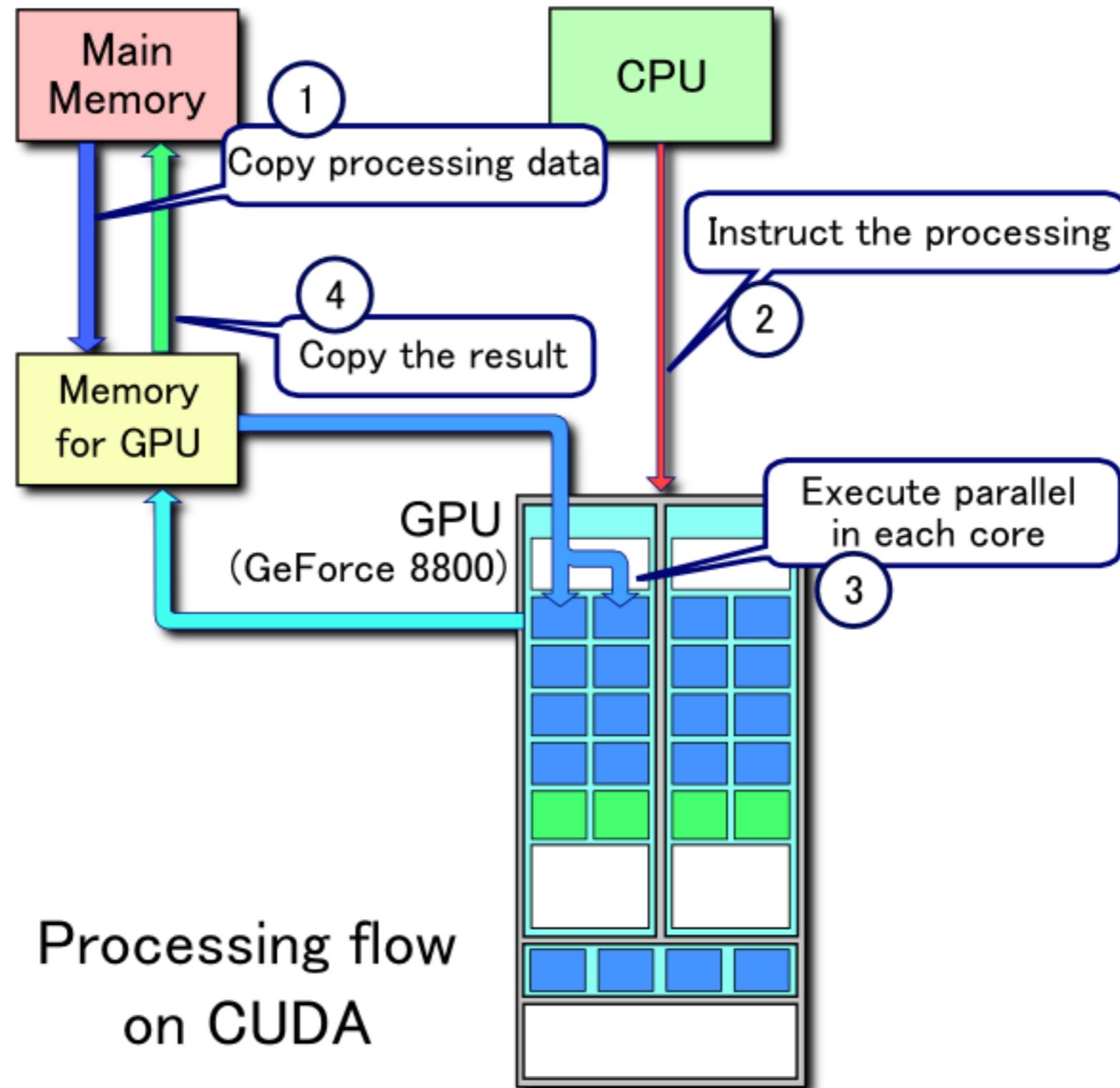
The data visualization is available at OurWorldinData.org. There you find more visualizations and research on this topic.

Licensed under CC-BY-SA by the author Max Roser.

Graphical Processing Unit (GPU)

- Provide efficient computation on arrays of data using Single-Instruction Multiple Data (SIMD) techniques pioneered in supercomputers
- No longer used just for rendering advanced graphics
 - Also used for general numerical processing
 - Physics simulations for games
 - Computations on large spreadsheets

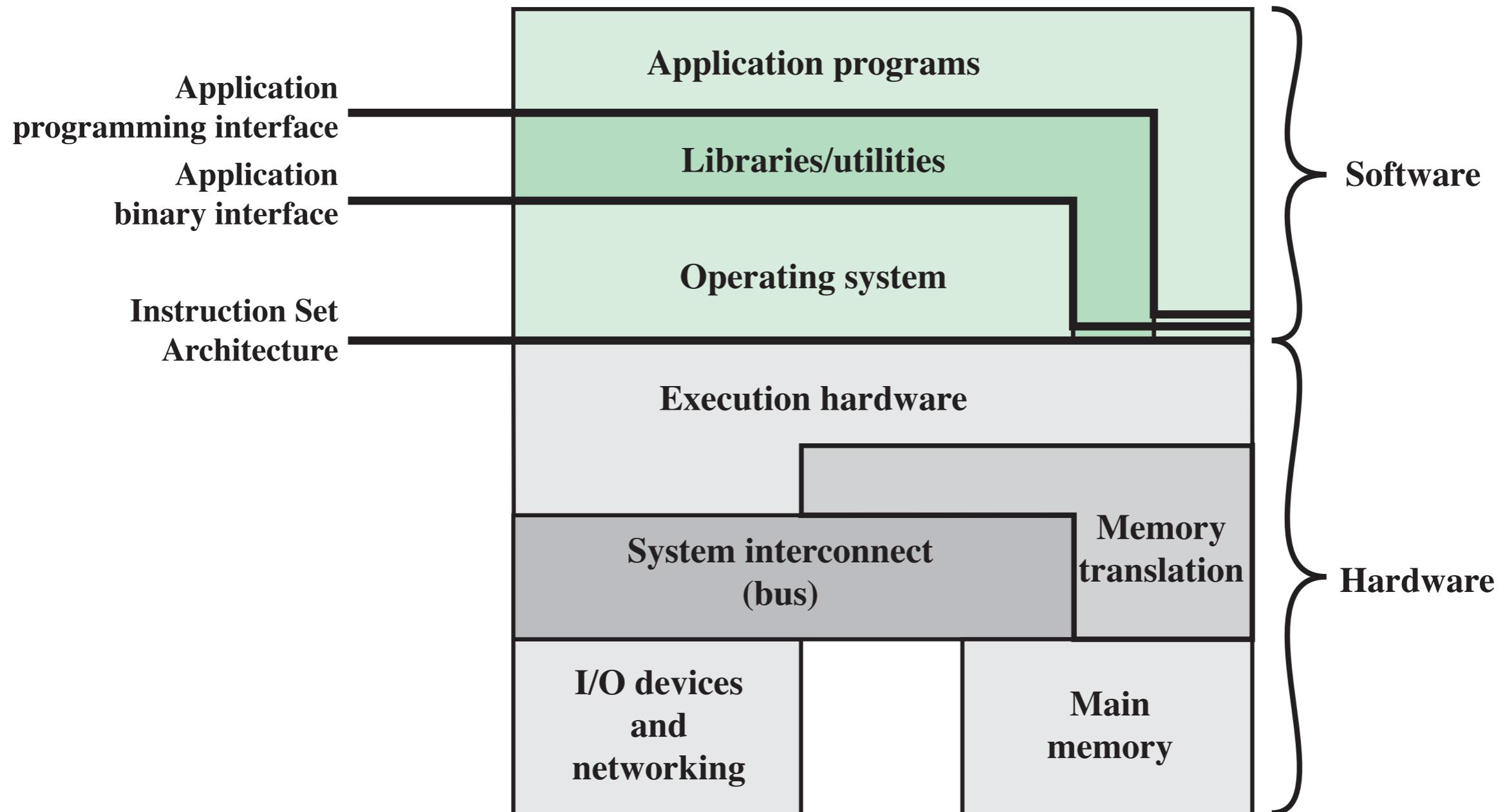
GPU Computing



Operating System

- What is an operating system?
 - A program that controls the execution of application programs
 - An interface between applications and hardware
- Operating System
 - Exploits the hardware resources of one or more processors
 - Provides a set of services to system users
 - Manages secondary memory and I/O devices

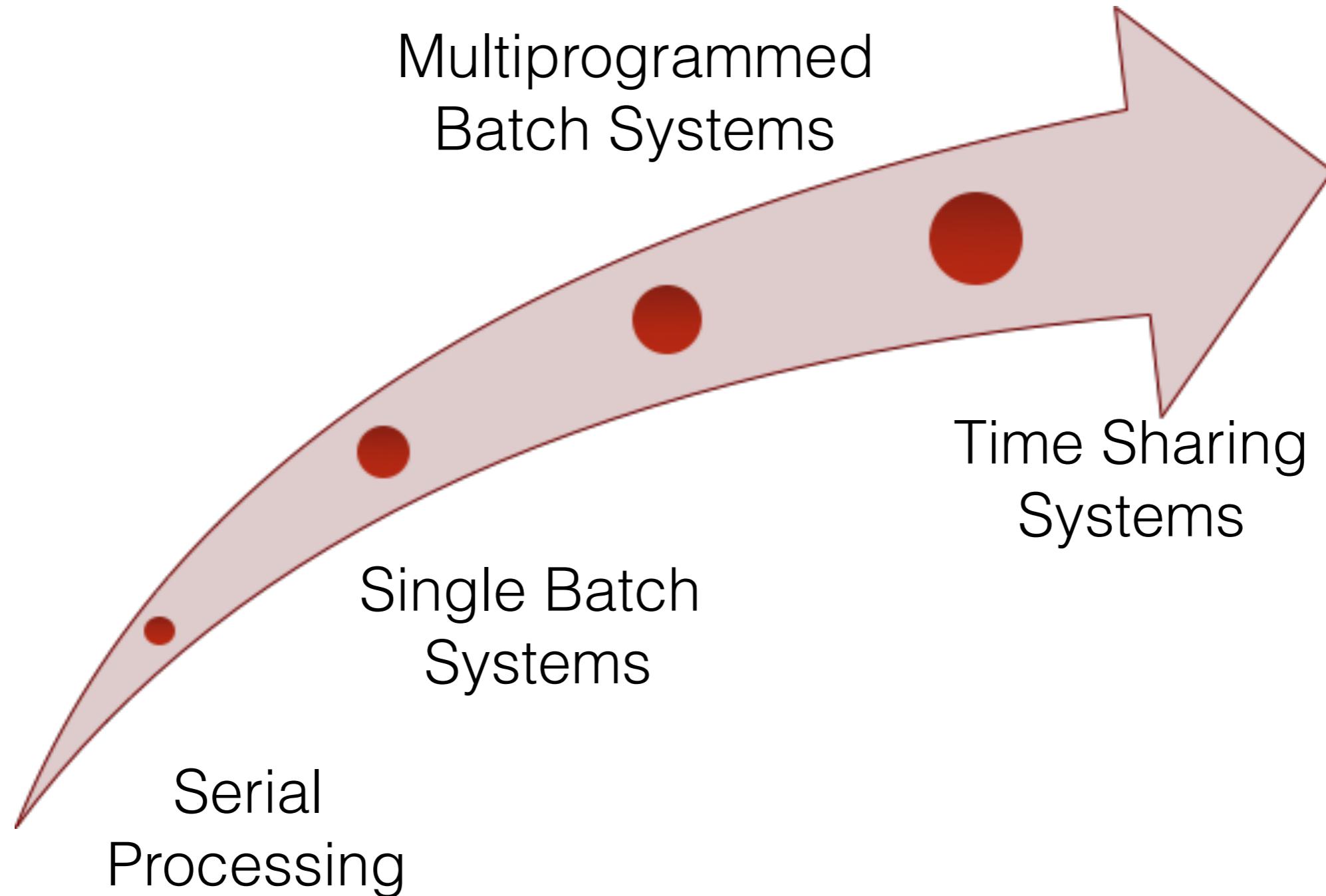
Computer Hardware and Software Structure



Operating System Services

- Program development
- Program execution
- Access I/O devices
- Controlled access to files
- System access
- Error detection and responds
- Accounting

Evolution of Operating Systems





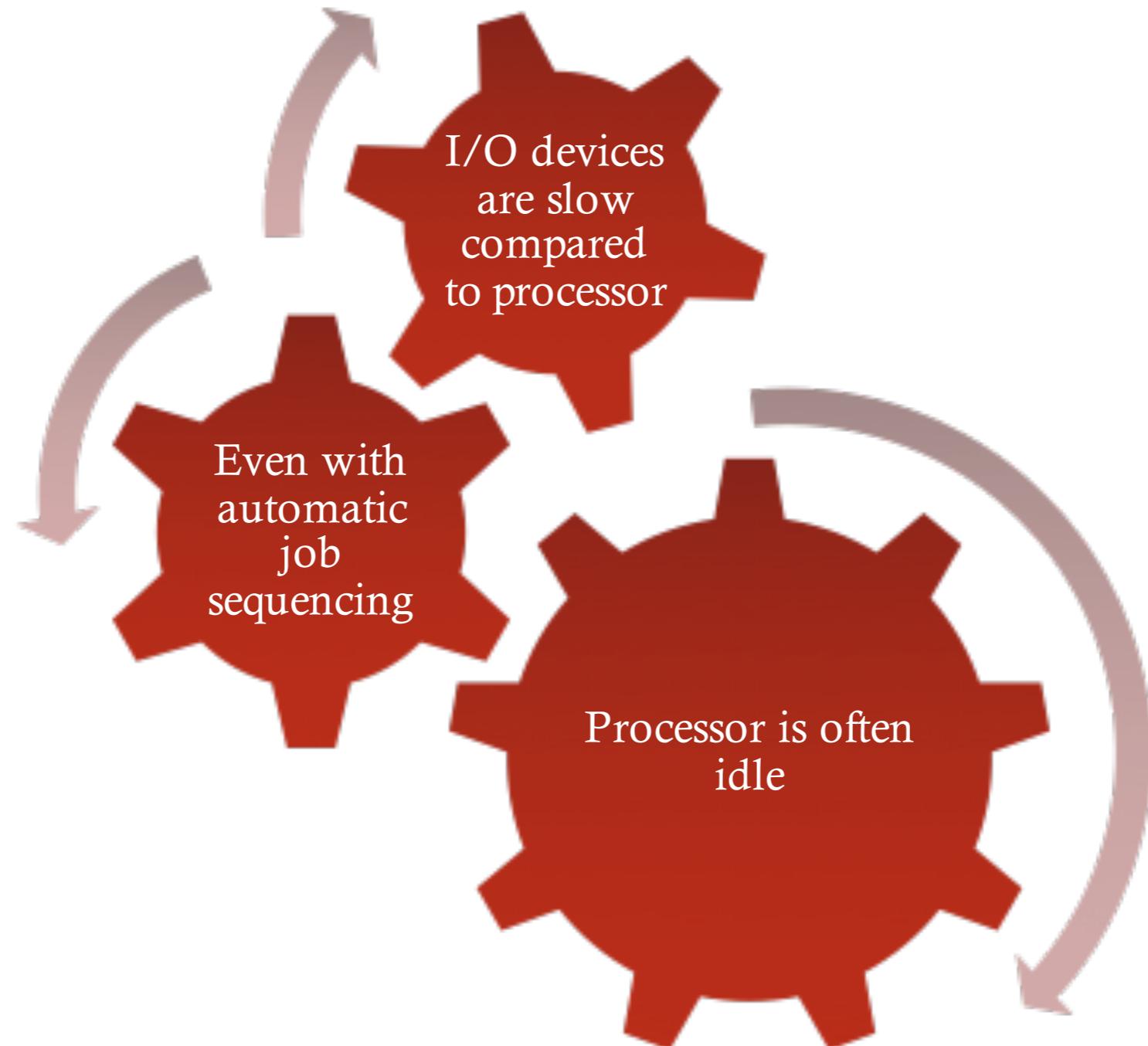
Serial Processing

- Earliest Computers:
 - No operating system
 - Programmers interacted directly with the computer hardware
 - Computers ran from a console with display lights, toggle switches, some form of input device, and a printer
 - Users have access to computer in "series"
- Problems:
 - Scheduling
 - Most installations used a hardcopy sign-up sheet to reserve computer time
 - Time allocations could run short or long, resulting in wasted computer time
 - Setup time
 - A considerable amount of time was spent on setting up the program to run

Simple Batch Systems

- Early computers were very expensive
 - Important to maximize processor utilization
- Monitor
 - User no longer has direct access to processor
 - Job is submitted to computer operator who batches them together and places them on an input device
 - Program branches back to the monitor when finished

Multiprogrammed Batch Systems



System Utilization Example

Read one record from file	$15 \mu\text{s}$
Execute 100 instructions	$1 \mu\text{s}$
Write one record to file	$\underline{15 \mu\text{s}}$
TOTAL	$31 \mu\text{s}$

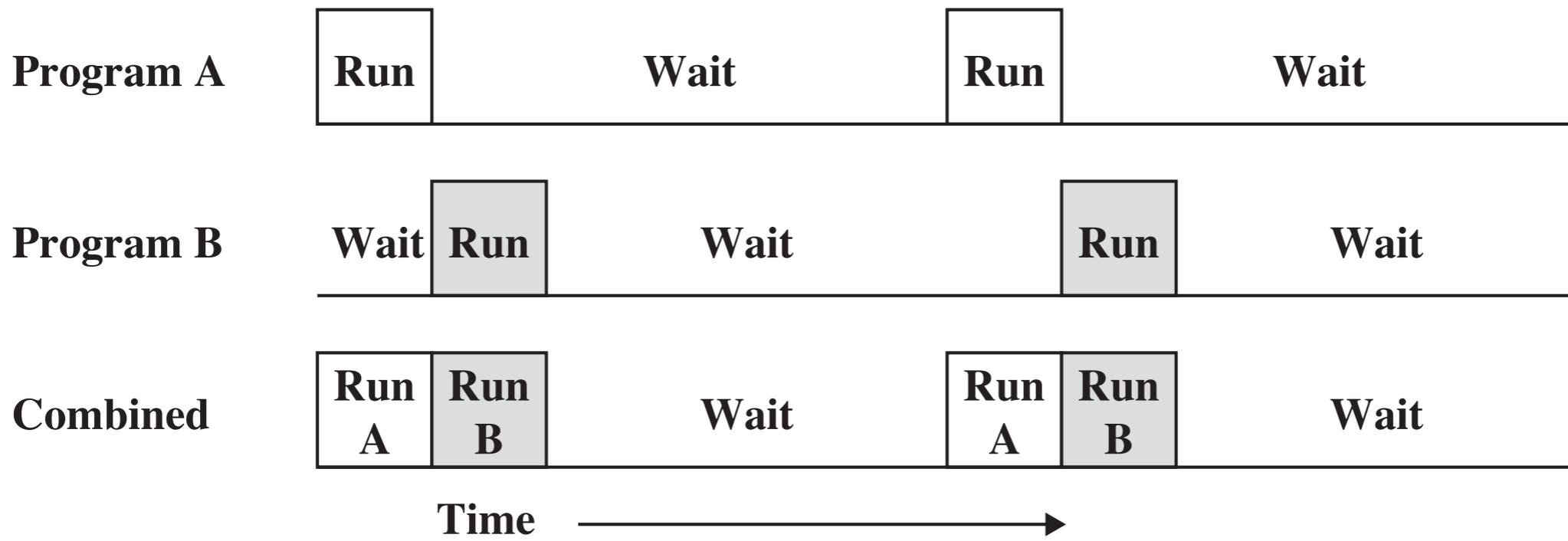
$$\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

Uniprogramming



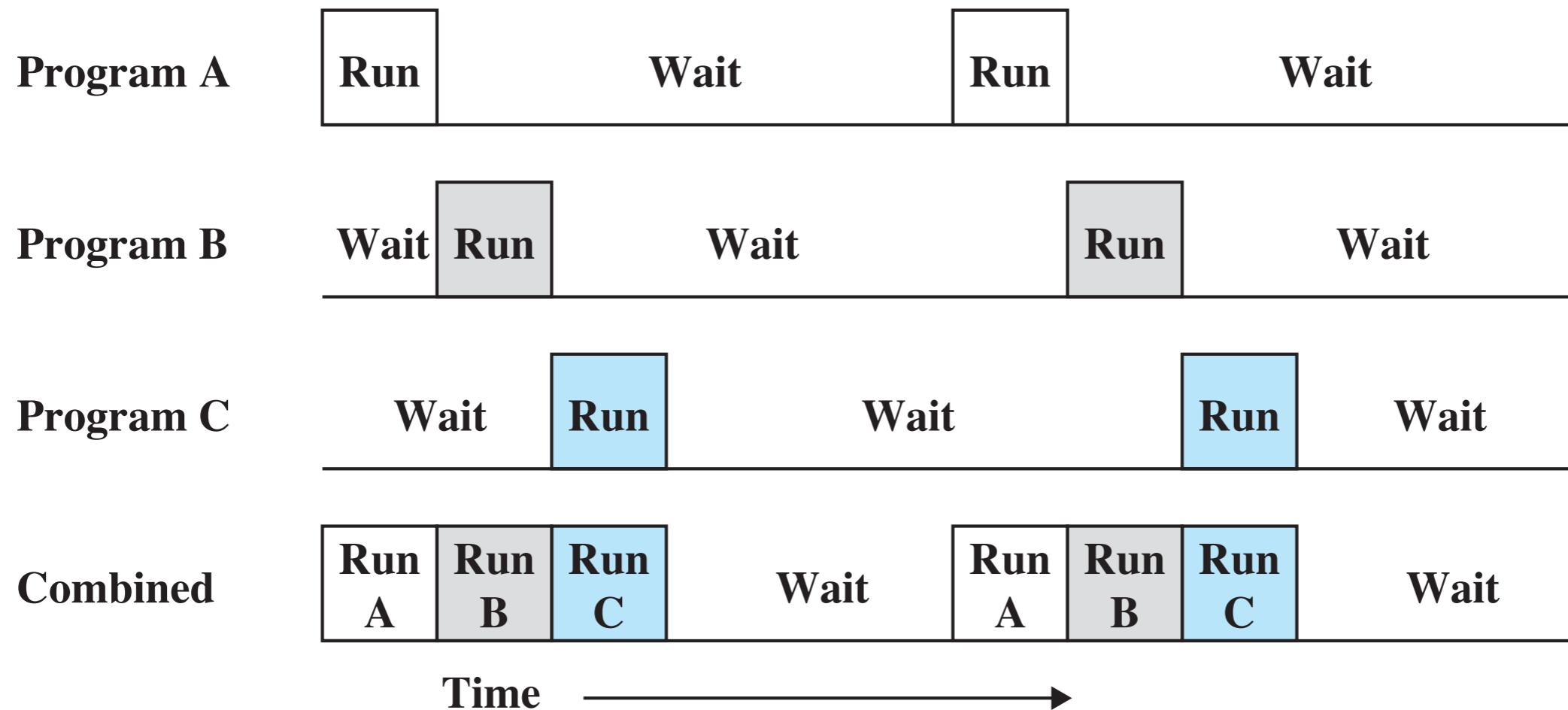
- The processor spends a certain amount of time executing, until it reaches an I/O instruction; it must then wait until that I/O instruction concludes before proceeding

Multiprogramming



- There must be enough memory to hold the OS (resident monitor) and one user program
- When one job needs to wait for I/O, the processor can switch to the other job, which is likely not waiting for I/O

Multiprogramming



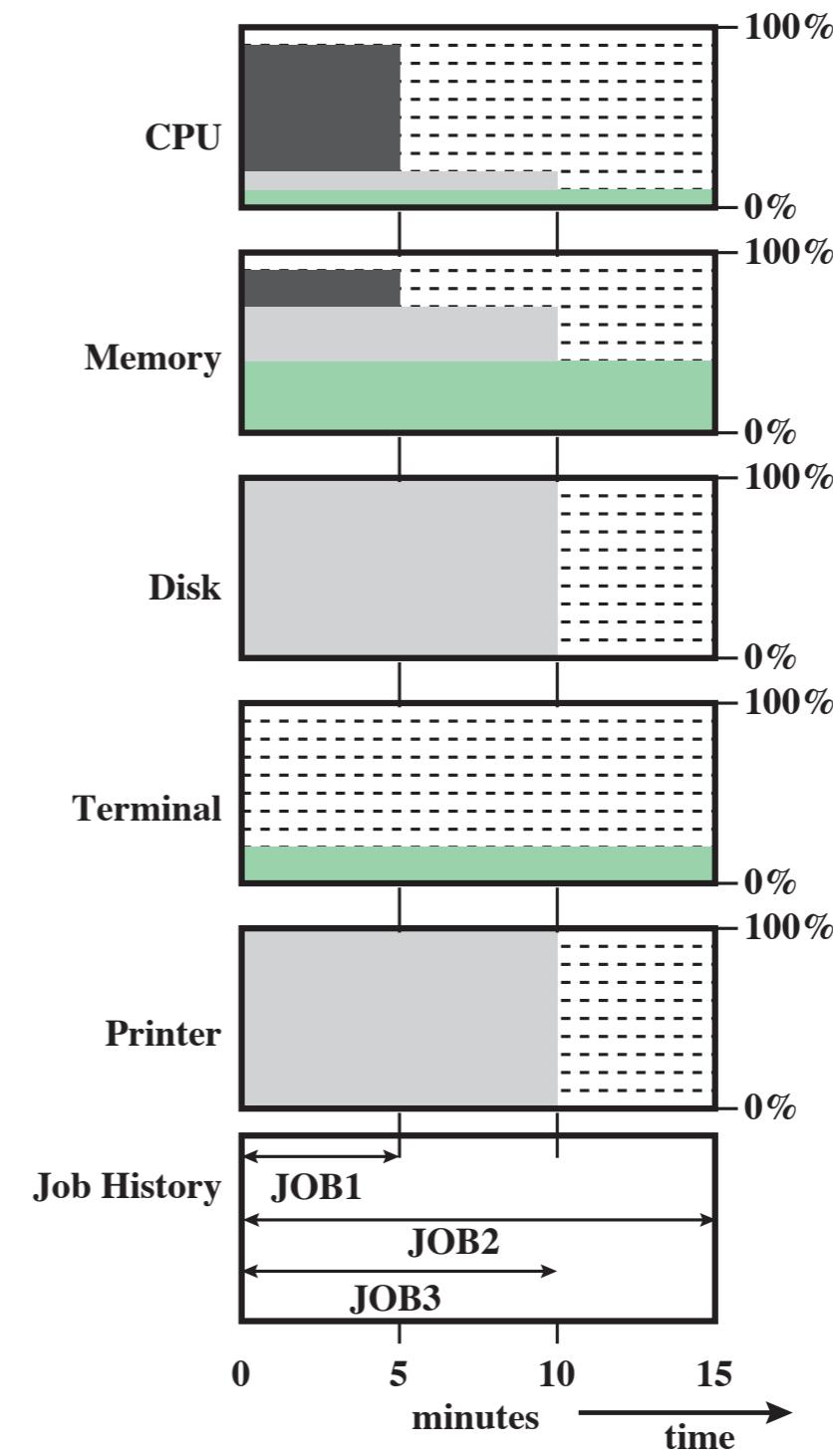
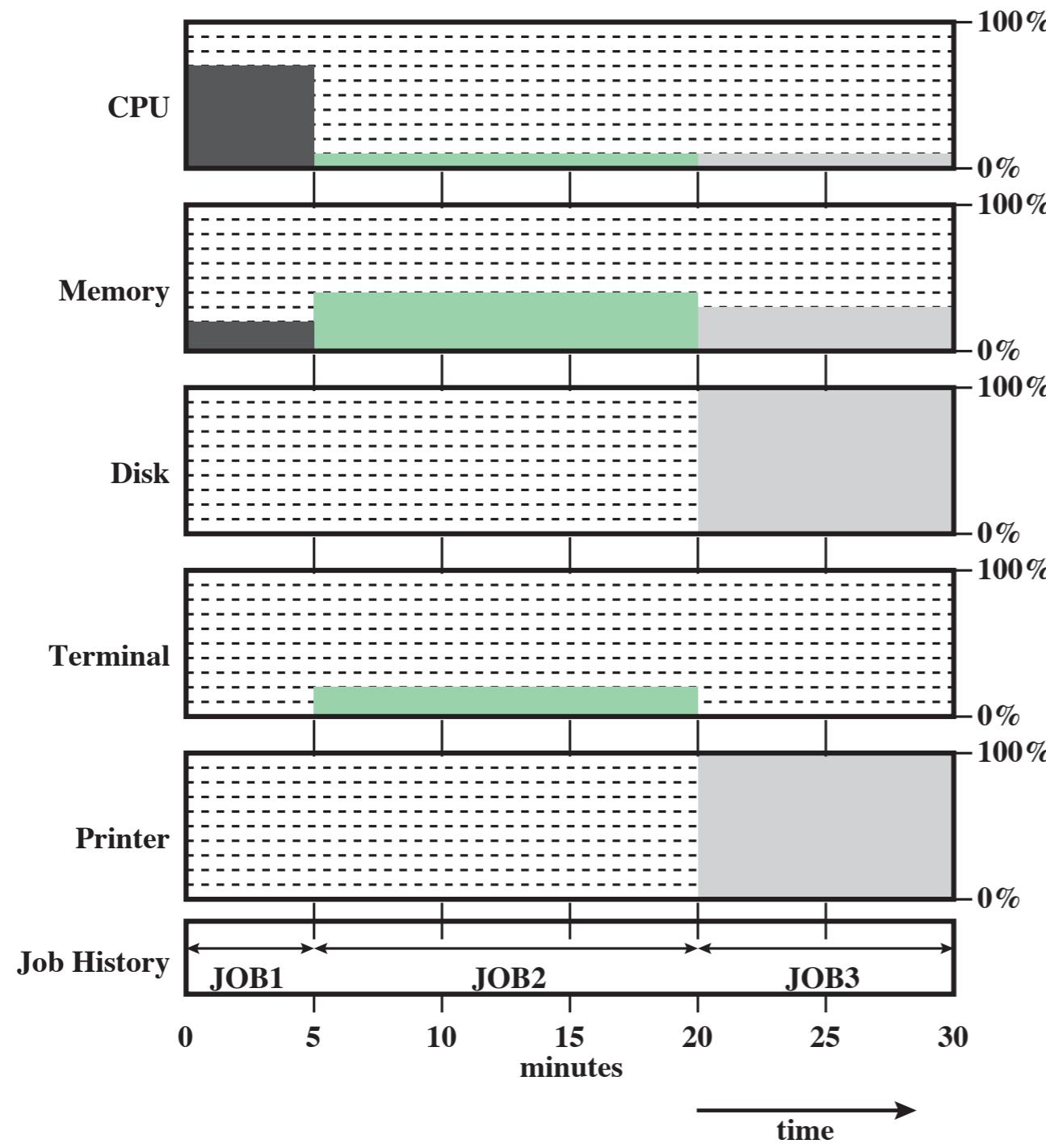
Multiprogramming Example

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

Effects of Multiprogramming on Resource Utilization

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

Utilization Histograms



Time-Sharing Systems

- Can be used to handle multiple interactive jobs
- Processor time is shared among multiple users
- Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation

Major Achievements

- Operating Systems are among the most complex pieces of software ever developed
- Major advances in development include:
 - Processes
 - Memory management
 - Information protection and security
 - Scheduling and resource management

Process

- Fundamental to the structure of operating system

A *process* can be defined as:

A program in execution

An instance of a running program

The entity that can be assigned to, and executed on, a processor

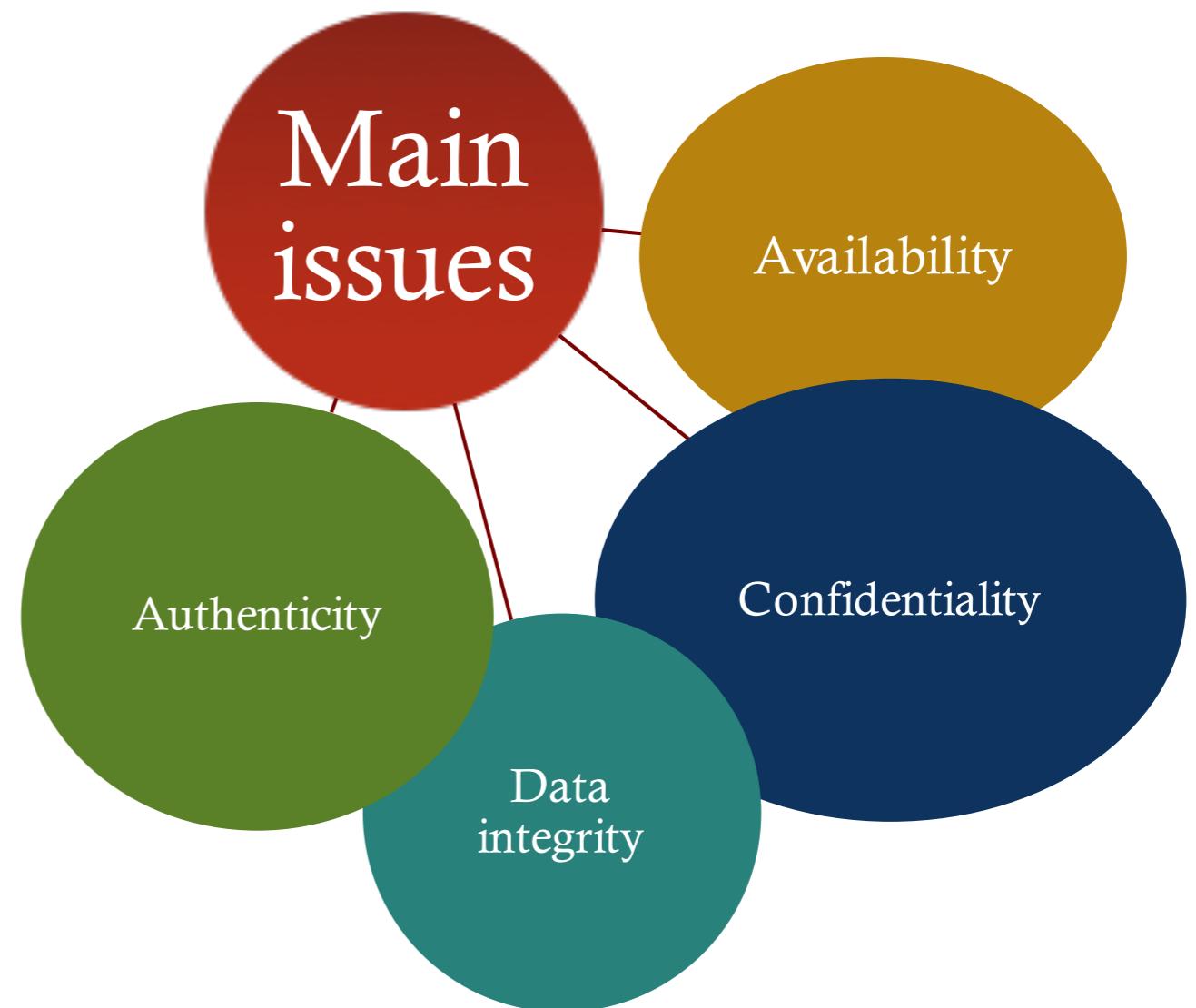
A unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources

Memory Management

- The OS has five principal storage management responsibilities:
 - Process isolation
 - Automatic allocation and management
 - Support of modular programming
 - Protection and access control
 - Long-term storage

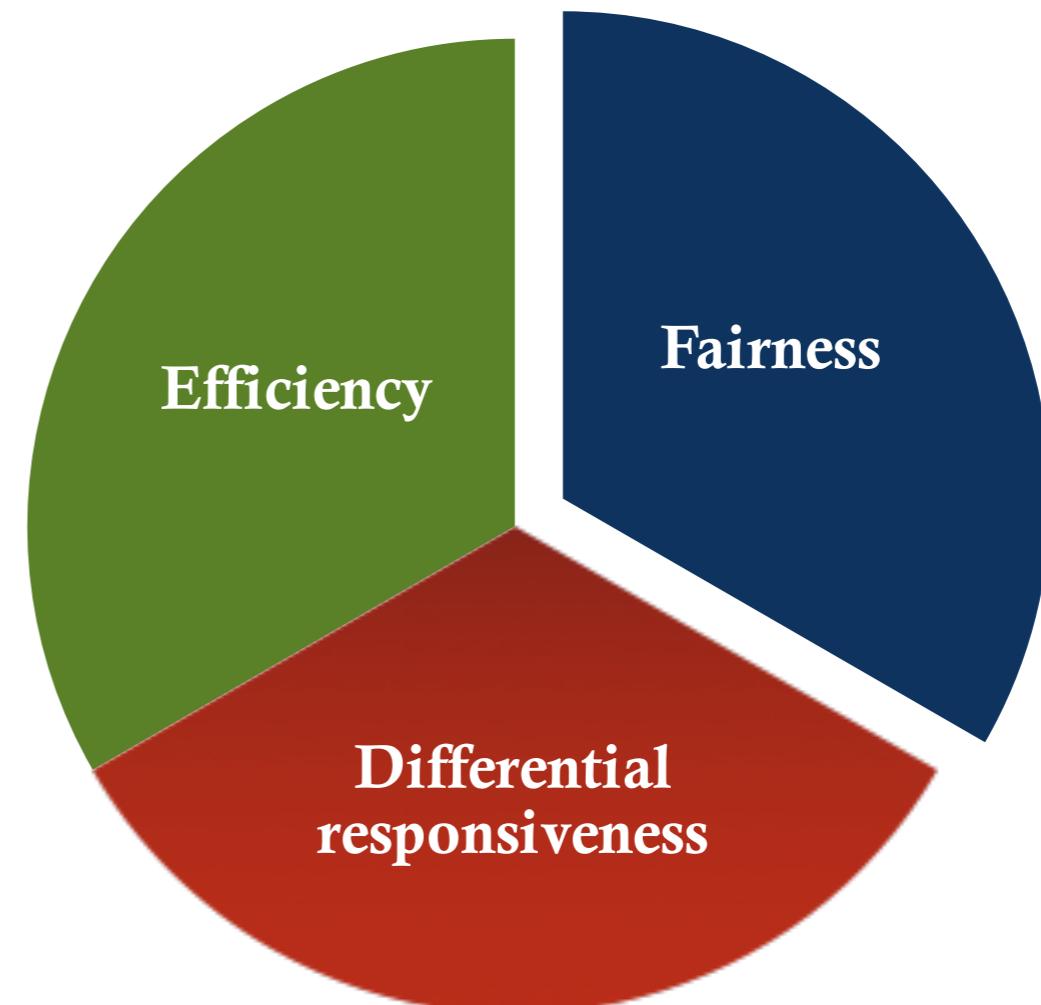
Information Protection and Security

- The nature of the threat that concerns an organization will vary greatly depending on the circumstances
- The problem involves controlling access to computer systems and the information stored in them

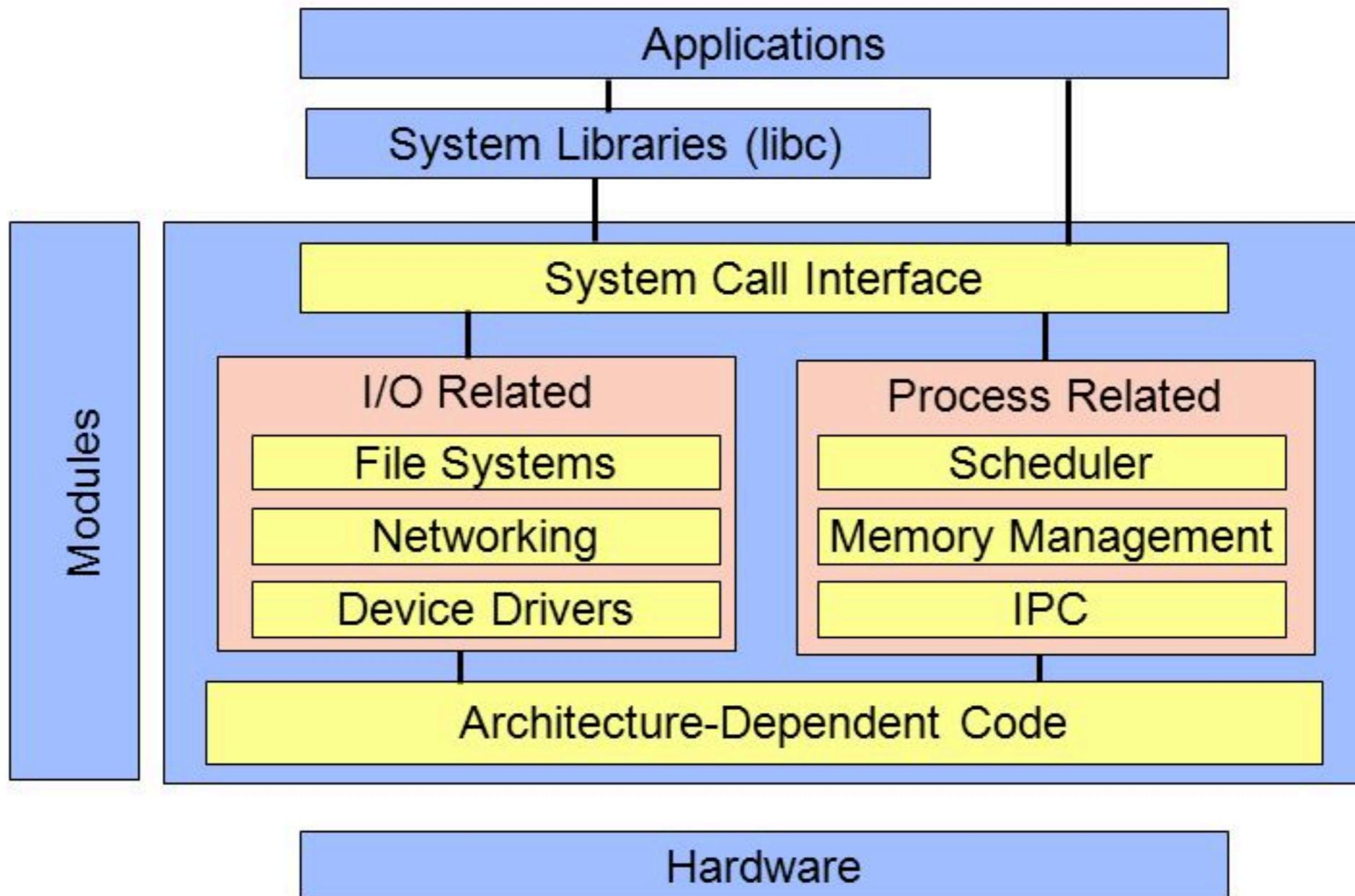


Scheduling and Resource Management

- Key responsibility of an OS is managing resources
- Resource allocation policies must consider:



Linux Architecture



Dual-Mode Operation

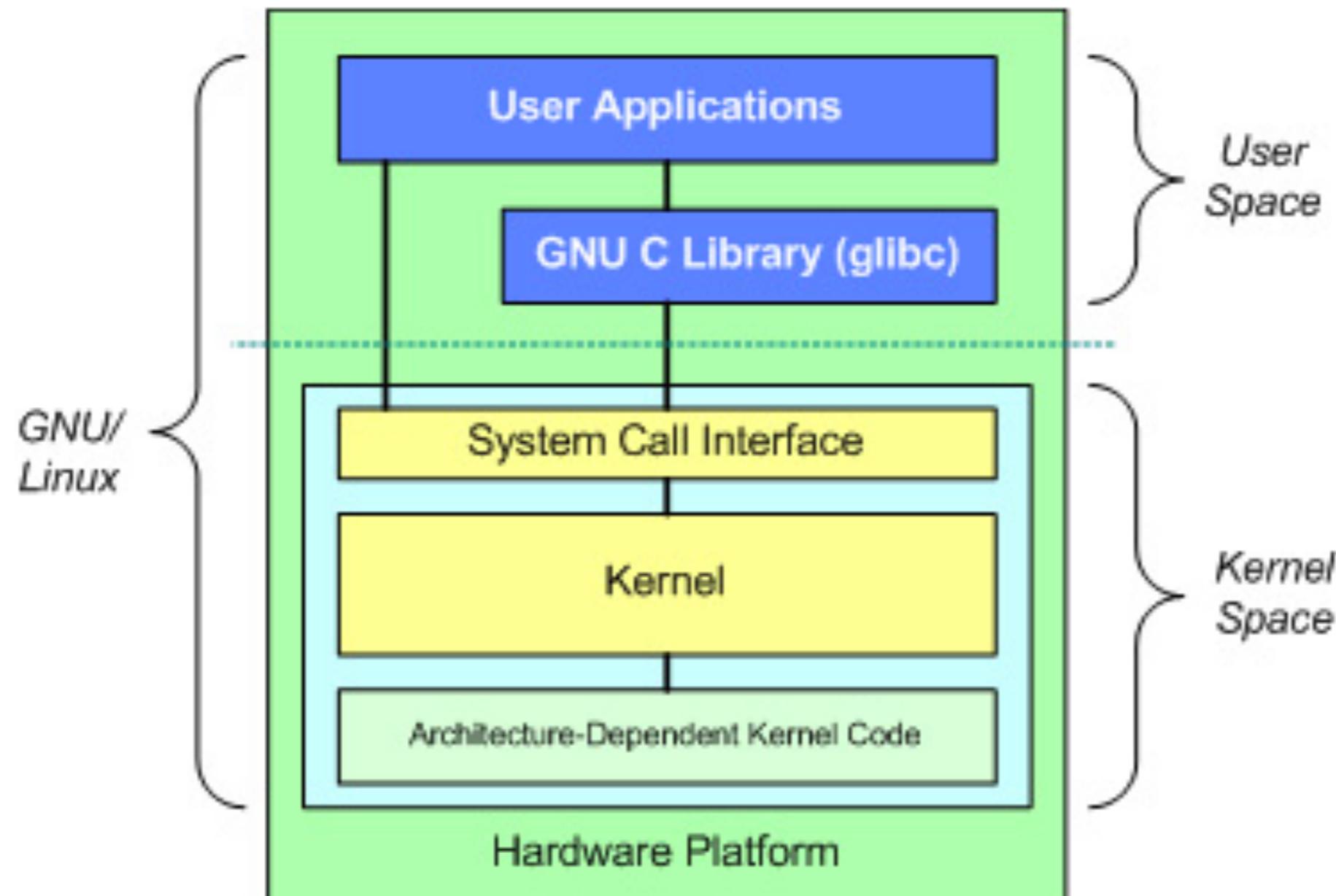
User Mode

- User program executes in user mode
- Certain areas of memory are protected from user access
- Certain instructions may not be executed

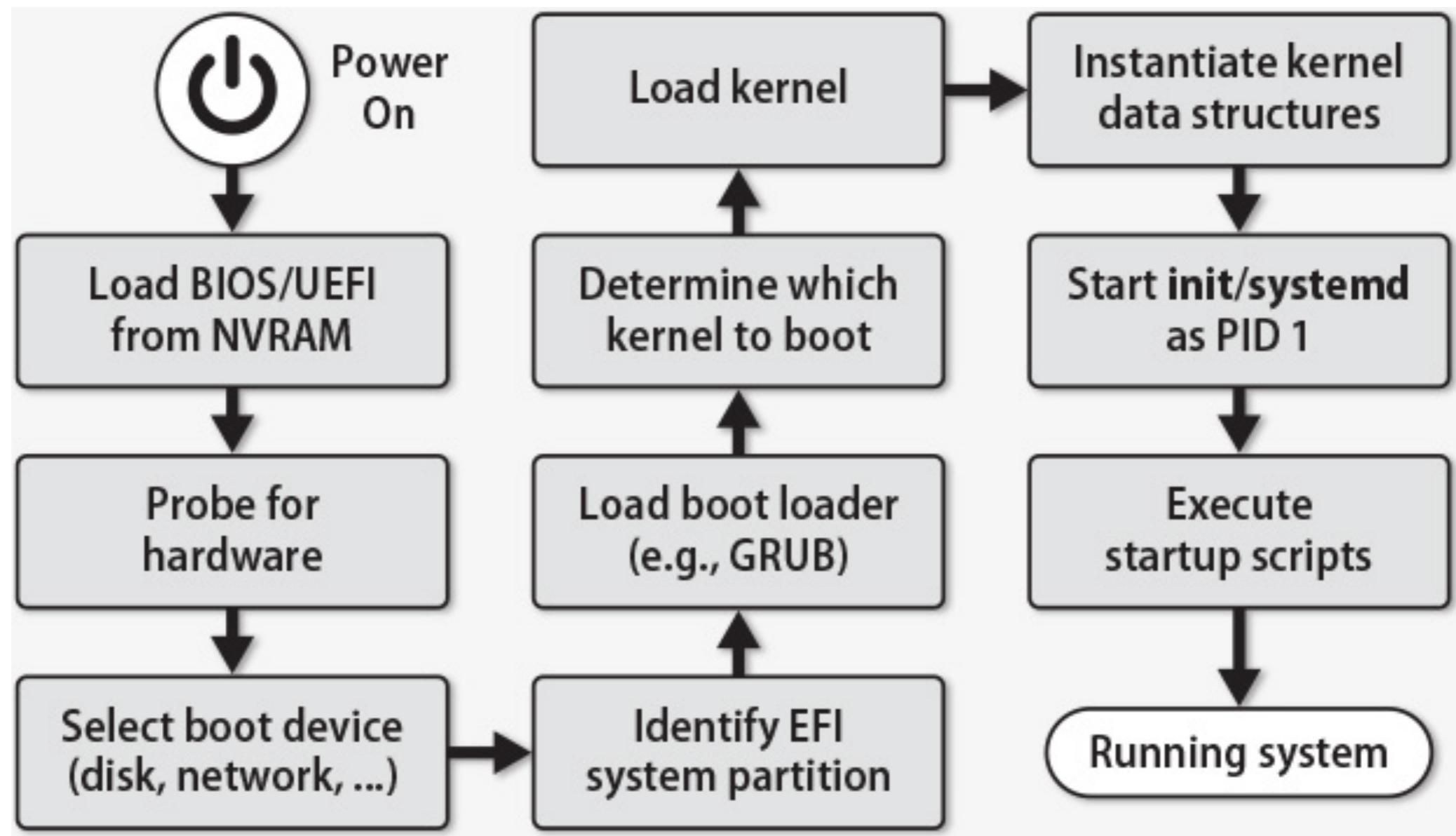
Kernel Mode

- Monitor executes in kernel mode
- Privileged instructions may be executed
- Protected areas of memory may be accessed

Linux Dual-Mode Operation



Linux Basics – Booting



Linux/UNIX booting process

Reboot and Shutdown

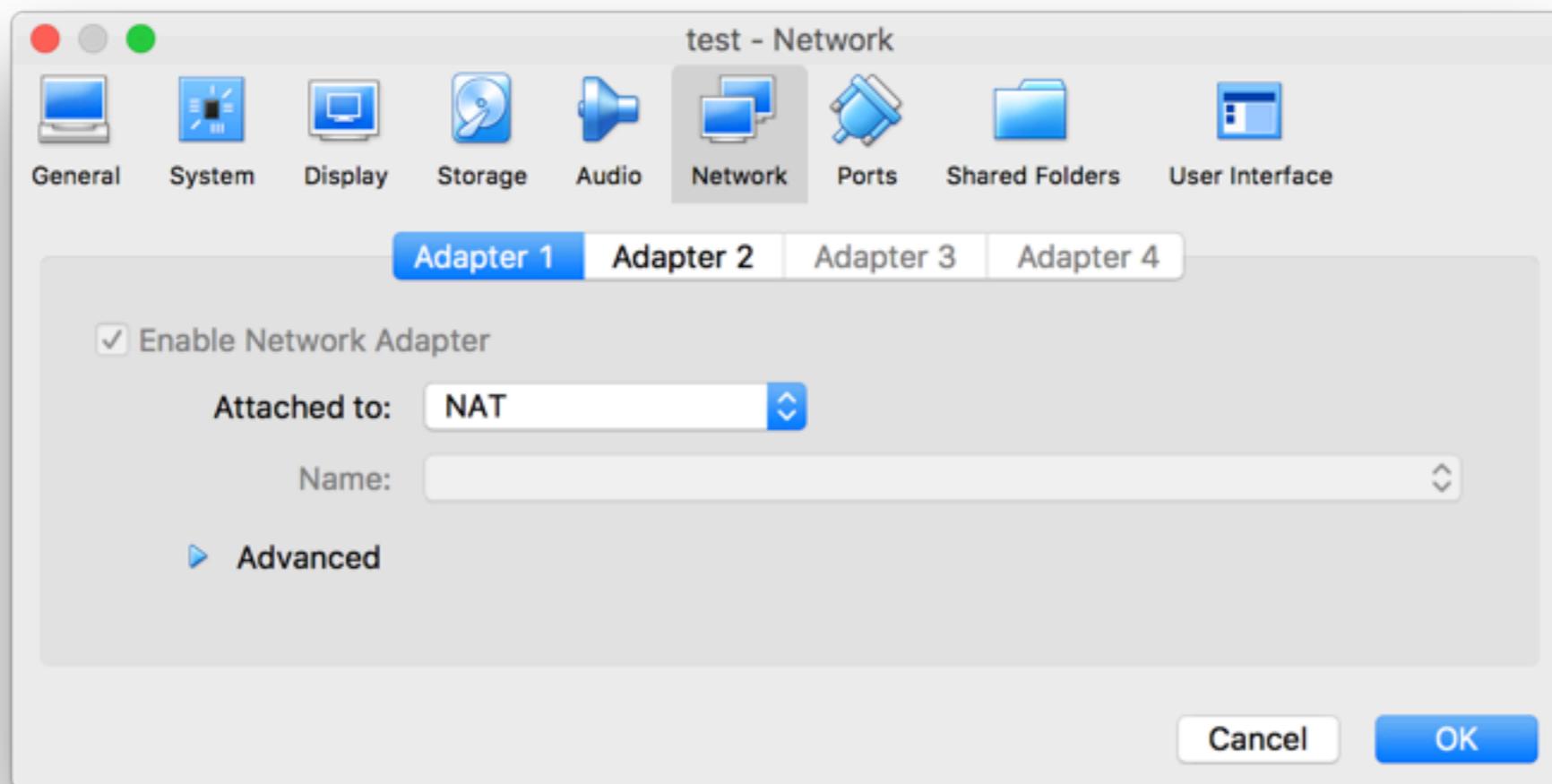
- **halt**: performs the essential duties required for shutting down the system
 - logs the shutdown
 - kills nonessential processes
 - flushes cached filesystem blocks to disk
 - halts the kernel
- **reboot**: identical to halt, but it causes the machine to reboot instead of halting
- **shutdown**:
 - provides for scheduled shutdowns
 - ominous warnings to logged-in users
- **poweroff**: shutdown the system immediately
 - you need root privilege to shutdown or reboot the machine

Linux commands basics

- Syntax of Linux commands:
 - command [option] [parameter]
 - case sensitive
 - use '-' before single letter option
 - [root@localhost ~]# ls -a -l
 - single letter options can be combined together
 - [root@localhost ~]# ls -al
 - use '- -' before word option
 - [root@localhost ~]# ls --help

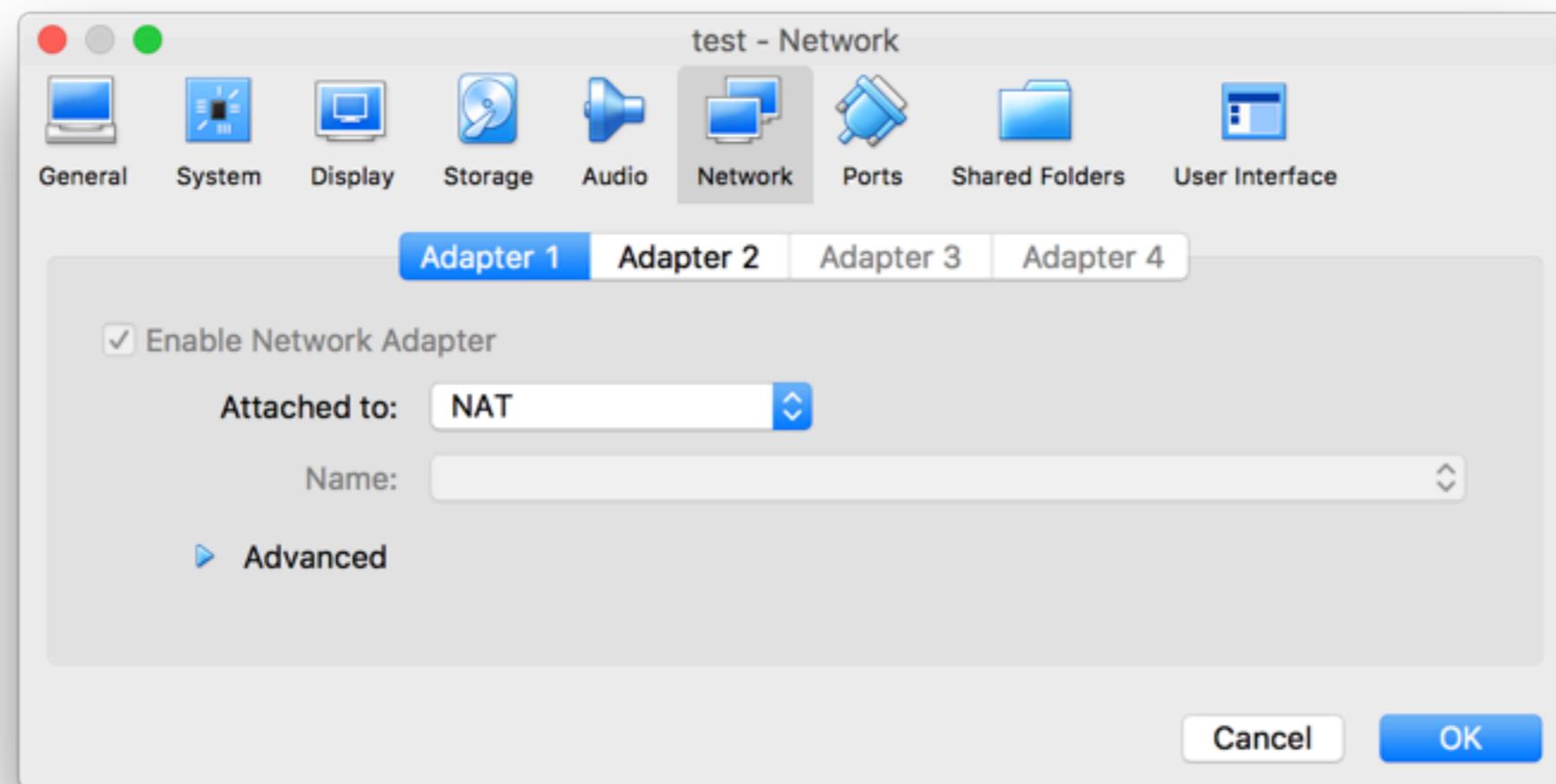
Remote access

- You can remotely access your virtual machine from network
- First, you need to connect your VM to network (WiFi)
- Go to your VM setting, select Network:



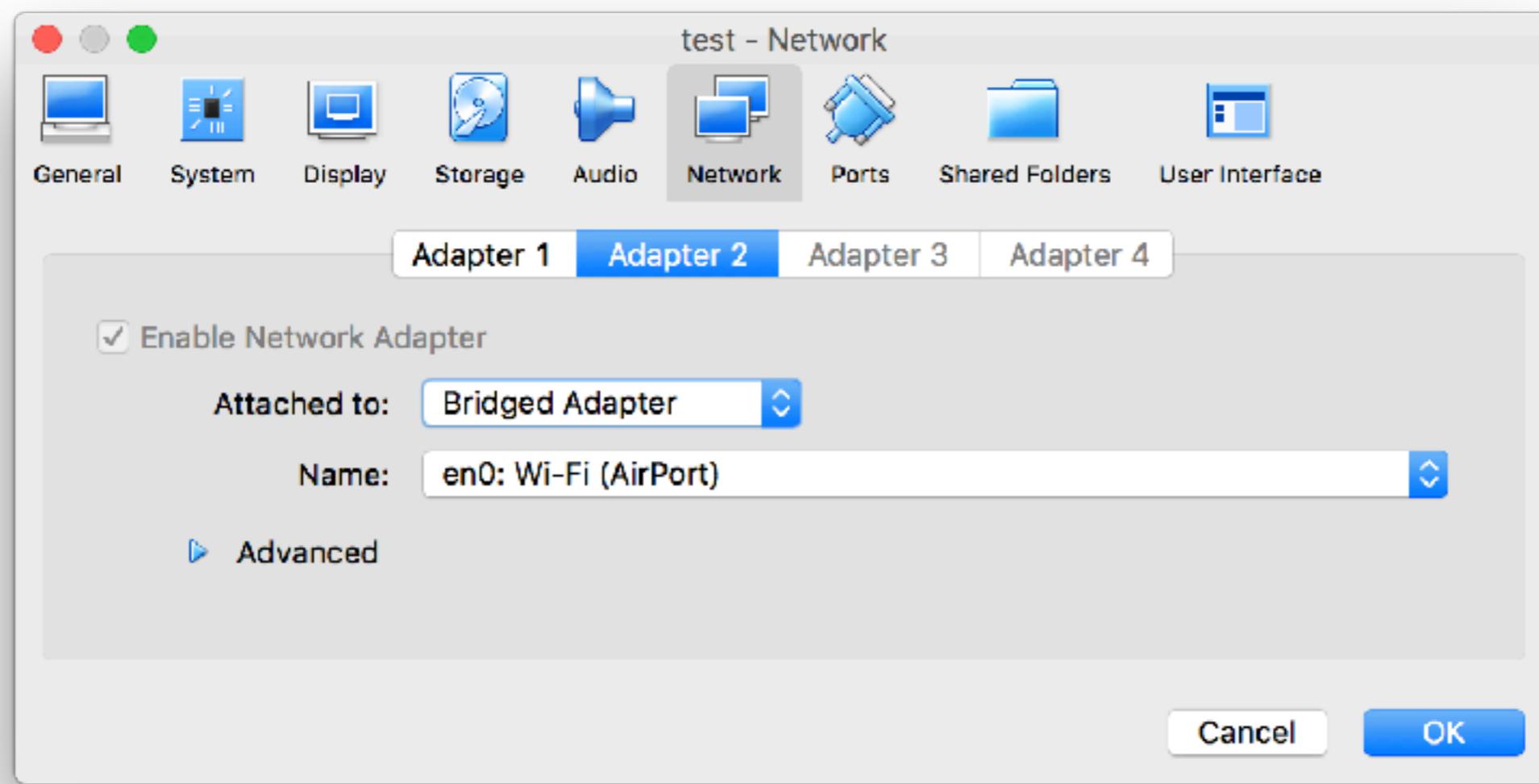
Remote access

- Select Adapter 1
- Attached to: NAT



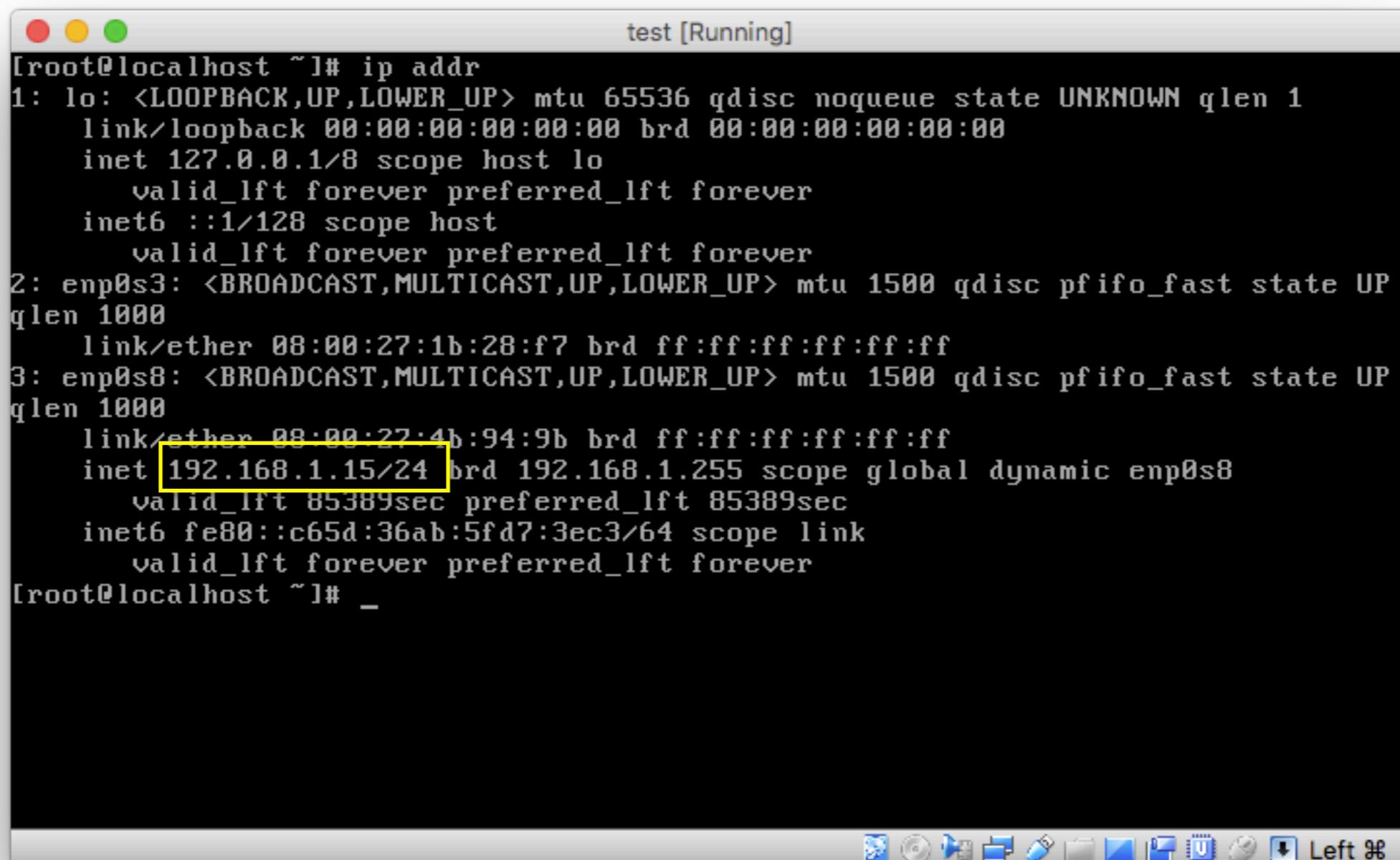
Remote access

- Select Adapter 2
- Check "Enable Network Adapter"
- Attached to: Bridged Adapter
- Name is your WiFi network card



Remote access

- Reboot your VM and login into your system
- use "ip addr" to find your VM's IP Address



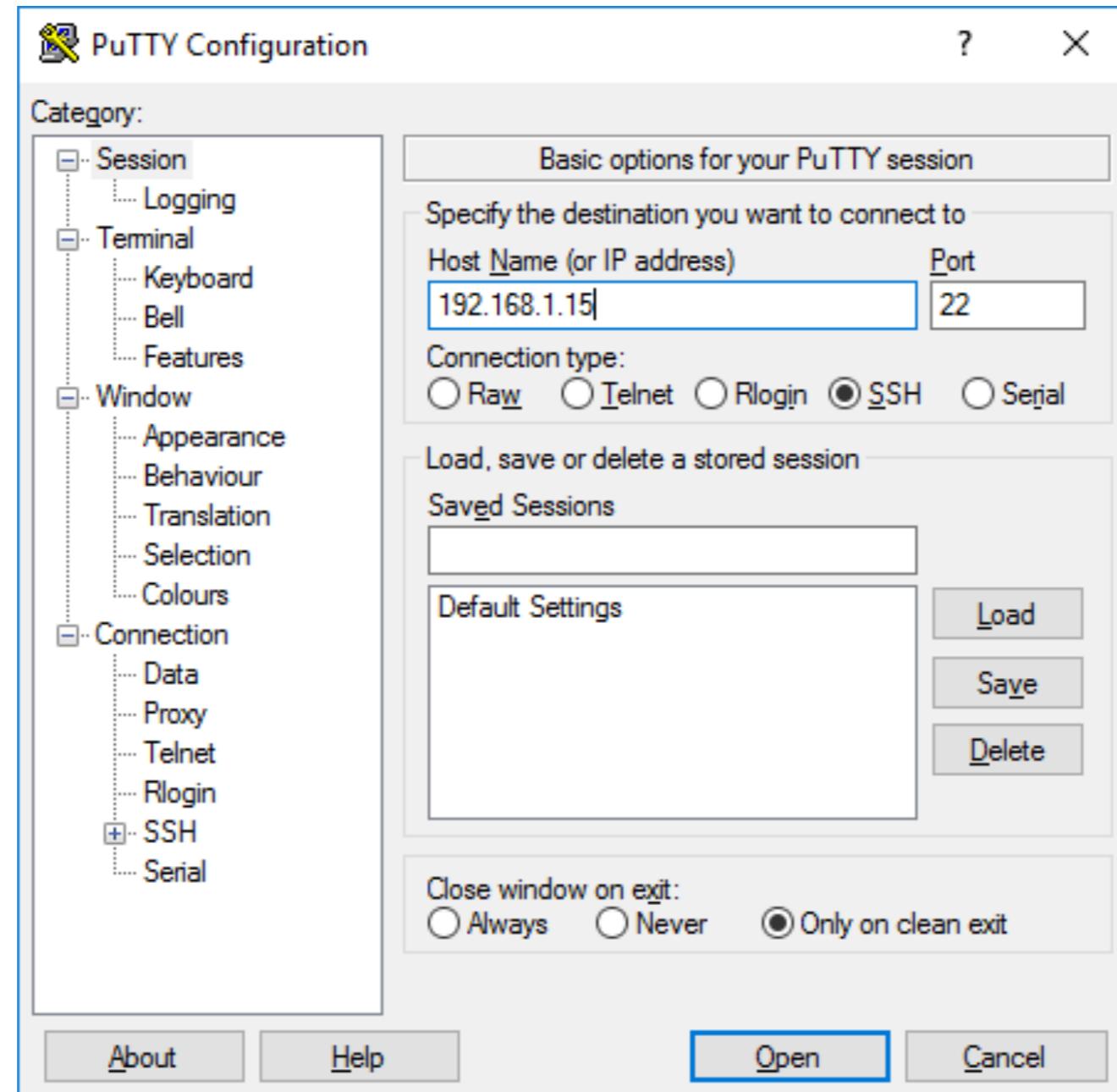
```
[root@localhost ~]# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
qlen 1000
    link/ether 08:00:27:1b:28:f7 brd ff:ff:ff:ff:ff:ff
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
qlen 1000
    link/ether 08:00:27:4b:94:9b brd ff:ff:ff:ff:ff:ff
        inet 192.168.1.15/24 brd 192.168.1.255 scope global dynamic enp0s8
            valid_lft 85389sec preferred_lft 85389sec
        inet6 fe80::c65d:36ab:5fd7:3ec3/64 scope link
            valid_lft forever preferred_lft forever
[root@localhost ~]# _
```

Remote access from your host machine

- If you are using MacOS:
 - open your terminal (LaunchPad → Other → Terminal)
- If you are using Windows OS:
 - download PuTTY: www.putty.org

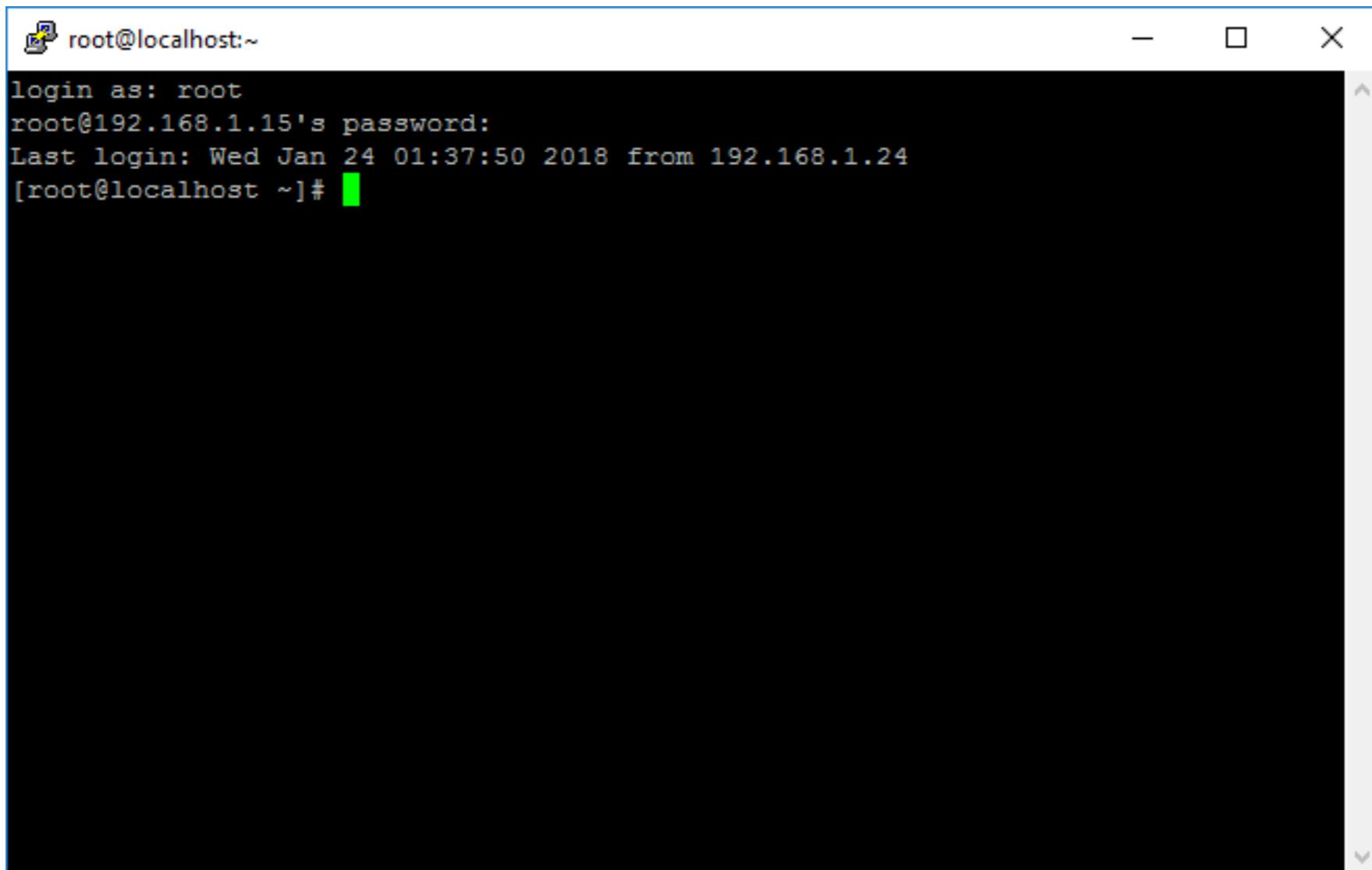
Remote access from your host machine (Windows)

- Open PuTTY
- Enter your VM's IP Address



Remote access from your host machine (Windows)

- Open PuTTY
- Enter your VM's IP Address
- Enter your user name (root) and passwd



The screenshot shows a PuTTY terminal window with a black background and white text. The title bar reads "root@localhost:~". The session log shows the following text:

```
login as: root
root@192.168.1.15's password:
Last login: Wed Jan 24 01:37:50 2018 from 192.168.1.24
[root@localhost ~]#
```

Remote access from your host machine (MacOS Terminal)

- use the following command:

```
Haos-MacBook-Pro:Scripts hao$ ssh -l root 192.168.1.15
```

- Replace the IP address for your VM

Execute command in different lines

- Use '\' to separate command to multiple lines:
- Be careful with the spaces

```
[root@localhost ~]# ls \
> -a \
> -l
```



Execute multiple commands in one line

- You can execute multiple commands in one line.
- Separate them with ';'

```
[root@localhost ~]# echo "What is your name?";echo "My name is Hao"  
What is your name?  
My name is Hao
```

help

- You can use 'help' command to review the Bash Shell information

```
[root@localhost ~]# help
GNU bash, version 4.2.46(2)-release (x86_64-redhat-linux-gnu)
These shell commands are defined internally. Type `help' to see this list.
Type `help name' to find out more about the function `name'.
Use `info bash' to find out more about the shell in general.
Use `man -k' or `info' to find out more about commands not in this list.
```

- You can use 'help [command]' to check the usage of a command

```
[root@localhost ~]# help cd
cd: cd [-L | [-P [-e]]] [dir]
      Change the shell working directory.
```

Change the current directory to DIR. The default DIR is the value of the HOME shell variable.

help Option

- All build-in commands have help option for usage information
- Using the following syntax:
`<command> --help`

```
[root@localhost ~]# ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.
```

Manual Pages

- The manual pages are a set of pages that explain every command available on your system including what they do, the specifics of how you run them and what command line arguments they accept.
- Syntax:
 - `man <command to look up>`

info

- info reads documentation in the info format
- Info is similar to man, with a more robust structure for linking pages together. Info pages are made using the texinfo tools, and can link with other pages, create menus and ease navigation in general.
- Syntax:
 - `info <command name>`

Bash Shell Operation

- Command-line completion:
 - Command completion:
 - the program automatically fills in partially typed commands.
 - Enter "Tab" to perform auto-completion
 - Enter "Tab" twice to list all the commands with the same partially typed commands

```
[root@localhost ~]# if
if           ifcfg        ifdown      ifenslave   ifstat      ifup
```

- File name completion

Command history

- The Bash Shell records the commands you used
- Use the up and down key to scroll through previously typed commands. Press [Enter] to execute them or use the left and right arrow keys to edit the command first.
- You can use 'history' command to lookup stored command history

```
[root@localhost ~]# history
 1 shutdown --help
 2 ssh
 3 ifconfig
 4 ifconfig -a
 5 cd /etc
 6 ls
 7 cd sysconfig/
```

Command history

- You can use the history command by:
 ! <num of command>

```
[root@localhost ~]# history
 1 shutdown --help
 2 ssh
 3 ifconfig
 4 ifconfig -a
 5 cd /etc
 6 ls
 7 cd sysconfig/
```

```
[root@localhost ~]# !5
cd /etc
[root@localhost etc]#
```

alias

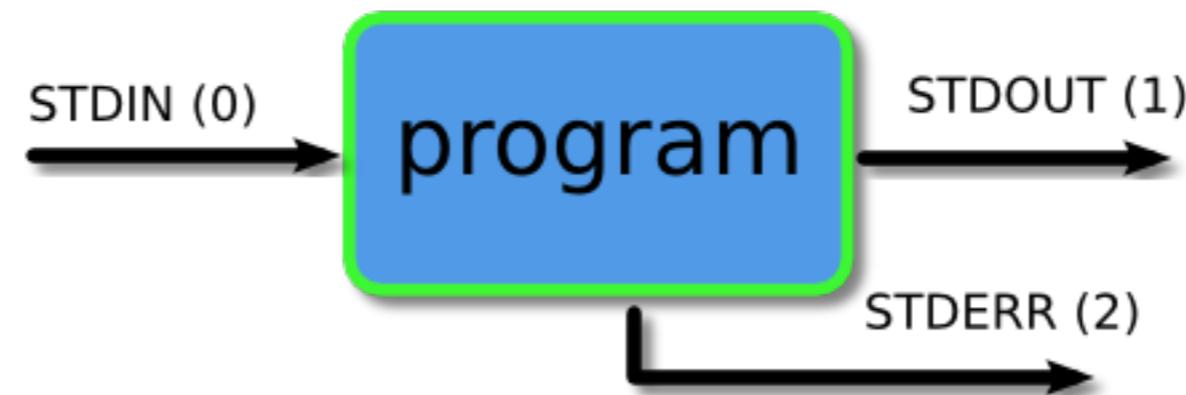
- alias:
 - an abbreviation
 - avoid typing a long command sequence repeatedly
 - Syntax:
 - `alias <abbr> = '<command>'`

```
[root@localhost etc]# alias ls='ls -a -l | more'
```

- Cancel alias:
 - Syntax:
 - `unalias <abbr>`

Piping and Redirection

- Every program running on the command line has three data streams connected to it:
 - STDIN (0) - Standard input (data fed into the program)
 - STDOUT (1) - Standard output (data printed by the program, defaults to the terminal)
 - STDERR (2) - Standard error (for error messages, also defaults to the terminal)



- We can connect these streams between programs and files

Redirecting to a File

- Normally, we will get our output on the screen
- We may save the output to a file instead of print it to the screen using '>'
- If the file exists, using '>' overwrites the file

```
[root@localhost ~]# ls  
anaconda-ks.cfg  dir.txt  
[root@localhost ~]# ls > dir.txt  
[root@localhost ~]# cat dir.txt  
anaconda-ks.cfg  
dir.txt
```

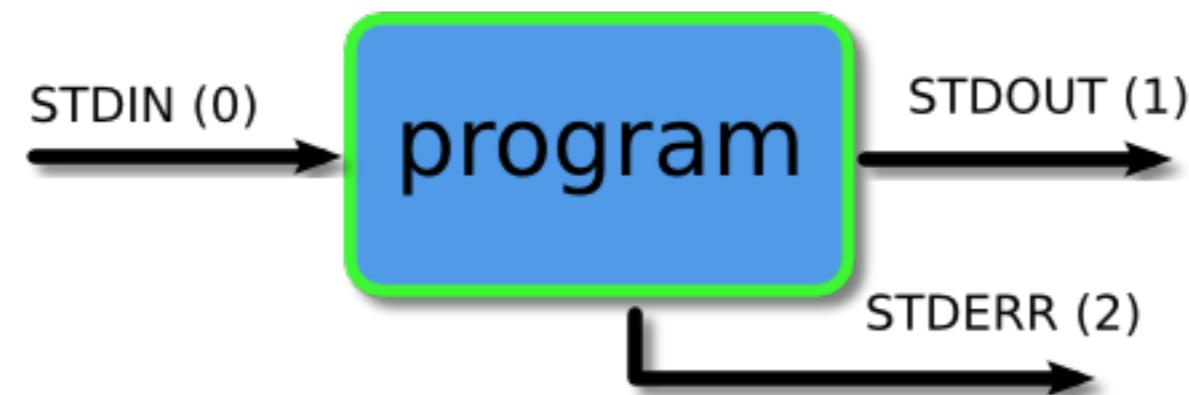
Redirecting to a File

- We can use '>>' to attach the existing file

```
[root@localhost ~]# ls  
anaconda-ks.cfg  dir.txt  
[root@localhost ~]# ls > dir.txt  
[root@localhost ~]# cat dir.txt  
anaconda-ks.cfg  
dir.txt  
[root@localhost ~]# ls >> dir.txt  
[root@localhost ~]# cat dir.txt  
anaconda-ks.cfg  
dir.txt  
anaconda-ks.cfg  
dir.txt
```

Redirecting STDERR

- The three streams have numbers associated with them
- STDERR is stream number 2 and we may use these numbers to identify the streams.
- If we place a number before the > operator then it will redirect that stream



Redirecting STDERR

- The three streams have numbers associated with them
- STDERR is stream number 2 and we may use these numbers to identify the streams.
- If we place a number before the > operator then it will redirect that stream

```
[root@localhost ~]# ls 2>err.txt  
anaconda-ks.cfg  dir.txt  err.txt  
[root@localhost ~]# cat err.txt  
[root@localhost ~]# ls e  
ls: cannot access e: No such file or directory  
[root@localhost ~]# ls e 2>err.txt  
[root@localhost ~]# cat err.txt  
ls: cannot access e: No such file or directory
```

Piping

- With redirection, we can send data to files
- We can also send data from one program to another
- pipe:
 - use 'l' between commands to send the data over

```
[root@localhost ~]# ls  
anaconda-ks.cfg  dir.txt  err.txt  
[root@localhost ~]# ls | head -2  
anaconda-ks.cfg  
dir.txt  
[root@localhost ~]# ls | head -2 |tail -1  
dir.txt
```



CSC424 System Administration

Instructor: Dr. Hao Wu

Week 3 File System

Filesystem

- Two definitions:
 - the entire hierarchy of directories (also referred to as the directory tree) that is used to organize files on a computer system.
 - refers to type of filesystem: how the storage of data is organized on a computer disk or on a partition on a hard disk. Each type of filesystem has its own set of rules for controlling the allocation of disk space to files and for associating data about each file with that file.

Linux Filesystem

- **"On a UNIX system, everything is a file; if something is not a file, it is a process."**

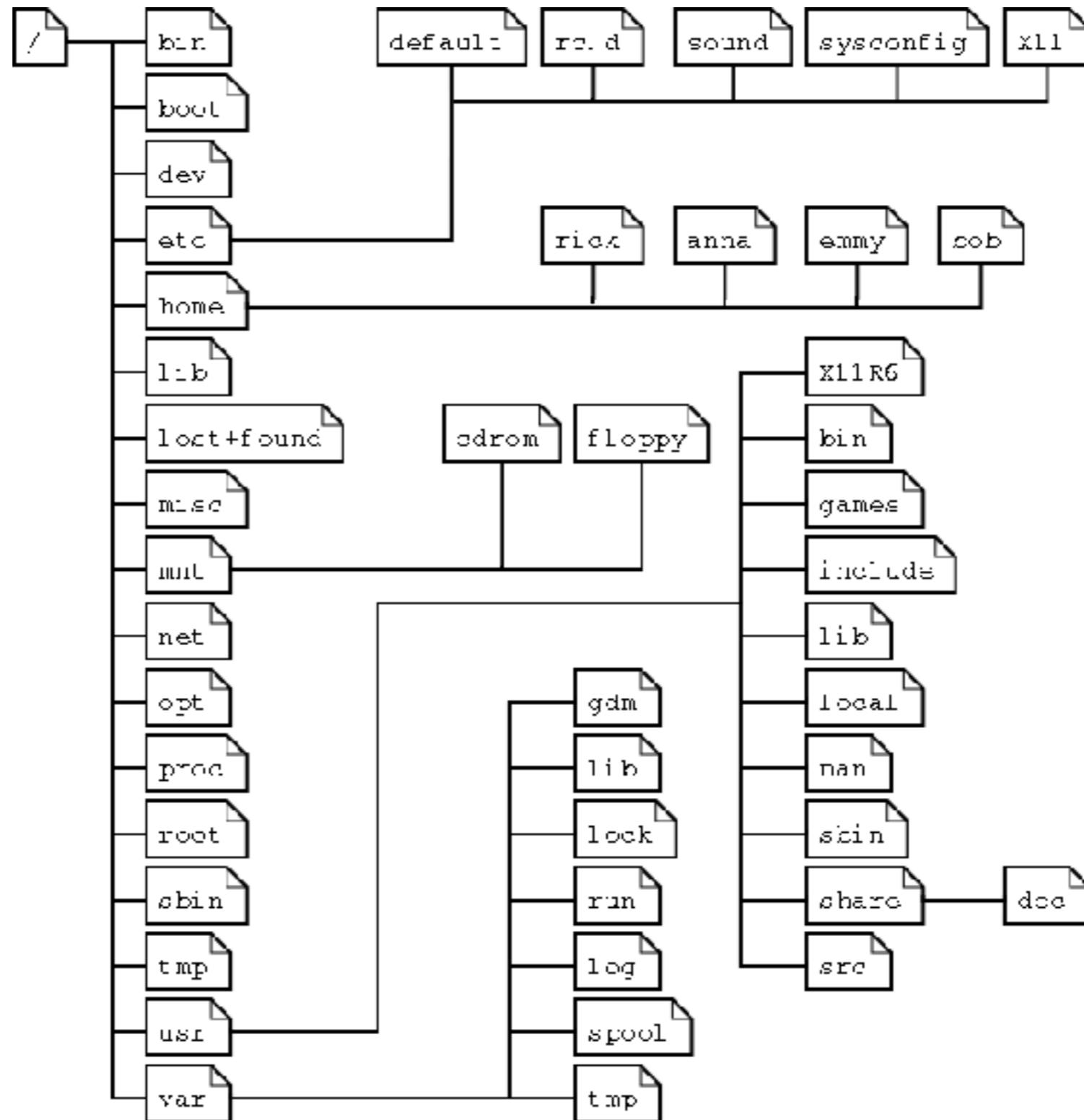
Linux Filesystem Structure

- Each filesystem system contains:
 - ***control block***: holds information about the filesystem
 - ***inodes***: contain information about individual files
 - ***data blocks***: contain the information stored in the individual files
- In Linux, users and kernel see filesystem in different ways
 - User
 - Filesystem appears as a hierarchical arrangement of directories that contain files and other directories.
 - Directories and files are identified by their names.
 - Kernel
 - Filesystem is flat, not have a hierarchical structure
 - no differences between directories and a file
 - identify files by name.

Linux Filesystem Structure: kernel's view

- inode: an entry in a list of inodes, it contains
 - inode number (a unique identification number)
 - the owner and group associated with the file
 - the file type
 - the file's permission list
 - the file creation, access and modification times
 - the size of the file
 - the disk address

Linux Filesystem Structure: user's view



Linux Filesystem Structure: user's view

- The tree of the file system starts at the trunk or *slash*, indicated by a forward slash (/). This directory, containing all underlying directories and files, is also called the *root directory* or "the root" of the file system.

```
[root@localhost ~]# ls /
bin  dev  home  lib64  mnt  proc  run  srv  tmp  var
boot  etc  lib   media  opt  root  sbin  sys  usr
```

Linux Filesystem Structure: subdirectories of the root

Pathname	Contents
/bin	Core operating system commands
/boot	Boot loader, kernel, and files needed by the kernel
/compat	On FreeBSD, files and libraries for Linux binary compatibility
/dev	Device entries for disks, printers, pseudo-terminals, etc.
/etc	Critical startup and configuration files
/home	Default home directories for users
/lib	Libraries, shared libraries, and commands used by /bin and /sbin
/media	Mount points for filesystems on removable media
/mnt	Temporary mount points, mounts for removable media
/opt	Optional software packages (rarely used, for compatibility)
/proc	Information about all running processes
/root	Home directory of the superuser (sometimes just /)
/run	Rendezvous points for running programs (PIDs, sockets, etc.)
/sbin	Core operating system commands ^a
/srv	Files held for distribution through web or other servers
/sys	A plethora of different kernel interfaces (Linux)
/tmp	Temporary files that may disappear between reboots
/usr	Hierarchy of secondary files and commands
/usr/bin	Most commands and executable files
/usr/include	Header files for compiling C programs
/usr/lib	Libraries; also, support files for standard programs
/usr/local	Local software or configuration data; mirrors /usr
/usr/sbin	Less essential commands for administration and repair
/usr/share	Items that might be common to multiple systems
/usr/share/man	On-line manual pages
/usr/src	Source code for nonlocal software (not widely used)
/usr/tmp	More temporary space (preserved between reboots)
/var	System-specific data and a few configuration files
/var/adm	Varies: logs, setup records, strange administrative bits
/var/log	System log files
/var/run	Same function as /run; now often a symlink
/var/spool	Spooling (that is, storage) directories for printers, mail, etc.
/var/tmp	More temporary space (preserved between reboots)

a. The distinguishing characteristic of /sbin was originally that its contents were statically linked and so had fewer dependencies on other parts of the system. These days, all binaries are dynamically linked and there is no real difference between /bin and /sbin.

Linux Filesystem: File Path

- File Path: is a unique location to a file or a directory in a filesystem. A path to a file is a combination of / and alphanumeric characters
 - Absolute path/Full path: the location of a file or directory from the root directory (/)
 - Relative path: the present working directory.
 - **pwd**: command to print name of current/working directory

```
[root@localhost sysconfig]# pwd  
/etc/sysconfig
```

Linux Filesystem: Basic Directory Operation

- **cd** : command used to enter a directory
- Example of usage:
 - **cd** : go to current user's home directory
 - **cd /path** : go to a directory (can be full path/relative path)
 - **cd ~/** : go to current user's home directory, can be followed by subdirectory name
 - **cd ..** : go to parent directory

Linux Filesystem: Basic Directory Operation

- **mkdir** : create a directory
- **rmdir** : remove a directory (empty)
- Example:

```
[root@localhost ~]# mkdir temp
[root@localhost ~]# ls
anaconda-ks.cfg  dir.txt  err.txt  temp
[root@localhost ~]# rmdir temp
[root@localhost ~]# ls
anaconda-ks.cfg  dir.txt  err.txt
[root@localhost ~]#
```

How to remove non-empty directories?

Linux Filesystem: disks and partition

- **fdisk**: command to manipulate disk partition table
- **df** : command to report file system disk space usage
- **mount** : command to mount a filesystem
 - All files accessible in a Unix system are arranged in one big tree, the file hierarchy, rooted at /. These files can be spread out over several devices. The **mount** command serves to attach the filesystem found on some device to the big file tree.
- **umount** : command will detach it again.

Exercise: Let's add a new hard drive to your system

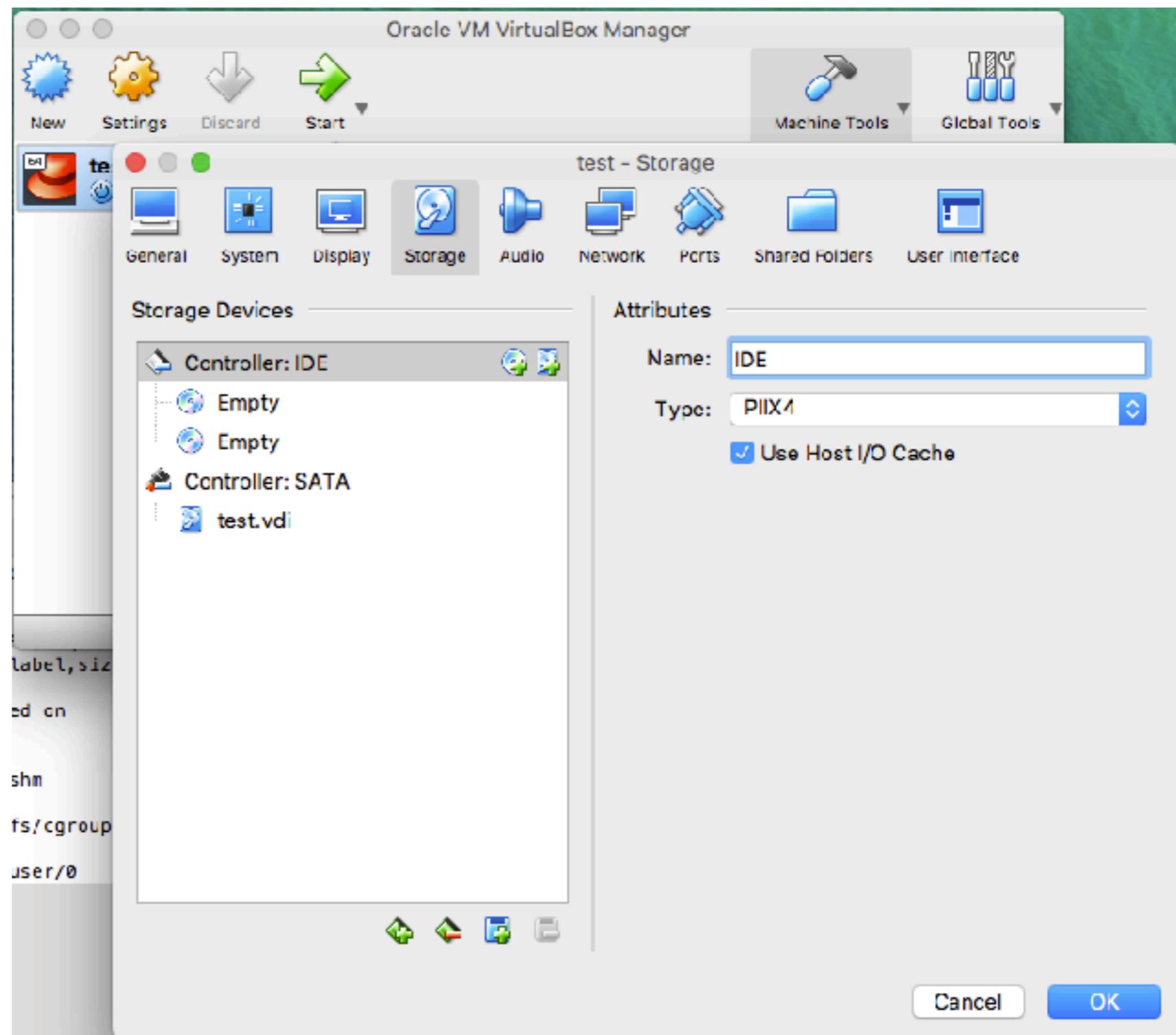
- Let's check the existing filesystems by execute the following command:

df -h

```
[root@localhost mnt]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/mapper/centos-root  6.2G  897M  5.4G  15% /
devtmpfs        486M    0  486M   0% /dev
tmpfs          497M    0  497M   0% /dev/shm
tmpfs          497M  6.6M  490M   2% /run
tmpfs          497M    0  497M   0% /sys/fs/cgroup
/dev/sda1       1014M 125M  890M  13% /boot
tmpfs          100M    0  100M   0% /run/user/0
```

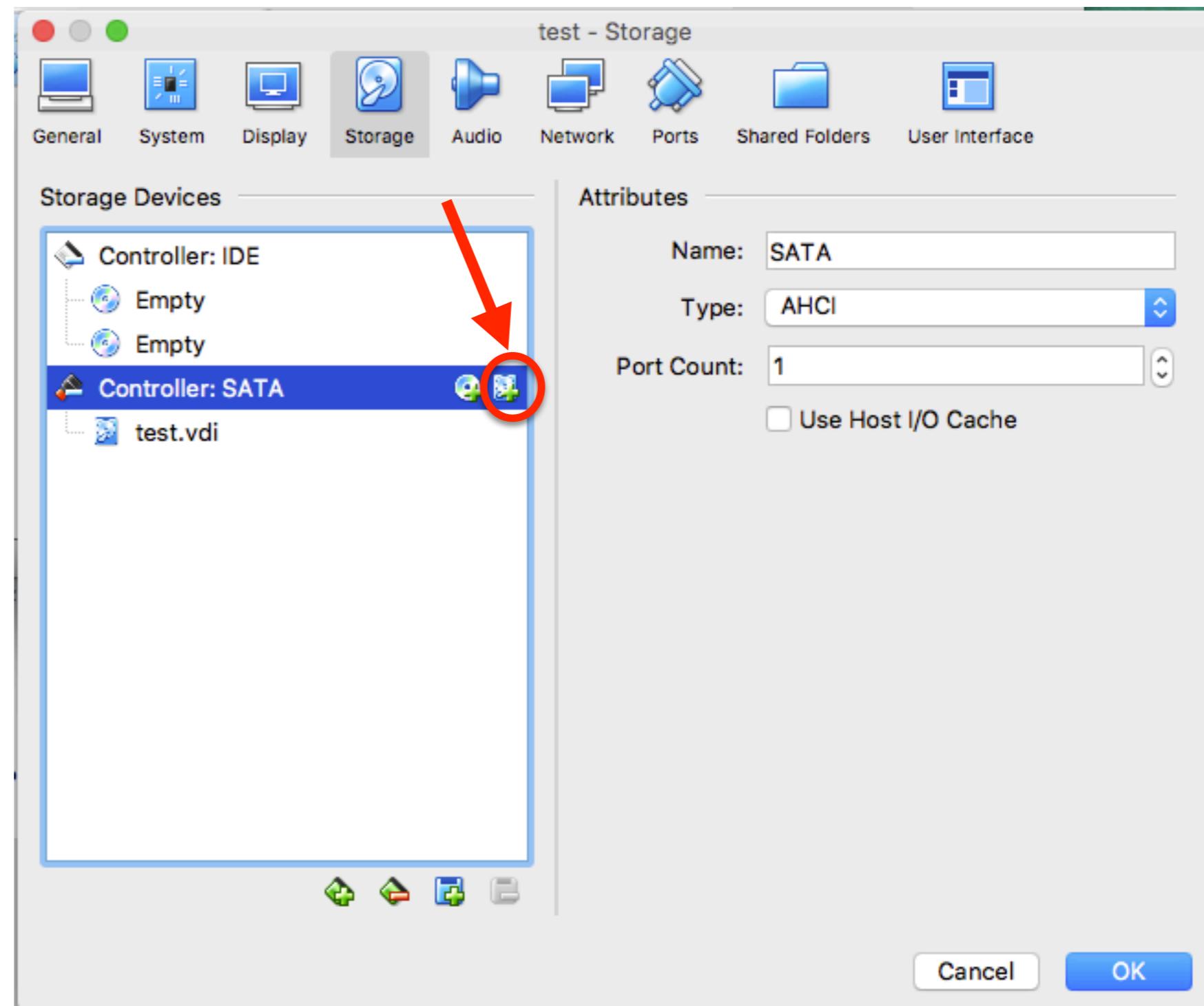
Exercise: Let's add a new hard drive to your system

- Poweroff your VM
- Go to 'Settings' for your VM



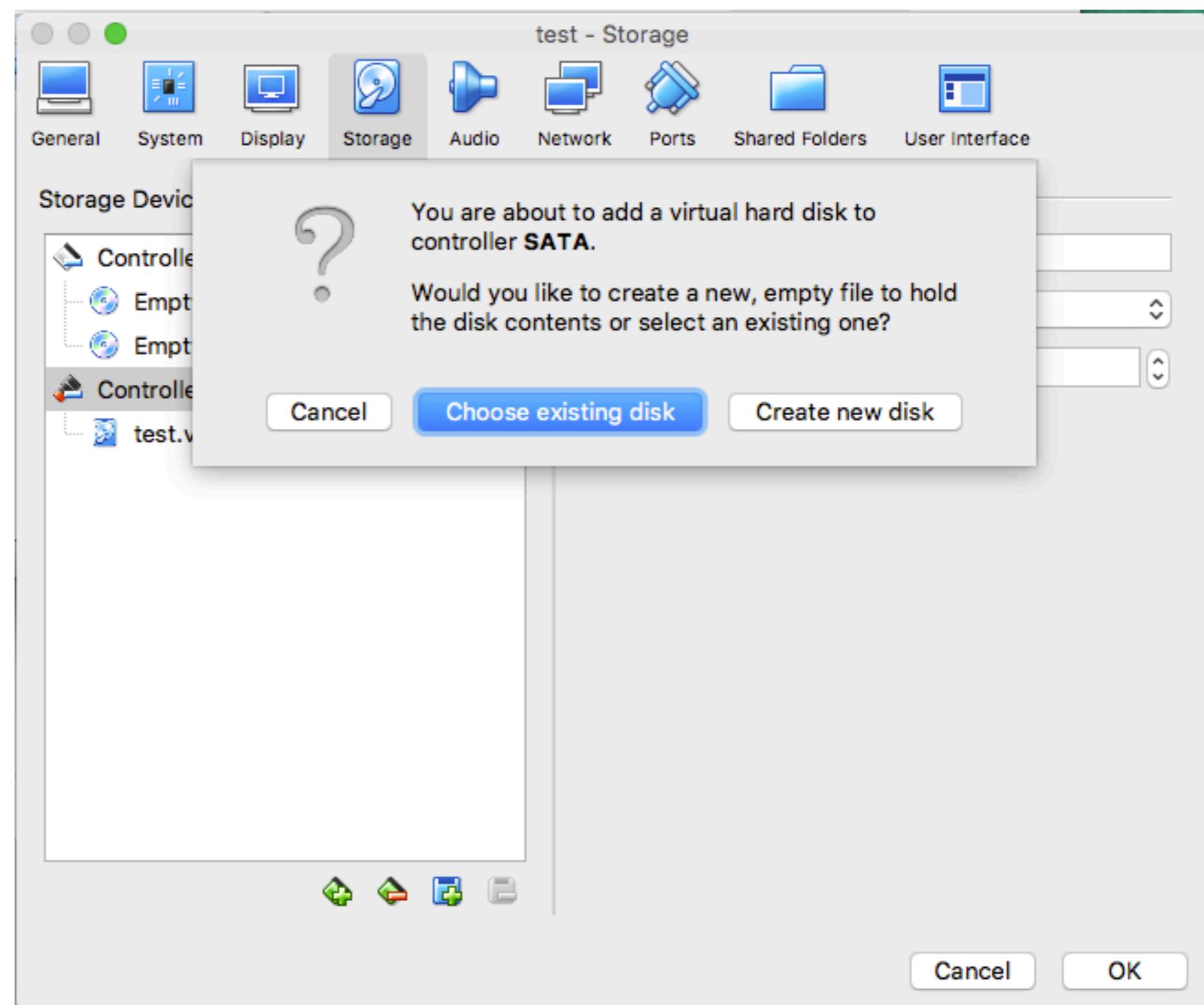
Exercise: Let's add a new hard drive to your system

- Add a new hard drive to your SATA Controller



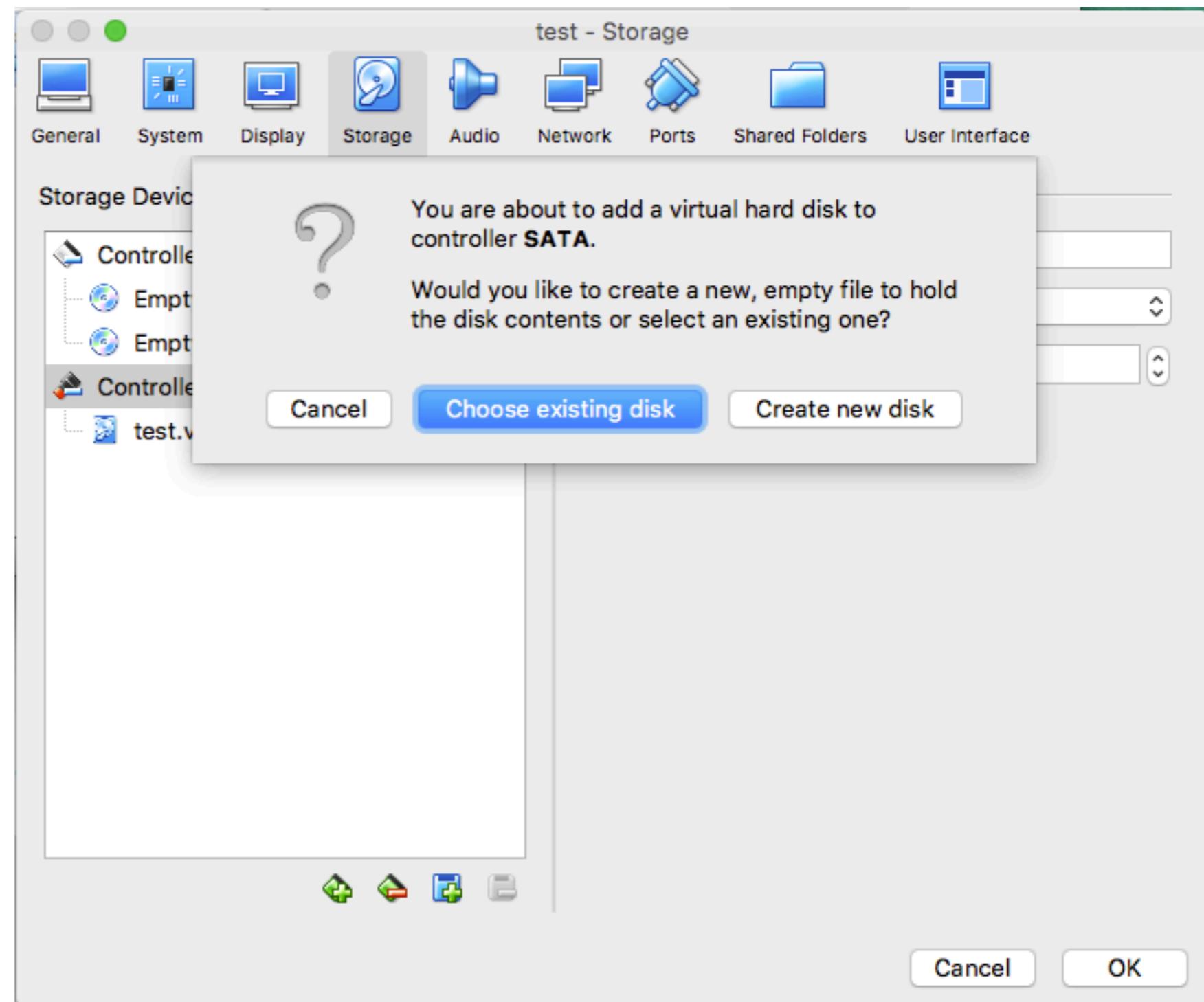
Exercise: Let's add a new hard drive to your system

- Create a new disk



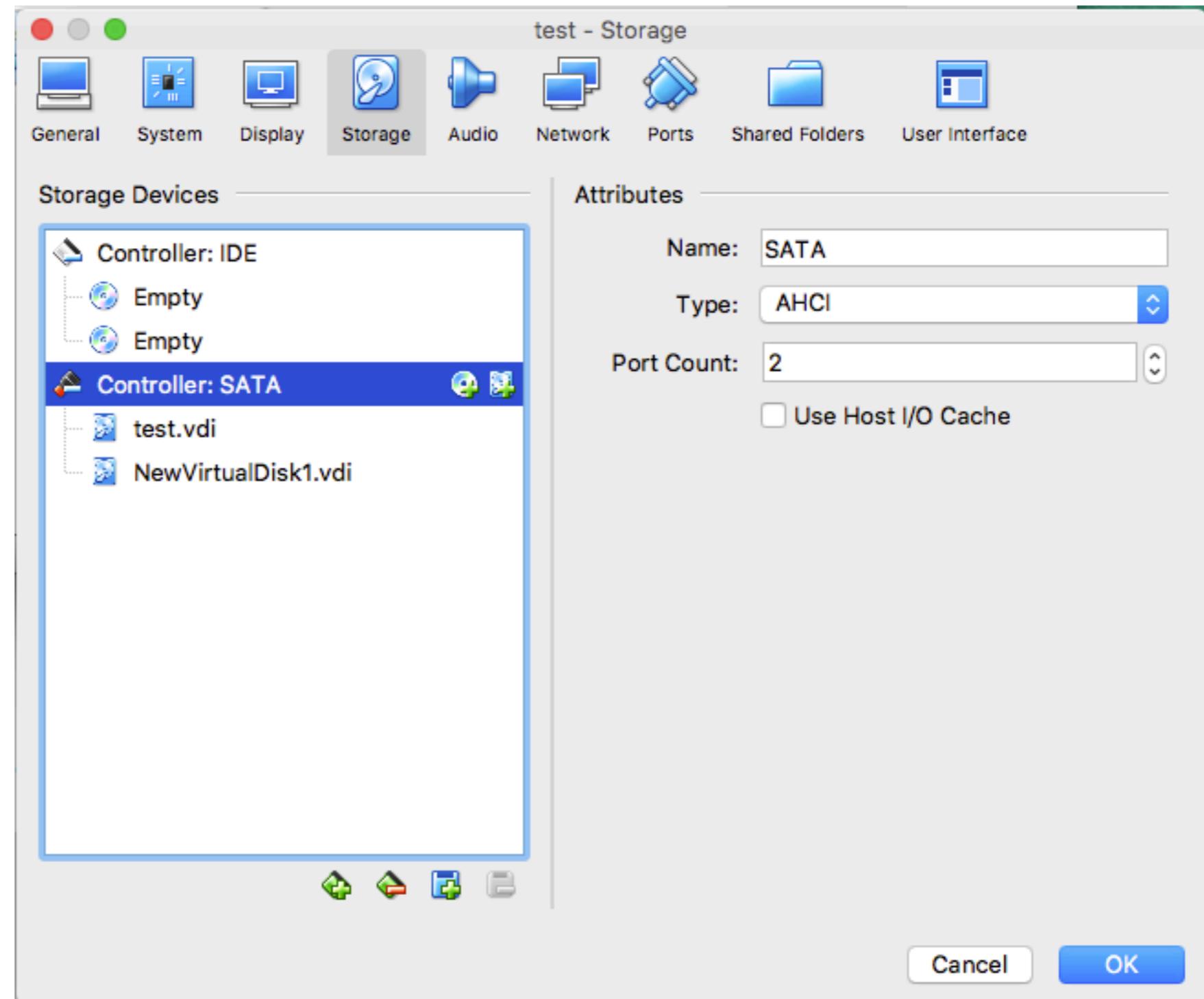
Exercise: Let's add a new hard drive to your system

- Create a new disk
- Follow the instructions to create a new disk with desired size



Exercise: Let's add a new hard drive to your system

- Create a new disk
- Follow the instructions to create a new disk with desired size



Exercise: Let's add a new hard drive to your system

- Poweron your VM
- Check your mounted filesystem with "df"
- Can you find your new HDD?

```
[root@localhost ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/mapper/centos-root  6.2G  897M  5.4G  15% /
devtmpfs        486M    0  486M   0% /dev
tmpfs          497M    0  497M   0% /dev/shm
tmpfs          497M  6.6M  490M   2% /run
tmpfs          497M    0  497M   0% /sys/fs/cgroup
/dev/sda1       1014M 125M  890M  13% /boot
tmpfs          100M    0  100M   0% /run/user/0
```

Exercise: Let's add a new hard drive to your system

- Use 'fdisk -l' to find your new HDD

```
[root@localhost ~]# fdisk -l

Disk /dev/sda: 8589 MB, 8589934592 bytes, 16777216 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x0000837e

      Device Boot   Start     End   Blocks   Id  System
/dev/sda1    *     2048  2099199   1048576   83  Linux
/dev/sda2     2099200 16777215   7339008   8e  Linux LVM

Disk /dev/sdb: 8589 MB, 8589934592 bytes, 16777216 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes

Disk /dev/mapper/centos-root: 6652 MB, 6652166144 bytes, 12992512 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes

Disk /dev/mapper/centos-swap: 859 MB, 859832320 bytes, 1679360 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
```

Exercise: Let's add a new hard drive to your system

- Use 'fdisk /dev/sdb' to partition your new HDD

```
[root@localhost ~]# fdisk /dev/sdb
Welcome to fdisk (util-linux 2.23.2).
```

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table
Building a new DOS disklabel with disk identifier 0xd3813d0d.

Command (m for help):

Exercise: Let's add a new hard drive to your system

- Enter m for help with different options

```
Command (m for help): m
Command action
  a  toggle a bootable flag
  b  edit bsd disklabel
  c  toggle the dos compatibility flag
  d  delete a partition
  g  create a new empty GPT partition table
  G  create an IRIX (SGI) partition table
  l  list known partition types
  m  print this menu
  n  add a new partition
  o  create a new empty DOS partition table
  p  print the partition table
  q  quit without saving changes
  s  create a new empty Sun disklabel
  t  change a partition's system id
  u  change display/entry units
  v  verify the partition table
  w  write table to disk and exit
  x  extra functionality (experts only)

Command (m for help):
```

Exercise: Let's add a new hard drive to your system

- Enter **n** to create a new partition
- Use default for partition type, partition number and first sector
- Set your size of the partition:
 - in this case, we want to partition a 8G HDD into two partitions, each has 4G space
 - type '**+4G**'

```
Command (m for help): n
Partition type:
  p  primary (0 primary, 0 extended, 4 free)
  e  extended
Select (default p):
Using default response p
Partition number (1-4, default 1):
First sector (2048-16777215, default 2048):
Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-16777215, default 16777215): +4G
Partition 1 of type Linux and of size 4 GiB is set
```

Exercise: Let's add a new hard drive to your system

- Enter 'w' to write the partition table to disk

```
Command (m for help): w
The partition table has been altered!
Calling ioctl() to re-read partition table.
Syncing disks.
```

Exercise: Let's add a new hard drive to your system

- Use 'fdisk -l' to check your partition:

```
[root@localhost ~]# fdisk -l

Disk /dev/sda: 8589 MB, 8589934592 bytes, 16777216 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x000083e

      Device Boot   Start     End   Blocks   Id  System
/dev/sda1  *    2048 2099199 1048576  83  Linux
/dev/sda2        2099200 16777215 7339008  8e  Linux LVM

Disk /dev/sdb: 8589 MB, 8589934592 bytes, 16777216 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk label type: dos
Disk identifier: 0x74d100b5

      Device Boot   Start     End   Blocks   Id  System
/dev/sdb1          2048 8390655 4194304  83  Linux
/dev/sdb2        8390656 16777215 4193280  83  Linux
```

Exercise: Let's add a new hard drive to your system

- before you can use any partitions, you have to create a filesystem (type of filesystem) in your partitions
- use 'df -T' to check the existing filesystem type

```
[root@localhost ~]# df -hT
Filesystem           Type      Size  Used   Avail Use% Mounted on
/dev/mapper/centos-root xfs       6.2G  897M  5.4G  15% /
devtmpfs              devtmpfs  486M    0  486M   0% /dev
tmpfs                 tmpfs     497M    0  497M   0% /dev/shm
tmpfs                 tmpfs     497M   6.6M  490M   2% /run
tmpfs                 tmpfs     497M    0  497M   0% /sys/fs/cgroup
/dev/sda1              xfs      1014M  125M  890M  13% /boot
tmpfs                 tmpfs    100M    0  100M   0% /run/user/0
[root@localhost ~]#
```

Exercise: Let's add a new hard drive to your system

- use 'mkfs.xfs' to format both of your new partitions to xfs filesystem

```
[root@localhost ~]# mkfs.xfs /dev/sdb1
meta-data=/dev/sdb1
              =               isize=512    agcount=4, agsize=262144 blks
              =               sectsz=512   attr=2, projid32bit=1
              =               crc=1      finobt=0, sparse=0
data        =               bsize=4096   blocks=1048576, imaxpct=25
              =               sunit=0     swidth=0 blks
naming      =version 2    bsize=4096   ascii-ci=0 ftype=1
log         =internal log  bsize=4096   blocks=2560, version=2
              =               sectsz=512   sunit=0 blks, lazy-count=1
realtime    =none          extsz=4096   blocks=0, rtextents=0
```

Exercise: Let's add a new hard drive to your system

- use 'mkfs.xfs' to format both of your new partitions to xfs filesystem

```
[root@localhost ~]# mkfs.xfs /dev/sdb2
meta-data=/dev/sdb2
              isize=512    agcount=4, agsize=262080 blks
              =                     attr=2, projid32bit=1
              =                     crc=1
data        =                     sectsz=512   attr=2, projid32bit=1
              =                     finobt=0, sparse=0
              =                     bsize=4096   blocks=1048320, imaxpct=25
              =                     sunit=0     swidth=0 blks
naming      =version 2    bsize=4096   ascii-ci=0 ftype=1
log         =internal log bsize=4096   blocks=2560, version=2
              =                     sectsz=512   sunit=0 blks, lazy-count=1
realtime    =none        extsz=4096   blocks=0, rtextents=0
```

Exercise: Let's add a new hard drive to your system

- Now we need to create a mount point to mount the new hard drive so that we can access the new partitions through file system
- Let's create two new directories in root directory

```
[root@localhost ~]# mkdir /newPartition1 /newPartition2
[root@localhost ~]# ls /
bin  dev  home  lib64  mnt          newPartition2  proc  run  srv  tmp  var
boot etc  lib   media  newPartition1  opt           root  sbin  sys  usr
```

Exercise: Let's add a new hard drive to your system

- Let's mount partition /dev/sdb1 on /newPartition1 and mount partition /dev/sdb2 on /newPartition2:
 - `mount /dev/sdb1 /newPartition1`
 - `mount /dev/sdb2 /newPartition2`
- use 'df' to check the disk usage

```
[root@localhost ~]# mount /dev/sdb1 /newPartition1
[root@localhost ~]# mount /dev/sdb2 /newPartition2
[root@localhost ~]# df -lh
Filesystem           Size   Used  Avail Use% Mounted on
/dev/mapper/centos-root  6.2G  897M  5.4G  15% /
devtmpfs              486M     0  486M   0% /dev
tmpfs                 497M     0  497M   0% /dev/shm
tmpfs                 497M   6.6M  490M   2% /run
tmpfs                 497M     0  497M   0% /sys/fs/cgroup
/dev/sda1              1014M 125M  890M  13% /boot
tmpfs                 100M     0  100M   0% /run/user/0
/dev/sdb1               4.0G  33M  4.0G   1% /newPartition1
/dev/sdb2               4.0G  33M  4.0G   1% /newPartition2
```

File Types

- Regular files
- Directories
- Character device files
- Local domain sockets
- Named pipes (FIFOs)
- Symbolic links

Check File Information

- `file` command is used to determine the file types
- `ls` command is used to list directory contents
 - commonly used options:
 - `-a`: list all files including entries starting with `.`
 - `-d`: list directories
 - `-l`: list detailed information
 - `-h`: list files with human readable format
 - `-R`: Recursively list Sub-Directories
 - `-r`: Reverse output order
 - `-S`: Sort files by file size
 - `-i`: print the inode number

Understand detailed file list information

- Use `ls -al` to list all files in your directory

```
total 36
dr-xr-x---.  2 root root  165 Jan 24 01:28 .
dr-xr-xr-x. 17 root root  224 Jan 23 23:24 ..
-rw-----.  1 root root 1235 Jan 23 23:25 anaconda-ks.cfg
-rw-----.  1 root root  270 Jan 24 01:38 .bash_history
-rw-r--r--.  1 root root   18 Dec 28 2013 .bash_logout
-rw-r--r--.  1 root root  176 Dec 28 2013 .bash_profile
-rw-r--r--.  1 root root  176 Dec 28 2013 .bashrc
-rw-r--r--.  1 root root  100 Dec 28 2013 .cshrc
-rw-r--r--.  1 root root   48 Jan 24 01:25 dir.txt
-rw-r--r--.  1 root root   47 Jan 24 01:29 err.txt
-rw-r--r--.  1 root root  129 Dec 28 2013 .tcshrc
```

Understand detailed file list information

- Use `ls -al` to list all files in your directory
- Meaning of each field:
 - File permissions
 - Number of links
 - Owner name
 - Owner group
 - File size
 - Time of last modification
 - File/Directory name

Understand detailed file list information

- Use `ls -al` to list all files in your directory
- Meaning of each field:
 - File permissions: 10 characters
 - First character is file type

File type	Symbol	Created by	Removed by
Regular file	-	editors,cp,etc.	rm
Directory	d	mkdir	rmdir, rm -r
Character device file	c	mknod	rm
Block device file	b	mknod	rm
Local domain socket	s	socket (system call)	rm
Named pipe	p	mknod	rm
Symbolic link	l	ln -s	rm

Understand detailed file list information

- Use `ls -al` to list all files in your directory
- Meaning of each field:
 - File permissions: 10 characters
 - First character is file type
 - Three sets of characters, three times, indicating permissions for owner, group and other:
 - r : readable
 - w: writable
 - x: executable

Regular files

- Most files are just regular files
- Consist of a series of bytes
- Filesystems impose no structure on their contents
- Text files, data files, executable programs, shared libraries and etc.
- Both sequential and random access are allowed

Create a File

- File creation: there are many ways to create a file
- We can create a file by redirecting output to a file (>, >>)
 - `echo 'xxx' > file`
- We can use 'touch' command
 - `touch filename`
- We can use editors (vim) to create file

File Name

- General rules for file naming:
 - All file names are case sensitive.
 - You can use upper and lowercase letters, numbers, “.” (dot), and “_” (underscore) symbols.
 - You can use other special characters such as blank space, but they are hard to use and it is better to avoid them.
 - Most modern Linux and UNIX limit filename to 255 characters (255 bytes). However, some older version of UNIX system limits filenames to 14 characters only.
 - A filename must be unique inside its directory.
 - No file extension in Linux
 - files start with '.' are hidden files

File Name

- Avoid using the following characters from appearing in file names
 - /
 - >
 - <
 - |
 - :
 - &

File Operation

- Move or rename a file command: `mv`
 - Move a file to another directory:
 - `mv file_name target_directory`
 - Rename a file: move file in the same directory
 - `mv file new_filename`
 - Move a directory to another directory:
 - `mv dir new_dir`
 - Rename a directory:
 - `mv dir new_dir_name`
 - Move multiple files at a time:
 - `mv file1 file2 file3 new_dir`

File Operation

- Copy a file/directory:
 - Copy a file to another directory:
 - `cp file_name target_directory`
 - Copy a directory to another directory (recursively copy all sub dirs):
 - `mv -R dir new_dir`
- Remove a file:
 - `rm file_name`
 - Option: `-f` (force to remove without confirmation)
 - remove all files in a directory:
 - `rm *`
 - remove a directory:
 - `rm -fr dir_name`

View a File

- **cat**: Concatenate files and print on the standard output
- Commonly used options:
 - **-n** : number all output lines
 - **-v** : show nonprinting
- **cat** is not good for long files
- For crt viewing:
 - **more** : a filter for paging through text one screenful at a time.
 - **less** : provides **more** emulation plus extensive enhancements.

File Operation

- **diff** : file comparison (line by line)
 - commonly used options:
 - -q : report only when files differ
 - -s : report only when files are the same
 - -c : output NUM (default 3) lines of copied context
- **wc** : word count of a file (newline, word, and byte)
 - commonly used options:
 - -c : print the byte counts
 - -m : print the character counts
 - -l : print the newline counts
 - -w : print the word counts

Directories

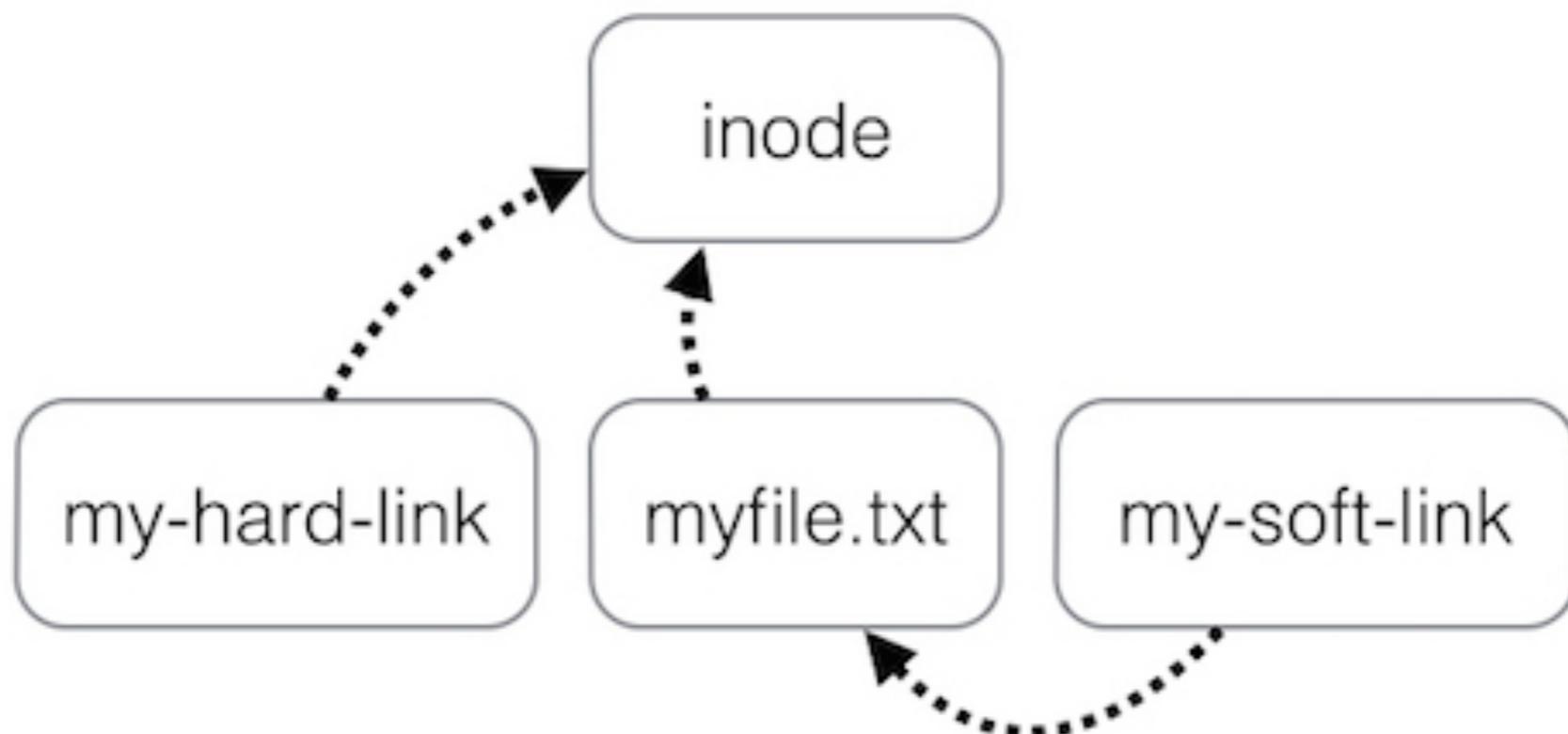
- A directory contains named references to other files.
- "." : refers to directory itself
- ".." : refers to its parent directory

Hard links

- A file name is stored within its parent directory, not with the file itself
- More than one directory can refer to a file at one time, and the references can have different names
- More than one entry in a single directory can refer to a file at one time, and the references can have different names
- These additional references (Hard links) are synonymous with the original file
- System maintains a count of the number of links
- Cannot be created for directories
- Cannot cross filesystem boundaries or span across partitions

Symbolic Links (soft links)

- A symbolic link (soft link) points to a file by name



Create hard and symbolic links

- Create a hard link:
 `ln oldfile newfile`
- Create a symbolic link:
 `ln -s oldfile newfile`



CSC424 System Administration

Instructor: Dr. Hao Wu

Week 5 User Management, Access Control and Rootly
Powers

Adding Users

- Mechanically, the process of adding a new user consists of several steps required by the system and a few more that establish a useful environment for the new user and incorporate the user into your local administrative system.
- Required:
 - Edit the **passwd** and **shadow** files to define the user's account
 - Add the user to the **/etc/group** file (not really necessary, but nice)
 - Set an initial password
 - Create, **chown**, and **chmod** the user's home directory
 - Configure roles and permissions

Adding a user

- Manual maintenance of the **passwd** and **group** files is error prone and inefficient
- We use higher-level tools such as:
 - **useradd, adduser**: add a new user to system
 - **usermod**: user management
 - **passwd**: change password
 - **userdel**: delete a user

useradd, adduser

- Use ls command to check the **useradd** and **adduser** scripts

```
[root@localhost ~]# ls -l /usr/sbin/useradd /usr/sbin/adduser
lrwxrwxrwx. 1 root root    7 Feb  5 14:09 /usr/sbin/adduser -> useradd
-rwxr-x---. 1 root root 118192 Nov  5 2016 /usr/sbin/useradd
```

- **adduser** is actually a soft link of **useradd**
- Syntax for **useradd** command:

useradd [option] username

- Commonly used options:
 - g: name or ID of the primary group of the new user (group must exist)
 - G: groups

Adding a user

- Let's create a new user of your own (I'm using my name as username):

useradd hao

- Create a passwd for your new user:

passwd hao

```
[root@localhost ~]# passwd hao
Changing password for user hao.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

Adding a user

- When a new user is added to system using **useradd**, the system will:
 - create a home directory for user: **/home/username**

```
[root@localhost ~]# ls -al /home
total 0
drwxr-xr-x. 3 root root 17 Feb 13 21:27 .
dr-xr-xr-x. 17 root root 224 Feb 5 14:14 ..
drwx-----. 2 hao hao 83 Feb 13 21:32 hao
```

- copy startup files into user's home directory

```
[root@localhost ~]# ls -al /home/hao
total 16
drwx-----. 2 hao hao 83 Feb 13 21:32 .
drwxr-xr-x. 3 root root 17 Feb 13 21:27 ..
-rw-----. 1 hao hao 9 Feb 13 21:32 .bash_history
-rw-r--r--. 1 hao hao 18 Aug 2 2017 .bash_logout
-rw-r--r--. 1 hao hao 193 Aug 2 2017 .bash_profile
-rw-r--r--. 1 hao hao 231 Aug 2 2017 .bashrc
```

- create a mailbox for user: **/var/spool/mail/username**

Change user's password

- passwd: command to change the user's password
- Syntax:
passwd [username]
 - When no username is given, system will change the password for current user
 - When username is given, system will change the password for that particular user (**Only root can change password for other users**)

Delete a user

- userdel: command to delete a user
- Syntax:
userdel [option] username
- Commonly used option:
- r: Files in the user's home directory will be removed along with the home directory itself and the user's mail spool.

User Management

- usermod: modify a user account
- Syntax:
usermod [option] username
- Commonly used options:
 - d: The user's new home directory
 - e: The date on which the user account will be disabled. The date is specified in the format YYYY-MM-DD
 - g: The group name of the user's initial login group.
 - l: Change new username
 - L: Lock a user's password
 - U: Unlock a user's password
 - m: move the content of the user's home directory to the new location.

The /etc/passwd file

- **/etc/passwd**: a file contain a list of users recognized by the system.
 - Originally, each user's encrypted password was also stored in the **/etc/passwd** file, which is world-readable
 - Now, the password is stored in **/etc/shadow**, which is not world-readable
 - The file contains 7 fields separated by colons:
 - Login name
 - Encrypted password placeholder
 - UID (user ID) number
 - Default GID (group ID) number
 - Optional "GECOS" information: full name, office, extension, home phone
 - Home directory
 - Login shell

Sample /etc/passwd file

```
root@localhost ~]# cat /etc/passwd
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
adm:x:3:4:adm:/var/adm:/sbin/nologin
lp:x:4:7:lp:/var/spool/lpd:/sbin/nologin
sync:x:5:0:sync:/sbin:/bin/sync
shutdown:x:6:0:shutdown:/sbin:/sbin/shutdown
halt:x:7:0:halt:/sbin:/sbin/halt
mail:x:8:12:mail:/var/spool/mail:/sbin/nologin
operator:x:11:0:operator:/root:/sbin/nologin
games:x:12:100:games:/usr/games:/sbin/nologin
ftp:x:14:50:FTP User:/var/ftp:/sbin/nologin
nobody:x:99:99:Nobody:/:/sbin/nologin
systemd-network:x:192:192:systemd Network Management:/:/sbin/nologin
dbus:x:81:81:System message bus:/:/sbin/nologin
polkitd:x:999:997:User for polkitd:/:/sbin/nologin
postfix:x:89:89::/var/spool/postfix:/sbin/nologin
chrony:x:998:996::/var/lib/chrony:/sbin/nologin
sshd:x:74:74:Privilege-separated SSH:/var/empty/sshd:/sbin/nologin
hao:x:1000:1000::/home/hao:/bin/bash
```

/etc/shadow file

- On Linux, the shadow password file is readable only by the superuser and serves to keep encrypted passwords safe from prying eyes and password cracking programs.

```
[root@localhost ~]# ls -al /etc/shadow
-----. 1 root root 707 Feb 13 21:34 /etc/shadow
[root@localhost ~]# ls -al /etc/passwd
-rw-r--r--. 1 root root 883 Feb 13 21:32 /etc/passwd
```

/etc/shadow file

- Nine fields for each line, separated by colons:
 - Login name
 - Encrypted password
 - Date of last password change
 - Minimum number of days between password change
 - Maximum number of days between password change
 - Number of days in advance to warn users about password expiration
 - Days after password expiration that account is disabled
 - Account expiration date
 - A field reserved for future use which is currently always empty

Sample /etc/shadow file

```
[root@localhost ~]# cat /etc/shadow
root:$6$GsG/Q1mpnPv7cnfJ$XTYn.oq2jvX896REmeQfwDfwZ2R0/
MYamT4vL0Y6uDxNIplc0GjkX125XR0SNjUmvt7CoKXT3r9cQ.LLse730::0:99999:7:::
bin:*:17110:0:99999:7:::
daemon:*:17110:0:99999:7:::
adm:*:17110:0:99999:7:::
lp:*:17110:0:99999:7:::
sync:*:17110:0:99999:7:::
shutdown:*:17110:0:99999:7:::
halt:*:17110:0:99999:7:::
mail:*:17110:0:99999:7:::
operator:*:17110:0:99999:7:::
games:*:17110:0:99999:7:::
ftp:*:17110:0:99999:7:::
nobody:*:17110:0:99999:7:::
systemd-network:!:17567::::::::::
dbus:!:17567::::::::::
polkitd:!:17567::::::::::
postfix:!:17567::::::::::
chrony:!:17567::::::::::
sshd:!:17567::::::::::
hao:$6$7QdC2H5/$K.
7KBgP1zqc6XNdZ7bQClydaLHmBwYeJxjpFYVZAmbu4J0drw5k3KKLyZARPM08gT0h1N7wlWpKih.pq0
QlNw/:17576:0:99999:7:::
```

Group Management

- groupadd: command to create a new group, syntax:
groupadd [options] groupName
- groupmod: command to modify a group definition on the system, syntax:
groupmod [option] groupName
- groupdel: command to delete a group, syntax:
groupdel [option] groupName

The /etc/group file

- The /etc/group file contains the names of groups and a list of each group's members. Each line represents one group and contains 4 fields:
- Group name
- Encrypted password or a placeholder
- GID number
- List of members, separated by commas

Standard UNIX Access Controls

- The standard UNIX access control model has remained largely unchanged for decades. The scheme follows a few basic rules:
 - Access control decisions depend on which user is attempting to perform an operation, or in some cases, on that user's membership in a UNIX group
 - Objects(e.g., files and processes) have owners. Owners have broad (but not necessarily unrestricted) control over their objects.
 - You own the objects you create
 - The special user account called "root" can act as the owner of any object
 - Only root can perform certain sensitive administrative operations.

Standard UNIX Access Controls

- Certain systems calls are restricted to root
 - checks the identity of the current user and rejects the operation if the user is not root
- Other system calls implement different calculations that involve both ownership matching and special provisions for root
- Filesystems have their own access control system

Filesystem and process ownership

- Each file has
 - an owner
 - a group
- Each process has
 - an owner
 - can send the process signals
 - can reduce the process's scheduling priority

The root account

- The root account is UNIX's omnipotent administrative user.
- It is also known as the superuser account.
- UID of root is 0
- Traditional UNIX allows the superuser to perform any valid operation on any file or process.
- Some example of restricted operations are:
 - Creating device files
 - Setting the system clock
 - Setting system's hostname
 - Configuring network interfaces
 - Opening privileged network ports (those numbered below 1024)
 - Shutting down the system

Management of the Root Account

- Root access is required for system administration, and it's also a pivot point for system security.
- Drawbacks:
 - root logins leave no record of what operations were performed as root
 - even worse when an access was unauthorized
 - log-in-as-root leaves no record of who was actually doing the work
- Most systems allow root logins to be disabled on terminals

Disable root access

- Disabling root access via any console device (tty)
 - Prevents access to the root account via the console or the network.
 - An empty /etc/securetty file prevents root login on any devices attached to the computer.
- Disabling root SSH logins
 - Prevents root access via the OpenSSH suit of tools
 - Edit the /etc/ssh/sshd_config file and set the PermitRootLogin parameter to no

su: substitute user identity

- **su:** a command to change user identity
 - a marginally better way to access the root account
 - if invoked without argument, su prompts for the root password and then starts up a root shell
 - Root privileges remain in effect until you terminate the shell by typing **<Control-D>** or the **exit** command
 - doesn't record the commands executed as root
 - create a log entry that states who became root and when
 - Can also substitute identities other than root: **su username**

sudo

- sudo: a command to execute command as root or as another restricted user
 - recommend as the primary method of access to the root account
 - /etc/sudoers lists the people who are authorized to use sudo and the commands they are allowed to run on each host
 - if the proposed command is permitted, sudo prompts for the user's own password and execute the command
 - sudo can be executed without having to type a password until a five-minute period (configurable) has elapsed with no further sudo activity
 - sudo keeps a log of the command lines that were executed, the hosts on which they were run, the people who ran them, the directories from which they were run, and the times at which they were invoked.

sudo

- Advantages
 - Accountability is much improved because of command logging
 - Users can do specific chores without having unlimited root privileges
 - The real root password can be known to only one or two people
 - Using sudo is faster than using su or logging in as root
 - Privileges can be revoked without the need to change the root password
 - A canonical list of all users with root privileges is maintained
 - The chance of a root shell being left unattended is lessened
 - A single file can control access for an entire network

chmod: change file permission

- chmod: command and system call which may change the access permissions to file system objects (files and directories). It may also alter special mode flags. The request is filtered by the umask. The name is an abbreviation of change mode.
- Syntax:
 - `chmod [options] mode[,mode] file1 [file2, ...]`
- Options:
 - R: recursive, i.e., files in the subdirectories
 - f: force

chmod: Octal modes

- The chmod numerical format accepts up to four octal digits.
- The three rightmost digits refer to permissions for the file owner, the group, and other users.

#	Permission	rwx
7	Read, write, execute	rwx
6	Read, write	rw-
5	Read, execute	r-x
4	Read only	r -
3	Write, execute	-wx
2	Write only	-w-
1	Execute	- - x
0	None	---

Chmod references

- The references (or classes) are used to distinguish the users to whom the permissions apply. If no references are specified it defaults to “all” but modifies only the permissions allowed by the `umask`. The references are represented by one or more of the following letters:

Reference	Class	Description
u	Owner	File's owner
g	group	users who are members of the file's group
o	Other	users who are neither the file's owner nor members of the file's group
a	All	all three of the above, same as ugo

chmod

- The chmod program uses an operator to specify how the modes of a file should be adjusted. The following operators are accepted:=

Operator	Description
+	adds the specified modes to the specified classes
-	removes the specified modes from the specified classes
=	the modes specified are to be made the exact modes for the specified classes

chown: change owner

- chown: command to change the owner of the file
- syntax:
- `chown [option] [owner][:group] file [file2 ...]`

- chgrp: command to change the group of the file
- syntax:
- `chgrp [option] group file`



CSC424 System Administration

Instructor: Dr. Hao Wu

Week 5 User Management, Access Control and Rootly
Powers

Process

- Fundamental to the structure of operating system

A *process* can be defined as:

A program in execution

An instance of a running program

The entity that can be assigned to, and executed on, a processor

A unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources

Components of a process

- An address space
 - a set of memory pages that the kernel has marked for the process's use
 - these pages contain the code and libraries, process's variable, its stacks, and various extra information needed by the kernel while the process is running
- A set of data structures within the kernel
 - The process's address space map
 - The current status of the process
 - The execution priority of the process
 - Information about the resources the process has used
 - Information about the files and network ports the process has opened
 - The process's signal mask
 - The owner of the process

Thread

- A thread is an execution context within a process
 - Every process has at least one thread
 - Some processes have many threads
 - Each thread has its own stack and CPU context but operates within the address space of its enclosing process
 - A process's threads can run simultaneously on different cores

Process information

- PID: process ID number
 - The kernel assigns a unique ID number to every process
 - Most commands that manipulate processes require you to specify a PID to identify the target of the operation.
 - PIDs are assigned in order as processes are created
- PPID: parent PID
 - Linux creates a new process in two separate steps
 - First, an existing process must clone itself to create a new process
 - The clone then exchanges the program it's running for a different one
 - When a process is cloned, the original process is referred to as the parent

Process information

- UID: real user ID
 - The user identification number of the person who created it
 - Usually, only the creator and the superuser can manipulate a process
- EUID: effective user ID
 - an extra UID that determines what resources and files a process has permission to access at any given moment
 - For most processes, the UID and EUID are the same
- UID v.s. EUID:
 - it's useful to maintain a distinction between identity and permissions
- GID and EGID: real and effective group ID

Process information

- Niceness
 - A process's scheduling priority determines how much CPU time it receives
 - The kernel also pays attention to an administratively set value that's usually called the "nice value" or "niceness". It specifies how nice you are planning to be to other users of the system.
- Control terminal
 - Most nondaemon processes have an associated control terminal
 - The control terminal determines the default linkages for the standard input, standard output, and standard error channels
 - It also distributes signals to processes in response to keyboard events such as <Control-C>

The life cycle of a process

- **Fork:** to create a new process, a process copies itself with the fork system call.
- After a fork, the child process often uses one of the **exec** family of routines to begin the execution of a new program.
- These calls change the program that the process is executing and reset the memory segments to a predefined initial state
- When the system boots, the kernel autonomously creates and installs several processes. The most notable of these is **init** or **systemd**, which is always process number 1.
- **init**(or **systemd**) also plays another important role in process management. When a process completes, it calls a routine named **_exit** to notify the kernel that it is ready to die.

Signals

- Signals are process-level interrupt requests. About thirty different kinds are defined, and they're used in a variety of ways:
 - They can be sent among processes as a means of communication
 - They can be sent by the terminal driver to kill, intercept, or suspend processes when keys such as <Control-C> and <Control-Z> are pressed
 - They can be sent by an administrator (with kill) to achieve various ends
 - They can be sent by the kernel when a process commits an infraction such as division by zero
 - They can be sent by the kernel to notify a process of an "interesting" condition such as the health of a child process or the availability of data on an I/O channel.

Signals every administrator should know

#	Name	Description	Default	Can catch?	Can block?	Dump core?
1	HUP	hangup	Terminate	Yes	Yes	No
2	INT	Interrupt	Terminate	Yes	Yes	No
3	QUIT	Quit	Terminate	Yes	Yes	Yes
9	KILL	Kill	Terminate	No	No	No
10	BUS	Bus error	Terminate	Yes	Yes	Yes
11	SEGV	Segmentation fault	Terminate	Yes	Yes	Yes
15	TERM	Software termination	Terminate	Yes	Yes	No
17	STOP	Stop	Stop	No	No	No
18	TSTP	Keyboard stop	Stop	Yes	Yes	No
19	CONT	Continue after stop	Ignore	Yes	No	No
28	WINCH	Window changed	Ignore	Yes	Yes	No
30	USR1	User-defined #1	Terminate	Yes	Yes	No
31	USR2	User-defined #2	Terminate	Yes	Yes	No

Signals

- The signals KILL, INT, TERM, HUP, and QUIT all sound as if they mean approximately the same thing, but their uses are actually quite different:
 - KILL: is unblockable and terminates a process at the kernel level. A process can never actually receive or handle this signal
 - INIT: is sent by the terminal driver when the user press <Control-C>. It's a request to terminate the current operation. Simple programs should quit or simply allow themselves to be killed, which is the default if the signal is not caught.
 - TERM: is a request to terminate execution completely. It's expected that the receiving process will clean up its state and exit.
 - QUIT: is similar to TERM, except that it defaults to producing a core dump if not caught.

Signals

- HUP: has two common interpretations.
 - First, it's understood as a reset request by many daemons. If a daemon is capable of rereadding its configuration file and adjusting to changes without restarting, a HUP can generally trigger this behavior
 - Second, HUP signals are sometimes generated by the terminal driver in an attempt to "clean up" the processes attached to a particular terminal.
 - **nohup**: makes a process running in background
 - What happened when you execute the following command:

```
nohup yes sir&
```

kill: send signals

- kill: a command most often been used to terminate a process.
It can send any signal, but by default it sends a TERM.
- Syntax:
 - `kill [-signal] pid`
- A kill without a signal number does not guarantee that the process will die, because the TERM signal can be caught, blocked, or ignored.
- Command:
 - `kill -9 pid`
- guarantees that the process will die

Send signals

- Command:
killall pname
 - kills processes by name.
-
- Command:
pkill -u pname
 - searches for processes by name (or other attributes, such as EUID) and sends the specified signal.

ps: monitor processes

- The ps command is the system administrator's main tool for monitoring processes.
- Most commonly used options:
 - a : show all processes
 - x : show even processes that don't have a control terminal
 - u : selects the "user oriented" output format

Example of "ps aux"

```
[root@node1 ~]# ps aux
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      1  0.0  0.6 128164  6824 ?        Ss   19:52  0:01 /usr/lib/systemd/
systemd --s
root      2  0.0  0.0      0     0 ?        S    19:52  0:00 [kthreadd]
root      3  0.0  0.0      0     0 ?        S    19:52  0:00 [ksoftirqd/0]
root      4  0.4  0.0      0     0 ?        R    19:52  0:20 [kworker/0:0]
root      5  0.0  0.0      0     0 ?        S<   19:52  0:00 [kworker/0:0H]
root      6  0.0  0.0      0     0 ?        S    19:52  0:00 [kworker/u2:0]
root      7  0.0  0.0      0     0 ?        S    19:52  0:00 [migration/0]
root      8  0.0  0.0      0     0 ?        S    19:52  0:00 [rcu_bh]
root      9  0.0  0.0      0     0 ?        R    19:52  0:00 [rcu_sched]
root     10  0.0  0.0      0     0 ?        S    19:52  0:00 [watchdog/0]
root     12  0.0  0.0      0     0 ?        S    19:52  0:00 [kdevtmpfs]
root     13  0.0  0.0      0     0 ?        S<   19:52  0:00 [netns]
root     14  0.0  0.0      0     0 ?        S    19:52  0:00 [khungtaskd]
root     15  0.0  0.0      0     0 ?        S<   19:52  0:00 [writeback]
```

Understand 'ps aux'

- USER: Username of the process's owner
- PID: process ID
- %CPU: Percentage of the CPU this process is using
- %MEM: Percentage of real memory this process is using
- VSZ: Virtual size of the process
- RSS: Resident set size (number of pages in memory)
- TTY: Control terminal ID
- STAT: Current process status
- TIME: CPU time the process has consumed
- COMMAND: Command name and arguments

Understand 'ps aux'

- Process status 'STAT':
 - R = Runnable
 - S = Sleeping(<20 sec)
 - Z = Zombie
 - D = In uninterruptible sleep
 - T = Traced or stopped
- Additional flags:
 - W = Process is swapped out
 - < = Process has higher than normal priority
 - N = Process has lower than normal priority
 - L = Some pages are locked in core
 - s = Process is a session leader

top: Interactive monitoring

- Command ps show you a snapshot of the system as it was at the time. Often, that limited sample is insufficient to convey the big picture of what's really going on.
- top is a sort of real-time version of ps that gives a regularly updated, interactive summary of process and their resource usage.

```
top - 21:19:12 up 1:26, 2 users, load average: 0.00, 0.01, 0.05
Tasks: 100 total, 3 running, 97 sleeping, 0 stopped, 0 zombie
%Cpu(s): 0.1 us, 0.2 sy, 0.0 ni, 99.7 id, 0.0 wa, 0.0 hi, 0.1 si, 0.0 st
KiB Mem : 1016232 total, 66320 free, 191644 used, 758268 buff/cache
KiB Swap: 839676 total, 839676 free, 0 used. 586052 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1	root	20	0	128164	6824	4060	S	0.0	0.7	0:01.39	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	S	0.0	0.0	0:00.24	ksoftirqd/0
5	root	0	-20	0	0	0	S	0.0	0.0	0:00.00	kworker/0:0H
6	root	20	0	0	0	0	S	0.0	0.0	0:00.28	kworker/u2:0
7	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
8	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_bh
9	root	20	0	0	0	0	R	0.0	0.0	0:00.65	rcu_sched

cron: schedule commands

- The cron daemon is the traditional tool for running commands on a predetermined schedule.
- It starts when the system boots and runs as long as the system is up.
- cron reads configuration files containing lists of command lines and times at which they are to be invoked.
- A cron configuration file is called a 'crontab', short for 'cron table'.
- Crontabs for individual users are stored under /var/spool/cron

The format of crontab files

- All the crontab files on a system share a similar format. Comments are introduced with a pound sign (#) in the first column of a line. Each non-comment line contains six fields and represents one command:
- *minute hour dom month weekday command*

Field	Description	Range
minute	Minute of the hour	0 to 59
hour	Hour of the day	0 to 23
dom	Day of the month	1 to 31
month	Month of the year	1 to 12
weekday	Day of the week	0 to 6 (0=Sunday)

The format of crontab files

- Each of the time-related fields can contain:
 - A star, which matches everything
 - A single integer, which matches exactly
 - Two integers separated by a dash, matching a range of values
 - A range followed by a slash and a step value, e.g. 1-10/2
 - A comma-separated list of integers or ranges, matching any value



CSC424 System Administration

Instructor: Dr. Hao Wu

Week 6 Bash Scripting - 1

Outline

- Filename Matching
- Regular Expression
- Linux Text Processing Tools
- Variables
- Arithmetic
- Control Flow
- Loops
- Function

Filename Matching

- Wildcards (meta characters) are symbols or special characters that represent other characters. You can use them with any command such as ls, cp, mv, rm to perform file operations matching a given criteria.
- Three main wildcards in Linux:
 - * : matches one or more occurrences of any character, including no character
 - ? : represents or matches a single occurrence of any character
 - [] : matches any occurrence of character enclosed in the square bracket. It is possible to use different types of characters (alphanumeric characters): numbers, letters, other special characters etc.)

Zero or multiple characters matching (*)

1. If we want to list all the files start with 'a':

ls a*

```
[hao@localhost sbin]$ ls a*
accessdb      agetty          arpd        auditd       authconfig
addgnupghome  alternatives    arp[ing]   augenrules  authconfig-tui
addpart       anacron         audispd    aureport    autrace
adduser       applygnupgdefaults auditctl   ausearch    avcstat
```

2. If we want to list all the files end with 'a'

ls *a

```
[hao@localhost sbin]$ ls *a
thin_delta  xfs_quota
```

Zero or multiple characters matching

1. If we want to list all the files start with 'a' and end with 't':

ls a*t

```
[hao@localhost sbin]$ ls a*t  
addpart  aureport  avcstat
```

Single character matching (?)

1. If we want to list a file whose name contains 2 characters and starts with 's'

ls s?

```
[hao@localhost sbin]$ ls s?  
ss
```

2. If we want to list a file whose name contains 2 characters and ends with 's'

ls ?s

```
[hao@localhost sbin]$ ls ?s  
ss
```

Single character matching (?)

1. If we want to list a file whose name contains 2 characters
ls ??

```
[hao@localhost sbin]$ ls ??  
ip ss tc
```

Range of character matching ([])

1. If we want to list files whose names start with 'a' or 'b'

ls [ab]*

```
[hao@localhost sbin]$ ls [ab]*
```

accessdb	audispd	badblocks	btrfs-convert
addgnupghome	auditctl	biosdecode	btrfs-debug-tree
addpart	auditd	biosdevname	btrfs-find-root
adduser	augenrules	blkdeactivate	btrfs-image
agetty	aureport	blkdiscard	btrfs-map-logical
alternatives	ausearch	blkid	btrfs-select-super
anacron	authconfig	blockdev	btrfstune
applygnupgdefaults	authconfig-tui	bridge	btrfs-zero-log
arpd	autrace	btrfs	build-locale-
archive			
arping	avcstat	btrfsck	

Range of character matching ([])

1. If we want to list files whose names contain numbers

```
ls *[0-9]*
```

```
[hao@localhost sbin]$ ls *[0-9]*
dumpe2fs      glibc_post_upgrade.x86_64    grub2-setpassword      mkfs.ext3
e2freefrag    grub2-bios-setup              grub2-sparc64-setup    mkfs.ext4
e2fsck        grub2-get-kernel-settings    iconvconfig.x86_64     pam_tally2
e2image       grub2-install                intel-microcode2ucode ping6
e2label       grub2-mkconfig               ip6tables                resize2fs
e2undo        grub2-ofpathname           ip6tables-restore
sasldblistusers2
e4defrag      grub2-probe                 ip6tables-save          saslpasswd2
fsck.ext2     grub2-reboot                killall5
fsck.ext3     grub2-rpm-sort             mke2fs
fsck.ext4     grub2-set-default          mkfs.ext2

```

Range of character matching ([])

1. If we want to list files whose names end with 'a-d'

```
ls *[a-d]
```

```
[hao@localhost sbin]$ ls *[a-d]
accessdb      dmfilemapd          lid           nl-class-add    thin_delta
arpd          firewalld          lpasswd       nl-cls-add     tuned
audispd       fxload            lsmod         nl-qdisc-add   unix_chkpwd
auditd        genhostid         luseradd     parted        useradd
blkdiscard    groupadd          lusermod     plymouthd    usermod
blkid         groupmod          lvextend     rdisc         vgextend
chpasswd      grub2-setpassword insmod       rmmod        xfs_db
chronyd       insmod           lvmtrash    rsyslogd    xfs_quota
crond         kexec            lvmpoll     selinuxenabled zic
depmod        lgroupadd         lvmsadc    mkdumpfd   sshd
dmeventd      lgroupmod         mklost+found tc
```

Regular Expression

- Regular expressions are standardized patterns that parse and manipulate text. For example, the regular expression:
 - I sent you a che(quelck) for the gr[ae]y-colou?red alumni?um
 - matches sentences that use either American or British spelling conventions.
- Regular expressions are supported by most modern languages.
- Wildcard is not a form of regular expression!

Special characters in regular expression

Symbol	What it matches or does
.	Matches any character
[]	Matches any character from a given set
^	Matches the beginning of a line
\$	Matches the end of a line
\w	Matches any "word" character
\s	Matches any whitespace character
\d	Matches any digits
	Matches either the element to its left or right
?	Allows zero or one match of the preceding element
*	Allows zero, one, or many matches of the preceding elements
+	Allows one or more matches of the preceding element
{n}	Matches exactly n instances of the preceding elements
{min,}	Matches at least min instance of
{min, max}	Matches any number of instances from min to max

Text Processing Tools: grep

- grep: print lines matching a pattern
- Syntax:
 grep [option] pattern [file]
- Commonly used options:
 - i: ignore case distinctions in both the pattern and the input files.
 - n: Prefix each line of output with the line number within its input file
 - v: Invert the sense of matching, to select non-matching lines
- In basic regular expressions the meta-characters ?, +, {, |, (, and) lose their special meaning; instead use the backslashed versions \?, \+, \{, \|, \(), and \).

grep examples:

- We will first create a file for demonstration:
ps aux > grep.demo
- If we want to find the lines contain 'root'
grep root grep.demo

```
[hao@localhost ~]$ grep root grep.demo
root      1 0.0 0.6 128164 6824 ?          Ss    Feb26   0:01 /usr/lib/systemd/
systemd --switched-root --system --deserialize 21
root      2 0.0 0.0      0      0 ?          S     Feb26   0:00 [kthreadd]
root      3 0.0 0.0      0      0 ?          S     Feb26   0:00 [ksoftirqd/0]
root      5 0.0 0.0      0      0 ?          S<    Feb26   0:00 [kworker/0:0H]
root      6 0.0 0.0      0      0 ?          S     Feb26   0:00 [kworker/u2:0]
root      7 0.0 0.0      0      0 ?          S     Feb26   0:00 [migration/0]
root      8 0.0 0.0      0      0 ?          S     Feb26   0:00 [rcu_bh]
```

grep examples:

- If we want to find the lines do not contain 'root'
 grep -v root grep.demo

```
[hao@localhost ~]$ grep -v root grep.demo
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
polkitd    626  0.0  1.2 534892 12884 ?          Ssl  Feb26   0:00 /usr/lib/polkit-1/
polkitd --no-debug
dbus       627  0.0  0.1 32776  1856 ?          Ssl  Feb26   0:00 /bin/dbus-daemon --
system --address=systemd: --nofork --nopidfile --systemd-activation
chrony     635  0.0  0.1 115640  1780 ?          S    Feb26   0:00 /usr/sbin/chronyd
postfix   1113  0.0  0.3 89716   4028 ?          S    Feb26   0:00 qmgr -l -t unix -u
hao        2160  0.0  0.2 145700  2400 ?          S    Feb26   0:00 sshd: hao@pts/0
hao        2161  0.0  0.2 115392  2132 pts/0      Ss   Feb26   0:00 -bash
postfix   3031  0.0  0.3 89648   4004 ?          S    00:05   0:00 pickup -l -t unix -
u
hao        3070  0.0  0.1 151064  1808 pts/0      R+   00:15   0:00 ps aux
```

grep examples:

- If we want to find the lines contain 'root' and 'xfs'
grep root grep.demo |grep xfs

```
[hao@localhost ~]$ grep root grep.demo |grep xfs
root      388  0.0  0.0      0      0 ?          S<    Feb26   0:00  [xfsalloc]
root      389  0.0  0.0      0      0 ?          S<    Feb26   0:00  [xfs_mru_cache]
root      390  0.0  0.0      0      0 ?          S<    Feb26   0:00  [xfs_buf/dm-0]
root      391  0.0  0.0      0      0 ?          S<    Feb26   0:00  [xfs_data/dm-0]
root      392  0.0  0.0      0      0 ?          S<    Feb26   0:00  [xfs_conv/dm-0]
root      393  0.0  0.0      0      0 ?          S<    Feb26   0:00  [xfs_cil/dm-0]
root      394  0.0  0.0      0      0 ?          S<    Feb26   0:00  [xfs_reclaim/dm-]
root      395  0.0  0.0      0      0 ?          S<    Feb26   0:00  [xfs_log/dm-0]
root      396  0.0  0.0      0      0 ?          S<    Feb26   0:00  [xfs_eofblocks/d]
root      397  0.0  0.0      0      0 ?          S     Feb26   0:01  [xfsaild/dm-0]
root      543  0.0  0.0      0      0 ?          S<    Feb26   0:00  [xfs_buf/sda1]
root      546  0.0  0.0      0      0 ?          S<    Feb26   0:00  [xfs_data/sda1]
root      547  0.0  0.0      0      0 ?          S<    Feb26   0:00  [xfs_conv/sda1]
root      549  0.0  0.0      0      0 ?          S<    Feb26   0:00  [xfs_cil/sda1]
root      551  0.0  0.0      0      0 ?          S<    Feb26   0:00  [xfs_reclaim/sda]
root      553  0.0  0.0      0      0 ?          S<    Feb26   0:00  [xfs_log/sda1]
root      555  0.0  0.0      0      0 ?          S<    Feb26   0:00  [xfs_eofblocks/s]
root      566  0.0  0.0      0      0 ?          S     Feb26   0:00  [xfsaild/sda1]
```

grep examples:

- If we want to find the lines contain 'root' or 'xfs'
grep "root\|xfs" grep.demo

```
[hao@localhost ~]$ grep "root\|xfs" grep.demo
root      1 0.0 0.6 128164 6824 ?          Ss    Feb26   0:01 /usr/lib/systemd/
systemd --switched-root --system --deserialize 21
root      2 0.0 0.0      0      0 ?          S     Feb26   0:00 [kthreadd]
root      3 0.0 0.0      0      0 ?          S     Feb26   0:00 [ksoftirqd/0]
root      5 0.0 0.0      0      0 ?          S<    Feb26   0:00 [kworker/0:0H]
root    389 0.0 0.0      0      0 ?          S<    Feb26   0:00 [xfs_mru_cache]
root    390 0.0 0.0      0      0 ?          S<    Feb26   0:00 [xfs-buf/dm-0]
root    391 0.0 0.0      0      0 ?          S<    Feb26   0:00 [xfs-data/dm-0]
root    392 0.0 0.0      0      0 ?          S<    Feb26   0:00 [xfs-conv/dm-0]
root    393 0.0 0.0      0      0 ?          S<    Feb26   0:00 [xfs-cil/dm-0]
root    394 0.0 0.0      0      0 ?          S<    Feb26   0:00 [xfs-reclaim/dm-]
root    395 0.0 0.0      0      0 ?          S<    Feb26   0:00 [xfs-log/dm-0]
```

grep examples:

- If we want to find the lines contain 'ss'
grep "ss" grep.demo

```
[hao@localhost ~]$ grep ss grep.demo
dbus      627  0.0  0.1  32776  1856 ?          Ssl  Feb26  0:00 /bin/dbus-daemon --
system --address=systemd: --nofork --nopidfile --systemd-activation
root     1009  0.0  0.4 105996  4072 ?          Ss   Feb26  0:00 /usr/sbin/sshd -D
root     2156  0.0  0.5 145700  5276 ?          Ss   Feb26  0:00 sshd: hao [priv]
hao      2160  0.0  0.2 145700  2400 ?          S    Feb26  0:00 sshd: hao@pts/0
```

grep examples:

- If we want to find the lines contain 'ss' or 'SS' or 'Ss' or 'sS'
grep -i "ss" grep.demo

```
[hao@localhost ~]$ grep -i ss grep.demo
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      1  0.0  0.6 128164  6824 ?        Ss   Feb26   0:01 /usr/lib/systemd/
systemd --switched-root --system --deserialize 21
root     464  0.0  0.3 34988  3788 ?        Ss   Feb26   0:00 /usr/lib/systemd/
systemd-journald
root     483  0.0  0.4 129552  4136 ?        Ss   Feb26   0:00 /usr/sbin/lvmetad -f
root     495  0.0  0.4 46772  4824 ?        Ss   Feb26   0:00 /usr/lib/systemd/
systemd-udevd
root     624  0.0  0.1 24204  1708 ?        Ss   Feb26   0:00 /usr/lib/systemd/
systemd-logind
polkitd  626  0.0  1.2 534892 12884 ?        Ssl  Feb26   0:00 /usr/lib/polkit-1/
polkitd --no-debug
```

grep examples:

- If we want to find the lines contain 'ps' from multiple files:
- We make another file called grep.demo1 by using command:
 ps > grep.demo1
- grep ps grep.demo grep.demo1

```
[hao@localhost ~]$ grep ps grep.demo grep.demo1
grep.demo:root          39  0.0  0.0      0      0 ?          S<   Feb26
0:00 [kpsmoused]
grep.demo:hao         3070  0.0  0.1 151064  1808 pts/0      R+   00:15
0:00 ps aux
grep.demo1: 3180 pts/0    00:00:00 ps
```

Using grep and regular expression

- grep command is usually been used with regular expression.
- If we want to list all the processes started in Feb:
grep 'Feb..' grep.demo

```
[hao@localhost ~]$ grep 'Feb..' grep.demo
root      1 0.0 0.6 128164 6824 ?          Ss  Feb26   0:01 /usr/lib/
systemd/systemd --switched-root --system --deserialize 21
root      2 0.0 0.0      0      0 ?          S  Feb26   0:00 [kthreadd]
root      3 0.0 0.0      0      0 ?          S  Feb26   0:00 [ksoftirqd/0]
root      5 0.0 0.0      0      0 ?          S< Feb26   0:00 [kworker/0:0H]
root      6 0.0 0.0      0      0 ?          S  Feb26   0:00 [kworker/u2:0]
root      7 0.0 0.0      0      0 ?          S  Feb26   0:00 [migration/0]
```

Using grep and regular expression

- grep command is usually been used with regular expression.
- If we want to list all the lines start with p :
grep '^p' grep.demo

```
[hao@localhost ~]$ grep '^p' grep.demo
polkitd      626  0.0  1.2 534892 12884 ?          Ssl  Feb26   0:00 /usr/lib/polkit-1/
polkitd --no-debug
postfix     1113  0.0  0.3  89716   4028 ?          S    Feb26   0:00 qmgr -l -t unix -u
postfix     3031  0.0  0.3  89648   4004 ?          S    00:05   0:00 pickup -l -t unix -u
```

Using grep and regular expression

- grep command is usually been used with regular expression.
- If we want to list all the lines end with a number :
grep '[0-9]\$' grep.demo

```
[hao@localhost ~]$ grep '^p' grep.demo
polkitd      626  0.0  1.2 534892 12884 ?
polkitd --no-debug
postfix     1113  0.0  0.3 89716   4028 ?
postfix     3031  0.0  0.3 89648   4004 ?
[hao@localhost ~]$ grep '[0-9]$' grep.demo
root         1  0.0  0.6 128164   6824 ?
systemd --switched-root --system --deserialize 21
root        1921  0.0  1.5 113372  15912 ?
root        2101  0.0  1.5 113372  15896 ?
hao          2160  0.0  0.2 145700  2400 ?

[hao@localhost ~]$ grep 'Ssl' grep.demo
Ssl  Feb26  0:00 /usr/lib/polkit-1/
S     Feb26  0:00 qmgr -l -t unix -u
S     00:05  0:00 pickup -l -t unix -u

[hao@localhost ~]$ grep 'Ss' grep.demo
Ss  Feb26  0:01 /usr/lib/systemd/
S     Feb26  0:00 /sbin/dhclient -d -q -
sf /usr/libexec/nm-dhcp-helper -pf /var/run/dhclient-enp0s3.pid -lf /var/lib/
NetworkManager/dhclient-dd33654a-838f-4664-abb4-38cf1b5980ac-enp0s3.lease -cf /var/lib/
NetworkManager/dhclient-enp0s3.conf enp0s3
S     Feb26  0:00 /sbin/dhclient -d -q -
sf /usr/libexec/nm-dhcp-helper -pf /var/run/dhclient-enp0s8.pid -lf /var/lib/
NetworkManager/dhclient-fad5ea84-d774-3cc6-a5b0-842365440f08-enp0s8.lease -cf /var/lib/
NetworkManager/dhclient-enp0s8.conf enp0s8
S     Feb26  0:00 sshd: hao@pts/0
```

Exercise

- Please create a file contain the following information:

ID	Name	City	Advisor	DOB	Year	GPA
282102	Andy	New Haven	Lisa	01/23/99	Sr	3.5
281734	Zoe	Bridgeport	Hao	05/14/00	Jr	3.2
292342	Alex	Cheshire	hao	12/25/01	Fresh	2.0
284323	Eric	Hartford	lisa	02/26/00	Soph	2.5
274342	Bill	Orange	Imad	04/12/99	Sr	4.0
293232	Alice	New Haven	IMAD	07/04/98	Jr	1.8
213852	Lee	bridgeport	lisa	06/22/02	Fresh	3.8

Questions:

- Find the students whose advisor is Lisa
- Find the students come from orange
- Find the students come from New Haven
- Find the students do not come from New Haven
- Find the students born after 2000
- Find the students have GPA higher than 3
- Find the students whose advisor is Hao and have GPA higher than 3
- Find the students with their ID start with 28

awk: text processing and data extraction tool

- awk: is a language for processing text files
- Syntax:
`awk [options] 'command' input_file`
- Commonly used options:
 - F: field separator
- commands:
 - build-in functions are enclosed in {}

awk: text processing and data extraction tool

- Example: separate field in /etc/passwd file and print out username:
- `awk -F: '{print "username:" $1}' /etc/passwd`
- `$1`: the first field in the line
- `$0`: the whole line

```
[root@localhost ~]# awk -F: '{print "username:" $1}' /etc/passwd
username:root
username:bin
username:daemon
username:adm
username:lp
username:sync
username:shutdown
```

awk with regular expression

- Example: separate field in /etc/passwd file and print out root :
- awk -F: '/root/{print "username:" \$1}' /etc/passwd

```
[root@localhost ~]# awk -F: '/root/{print "username:" $1}' /etc/
passwd
username:root
username:operator
```

awk with regular expression

- Example: separate field in /etc/passwd file and print out root :
- we only want first line
- `awk -F: '$1=="root"{print "username:" $1}' /etc/passwd`

```
[root@localhost ~]# awk -F: '$1=="root"{print "username:" $1}' /etc/passwd
username:root
```

Exercise:

- Print the students name whose advisor is Lisa
- Print the name and DOB of students who come from orange
- Print the name and GPA of students who come from New Haven
- Print the name and hometown of students do not come from New Haven
- Print the DOB of students born after 2000
- Print the name, GPA, and advisor of students have GPA higher than 3
- Print the GPA and name of students whose advisor is Hao and have GPA higher than 3
- Print the name and year of students with GPA less than 3

Solution for Exercise 1:

```
grep -i lisa students
grep -i orange students
grep "New Haven" students
grep -v "New Haven" students
grep "[0-9][0-9]/[0-9][0-9]/0[0-9]" students
grep "[3-9]\.[0-9]" students
grep "[3-9]\.[0-9]" students | grep -i hao
grep "^28" students
```

Solution for Exercise 2:

```
awk '/[Ll]isa/{print $2}' students
```

I can also print out the header before names:

```
head -1 students|awk '{print $2}';awk '/[lL]isa/{print  
$2}' students
```

```
awk '/Orange/{print $2" "$5}' students
```

```
awk '/New Haven/{print $2" "$8}' students
```

```
grep -v "New Haven" students|awk '{print $2" "$3}'
```

```
awk '/[0-9][0-9]\.[0-9][0-9]\.[0-9]/{print $5}'  
students
```

```
awk '/[34]\.[0-9]$/ {if ($4=="Haven") print $2" "$5" "$8;
```

```
else print $2" "$4" "$7}' students
```

```
awk '/[34]\.[0-9]$/ {if ($4=="Hao" || $4=="hao") print $7"  
"$2;}' students
```

```
awk '/[0-2]\.[0-9]$/ {if($4=="Haven") print $2" "$6;  
else print $2" "$5}' students
```



CSC424 System Administration

Instructor: Dr. Hao Wu

Week 7 Bash Scripting - 2

Outline

- Filename Matching
- Regular Expression
- Linux Text Processing Tools
- **Variables**
- **Arithmetic**
- Control Flow
- Loops
- Function

Your first bash script

- Create your first script named "hello"
- Type the following inside it:

```
#!/bin/bash  
echo "Hello World"
```

- The first line tells Linux to use the bash interpreter to run this script.
- In order to run the script, we need to make the script executable:

```
chmod +x hello
```

- To run the script:

```
./hello
```

Variables

- A variable is temporary store for a piece of information. There are two actions we may perform for variables:
 - Assign a value to a variable.
 - var=value (**no space before or after!**)
 - Read the value from a variable.
 - \$var or \${var}
- Uninitialized variables have no value
- Variables are untyped, interpreted based on context

Single and Double Quote

- When assigning character data containing spaces or special characters, the data must be enclosed in either single or double quotes.
- Using **double quotes** to show a string of characters will allow any variables in the quotes to be resolved

```
[hao@node1 ~]$ var="test string"
[hao@node1 ~]$ newvar="Value of var is $var"
[hao@node1 ~]$ echo $newvar
Value of var is test string
```

Single and Double Quote

- Using **single quotes** to show a string of characters will not allow variable resolution

```
[hao@node1 ~]$ var='test string'  
[hao@node1 ~]$ newvar='Value of var is $var'  
[hao@node1 ~]$ echo $newvar  
Value of var is $var
```

Environmental Variables

- There are two types of variables:
 - Local variables
 - Environmental variables
- Environmental variables are set by the system and can usually be found by using the env command.
- Shell variables are generally not visible to programs
- All environment variables are also shell variables, but not vice versa
- Environmental variables hold special values. For instance:

```
[hao@node1 ~]$ echo $PATH  
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin  
[hao@node1 ~]$ echo $SHELL  
/bin/bash
```

The export command

- The export command puts a variable into the environment so it will be accessible to child processes. For instance:

```
[hao@node1 ~]$ bash #Run a child shell  
[hao@node1 ~]$ echo $var #No value in var  
  
[hao@node1 ~]$ exit #Return to parent  
exit  
[hao@node1 ~]$ echo $var  
test string  
[hao@node1 ~]$ export var  
[hao@node1 ~]$ bash  
[hao@node1 ~]$ echo $var  
test string #It's there
```

The export command

- If the child modifies var, it will not modify the parent's original value.

```
[hao@node1 ~]$ export var
[hao@node1 ~]$ bash
[hao@node1 ~]$ echo $var
test string                                #It's there
[hao@node1 ~]$ var="change me"
[hao@node1 ~]$ echo $var
change me
[hao@node1 ~]$ exit
exit
[hao@node1 ~]$ echo $var
test string
```

Environment Variables

- LOGNAME: contains the user name
- HOSTNAME: contains the computer name.
- HOME: home directory
- PATH: list of directories to search
- TERM: type of terminal (vt100, ...)
- TZ: timezone (e.g., US/Eastern)
- RANDOM: random number generator
- PS1: sequence of characters shown before the prompt

Environment Variables

- PS1: sequence of characters shown before the prompt:

\t hour
\d date
\w current directory
\w last part of the current directory
\u user name
\\$ prompt character

- Example:

```
[hao@~]$PS1="This is \u]\$"  
[This is hao]$
```

Shell variables

Parameter	Meaning
\$0	Name of the current shell script
\$1-\$9	Positional parameters 1 through 9
\$#	The number of positional parameters
\$*	All positional parameters, “\$*” is one string
\$@	All positional parameters, “\$@” is a set of strings
\$?	Return status of most recently executed command
\$\$	Process id of current process

Shell variables: Command Line Arguments

- The 'set' command can be used to assign values to positional parameters

```
[hao@node1 ~]$ set arg1 arg2 arg3 arg4
[hao@node1 ~]$ echo $0
bash
[hao@node1 ~]$ echo $*
arg1 arg2 arg3 arg4
[hao@node1 ~]$ echo $#
4
[hao@node1 ~]$ echo $1
arg1
[hao@node1 ~]$ echo $3 $2
arg3 arg2
[hao@node1 ~]$ echo $$
2725
```

Array variables

- An array is a variable containing multiple values.
- Any variable may be used as an array.
- There is no maximum limit to the size of an array, nor any requirement that member variables be indexed or assigned contiguously.
- Arrays are zero-based: the first element is indexed with the number 0.
- Array variable assignment:
 - **array=(var1 var2 var3 ...)**

Array variables

- Accessing the value in array variables
 - In order to refer to the content of an item in an array, use curly braces

```
[hao@node1 ~]$ a=(one two three four) #create an array
[hao@node1 ~]$ echo ${a[*]}           #get all values
one two three four
[hao@node1 ~]$ echo ${a[2]}           #get the third value
three
[hao@node1 ~]$ a[2]=2                #change the third value
[hao@node1 ~]$ echo ${a[*]}
one two 2 four
[hao@node1 ~]$ a[4]=five             #add a new value to array
[hao@node1 ~]$ echo ${a[*]}
one two 2 four five
[hao@node1 ~]$ echo ${a[@]}
one two 2 four five
```

Array variables

- Deleting the value in array variables

```
[hao@node1 ~]$ unset a[1]
[hao@node1 ~]$ echo ${a[*]}
one 2 four five
[hao@node1 ~]$ unset a
[hao@node1 ~]$ echo ${a[*]}
```

Manipulating Strings

- Bash supports a number of string manipulation operations.

`${#string}` gives the string length

`${string:position}` extracts sub-string from \$string at \$position

`${string:position:length}` extracts \$length characters of sub-string from \$string at \$position

```
[hao@node1 ~]$ st=0123456789
[hao@node1 ~]$ echo ${#st}
10
[hao@node1 ~]$ echo ${st:6}
6789
[hao@node1 ~]$ echo ${st:6:2}
67
```

User Input

- shell allows to prompt for user input
- Syntax:
 - `read varname [more vars]`
 - or
 - `read -p "prompt" varname [more vars]`
- words entered by user are assigned to
- **varname** and “**more vars**”
- last variable gets rest of input line

User Input Example

```
#! /bin/sh
read -p "enter your name: " first last

echo "First name: $first"
echo "Last name: $last"
```

Command Substitution

- The backquote `` is different from the single quote ''. It is used for command substitution: `command`

```
[hao@node1 ~]$ LIST=`ls -a`
[hao@node1 ~]$ echo $LIST
. . . .bash_history .bash_logout .bash_profile .bashrc .ca
che .config hello .viminfo
```

- We can perform the command substitution by \$(command)

```
[hao@node1 ~]$ LIST=$(ls -a)
[hao@node1 ~]$ echo $LIST
. . . .bash_history .bash_logout .bash_profile .bashrc .ca
che .config hello .viminfo
```

Arithmetic Evaluation

- The `let` statement can be used to do mathematical functions:

```
[hao@node1 ~]$ let X=5+4*8  
[hao@node1 ~]$ echo $X  
37  
[hao@node1 ~]$ let Y=X*2-21  
[hao@node1 ~]$ echo $Y  
53
```

- An arithmetic expression can be evaluated by `$[expression]` or `$((expression))`

```
[hao@node1 ~]$ echo "$((121+111))"  
232  
[hao@node1 ~]$ x=$((12*12)  
[hao@node1 ~]$ echo ${x*2}  
288
```

Arithmetic Evaluation

- Available operators: +, -, /, *, %
- Example

```
[hao@node1 ~]$ cat arithmetic.sh
#!/bin/bash
echo -n "Enter the first number: "; read x
echo -n "Enter the second number: "; read y
add=$((x + y))
sub=$((x - y))
mul=$((x * y))
div=$((x / y))
mod=$((x % y))
# print out the answers:
echo "Sum: $add"
echo "Difference: $sub"
echo "Product: $mul"
echo "Quotient: $div"
echo "Remainder: $mod"
```



CSC424 System Administration

Instructor: Dr. Hao Wu

Week 8 Bash Scripting - 3

Outline

- Filename Matching
- Regular Expression
- Linux Text Processing Tools
- Variables
- Arithmetic
- **Control Flow**
- Loops
- Function

Conditional Statements

- **Conditionals** let us decide whether to perform an action or not, this decision is taken by evaluating an expression. The most basic form is:

```
if [ expression ];
```

```
then
```

```
    statements
```

```
elif [ expression ];
```

```
then
```

```
    statements
```

```
else
```

```
    statements
```

```
fi
```

- the **elif** (else if) and **else** sections are optional

- Put spaces after [and before], and around the operators and operands.

Expressions

- An expression can be: String comparison, Numeric comparison, File operators and Logical operators and it is represented by [expression]:
- String Comparisons:
 - = compare if two strings are equal
 - != compare if two strings are not equal
 - n evaluate if string length is greater than zero
 - z evaluate if string length is equal to zero
- Examples:
 - [s1 = s2] (true if s1 same as s2, else false)
 - [s1 != s2] (true if s1 not same as s2, else false)
 - [s1] (true if s1 is not empty, else false)
 - [-n s1] (true if s1 has a length greater than 0, else false)
 - [-z s2] (true if s2 has a length of 0, otherwise false)

Expressions

- Number Comparisons:
 - eq compare if two numbers are equal
 - ge compare if one number is greater than or equal to a number
 - le compare if one number is less than or equal to a number
 - ne compare if two numbers are not equal
 - gt compare if one number is greater than another number
 - lt compare if one number is less than another number
- Examples:

[n1 -eq n2]	(true if n1 same as n2, else false)
[n1 -ge n2]	(true if n1 greater then or equal to n2, else false)
[n1 -le n2]	(true if n1 less then or equal to n2, else false)
[n1 -ne n2]	(true if n1 is not same as n2, else false)
[n1 -gt n2]	(true if n1 greater then n2, else false)
[n1 -lt n2]	(true if n1 less then n2, else false)

Examples

```
$ cat user.sh
#!/bin/bash
echo -n "Enter your login name: "
read name
if [ "$name" = "$USER" ];
then
    echo "Hello, $name. How are you today ?"
else
    echo "You are not $USER, so who are you ?"
fi
```

Examples

```
$ cat number.sh
#!/bin/bash
echo -n "Enter a number 1 < x < 10: "
read num
if [ "$num" -lt 10 ]; then
    if [ "$num" -gt 1 ]; then
        echo "$num*$num=\$((\$num*\$num))"
    else
        echo "Wrong insertion !"
    fi
else
    echo "Wrong insertion !"
fi
```

Expressions

- Files operators:
 - d** check if path given is a **directory**
 - f** check if path given is a **file**
 - e** check if file name **exists**
 - r** check if **read permission** is set for file or directory
 - s** check if a file has a **length greater than 0**
 - w** check if **write permission** is set for a file or directory
 - x** check if **execute permission** is set for a file or directory

Expressions

- Examples:

- [-d fname] (true if fname is a directory, otherwise false)
- [-f fname] (true if fname is a file, otherwise false)
- [-e fname] (true if fname exists, otherwise false)
- [-s fname] (true if fname length is greater than 0, else false)
- [-r fname] (true if fname has the read permission, else false)
- [-w fname] (true if fname has the write permission, else false)
- [-x fname] (true if fname has the execute permission, else false)

Example

```
#!/bin/bash
echo "Enter a filename: "
read filename
if [ ! -r "$filename" ]
then
    echo "File is not read-able"
exit 1
fi
```

Example

```
#!/bin/bash
echo "Enter a path: "; read x
if cd $x; then
    echo "I am in $x and it contains"; ls
else
    echo "The directory $x does not exist";
    exit 1
fi
```

Exercise

- Write a shell script which:
 - accepts a file name
 - checks if file exists
 - if file exists, copy the file to the same name + .bak + the current date (if the backup file already exists ask if you want to replace it).
- When done you should have the original file and one with a .bak at the end.

Expressions

- Logical operators:
 - ! negate (**NOT**) a logical expression
 - && logically **AND** two logical expressions
 - || logically **OR** two logical expressions

Example: Using the ! operator

```
#!/bin/bash
```

```
read -p "Enter years of work: " Years
if [ ! "$Years" -lt 20 ]; then
    echo "You can retire now."
else
    echo "You need 20+ years to retire"
fi
```

Example: Using the && operator

```
#!/bin/bash

Bonus=500
read -p "Enter Status: " Status
read -p "Enter Shift: " Shift
if [[ "$Status" = "H" && "$Shift" = 3 ]]
then
    echo "shift $Shift gets \$Bonus bonus"
else
    echo "only hourly workers in"
    echo "shift 3 get a bonus"
fi
```

Example: Using the || operator

```
#!/bin/bash
```

```
read -p "Enter calls handled:" CHandle
read -p "Enter calls closed: " CClose
if [[ "$CHandle" -gt 150 || "$CClose" -gt 50 ]]
then
    echo "You are entitled to a bonus"
else
    echo "You get a bonus if the calls"
    echo "handled exceeds 150 or"
    echo "calls closed exceeds 50"
fi
```

Case Statement

- Used to execute statements based on specific values. Often used in place of an if statement if there are a large number of conditions.
- Value used can be an **expression**
- each set of statements must be ended by a **pair of semicolons**;
- a ***)** is used to accept any value not matched with list of values

```
case $var in
  val1)
    statements;;
  val2)
    statements;;
  *)
    statements;;
esac
```

Example (case.sh)

```
$ cat case.sh
#!/bin/bash
echo -n "Enter a number 1 < x < 10: "
read x
case $x in
    1) echo "Value of x is 1.";;
    2) echo "Value of x is 2.";;
    3) echo "Value of x is 3.";;
    4) echo "Value of x is 4.";;
    5) echo "Value of x is 5.";;
    6) echo "Value of x is 6.";;
    7) echo "Value of x is 7.";;
    8) echo "Value of x is 8.";;
    9) echo "Value of x is 9.";;
    0 | 10) echo "wrong number.";;
    *) echo "Unrecognized value.";;
esac
```

Loop Statements

- The **for structure** is used when you are looping through a range of variables.

```
for var in list  
do  
    statements  
done
```

- statements are executed with var set to each value in the list.

Example: for loop

```
#!/bin/bash  
let sum=0  
for num in 1 2 3 4 5  
  do  
    let "sum = $sum + $num"  
  done  
echo $sum
```

```
#!/bin/bash  
for x in paper pencil pen  
do  
  echo "The value of variable x is: $x"  
  sleep 1  
done
```

Iteration Statements

- if the list part is left off, var is set to each parameter passed to the script (\$1, \$2, \$3,...)

```
$ cat for1.sh
#!/bin/bash
for x
do
    echo "The value of variable x is: $x"
    sleep 1
done
$ for1.sh arg1 arg2
The value of variable x is: arg1
The value of variable x is: arg2
```

Example (old.sh)

```
$ cat old.sh
#!/bin/bash
# Move the command line arg files to old directory.
if [ $# -eq 0 ] #check for command line arguments
then
    echo "Usage: $0 file ..."
    exit 1
fi
if [ ! -d "$HOME/old" ]
then
    mkdir "$HOME/old"
fi
echo The following files will be saved in the old directory:
echo $*
for file in $* #loop through all command line arguments
do
    mv $file "$HOME/old/"
    chmod 400 "$HOME/old/$file"
done
ls -l "$HOME/old"
```

Example (args.sh)

```
$ cat args.sh
#!/bin/bash
# Invoke this script with several arguments: "one two three"
if [ ! -n "$1" ]; then
    echo "Usage: $0 arg1 arg2 ..." ; exit 1
fi
echo ; index=1 ;
echo "Listing args with \"\$*\":"
for arg in "$*" ;
do
    echo "Arg $index = $arg"
    let "index+=1" # increase variable index by one
done
echo "Entire arg list seen as single word."
echo ; index=1 ;
echo "Listing args with \"\$@\":"
for arg in "$@" ; do
    echo "Arg $index = $arg"
    let "index+=1"
done
echo "Arg list seen as separate words." ; exit 0
```

Using Arrays with Loops

- We can combine arrays with loops using a for loop:

```
for x in ${arrayname[*]}
do
    ...
done
```

A C-like for loop

- An **alternative** form of the **for** structure is

```
for (( EXPR1 ; EXPR2 ; EXPR3 ))  
do  
    statements  
done
```

- First, the arithmetic expression EXPR1 is evaluated. EXPR2 is then evaluated repeatedly until it evaluates to 0. Each time EXPR2 is evaluates to a non-zero value, statements are executed and EXPR3 is evaluated.

Example: A C-like for loop

```
$ cat for2.sh
#!/bin/bash
echo -n "Enter a number: "; read x
let sum=0
for (( i=1 ; $i<$x ; i=$i+1 )) ; do
    let "sum = $sum + $i"
done
echo "the sum of the first $x numbers is: $sum"
```

While Statements

- The while structure is a looping structure. Used to **execute a set of commands while a specified condition is true**. The loop terminates as soon as the condition becomes false. If condition never becomes false, loop will never exit.

while expression

do

statements

done

While Statements

```
$ cat while.sh
#!/bin/bash
echo -n "Enter a number: "; read x
let sum=0; let i=1
while [ $i -le $x ]; do
    let "sum = $sum + $i"
    i=$i+1
done
echo "the sum of the first $x numbers is: $sum"
```

Menu

```
$ cat menu.sh
#!/bin/bash
clear ; loop=y
while [ "$loop" = y ] ;
do
echo "Menu"; echo "===="
echo "D: print the date"
echo "W: print the users who are currently log on."
echo "P: print the working directory"
echo "Q: quit."
echo
read -s choice      # silent mode: no echo to terminal
case $choice in
  D | d) date ;;
  W | w) who ;;
  P | p) pwd ;;
  Q | q) loop=n ;;
  *) echo "Illegal choice." ;;
esac
echo
done
```

Find a Pattern and Edit

```
$ cat grepedit.sh
#!/bin/bash
# Edit argument files $2 ..., that contain pattern $1
if [ $# -le 1 ]
then
    echo "Usage: $0 pattern file ..." ; exit 1
else
    pattern=$1          # Save original $1
    shift                # shift the positional parameter to the left by 1
    while [ $# -gt 0 ]      # New $1 is first filename
        do
            grep "$pattern" $1 > /dev/null
            if [ $? -eq 0 ] ; then # If grep found pattern
                vi $1            # then vi the file
            fi
            shift
        done
    fi
$ grepedit.sh while ~
```

Continue Statements

- The **continue** command causes a jump to the next iteration of the loop, skipping all the remaining commands in that particular loop cycle.

```
$ cat continue.sh
#!/bin/bash
LIMIT=19
echo
echo "Printing Numbers 1 through 20 (but not 3 and 11)"
a=0
while [ $a -le "$LIMIT" ]; do
    a=$((a+1))
    if [ "$a" -eq 3 ] || [ "$a" -eq 11 ]
    then
        continue
    fi
    echo -n "$a "
done
```

Break Statements

- The **break** command **terminates the loop** (breaks out of it).

```
$ cat break.sh
#!/bin/bash
LIMIT=19
echo
echo "Printing Numbers 1 through 20, but something happens after 2 ... "
a=0
while [ $a -le "$LIMIT" ]
do
    a=$((a+1))
    if [ "$a" -gt 2 ]
    then
        break
    fi
    echo -n "$a "
done
echo; echo; echo
exit 0
```

Until Statements

- The **until** structure is very similar to the **while** structure. The **until** structure **loops until the condition is true**. So basically it is “until this condition is true, do this”.

until [expression]

do

statements

done

Until Statements

```
$ cat countdown.sh
#!/bin/bash
echo "Enter a number: "; read x
echo ; echo Count Down
until [ "$x" -le 0 ]; do
    echo $x
    x=$((x -1))
    sleep 1
done
echo ; echo GO !
```

Functions

- Functions make scripts easier to maintain. Basically it breaks up the program into smaller pieces. A function performs an action defined by you, and it can return a value if you wish.

```
#!/bin/bash
hello()
{
echo "You are in function hello()"
}

echo "Calling function hello()..."
hello
echo "You are now out of function hello()"
```

- In the above, we called the hello() function by name by using the line: hello . When this line is executed, bash searches the script for the line hello(). It finds it right at the top, and executes its contents.

Functions

```
$ cat function.sh
#!/bin/bash
function check() {
if [ -e "/home/$1" ]
then
    return 0
else
    return 1
fi
}
echo "Enter the name of the file: " ; read x
if check $x
then
    echo "$x exists !"
else
    echo "$x does not exists !"
fi.
```

Example: Picking a random card from a deck

```
#!/bin/bash  
# Count how many elements.
```

```
Suites="Clubs Diamonds Hearts Spades"
```

```
Denominations="2 3 4 5 6 7 8 9 10 Jack Queen King Ace"
```

```
# Read into array variable.
```

```
suite=($Suites)
```

```
denomination=($Denominations)
```

```
# Count how many elements.
```

```
num_suites=${#suite[*]}
```

```
num_denominations=${#denomination[*]}
```

```
echo -n "${denomination[$((RANDOM%num_denominations))]} of "
```

```
echo ${suite[$((RANDOM%num_suites))]}
```

```
exit 0
```

Example: Compare two files with a script

```
#!/bin/bash
ARGS=2                      # Two args to script expected.
if [ $# -ne "$ARGS" ]; then
    echo "Usage: `basename $0` file1 file2" ; exit 1
fi
if [[ ! -r "$1" || ! -r "$2" ]] ; then
    echo "Both files must exist and be readable." ; exit 2
fi
# /dev/null buries the output of the "cmp" command.
cmp $1 $2 &> /dev/null
# Also works with 'diff', i.e., diff $1 $2 &> /dev/null
if [ $? -eq 0 ]              # Test exit status of "cmp" command.
then
    echo "File \"\$1\" is identical to file \"\$2\"."
else
    echo "File \"\$1\" differs from file \"\$2\"."
fi
exit 0
```

Example: Suite drawing statistics

```
$ cat cardstats.sh
#!/bin/sh # -xv
N=100000
hits=(0 0 0 0) # initialize hit counters
if [ $# -gt 0 ]; then # check whether there is an argument
    N=$1
else # ask for the number if no argument
    echo "Enter the number of trials:"
    TMOUT=5 # 5 seconds to give the input
    read N
fi
i=$N
echo "Generating $N random numbers... please wait."
SECONDS=0 # here is where we really start
while [ $i -gt 0 ]; do # run until the counter gets to zero
    case $((RANDOM%4)) in
        0) let "hits[0]+=1";; # count the hits
        1) let "hits[1]=${hits[1]}+1";;
        2) let hits[2]="${hits[2]}+1";;
        3) let hits[3]="${hits[3]}+1";;
    esac
    let "i-=1" # count down
done
echo "Probabilities of drawing a specific color:"
# use bc - bash does not support fractions
echo "Clubs: "`echo ${hits[0]}*100/$N | bc -l`"
echo "Diamonds: "`echo ${hits[1]}*100/$N | bc -l`"
echo "Hearts: "`echo ${hits[2]}*100/$N | bc -l`"
echo "Spades: "`echo ${hits[3]}*100/$N | bc -l`"
echo ====="
echo "Execution time: $SECONDS"
```



CSC424 System Administration

Instructor: Dr. Hao Wu

Week 8 Bash Scripting - 3

Exercise 1

- Write a program that gives simple math quizzes. The program should display two random numbers that are to be added. such as
 - $$\begin{array}{r} 247 \\ + 129 \\ \hline \end{array}$$
 - The program should allow the student to enter the answer, If the answer is correct, a message of congratulations should be displayed. If the answer is incorrect, a message showing the correct answer should be displayed.

printf function

- printf function can be used for format control
- Syntax:
`printf "%w.pL\n" $varName`
- w - Minimum field width.
- p - Display number of digits after the decimal point (precision).
- L - a conversion character. It can be:
 - s - String
 - d - Integer
 - e - Exponential
 - f - Floating point

Exercise 2

- Write a program that generates 100 random numbers and keeps a count of how many of those random numbers are even, and how many of them are odd.

Exercise 3

- Write a program that generates 100 random numbers and write the numbers into a file called "data". Each line contain one number.

Exercise 4

- Write a program to read the numbers from data. Calculate the sum, average, min and max values.

Exercise 5

- Write a program to generate a database contains student records (at least 10 students). Each line contains one student's record. The format of the record is:
 - Name grade_1 grade_2 grade_3 ... grade_10
- Grade: A, B, C, D, F

Exercise 6

- Write a program that reads the student record from database.
The program should display each student's GPA. You can assume 3 credits for each course.
- GPA calculation:
 - A: 4.0
 - B: 3.0
 - C: 2.0
 - D: 1.0
 - F: 0.0



CSC424 System Administration

Instructor: Dr. Hao Wu

Week 8 Bash Scripting - 3

Exercise 1

- Write a script to simulate the progress bar. The script should show the dynamic change of the progress.

```
Haos-MacBook-Pro:scripts hao$ ./progress
```

```
[#####
#####] [100%] -
```

- Options for echo:

-n do not output the trailing newline

-e enable interpretation of backslash escapes

\r carriage return

Exercise 2

- Moving Car: write a script that displays a car moving on the screen

Exercise 3

- Write a script to get a user input and convert all lowercase letters to uppercase letters

Exercise 4

- Write a script to read a sentence from user input and reverse sentence. For instance, the user input is:
- "sentence"
- The output is:
- "ecnetnes"

Exercise 5

- Caesar cipher:
- Write a script to encrypt a sentence using caesar cipher.
- Caesar cipher is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example:
 - Plain: ABCDEFGHIJKLMNOPQRSTUVWXYZ
 - Cipher: XYZABCDEFGHIJKLMNPQRSTUVWXYZ

Exercise 6

- Write a program that randomly generate student records:
student name, student id (2018xxxx)

Exercise 7

- Write a script that reads student record from database file created by previous script. Create users account for each student and set their password as their student ID.



CSC424 System Administration

Instructor: Dr. Hao Wu

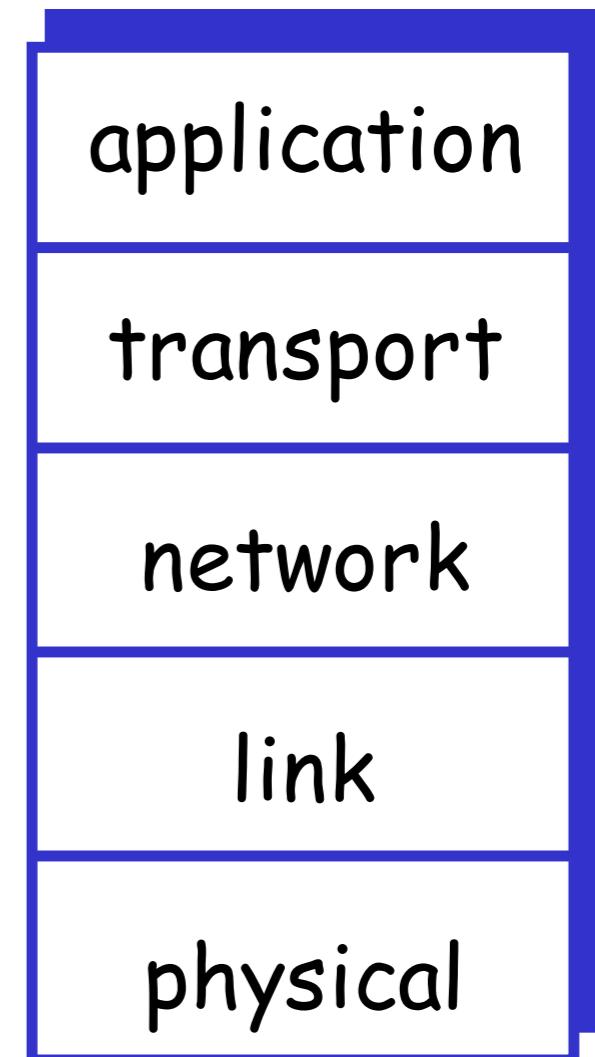
Week 11 Network and System Monitoring

Networking Basics

- Internet protocol layers
- MAC
- IP
- TCP/UDP
- Routing

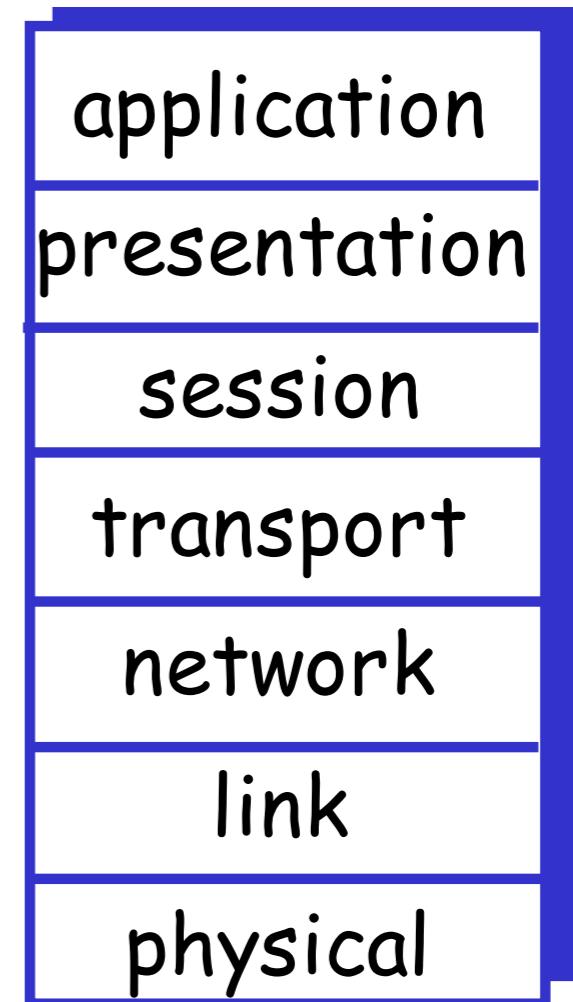
Internet protocol stack

- ❖ *application*: supporting network applications
 - FTP, SMTP, HTTP
- ❖ *transport*: process-process data transfer
 - TCP, UDP
- ❖ *network*: routing of datagrams from source to destination
 - IP, routing protocols
- ❖ *link*: data transfer between neighboring network elements
 - Ethernet, 802.111 (WiFi), PPP
- ❖ *physical*: bits “on the wire”

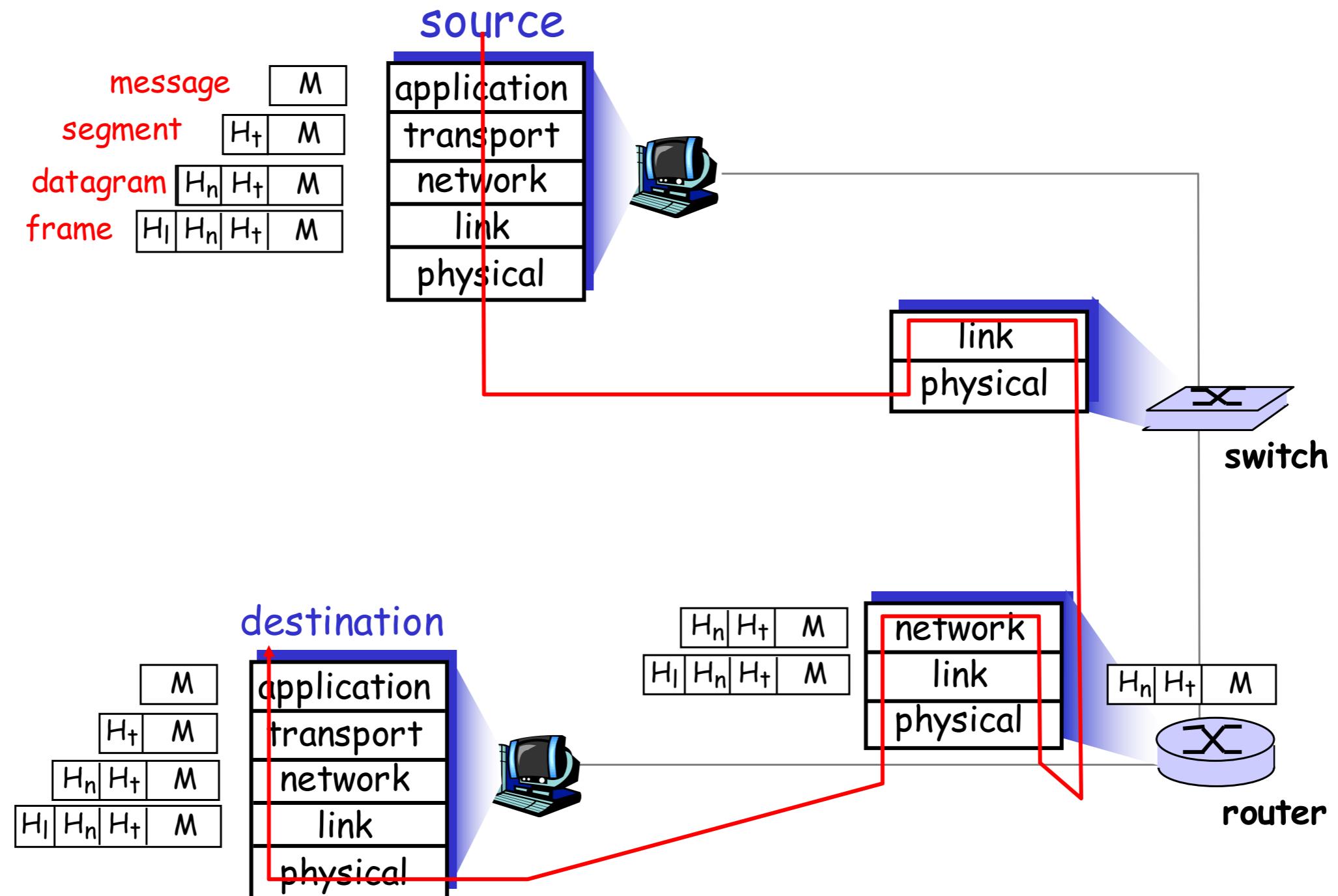


ISO/OSI (Open Systems Interconnection) reference model

- **presentation:** allow applications to interpret meaning of data, e.g., encryption, compression, machine-specific conventions
- **session:** synchronization, checkpointing, recovery of data exchange
- Internet stack “missing” these layers!
 - ❖ these services, *if needed*, must be implemented in application
 - ❖ needed?



Encapsulation

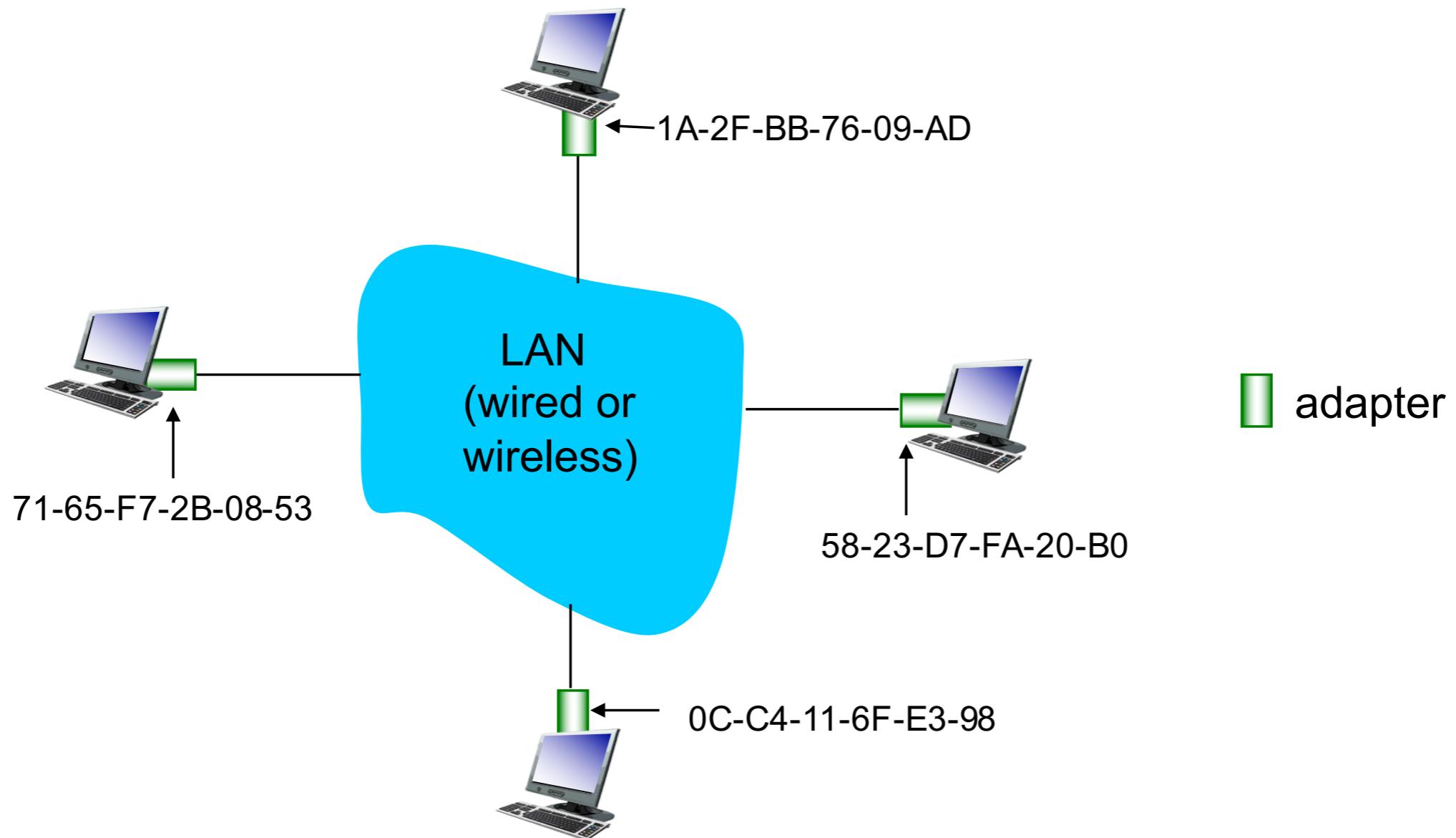


MAC addresses and ARP

- ❖ 32-bit IP address:
 - *network-layer* address for interface
 - used for layer 3 (network layer) forwarding
- ❖ MAC (or LAN or physical or Ethernet) address:
 - function: *used ‘locally’ to get frame from one interface to another physically-connected interface (same network, in IP-addressing sense)*
 - 48 bit MAC address (for most LANs) burned in NIC ROM, also sometimes software settable
 - e.g.: IA-2F-BB-76-09-AD

LAN addresses and ARP

each adapter on LAN has unique *LAN* address

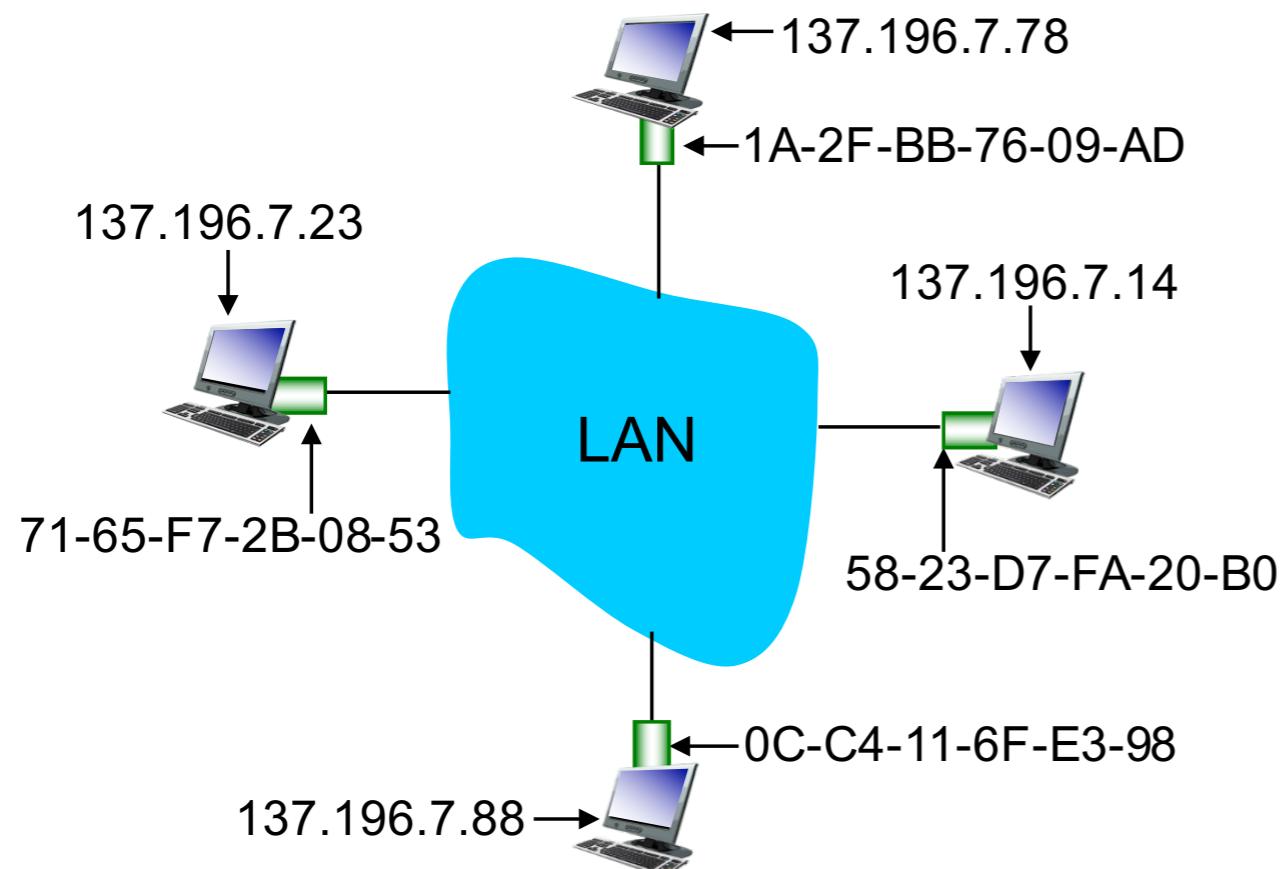


LAN addresses (more)

- ❖ MAC address allocation administered by IEEE
- ❖ manufacturer buys portion of MAC address space (to assure uniqueness)
- ❖ analogy:
 - MAC address: like Social Security Number
 - IP address: like postal address
- ❖ MAC flat address → portability
 - can move LAN card from one LAN to another
- ❖ IP hierarchical address *not portable*
 - address depends on IP subnet to which node is attached

ARP: address resolution protocol

Question: how to determine interface's MAC address, knowing its IP address?

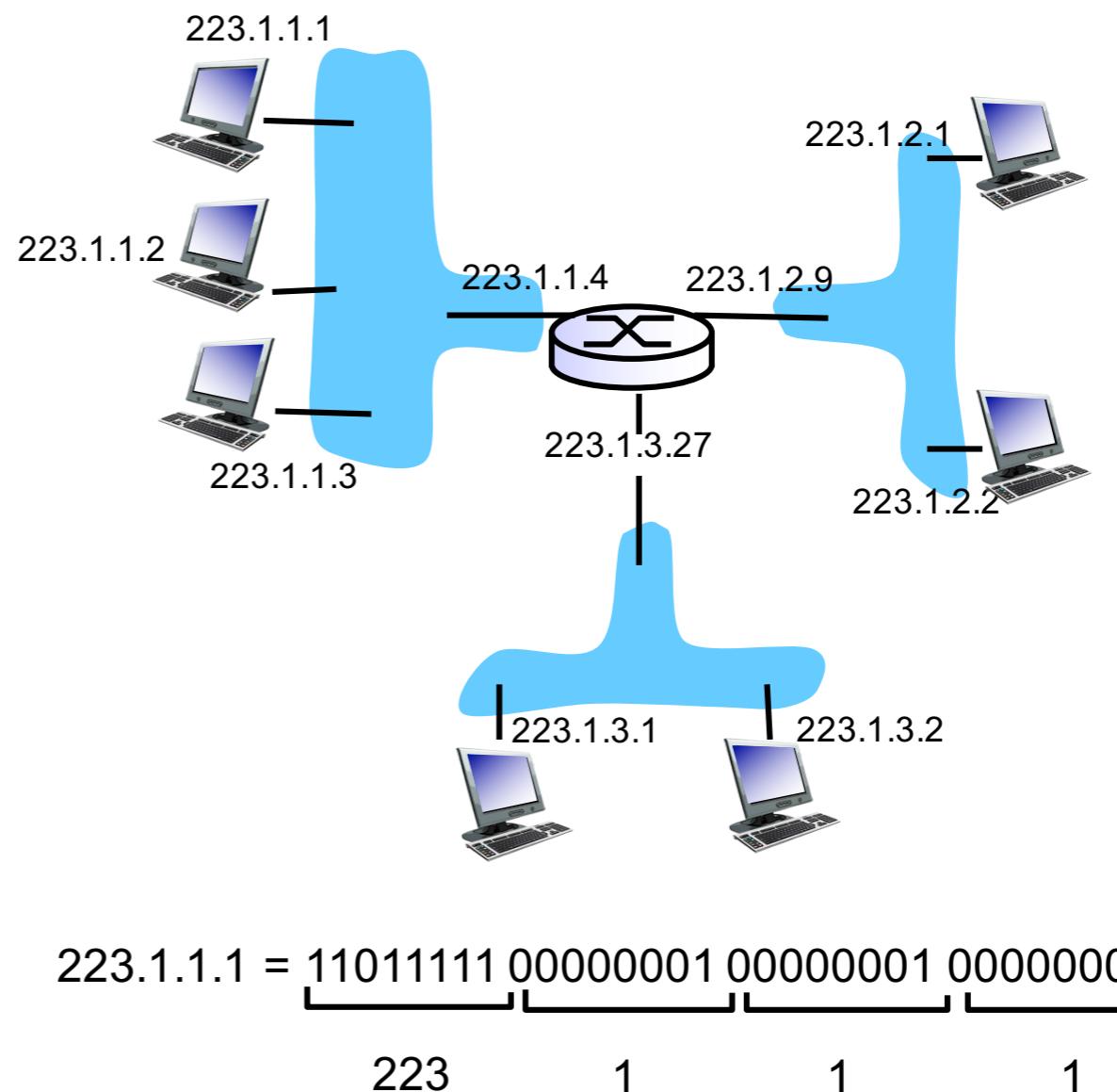


ARP table: each IP node (host, router) on LAN has table

- IP/MAC address mappings for some LAN nodes:
<IP address; MAC address; TTL>
- TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

IP addressing: introduction

- ❖ **IP address:** 32-bit identifier for host, 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)
- ❖ **IP addresses associated with each interface**



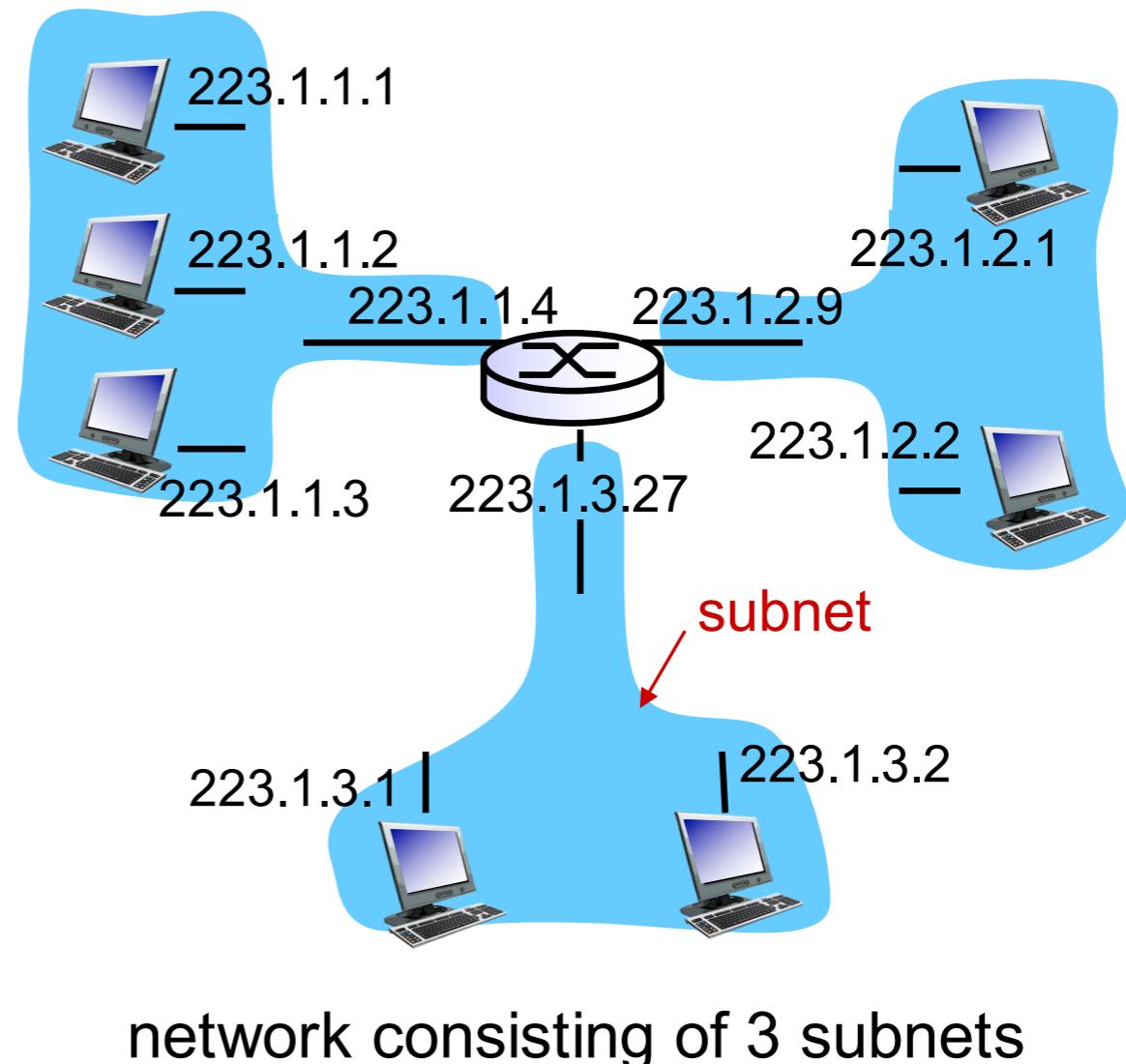
Subnets

❖ IP address:

- subnet part - high order bits
- host part - low order bits

❖ what's a subnet ?

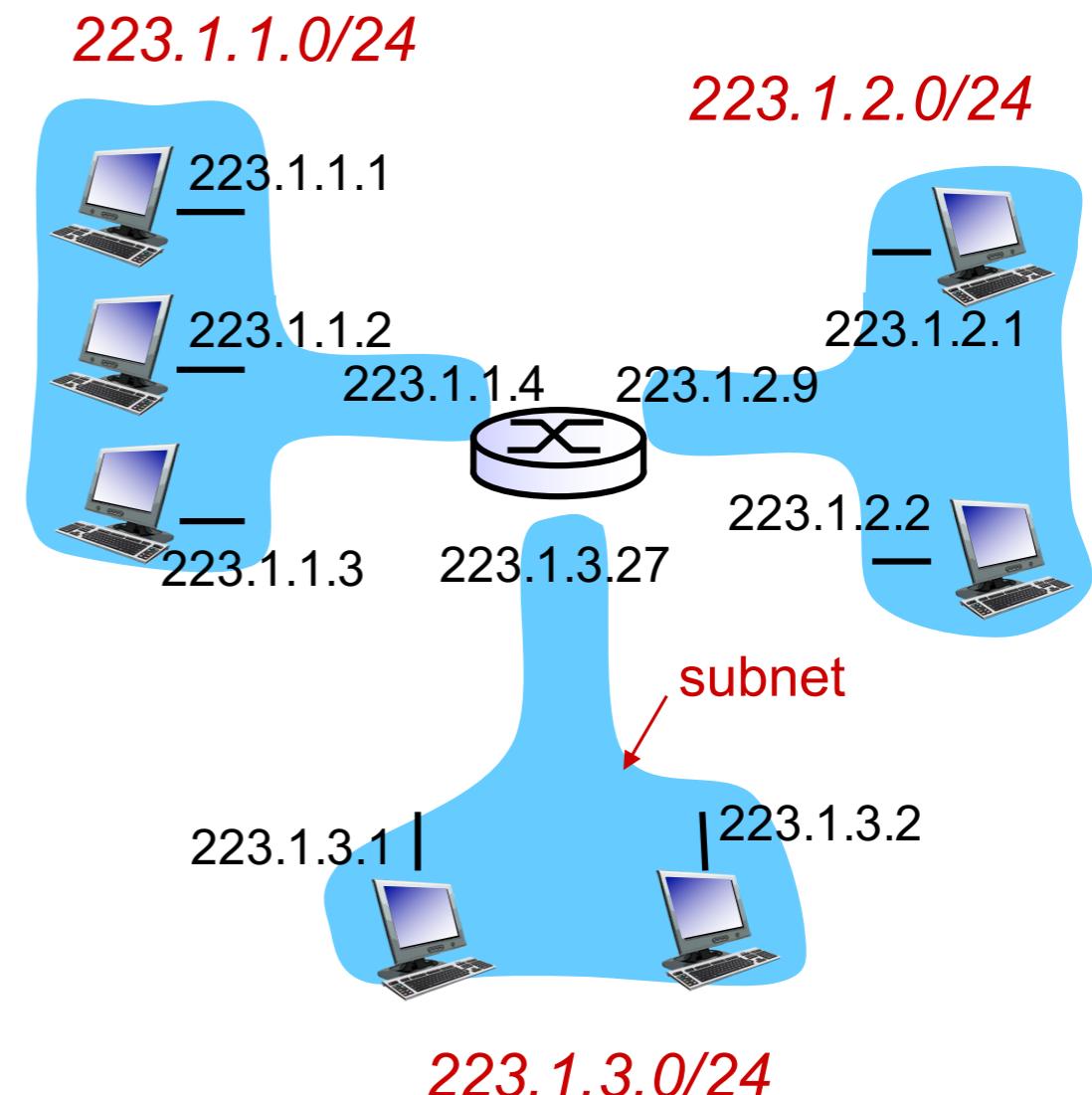
- device interfaces with same subnet part of IP address
- can physically reach each other *without intervening router*



Subnets

recipe

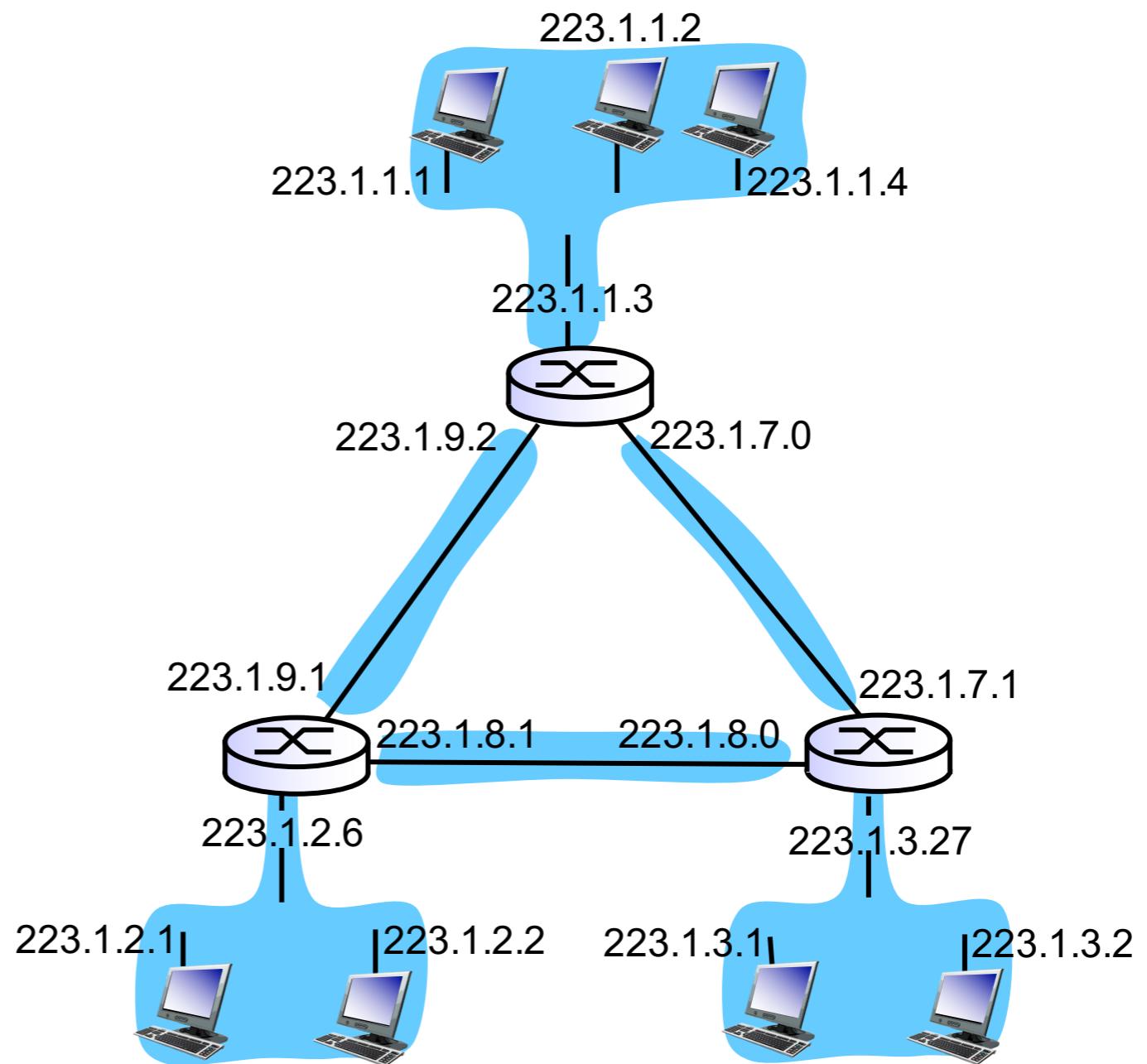
- ❖ to determine the subnets, detach each interface from its host or router, creating islands of isolated networks
- ❖ each isolated network is called a *subnet*



subnet mask: /24

Subnets

how many?



IP addressing: CIDR

CIDR: Classless InterDomain Routing

- 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?

Q: How does a *host* get IP address?

- ❖ hard-coded by system admin in a file
 - Windows: control-panel->network->configuration->tcp/ip->properties
 - UNIX: /etc/rc.config
- ❖ **DHCP: Dynamic Host Configuration Protocol:** dynamically get address from as server
 - “plug-and-play”

DHCP: Dynamic Host Configuration Protocol

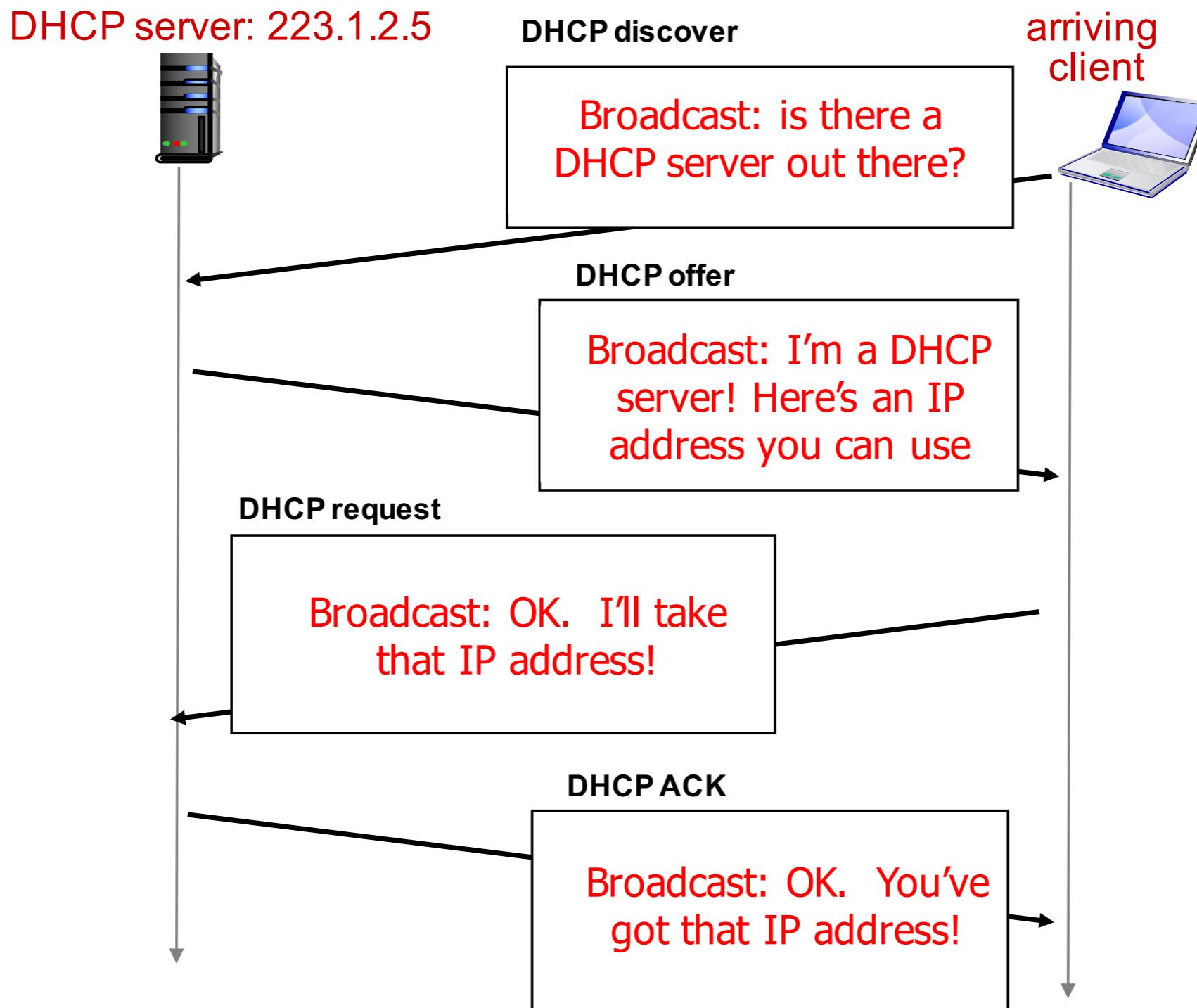
goal: allow host to *dynamically* obtain its IP address from network server when it joins network

- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/“on”)
- support for mobile users who want to join network (more shortly)

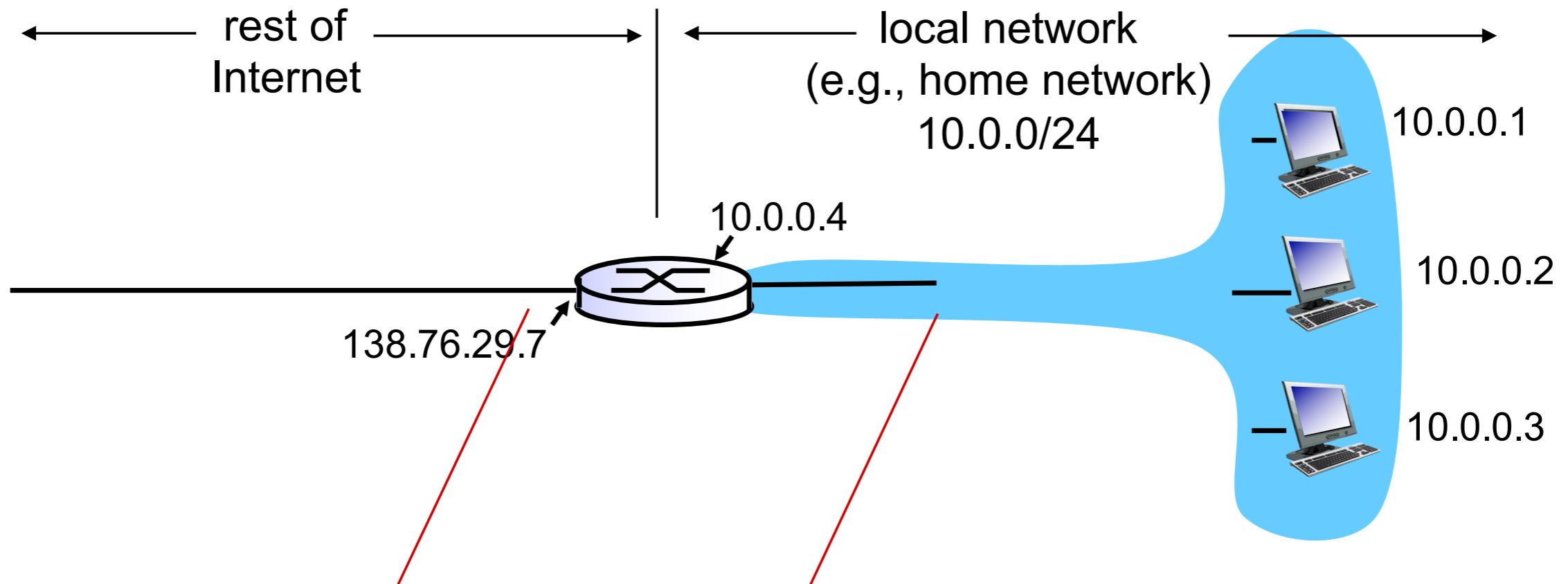
DHCP overview:

- 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



NAT: network address translation



all datagrams ***leaving*** local network have ***same*** single source NAT IP address:
138.76.29.7, 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

motivation: local network uses just one IP address as far as outside world is concerned:

- range of addresses not needed from ISP: just one IP address for all devices
- can change addresses of devices in local network without notifying outside world
- can change ISP without changing addresses of devices in local network
- devices inside local net not explicitly addressable, visible by outside world (a security plus)

NAT: network address translation

implementation: NAT router must:

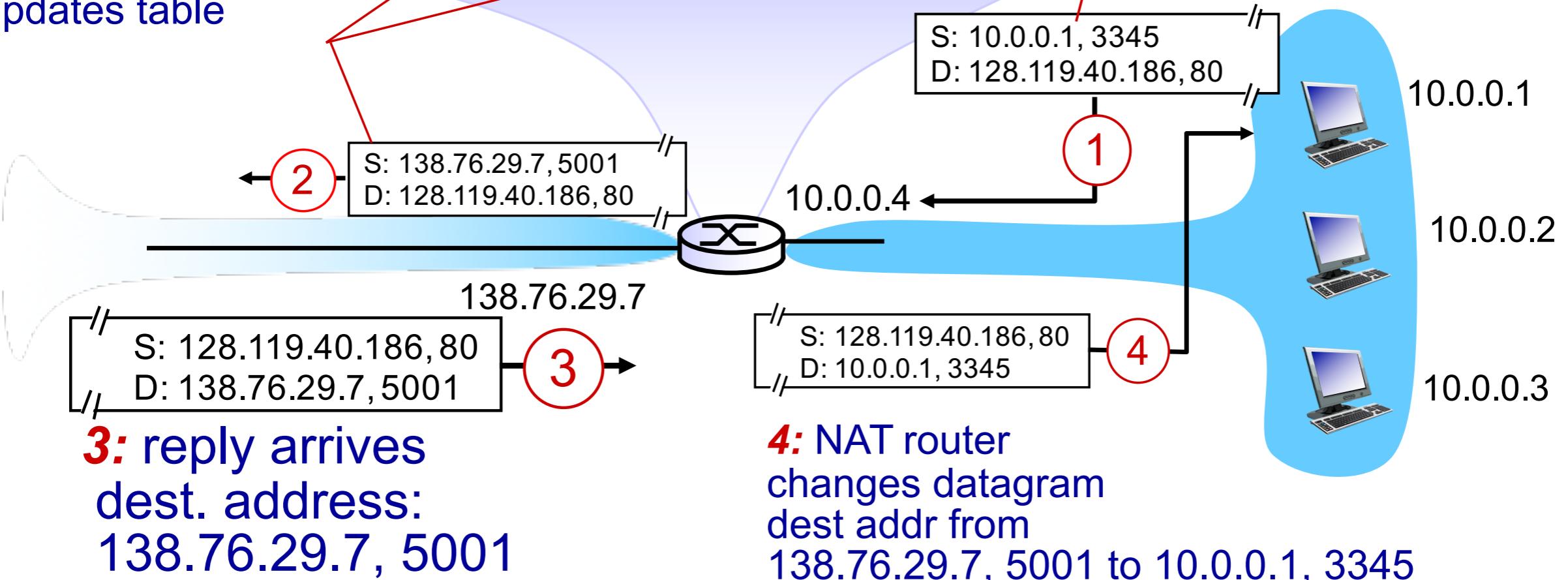
- *outgoing datagrams:* *replace* (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
. . . remote clients/servers will respond using (NAT IP address, new port #) as destination addr
- *remember (in NAT translation table)* every (source IP address, port #) to (NAT IP address, new port #) translation pair
- *incoming datagrams:* *replace* (NAT IP address, new port #) in dest 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 addr from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table

NAT translation table	
WAN side addr	LAN side addr
138.76.29.7, 5001	10.0.0.1, 3345
.....

1: host 10.0.0.1 sends datagram to 128.119.40.186, 80



NAT: network address translation

- ❖ 16-bit port-number field:
 - 60,000 simultaneous connections with a single LAN-side address!
- ❖ NAT is controversial:
 - routers should only process up to layer 3
 - violates end-to-end argument
 - NAT possibility must be taken into account by app designers, e.g., P2P applications
 - address shortage should instead be solved by IPv6

IPv6: motivation

- ❖ *initial motivation:* 32-bit address space soon to be completely allocated.
- ❖ additional motivation:
 - header format helps speed processing/forwarding
 - header changes to facilitate QoS

IPv6 datagram format:

- fixed-length 40 byte header
- no fragmentation allowed

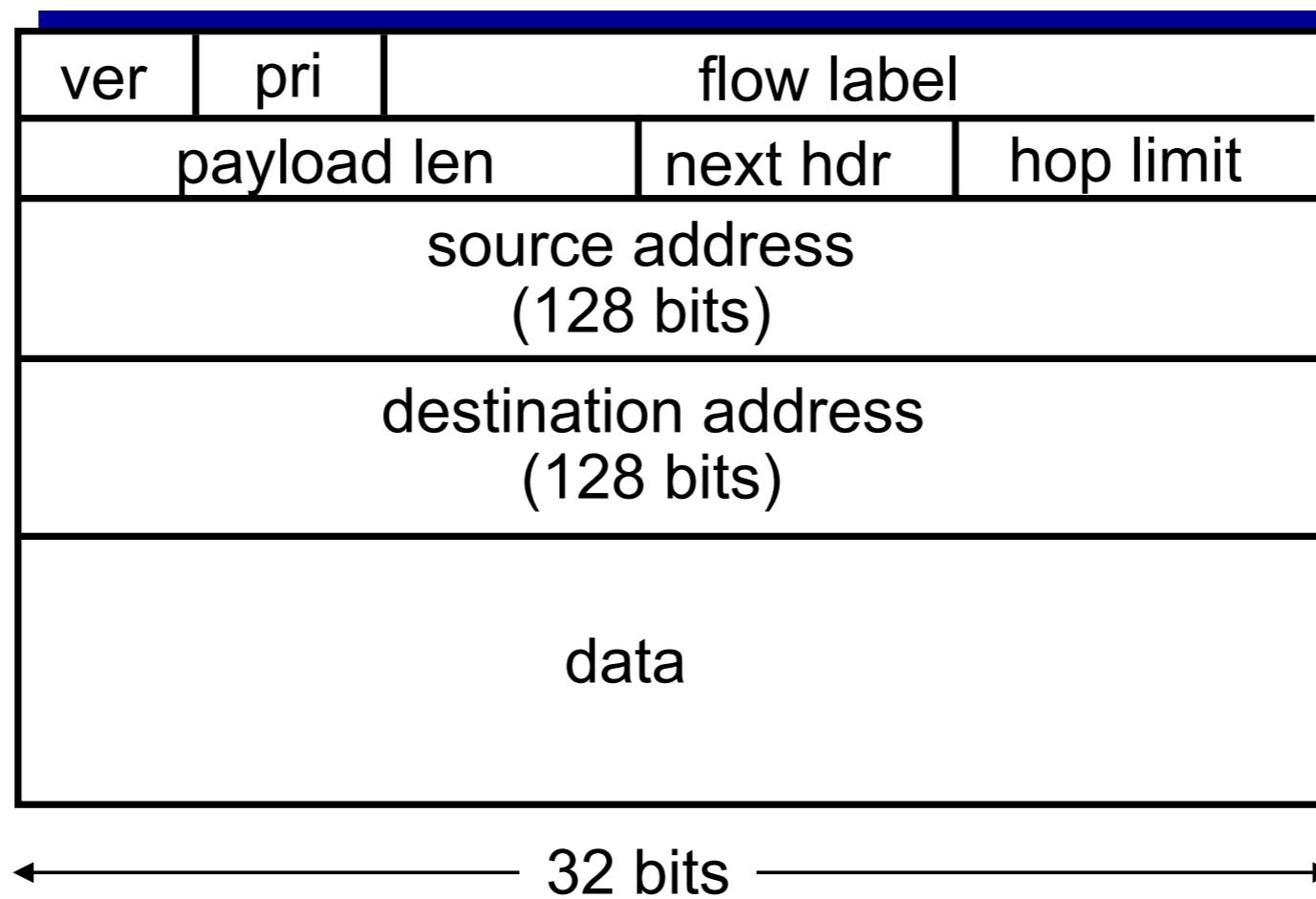
IPv6 datagram format

priority: identify priority among datagrams in flow

flow Label: identify datagrams in same “flow.”

(concept of “flow” not well defined).

next header: identify upper layer protocol for data



Network Interface Configuration

- Command for network interface configuration:
 ifconfig [option] [interface]
- If command not found, install the tool:

```
[root@localhost ~]# yum install net-tools
Loaded plugins: fastestmirror
Loading mirror speeds from cached hostfile
 * base: mirror.lug.udel.edu
 * extras: mirror.es.its.nyu.edu
 * updates: mirrors.lga7.us.voxel.net
```

Network Interface Configuration

- Usage: check all network interfaces:
 ifconfig -a

```
[root@localhost ~]# ifconfig -a
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        ether 08:00:27:12:ea:9a txqueuelen 1000 (Ethernet)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.1.31 netmask 255.255.255.0 broadcast 192.168.1.255
        inet6 fe80::ae7e:dca9:dfa6:fabe prefixlen 64 scopeid 0x20<link>
        ether 08:00:27:34:2b:0c txqueuelen 1000 (Ethernet)
        RX packets 10374 bytes 14649660 (13.9 MiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 1512 bytes 127449 (124.4 KiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
        inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
        loop txqueuelen 1 (Local Loopback)
        RX packets 0 bytes 0 (0.0 B)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 0 bytes 0 (0.0 B)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Network Interface Configuration

- Understanding output information:

```
[root@localhost ~]# ifconfig enp0s8
enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
          inet 192.168.1.31 netmask 255.255.255.0 broadcast 192.168.1.255
          inet6 fe80::ae7e:dca9:dfa6:fabe prefixlen 64 scopeid 0x20<link>
            ether 08:00:27:34:2b:0c txqueuelen 1000 (Ethernet)
              RX packets 10494 bytes 14663846 (13.9 MiB)
              RX errors 0 dropped 0 overruns 0 frame 0
              TX packets 1566 bytes 134435 (131.2 KiB)
              TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

- enp0s8: name of the device
- UP: This flag indicates that the kernel modules related to the Ethernet interface has been loaded.
- BROADCAST: Denotes that the Ethernet device supports broadcasting - a necessary characteristic to obtain IP address via DHCP.
- RUNNING: The interface is ready to accept data

Network Interface Configuration

- Understanding output information:

```
[root@localhost ~]# ifconfig enp0s8
enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
          inet 192.168.1.31 netmask 255.255.255.0 broadcast 192.168.1.255
          inet6 fe80::ae7e:dca9:dfa6:fabe prefixlen 64 scopeid 0x20<link>
            ether 08:00:27:34:2b:0c txqueuelen 1000 (Ethernet)
              RX packets 10494 bytes 14663846 (13.9 MiB)
              RX errors 0 dropped 0 overruns 0 frame 0
              TX packets 1566 bytes 134435 (131.2 KiB)
              TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

- MULTICAST: This indicates that the Ethernet interface supports multicasting.
- mtu: short form for Maximum Transmission Unit is the size of each packet received by the Ethernet card.
- inet: IPv4 address
- inet6: IPv6 address
- ether: MAC address
- txqueuelen: This denotes the length of the transmit queue of the device.

Network Interface Configuration

- Understanding output information:

```
[root@localhost ~]# ifconfig enp0s8
enp0s8: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.1.31 netmask 255.255.255.0 broadcast 192.168.1.255
        inet6 fe80::ae7e:dca9:dfa6:fabe prefixlen 64 scopeid 0x20<link>
        ether 08:00:27:34:2b:0c txqueuelen 1000 (Ethernet)
        RX packets 10494 bytes 14663846 (13.9 MiB)
        RX errors 0 dropped 0 overruns 0 frame 0
        TX packets 1566 bytes 134435 (131.2 KiB)
        TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

- RX and TX: total number of packets or bytes received and transmitted respectively.

Network Interface Configuration

- Change IP address using ifconfig:
- *ifconfig <interface> <address> netmask <netmask> [up|down]*
- *ifconfig <interface> <address> </prefixlen> [up | down]*
-

```
[root@localhost ~]# ifconfig enp0s3 192.168.2.3/24
[root@localhost ~]# ifconfig enp0s3
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
        inet 192.168.2.3  netmask 255.255.255.0 broadcast 192.168.2.255
        inet6 fe80::a00:27ff:fe12:ea9a  prefixlen 64  scopeid 0x20<link>
          ether 08:00:27:12:ea:9a  txqueuelen 1000  (Ethernet)
            RX packets 17  bytes 3135 (3.0 KiB)
            RX errors 0  dropped 0  overruns 0  frame 0
            TX packets 542  bytes 33328 (32.5 KiB)
            TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
```

Network Interface Configuration

- ifconfig:
 - changes network interface configurations temporarily
 - lose configurations after reboot
- Edit network configuration files:
 - Location: /etc/sysconfig/network-scripts/
 - Files start with "ifcfg-interfaceName" are configuration files for network interfaces



```
[root@localhost network-scripts]# ls
ifcfg-enp0s3  ifdown-isdn    ifdown-tunnel  ifup-isdn   ifup-Team
ifcfg-lo       ifdown-post   ifup          ifup-plip   ifup-TeamPort
ifdown        ifdown-ppp    ifup-aliases   ifup-plusb  ifup-tunnel
ifdown-bnep   ifdown-routes ifup-bnep     ifup-post   ifup-wireless
ifdown-eth    ifdown-sit    ifup-eth      ifup-ppp    init.ipv6-global
ifdown-ippn   ifdown-Team   ifup-ippn     ifup-routes network-functions
ifdown-ipv6   ifdown-TeamPort ifup-ipv6    ifup-sit    network-functions-ipv6
```

Network Interface Configuration

- Understand the configuration file:

- BOOTPROTO:

- boot-time protocol used
- none - No boot time protocol should be used
- dhcp - The DHCP protocol should be used
- static - Use static ip address

- ONBOOT:

- yes - This device should be activated at boot-time
- no - This device should not be activated at boot-time

```
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=dhcp
DEFROUTE=yes
IPV4_FAILURE_FATAL=no
IPV6INIT=yes
IPV6_AUTOCONF=yes
IPV6_DEFROUTE=yes
IPV6_FAILURE_FATAL=no
IPV6_ADDR_GEN_MODE=stable-privacy
NAME=enp0s3
UUID=dd33654a-838f-4664-
abb4-38cf1b5980ac
DEVICE=enp0s3
ONBOOT=no
```

Network Interface Configuration

- Config a static IP interface:
 - IPADDR: IPv4 address
 - NETMASK: network mask

```
TYPE=Ethernet
PROXY_METHOD=none
BROWSER_ONLY=no
BOOTPROTO=static
IPADDR=192.168.2.3
NETMASK=255.255.255.0
NAME=enp0s3
UUID=dd33654a-838f-4664-
abb4-38cf1b5980ac
DEVICE=enp0s3
ONBOOT=yes
~
```

Network Interface Configuration

- Take a network interface down:

```
[root@localhost network-scripts]# ifdown enp0s3  
Device 'enp0s3' successfully disconnected.
```

- Bring a network interface up:

```
[root@localhost network-scripts]# ifup enp0s3  
Connection successfully activated (D-Bus active path: /org/freedesktop/  
NetworkManager/ActiveConnection/3)
```

Hostname

- Change your hostname by command:
 - Temporary

```
[root@localhost ~]# hostname  
localhost.localdomain  
[root@localhost ~]# hostname myhost.localdomain  
[root@localhost ~]# hostname  
myhost.localdomain
```

- Change your hostname by editing configuration file:
- Location: /etc/hostname

```
[root@localhost ~]# echo myhost.localhost > /etc/hostname
```

Hosts files

- /etc/hosts: file retains a mapping of host names and their ip addresses
-

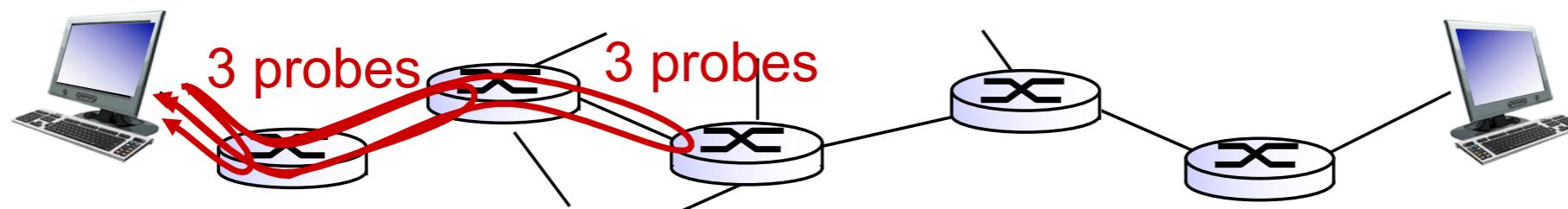
```
[root@localhost ~]# cat /etc/hosts
127.0.0.1    localhost localhost.localdomain localhost4 localhost4.localdomain4
::1          localhost localhost.localdomain localhost6 localhost6.localdomain6
```

Network Tools

- ping: used to test connections between hosts
- traceroute: used to test connections between hosts and routers
- netstat: Print network connections, routing tables, interface statistics, masquerade connections, and multicast memberships

Traceroute and ICMP

- ❖ source sends series of UDP segments to dest
 - first set has TTL =1
 - second set has TTL=2, etc.
 - unlikely port number
 - ❖ when *n*th set of datagrams arrives to *n*th router:
 - router discards datagrams
 - and sends source ICMP messages (type 11, code 0)
 - ICMP messages includes name of router & IP address
 - ❖ when ICMP messages arrives, source records RTTs
- stopping criteria:*
- ❖ UDP segment eventually arrives at destination host
 - ❖ destination returns ICMP “port unreachable” message (type 3, code 3)
 - ❖ source stops



netstat

```
[root@localhost ~]# netstat -tunpa
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address          Foreign Address        State      PID/Program name
tcp      0      0 0.0.0.0:22              0.0.0.0:*
                                         LISTEN
tcp      0      0 127.0.0.1:25             0.0.0.0:*
                                         LISTEN
tcp      0      0 192.168.1.31:22            192.168.1.8:61706 ESTABLISHED 1257/sshd: root@pts
tcp6     0      0 :::22                  :::*
                                         LISTEN
tcp6     0      0 ::1:25                 :::*
                                         LISTEN
udp      0      0 0.0.0.0:68              0.0.0.0:*
                                         LISTEN
udp      0      0 0.0.0.0:4317             0.0.0.0:*
                                         LISTEN
udp      0      0 127.0.0.1:323             0.0.0.0:*
                                         LISTEN
udp6     0      0 :::13571                :::*
                                         LISTEN
udp6     0      0 ::1:323                 :::*
                                         LISTEN
```

- Options:
 - a: Show both listening and non-listening sockets.
 - t: TCP
 - u: UDP
 - n: Show numerical addresses instead of trying to determine symbolic host, port or user names.
 - p: Show the PID and name of the program to which each socket belongs.