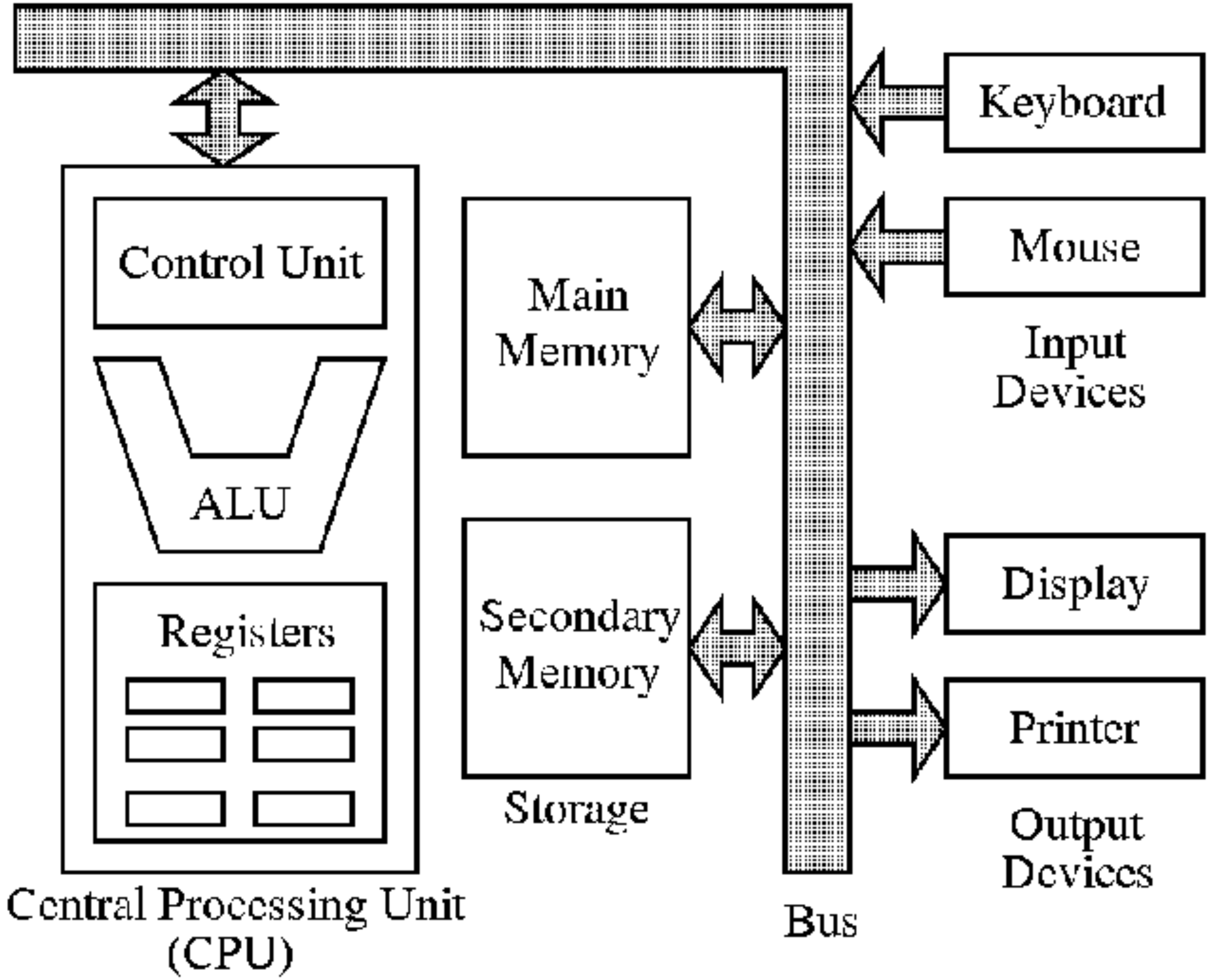
Southern Connecticut State University

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

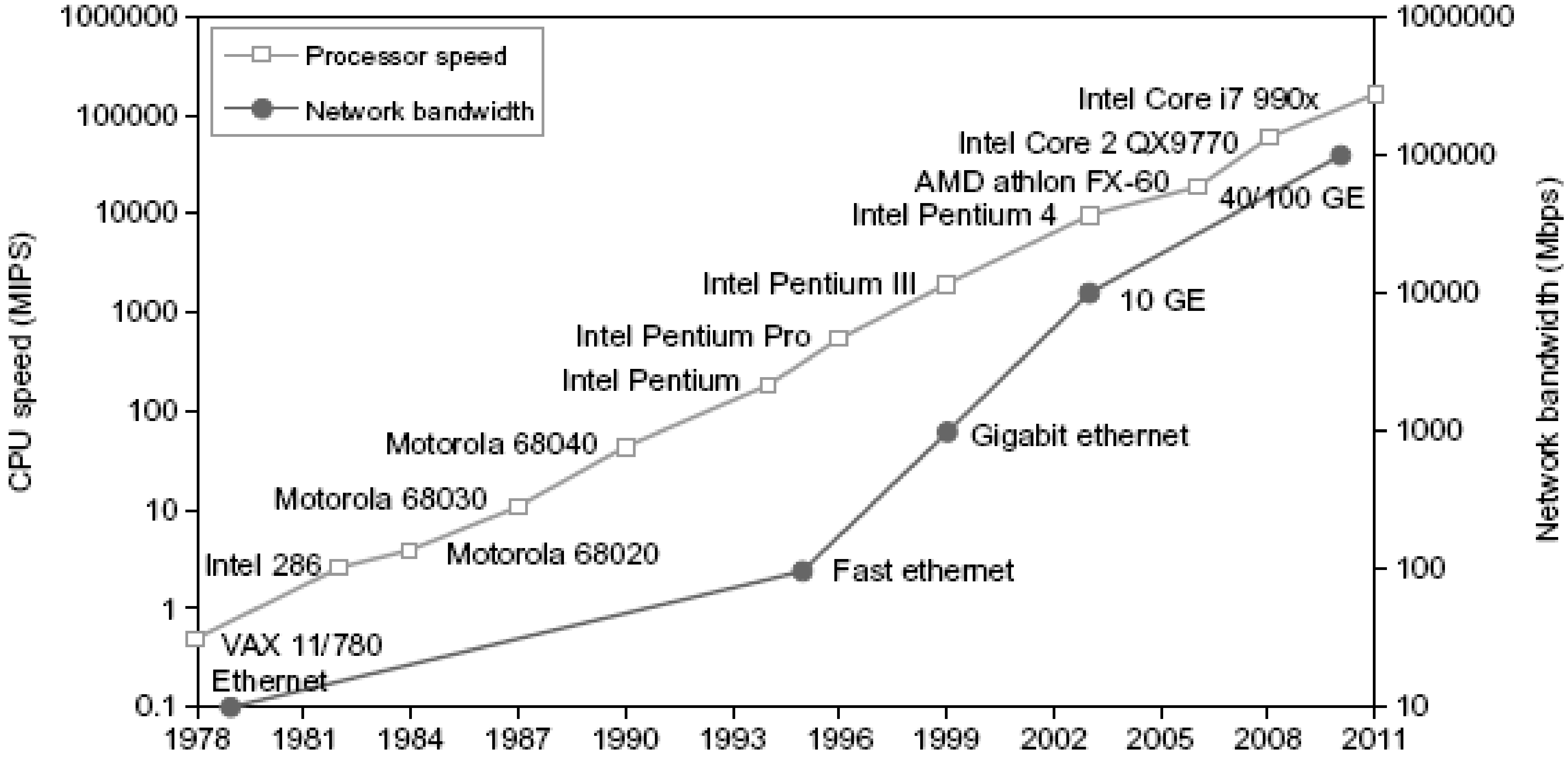
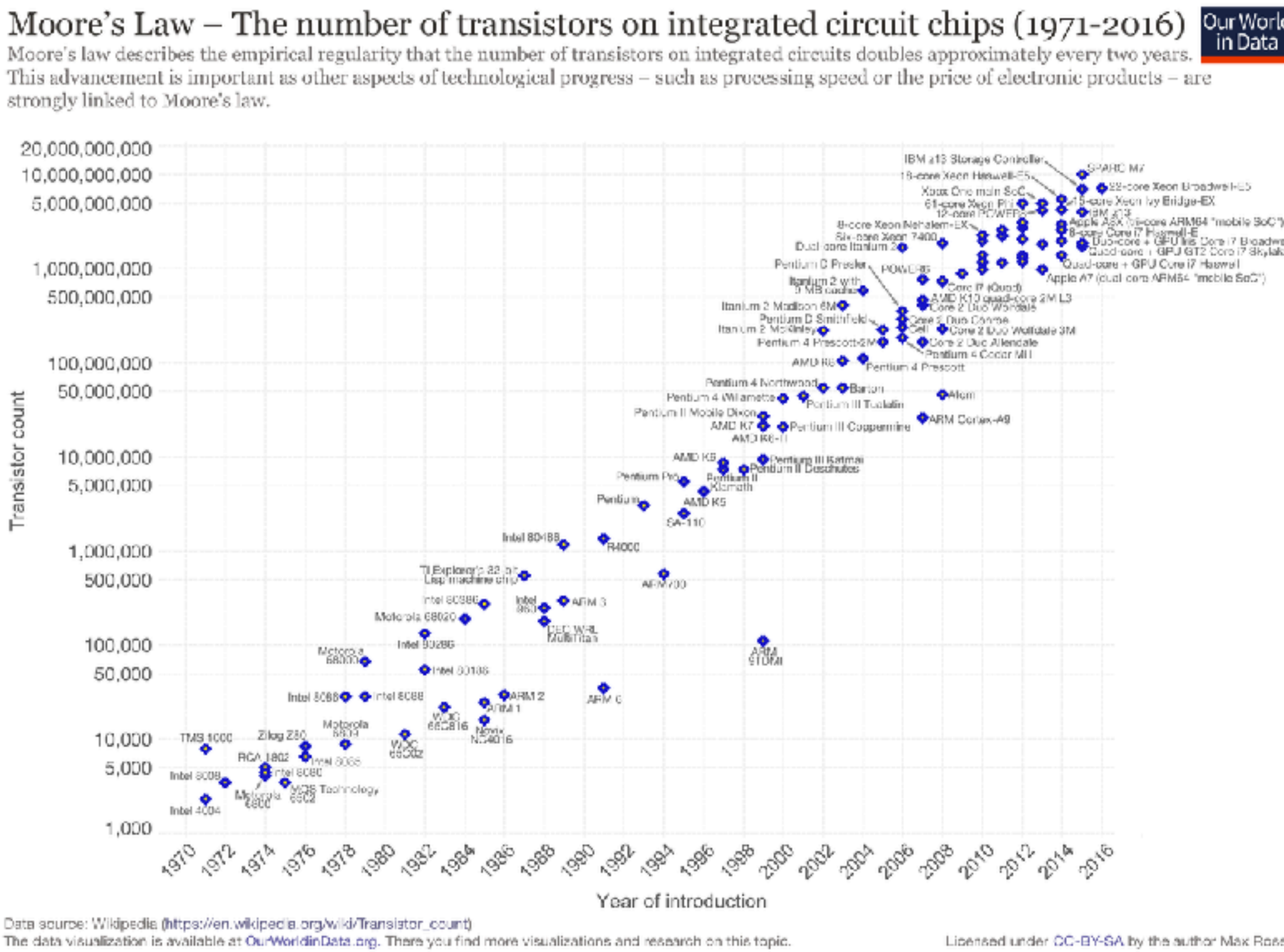


FIGURE 1.4

Improvement in processor and network technologies over 33 years.

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

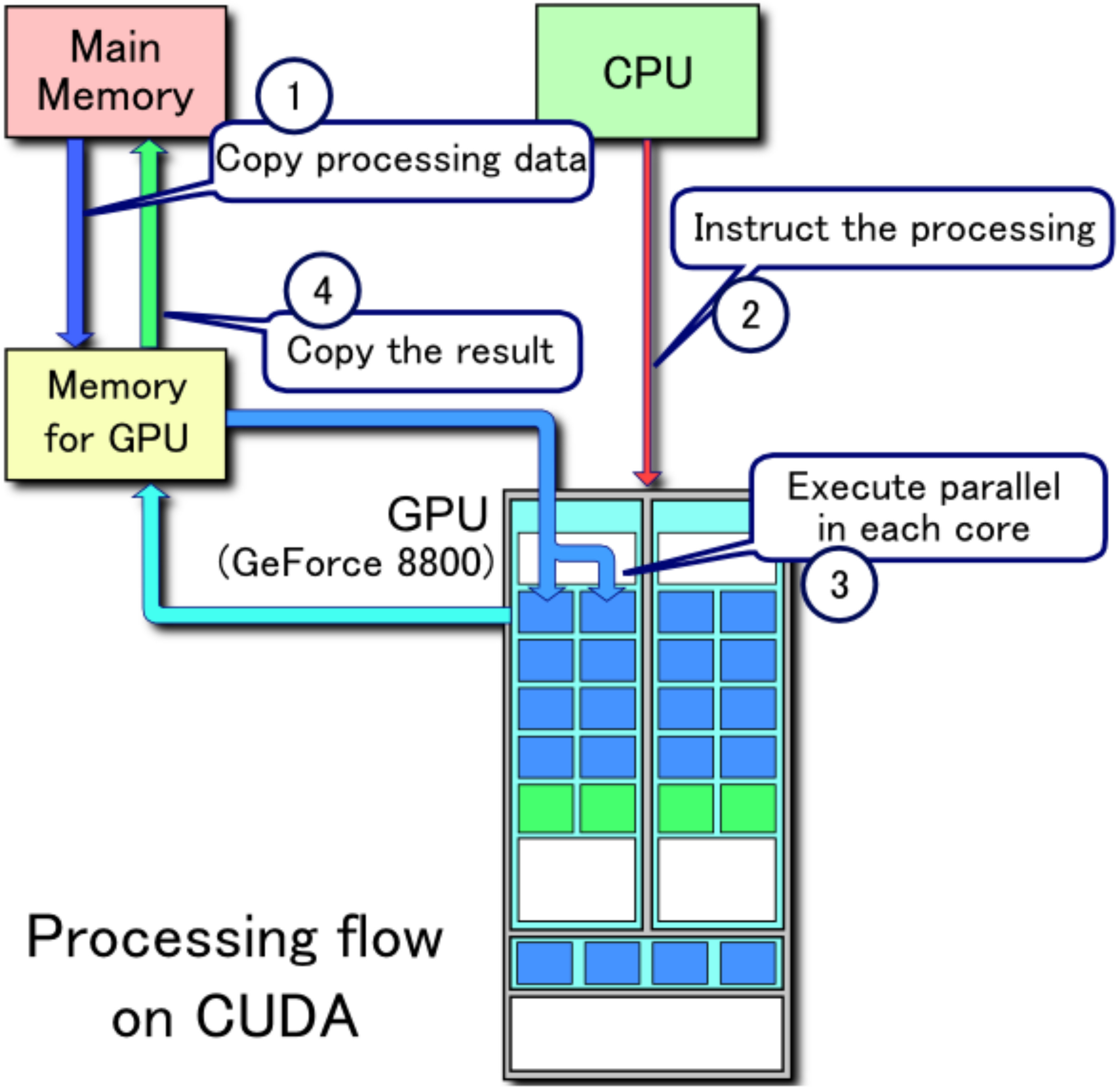
Moore's Law


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.
Transistor count
20,000,000,000
10,000,000,000
5,000,000,000
1,000,000,000
500,000,000
100,000,000
50,000,000
10,000,000
5,000,000
1,000,000
500,000
100,000
50,000
10,000
5,000
1,000
1970
1972
1974
1976
1978
1980
1982
1984
1986
1988
1990
1992
1994
1996
1998
2000
2002
2004
2006
2008
2010
2012
2014
2016
Year of introduction
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

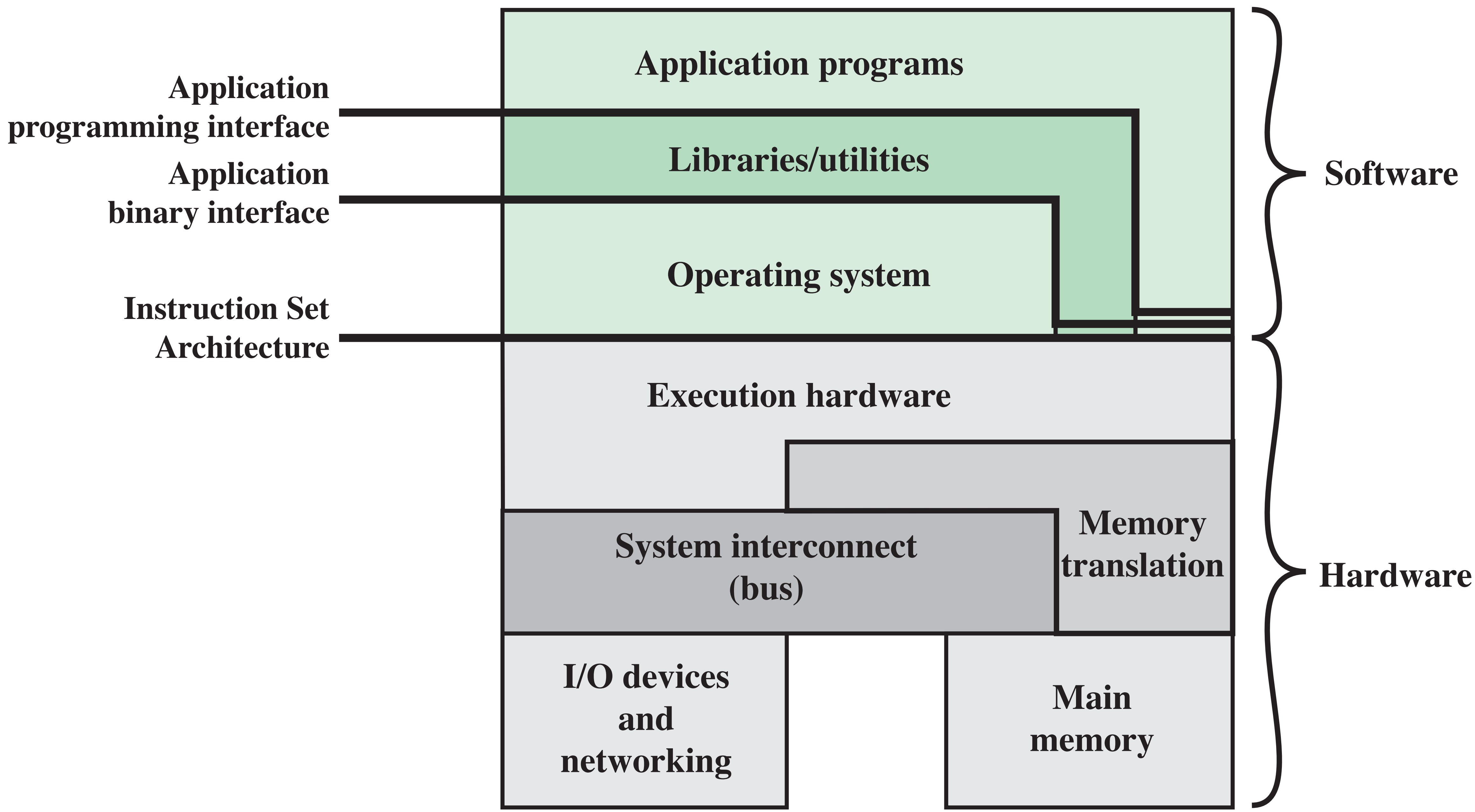


Operating System

- 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

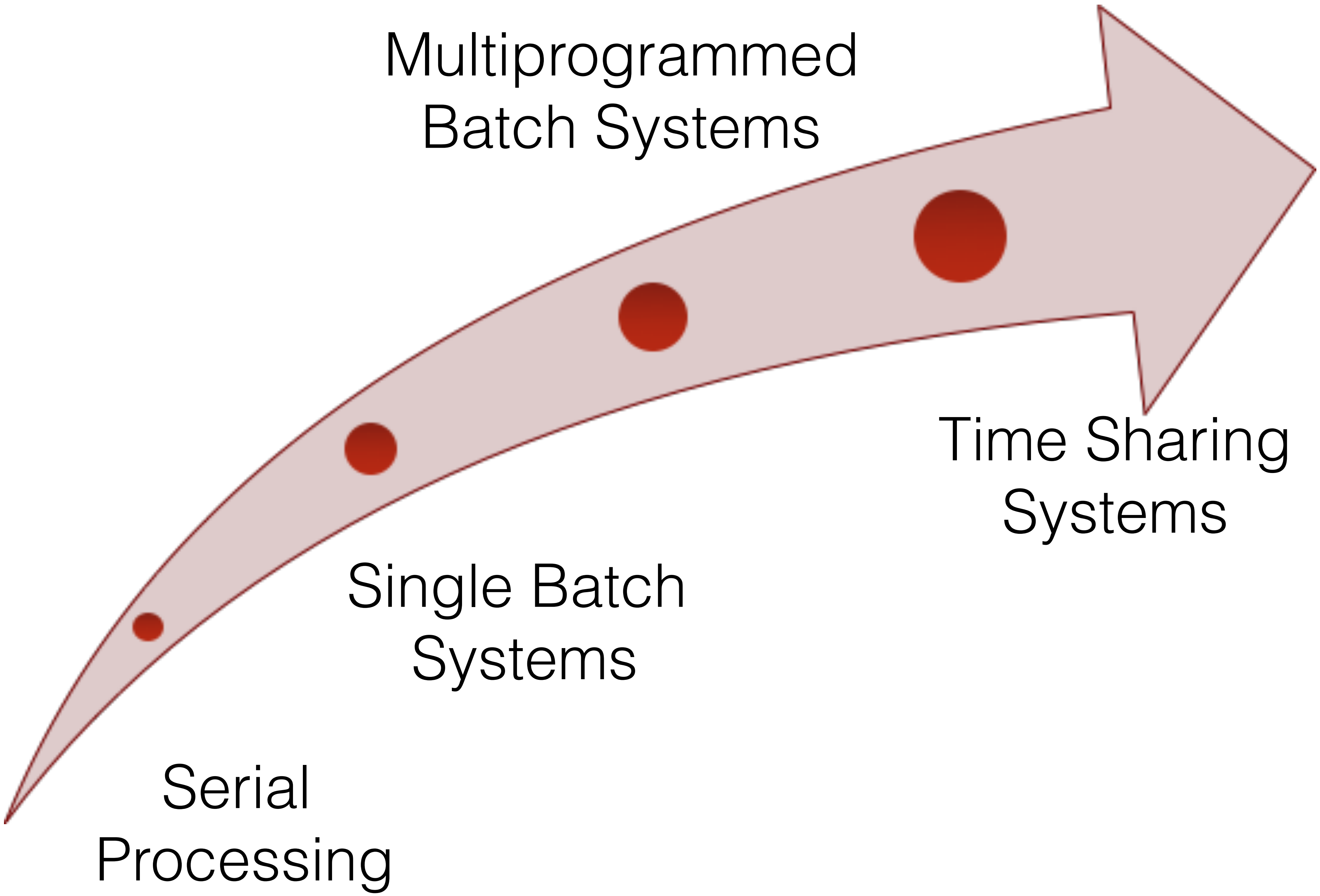
Computer Hardware and Software Structure



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

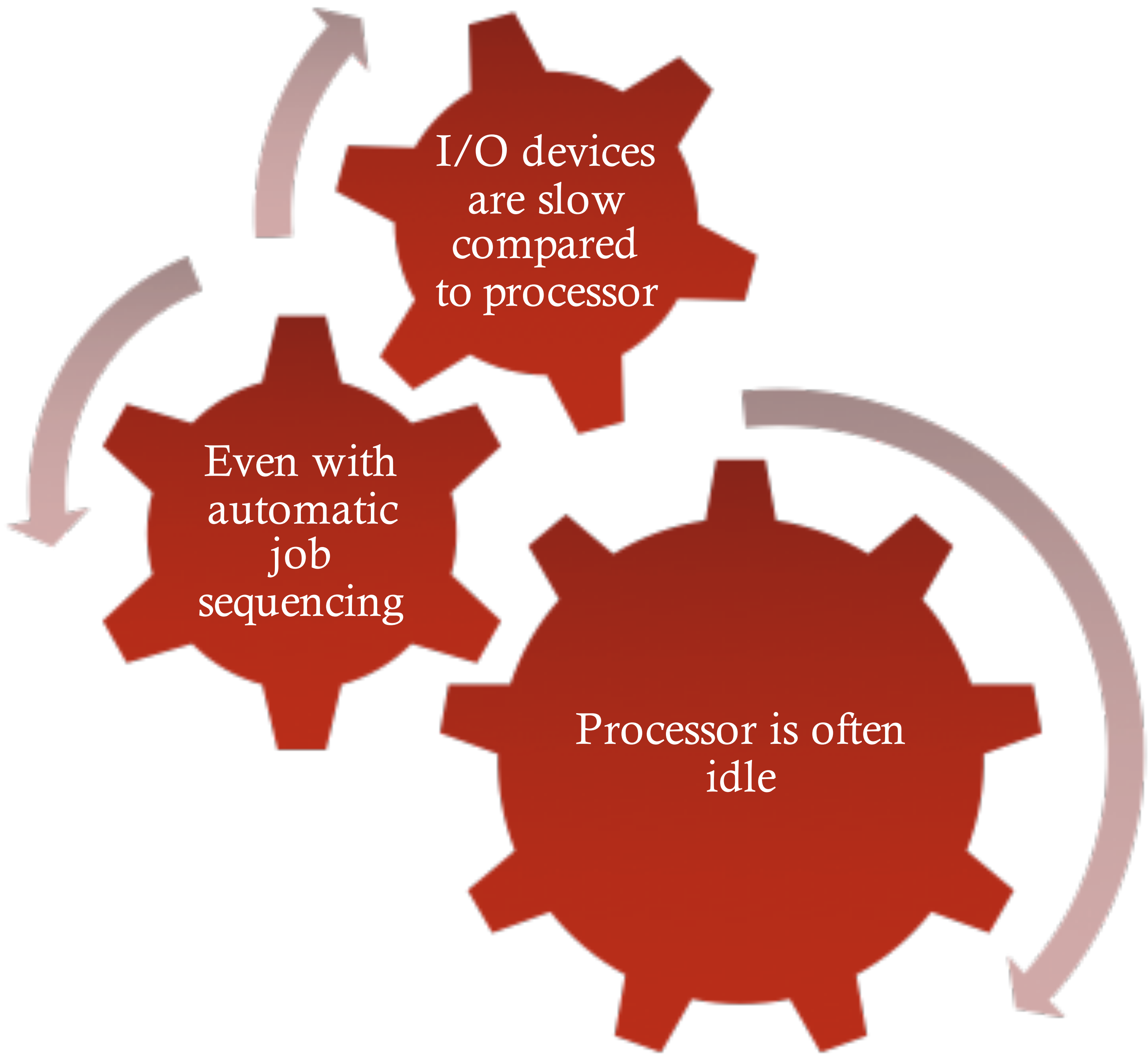
- Earliest Computers:
- No operating system
  - Programmers interacted directly with the computer hardware
- Computers ran from a console with display lights, toggle switches, some form of input device, and a printer
- Users have access to computer in "series"

- 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 μs
Execute 100 instructions	1 μs
Write one record to file	15 μs
TOTAL	31 μs

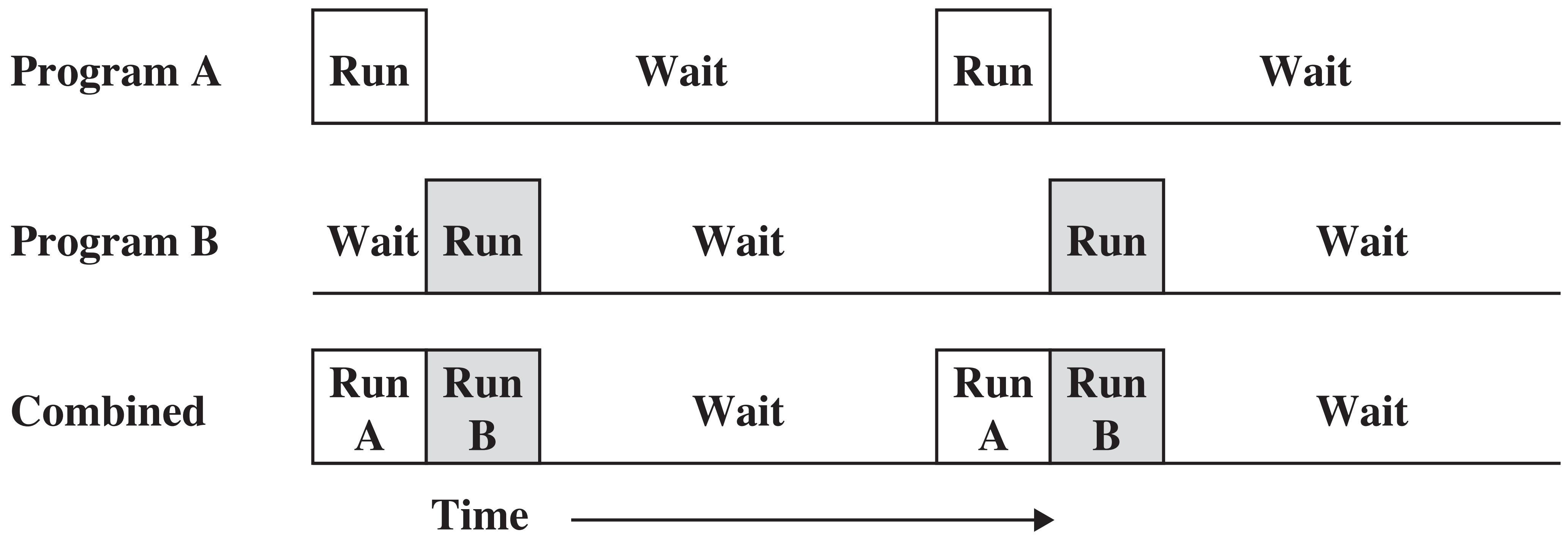
$$\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

Southern Connecticut State University
SCSU

Uniprogramming

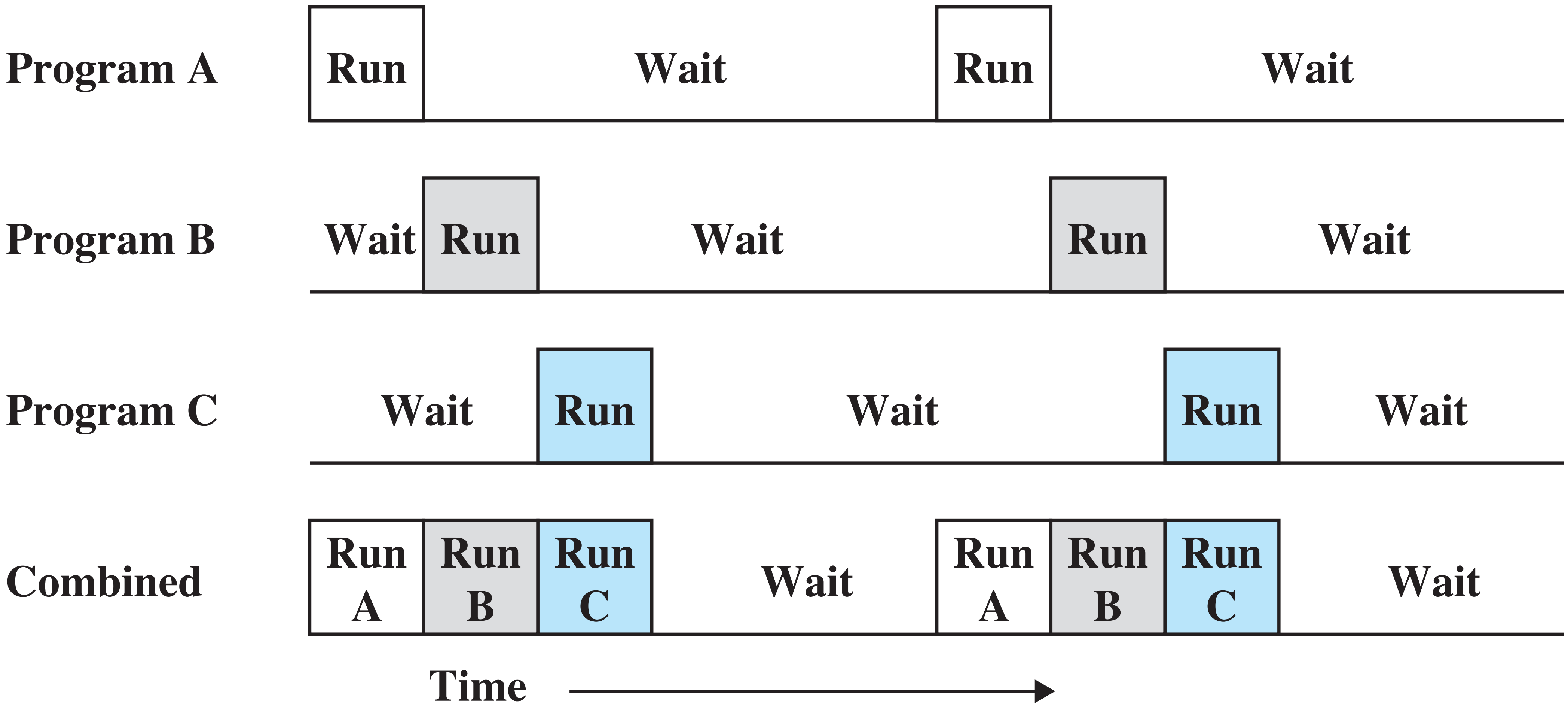
- The processor spends a certain amount of time executing, until it reaches an I/O instruction; it must then wait until that I/O instruction concludes before proceeding

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



- Also known as multitasking
- Memory is expanded to hold three, four, or more programs and switch among all of them

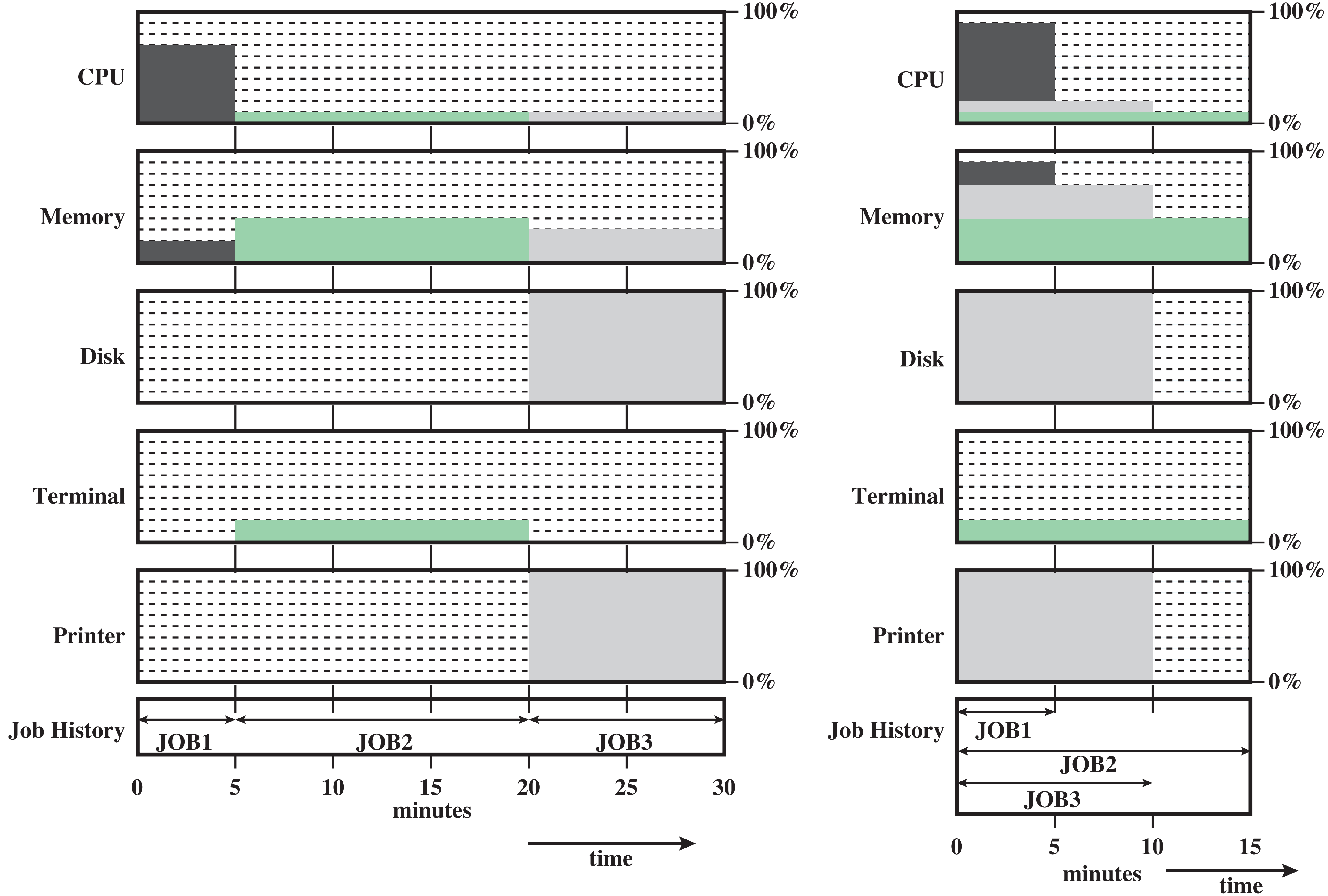
Multiprogramming Example

	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



(a) Uniprogramming

(b) Multiprogramming

Time-Sharing Systems

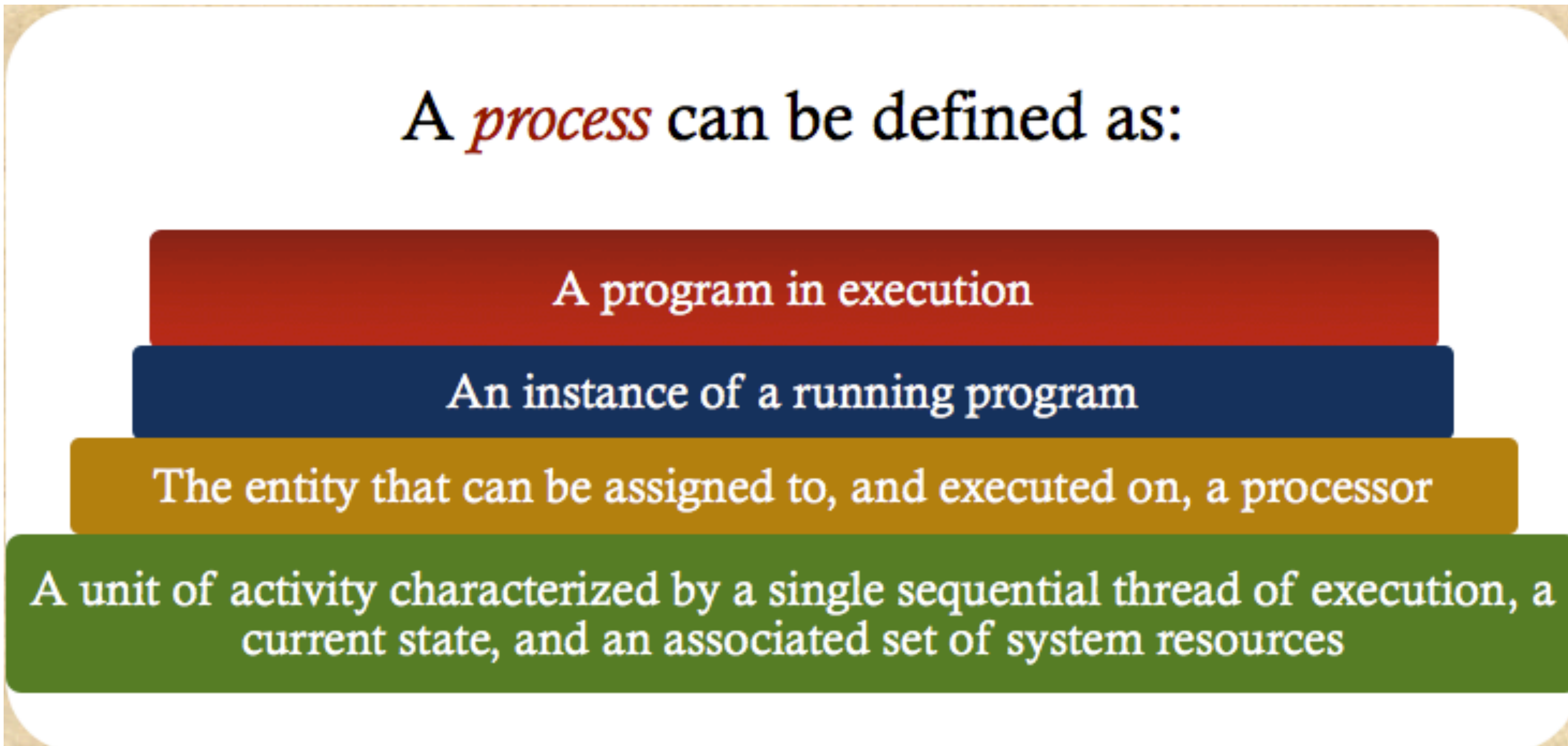
- Can be used to handle multiple interactive jobs
- Processor time is shared among multiple users
- Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation

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

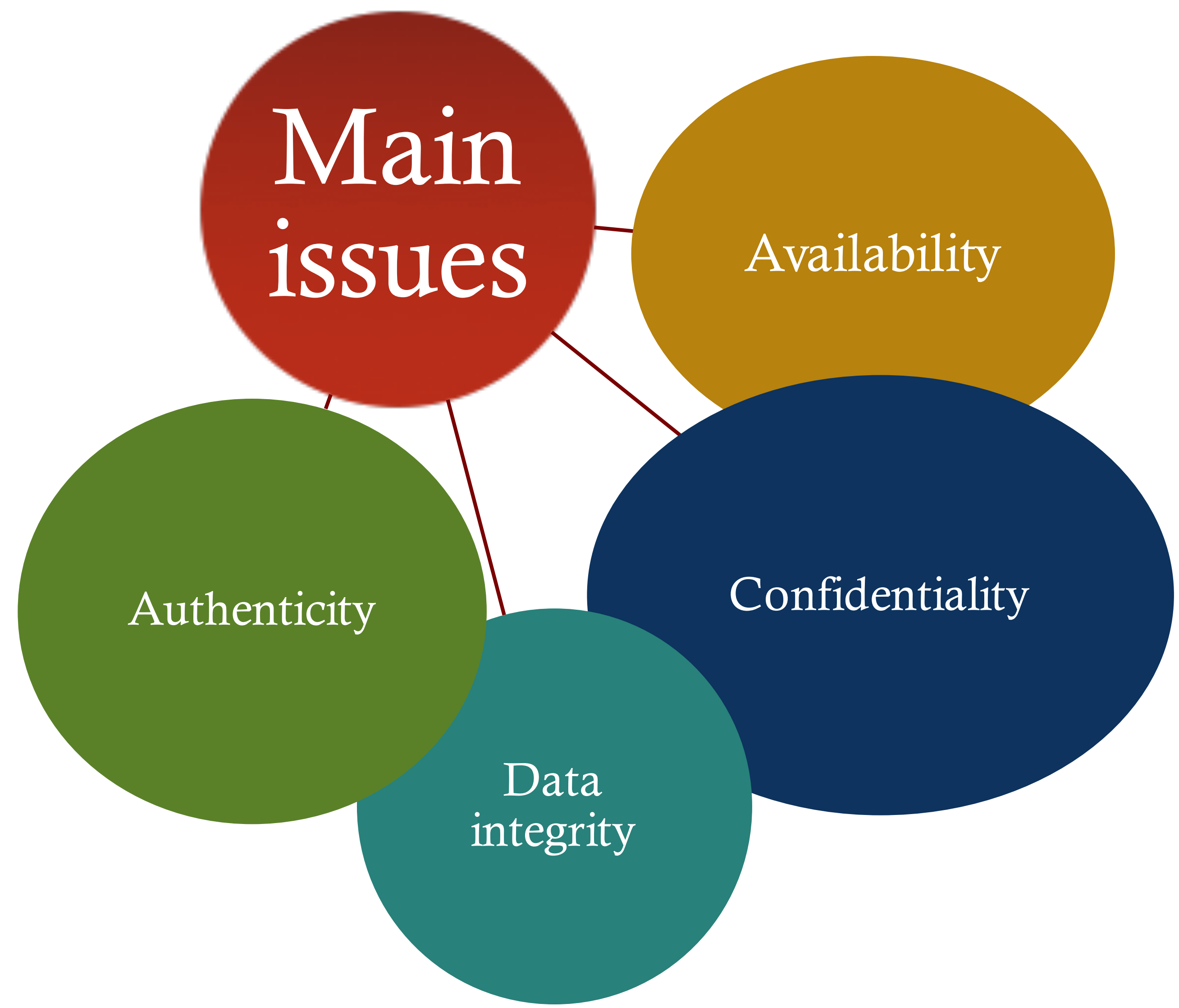


Memory Management

- 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

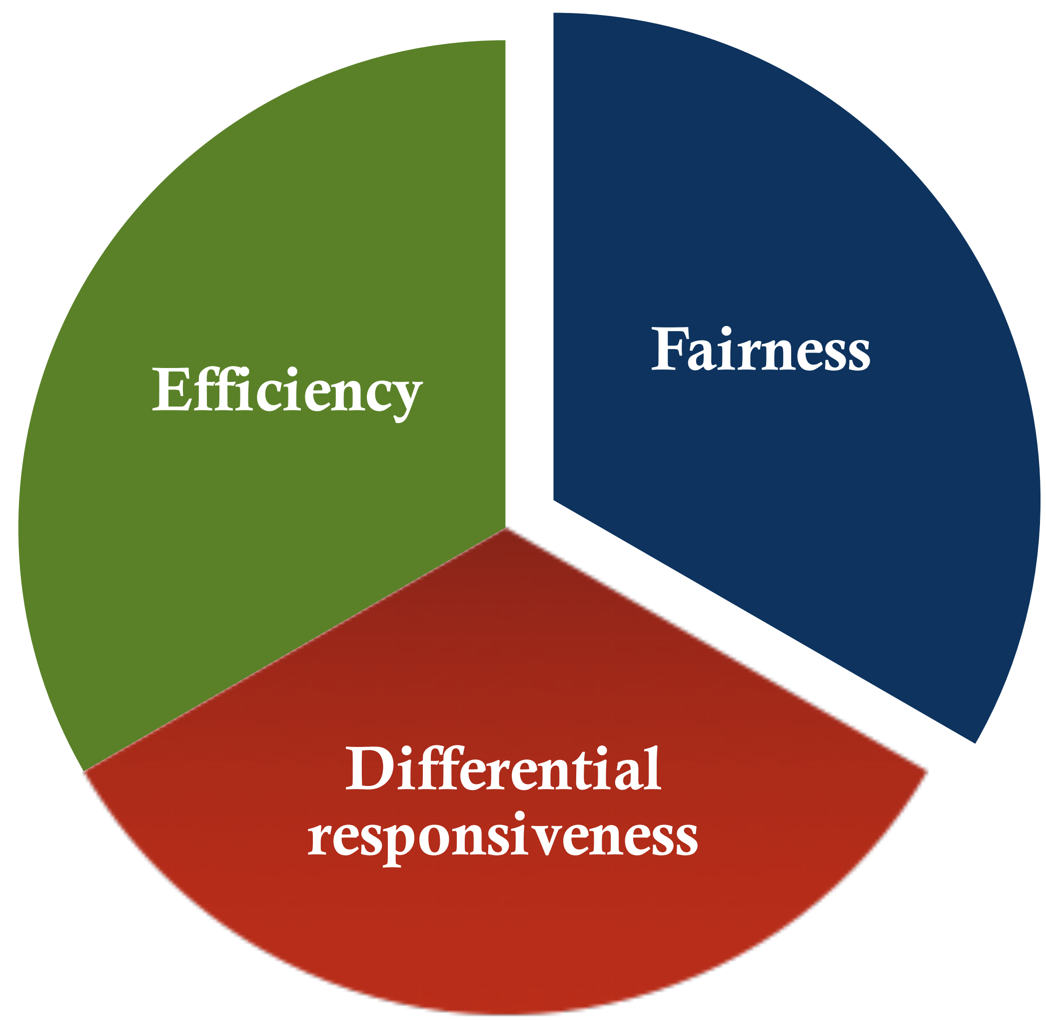
Information Protection and Security

- 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

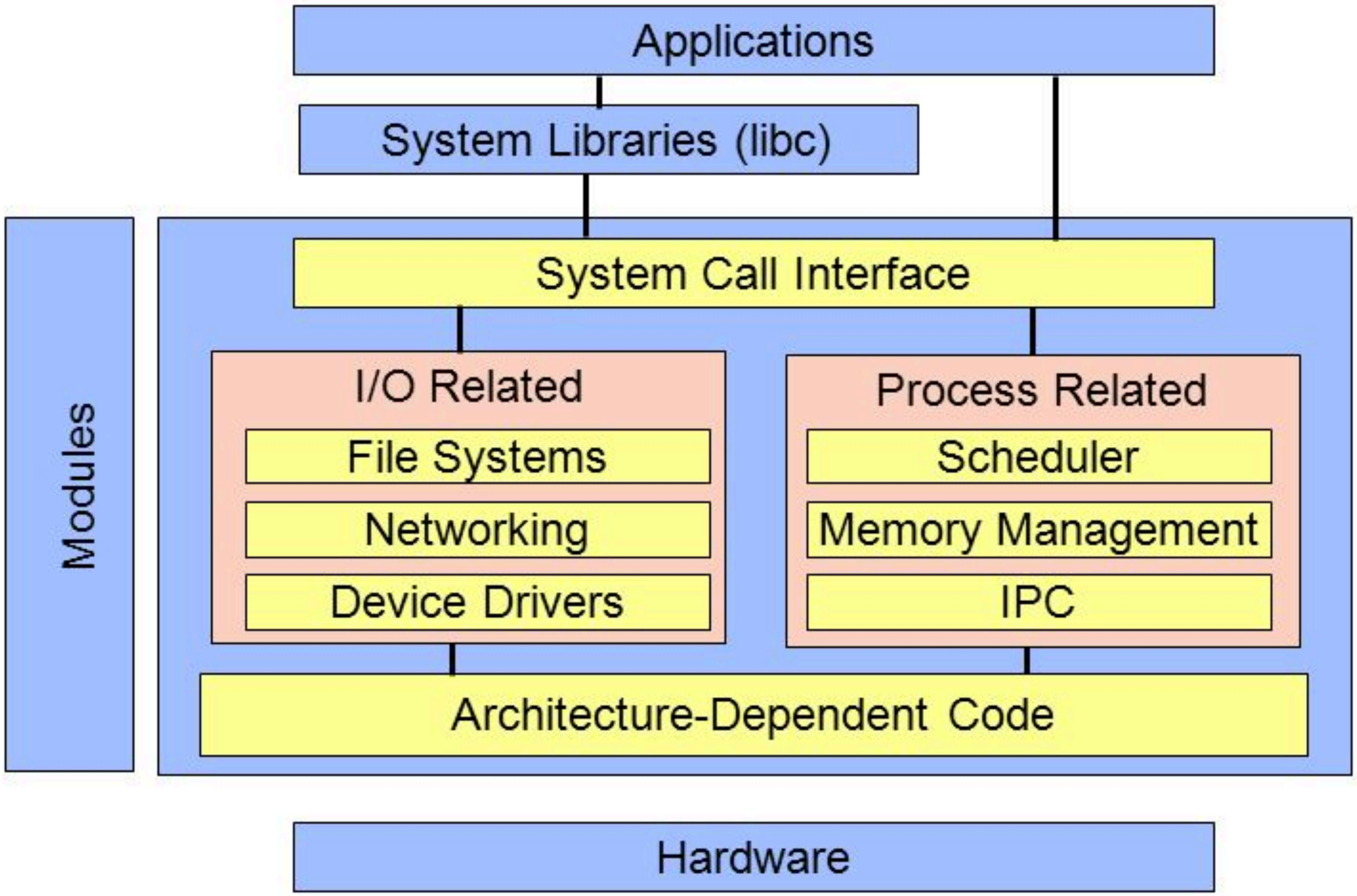


Scheduling and Resource Management

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



Linux Architecture



Dual-Mode Operation

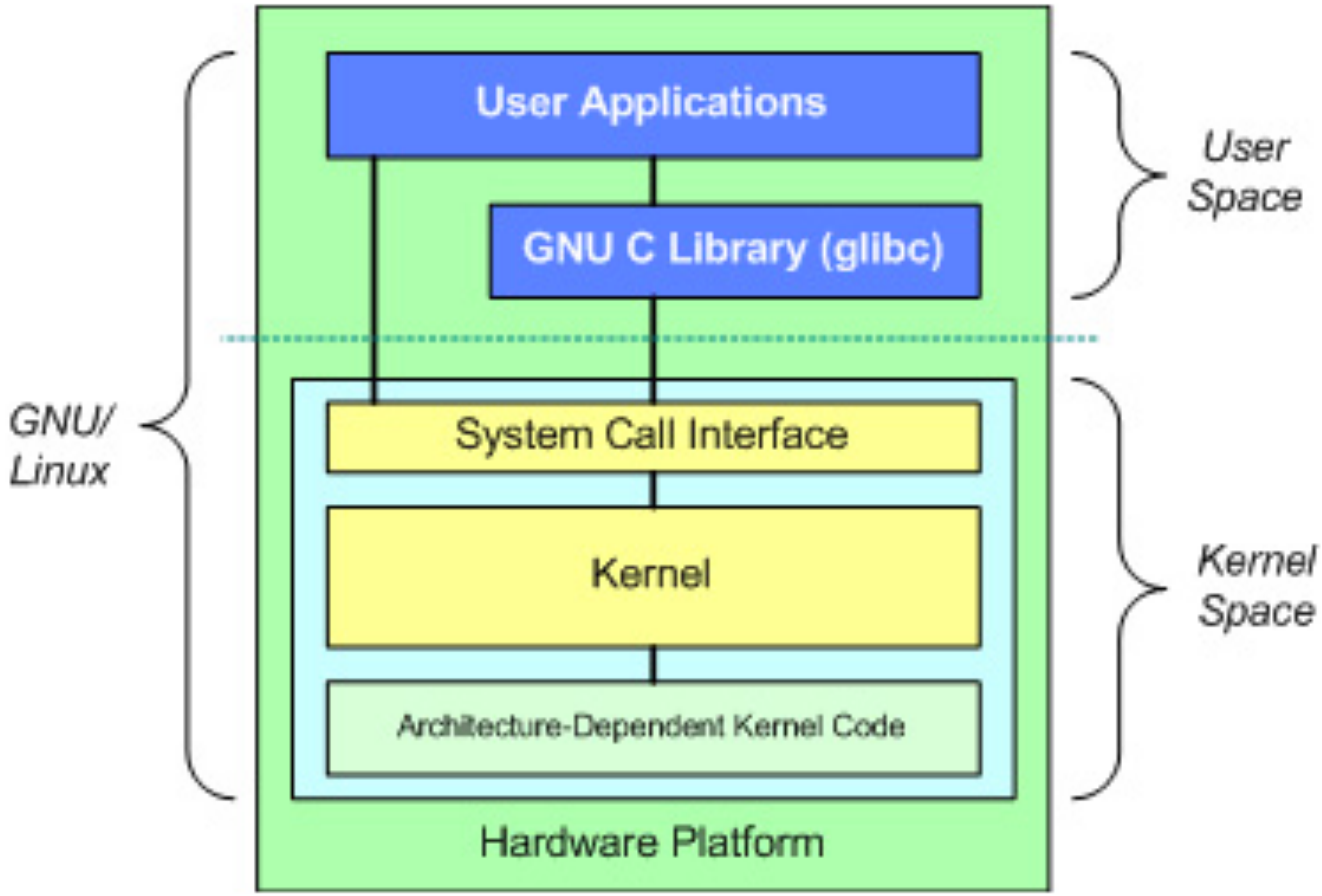
User Mode

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

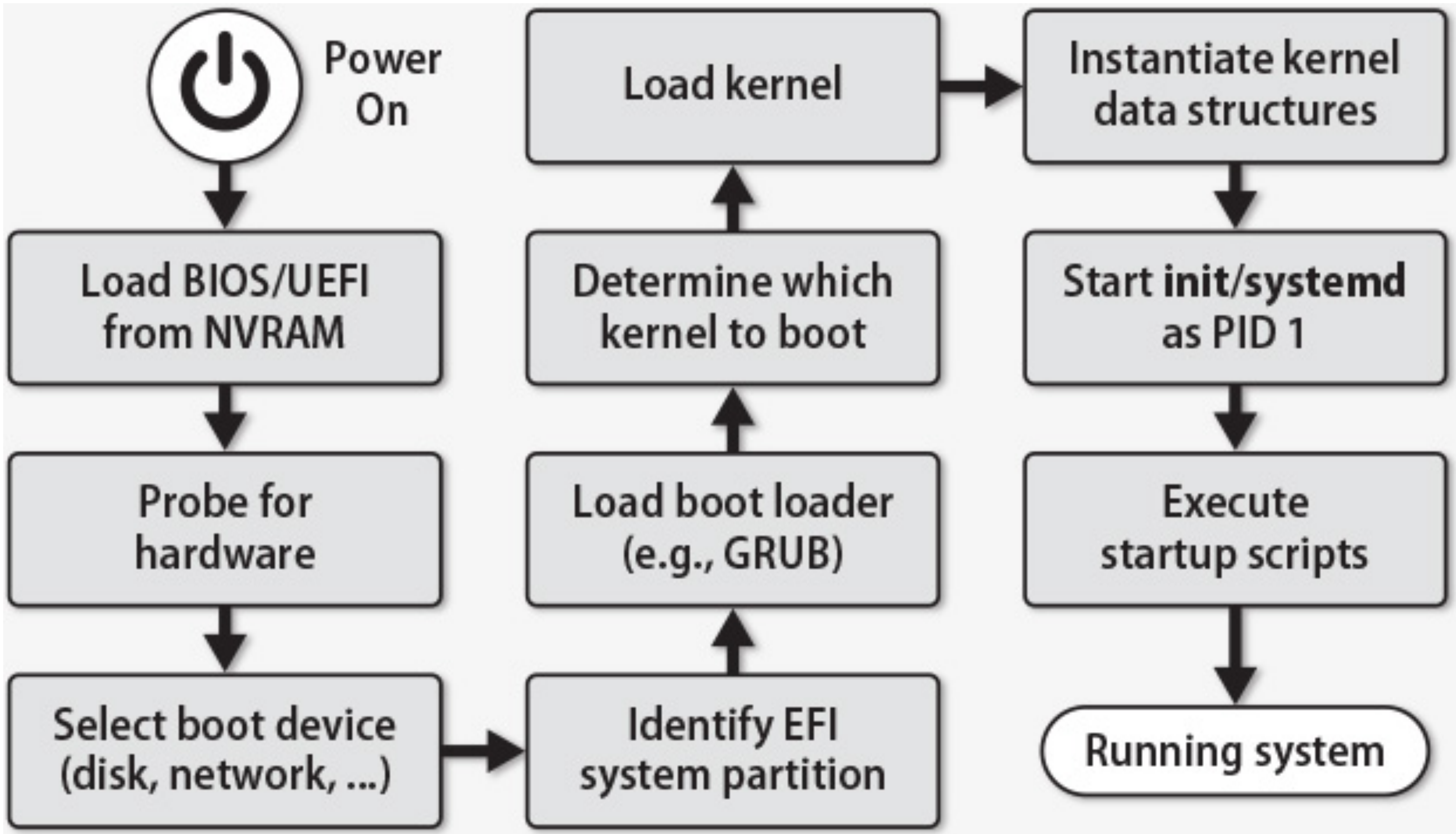
Kernel Mode

- 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

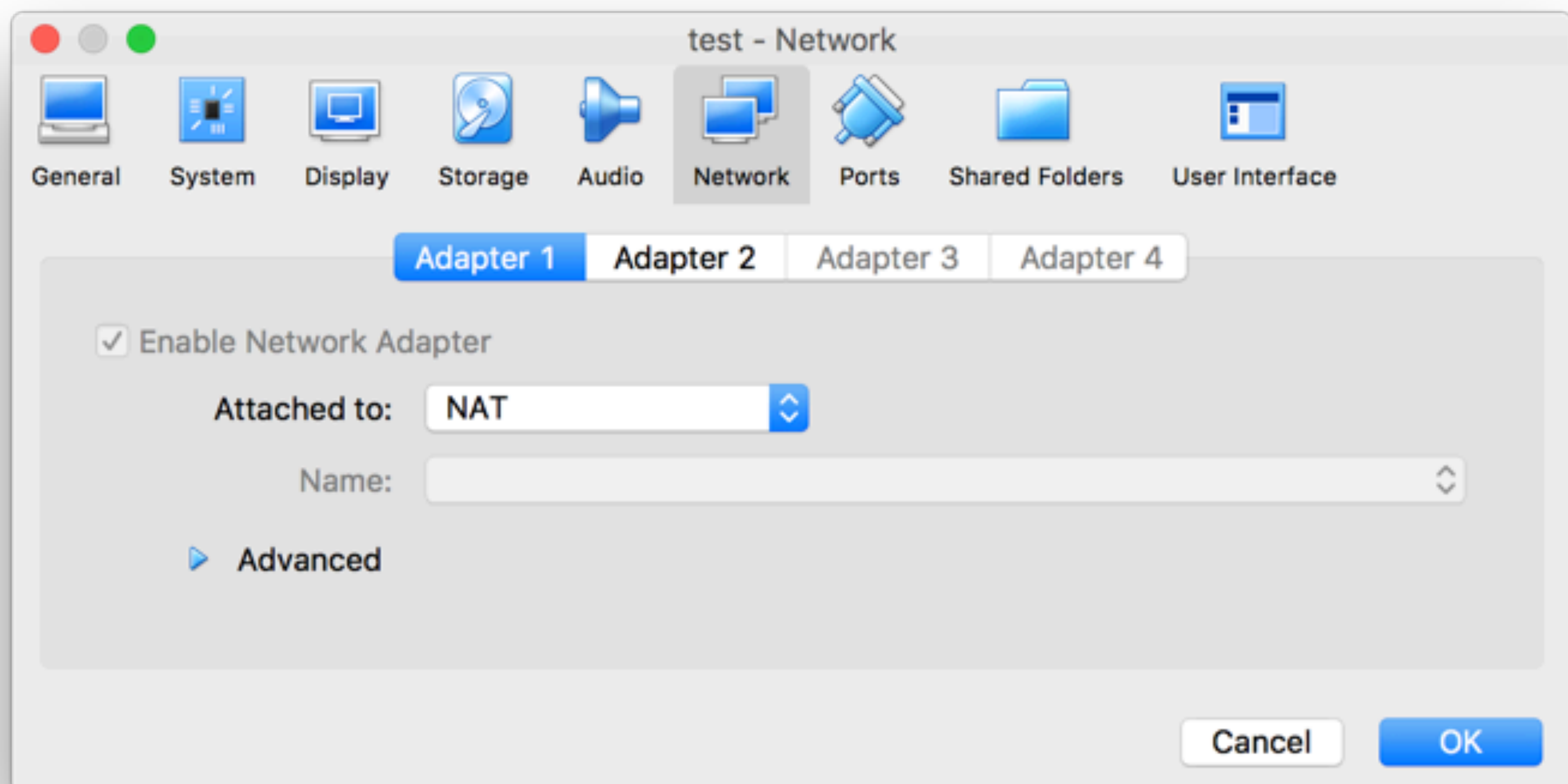
```
[root@localhost ~]# ls -al
```

  - use '- -' before word option

```
[root@localhost ~]# ls --help
```

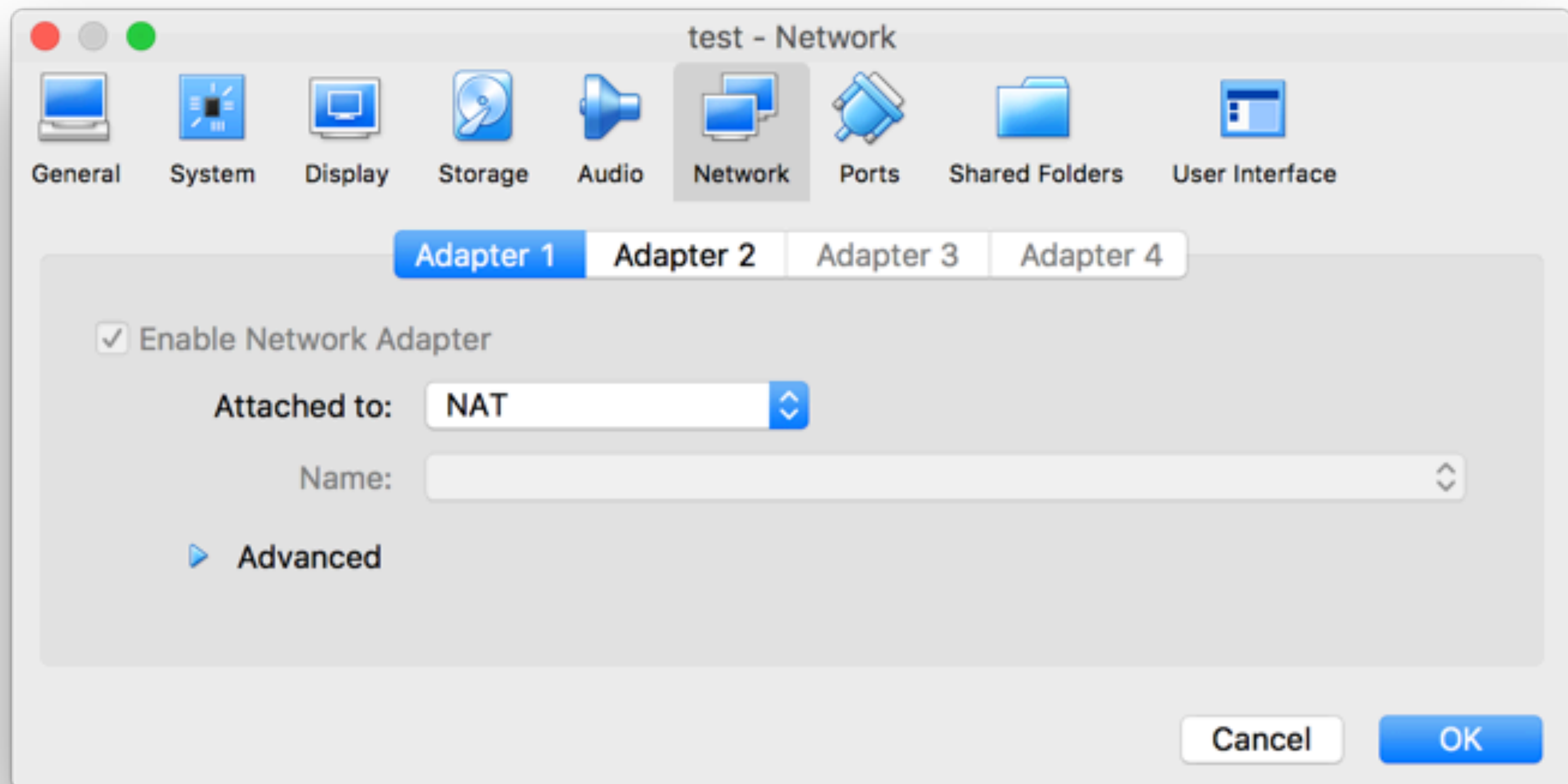
Remote access

- 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

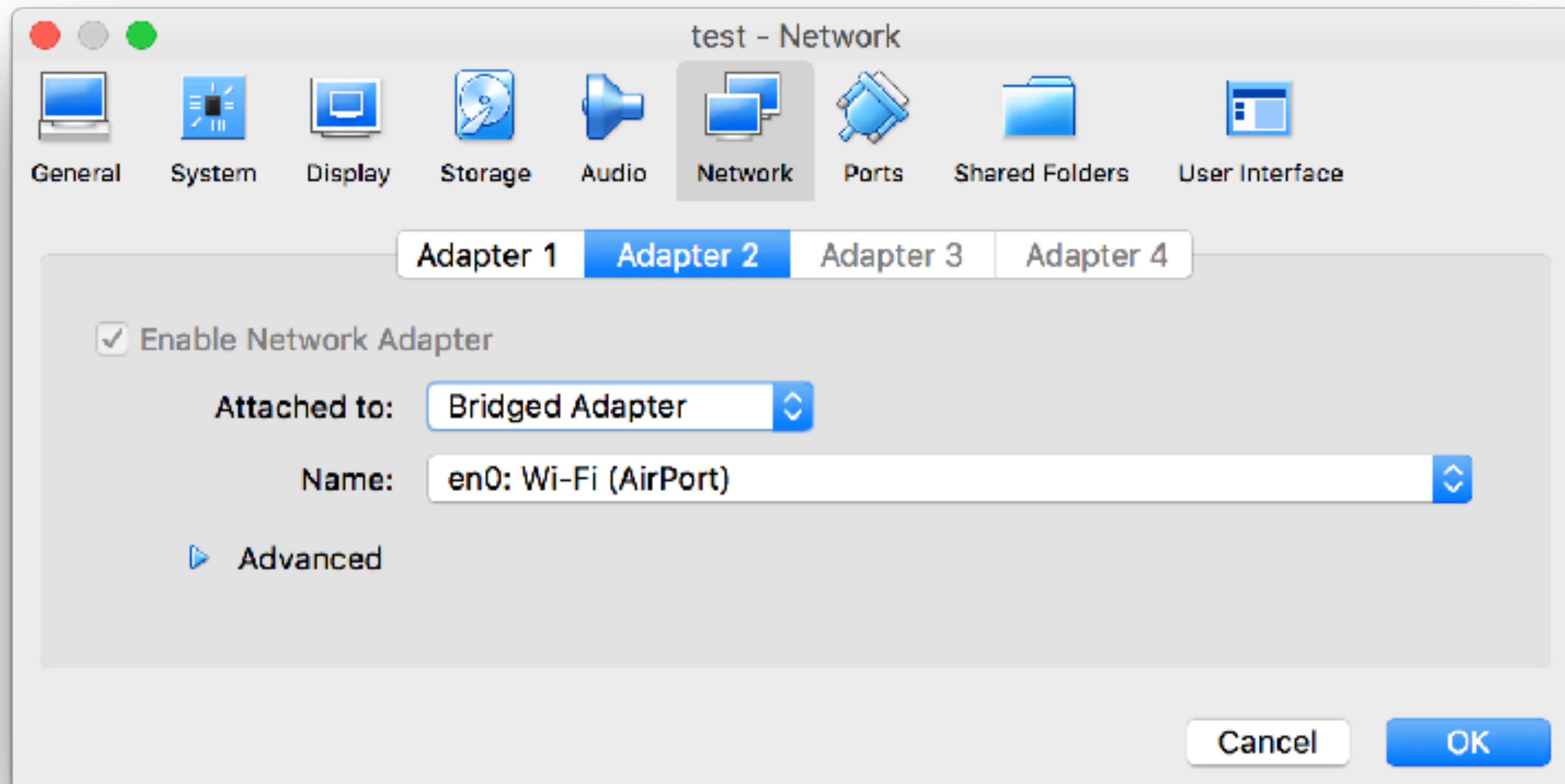
- Select Adapter 1
- Attached to: NAT



Southern Connecticut State University

Remote access

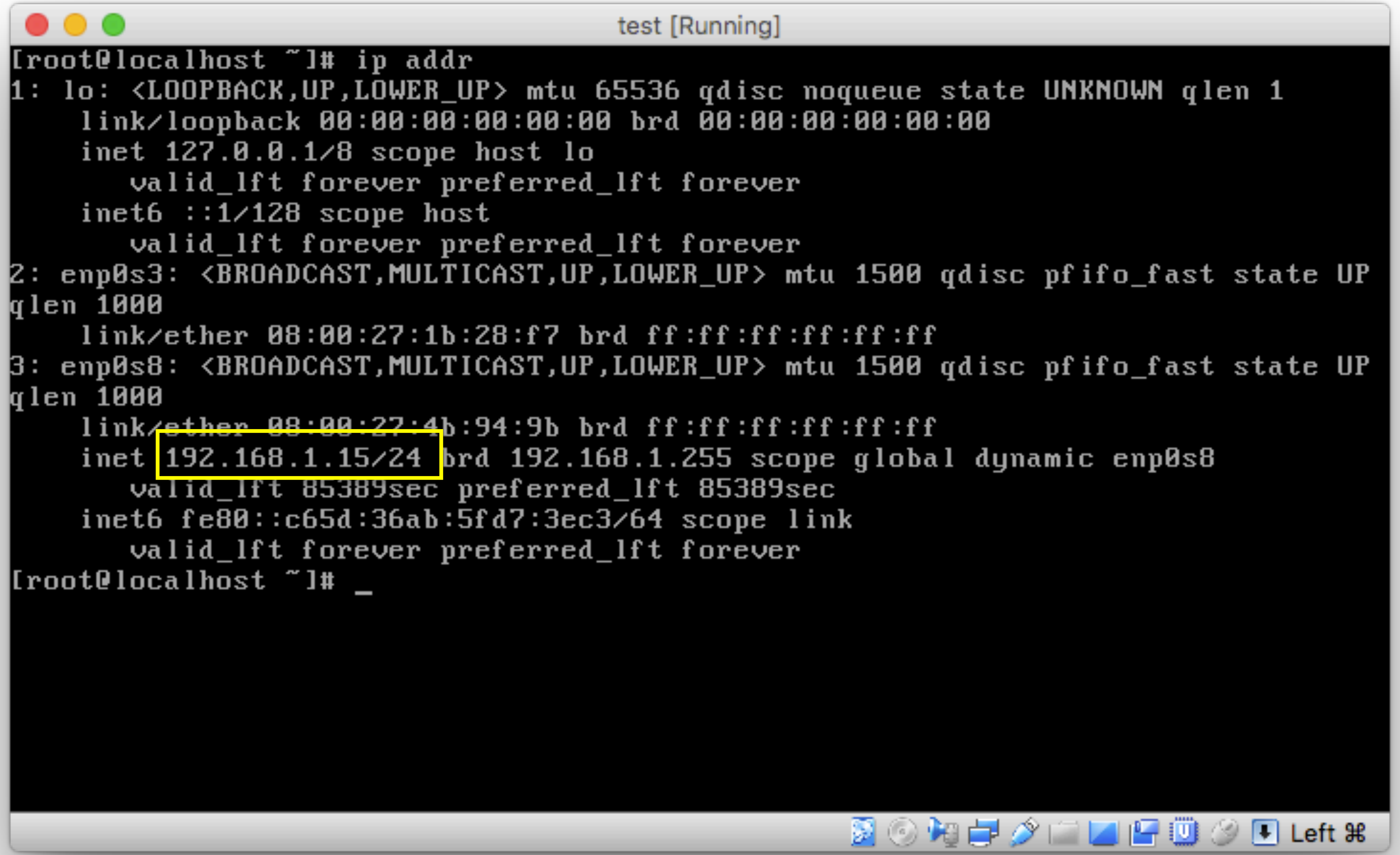
- Select Adapter 2
- Check "Enable Network Adapter"
- Attached to: Bridged Adapter
- Name is your WiFi network card



Southern Connecticut State University
SCSU

Remote access

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



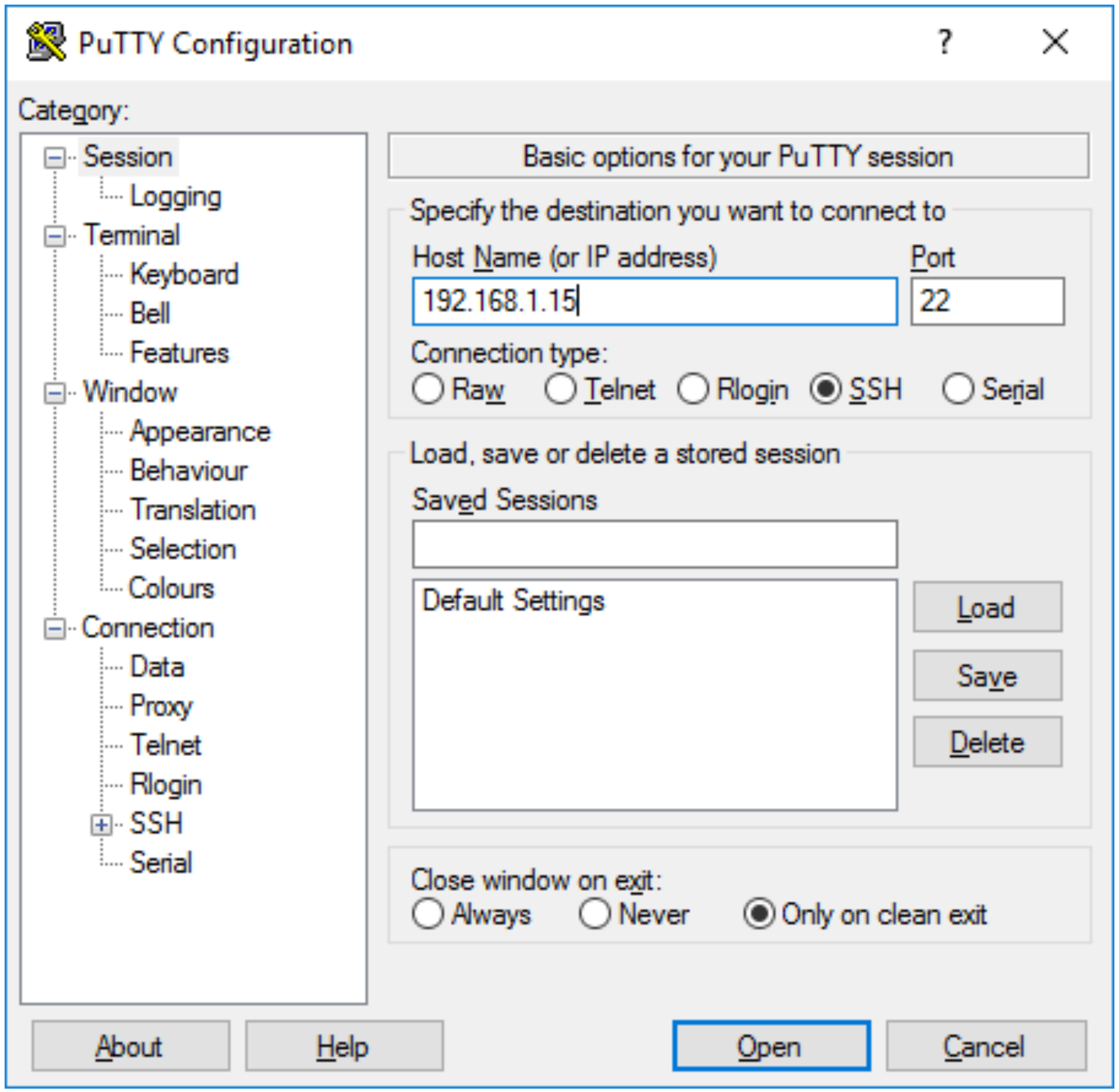
Remote access from your host machine

- 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)

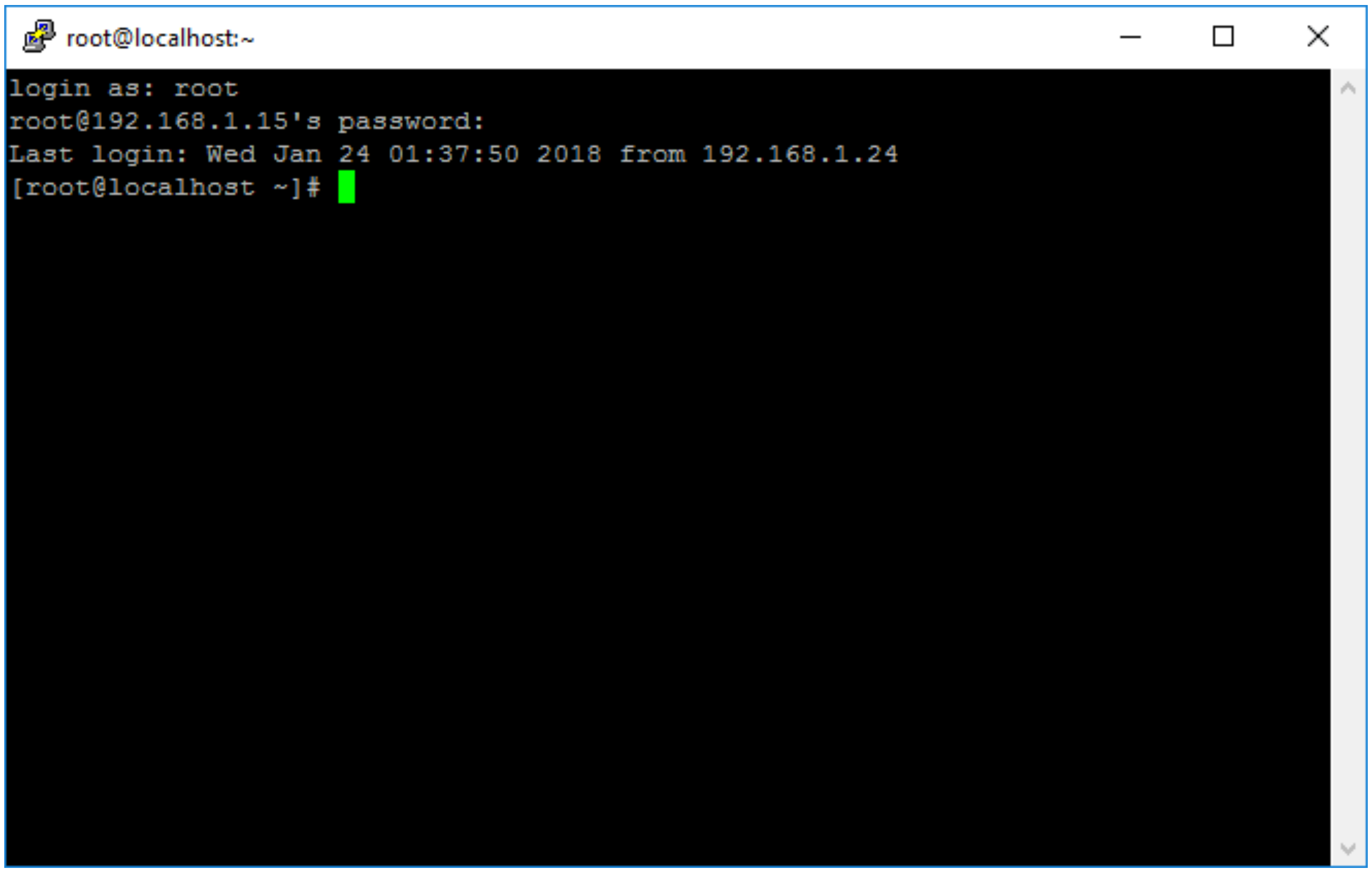
- Open PuTTY
- Enter your VM's IP Address



Southern Connecticut State University

Remote access from your host machine (Windows)

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



Southern Connecticut State University

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
```


Southern Connecticut State University
SC SU

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 '|' 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
```