Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Software Systems

Lectures Week 4

Sessions

Test 1

Introduction to C

Prof. Joseph Vybihal
Computer Science
McGill University

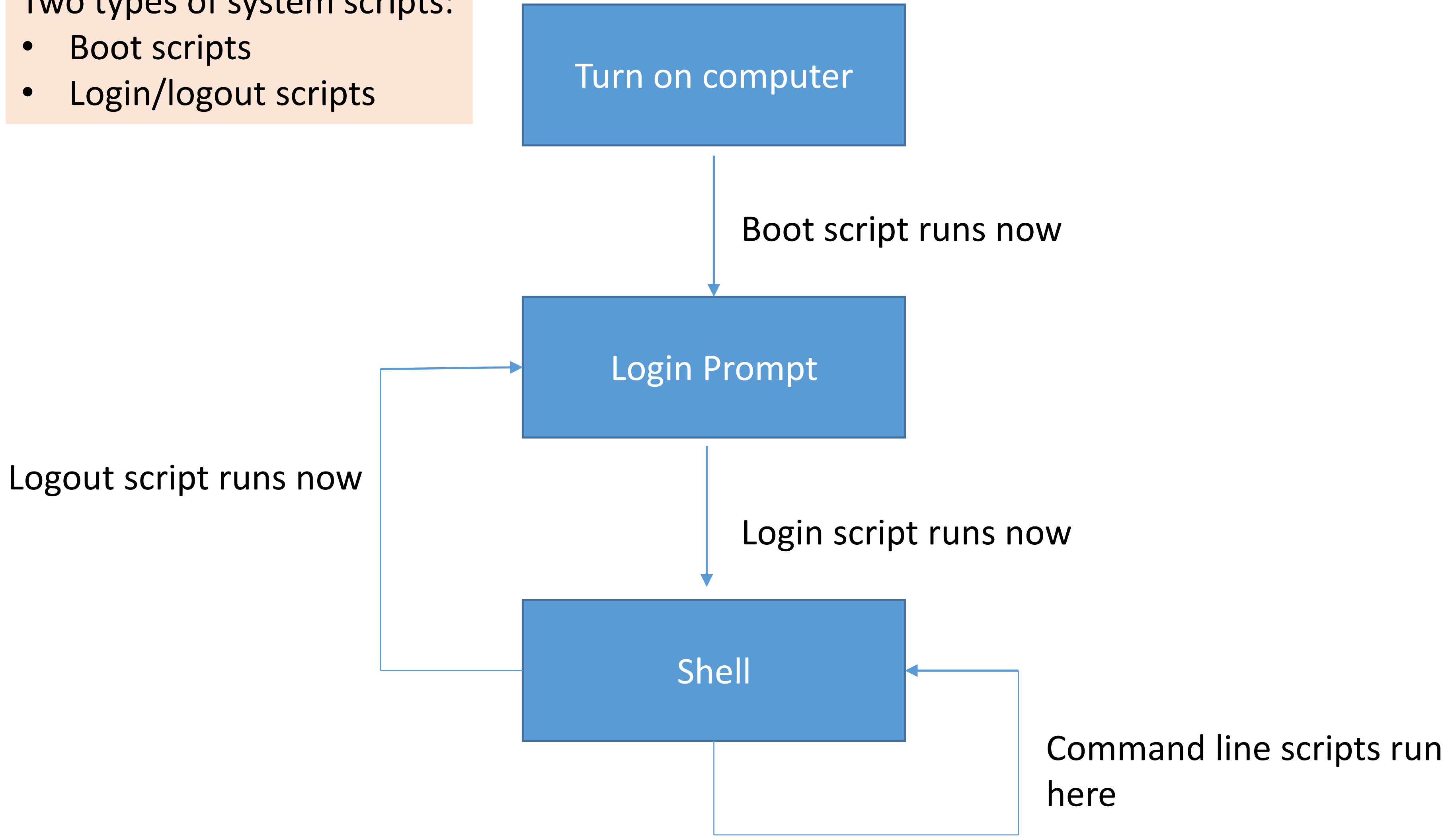
Part 1

System Scripts & Sessions

System Scripts

Two types of system scripts:

- Boot scripts
- Login/logout scripts



System Scripts

System scripts are used by the operating system or shell for configuration purposes.

Since they are similar to regular scripts all the regular script commands also work.

System Scripts

Boot scripts:

- Are created and managed by the root (or system operator).
- When the computer is turned on this script is executed.

Login scripts:

- Similar to a boot script but is managed by the user.
- When the user logs in this script is executed.

Logout scripts:

- Similar to the login script but not always supported by shells.
- When the user logs out this script is executed.

Default Script Files

- Session Script Files

.cshrc	csh login script
.kshrc	ksh login script
.login	sh login script
.bash_profile	bash login script
.bashrc	
.logout	sh and csh logout

- Not Script files

.plan	extra finger info
.forward	email forwarding

Note!

- Login scripts only run when you login, not from the command-line prompt.
- Login scripts require you to use **setenv** or **export** when changing the run-time environment.

Default script files are hidden

<u>Syntax</u>: .name

The dot makes the file hidden.

<u>To see them</u>:

You must list with the -a option to see hidden files.
Many of the system files are hidden files.

ls

ls -a

<u>They are hidden files</u>:

.cshrc

.bashrc

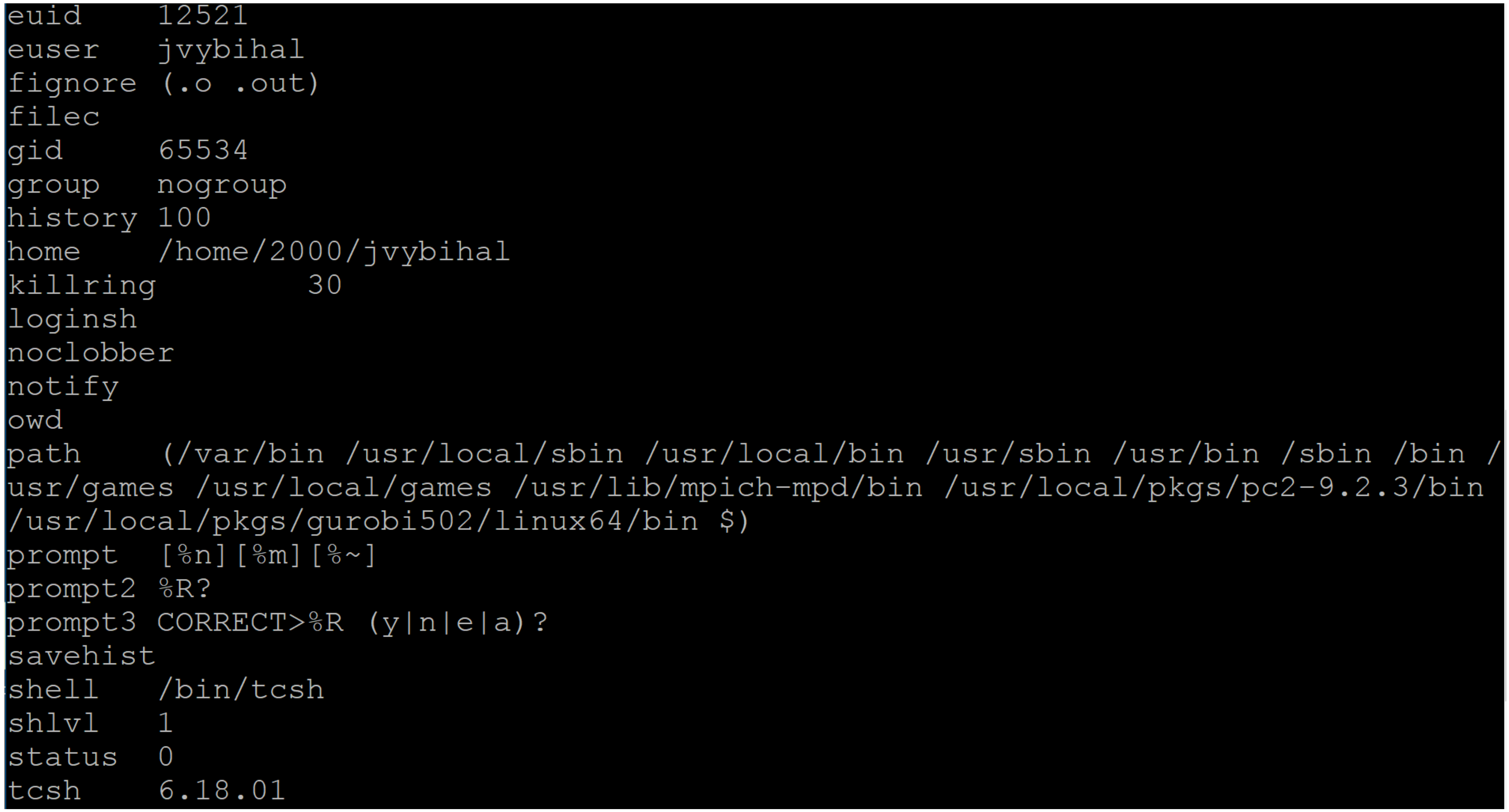
Session & Environment Variables

- These variables are created when you login.
- They contain information about your session.
  - Your IP address
  - The shell you are using
  - Your username
- Environment values can be changed later on, for example:
  - Your prompt
  - Text and background colours
  - The terminal type to process special keyboard keys, eg. F-keys

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Example Session Variables



```
euid    12521
euser   jvybihal
fignore (.o .out)
filec
gid     65534
group   nogroup
history 100
home    /home/2000/jvybihal
killring        30
loginsh
noclobber
notify
owd
path    (/var/bin /usr/local/sbin /usr/local/bin /usr/sbin /usr/bin /sbin /bin /
usr/games /usr/local/games /usr/lib/mpich-mpd/bin /usr/local/pkgs/pc2-9.2.3/bin
/usr/local/pkgs/gurobi502/linux64/bin $)
prompt  [%n][%m][%~]
prompt2 %R?
prompt3 CORRECT>%R (y|n|e|a)?
savehist
shell   /bin/tcsh
shlvl   1
status  0
tcsh    6.18.01
```

Use the **set** command to see your variables.

Environment Variables

- In Bash:

```
export SHELL_VAR=value
```

or

```
SHELL_VAR=value
```

- In tcsh:

```
setenv SHELL_VAR value
```

Environment Variables Example

- Setting your prompt:
  - set prompt="I am the best>>"
  - export ps1="I am the best>>"
- Setting the terminal type:
  - set TERM="VT100"
  - export TERM="VT100"

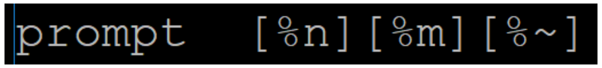
The prompt is the command-line symbol that is displayed when the shell is waiting for your next command.

The TERM, or terminal, describes your keyboard and screen. VT100 is a standard simple 256 color screen with a keyboard that has F-keys.

Login Scripting

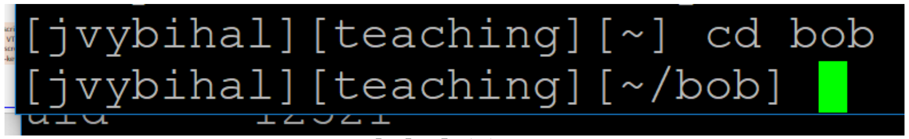
- To customize your account:
  - set prompt = "Best Student $home> "
  - setenv prompt "Best Student $home> "
  - set ps1="Best Student $home>"

Notice the use of $variables within your configuration.



```
prompt   [%n][%m][%~]
```

%n → user name
%m → machine name
%~ → current directory



```
[jvybihal][teaching][~] cd bob
[jvybihal][teaching][~/bob] █
```

Login Scripting

- To customize your account:

  – set history = 100

This will remember the commands you enter at the keyboard. In the example about 100 commands you type.

Using the up and down arrow keys you can cycle through the commands you have been using. Once you found the command you want pressing the enter key will execute that command.

You can directly invoke a command from the command-line prompt by using the “bang” command, the exclamation mark, !, following by the index number of the command. Using the example above, it would be the position in the list from 0 to 99, since we have 100.

Login Scripting

- To customize your account:

  – alias yourTag oneWordCommand

  – alias yourTag 'multi-word command'

  – Example:

```
alias ll 'ls -l-a'
alias dir ls
```

Login Scripting

- The PATH is a set of directories a shell searches for executables.
  - In Unix, it is a colon (:) separated list.
    - You can use the **which** command to figure out what file path is needed.

- The CLASSPATH is the set of directories the JVM searches when loading classes.

Path and Classpath

- set path=/home/foo:/bin/exe
  - set path=$path:/bla/folder:/bla2

Notice the reuse of $path in the configuration.

- export path=/home/foo:/bin/exe

- set classpath=/home/java:bin/java

```
path     (/var/bin /usr/local/sbin /usr/local/bin /usr/sbin /usr/bin /sbin
usr/games /usr/local/games /usr/lib/mpich-mpd/bin /usr/local/pkgs/pc2-9.2
/usr/local/pkgs/gurobi502/linux64/bin $)
```

Other “start-up” things . . .

- You can set your default editor.
  - EDITOR=vi
- Some applications might require you to set up an environment variable.
  - PVM-ROOT=/usr/local
  - set FILE=”*.txt”
  - set DIR=”/usr/jack/backup

Using the environment variables from the prev slide:

```
#!/bin/bash
cp “$FILE $DIR”
```

We would execute the program without the need of command line argument:

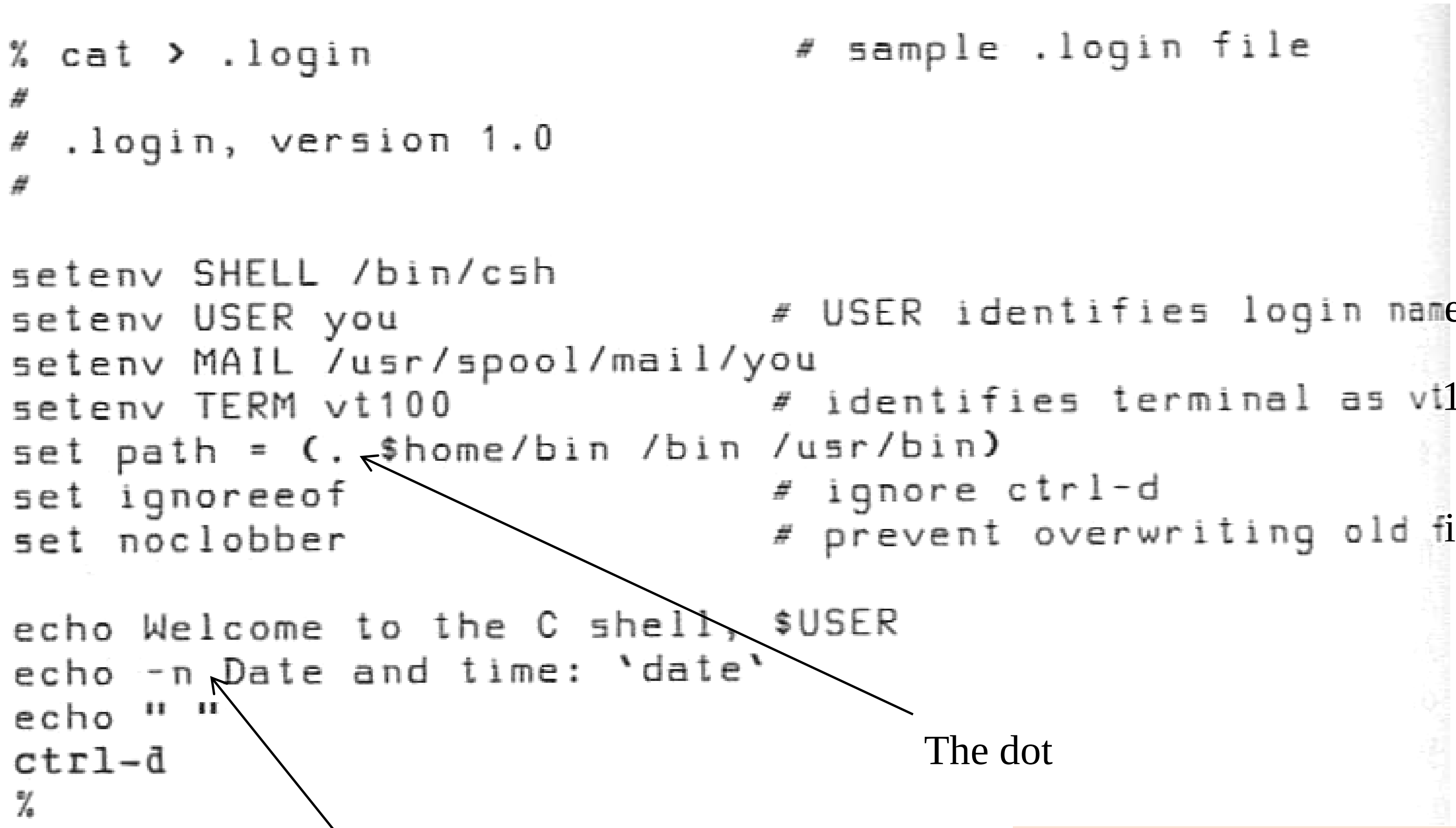
```
$ ./backup
```

The variables $FILE and $DIR are using the the ./backup script.

Other defaults…

•HOME	path to home directory
•SHELL	path to your shell
•TERM	type of terminal I/O
•USER	your user name
•PWD	your current directory

SH .login Example



```
% cat > .login                    # sample .login file
#
# .login, version 1.0
#

setenv SHELL /bin/csh
setenv USER you                   # USER identifies login name
setenv MAIL /usr/spool/mail/you
setenv TERM vt100                 # identifies terminal as vt100
set path = (. $home/bin /bin /usr/bin)
set ignoreeof                     # ignore ctrl-d
set noclobber                     # prevent overwriting old file

echo Welcome to the C shell, $USER
echo -n Date and time: `date`
echo " "
ctrl-d
%
```

The dot

-n do not print trailing new line

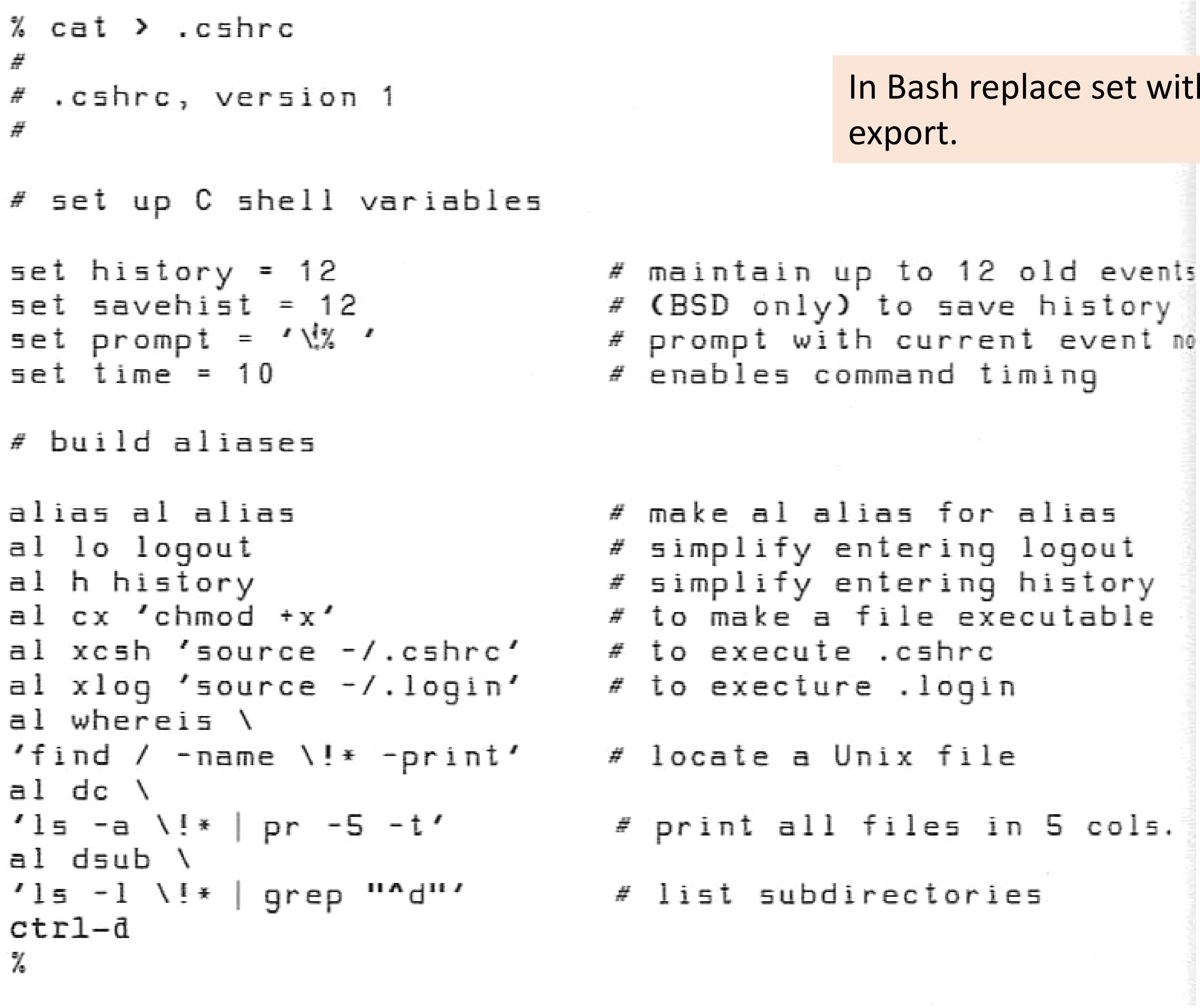
In Bash replace setenv and set with export.

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

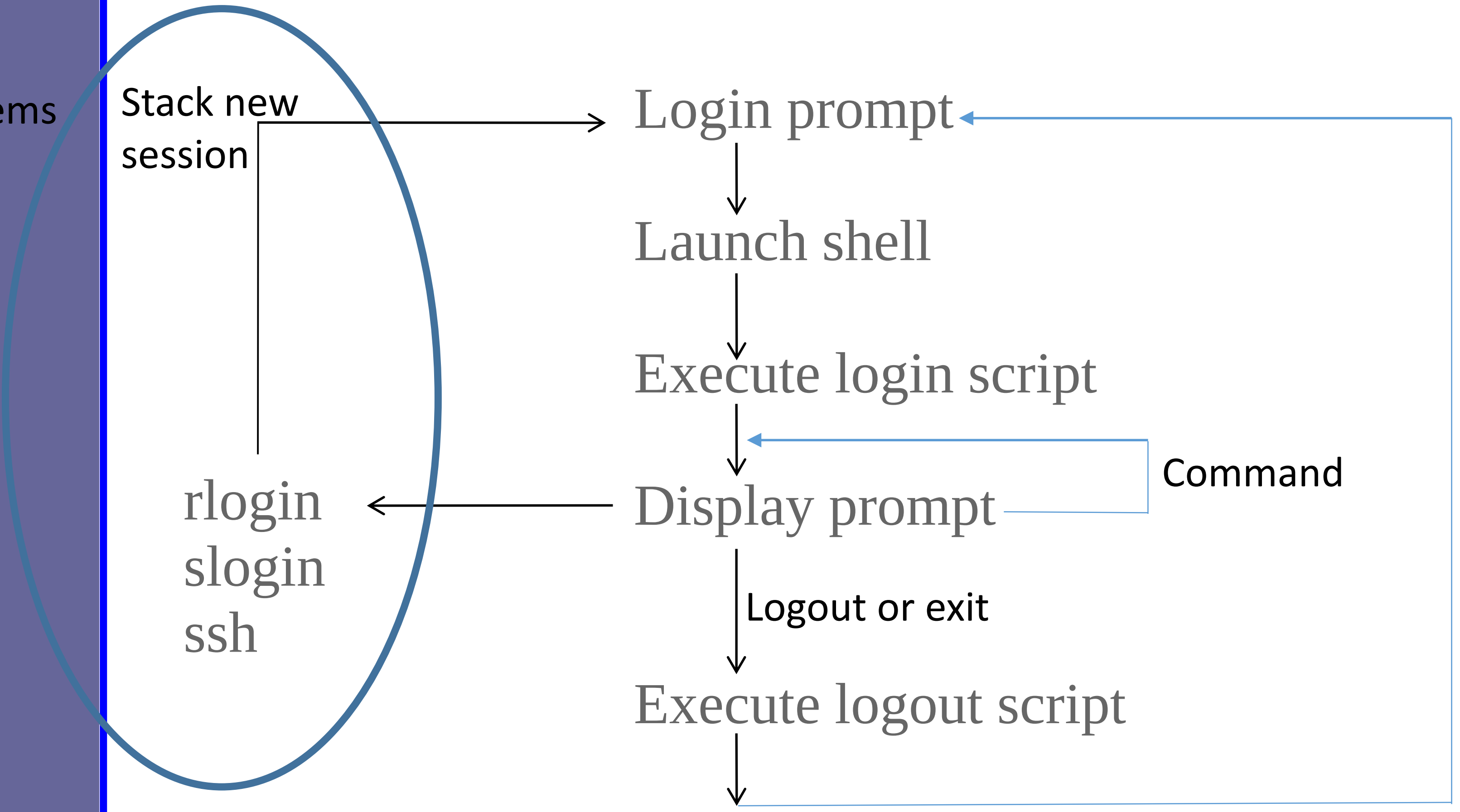
CSH .cshrc Example

In Bash replace set with export.



```
% cat > .cshrc
#
# .cshrc, version 1
#

# set up C shell variables

set history = 12                 # maintain up to 12 old events
set savehist = 12                # (BSD only) to save history
set prompt = '\!% '              # prompt with current event no
set time = 10                    # enables command timing

# build aliases

alias al alias                   # make al alias for alias
al lo logout                     # simplify entering logout
al h history                     # simplify entering history
al cx 'chmod +x'                 # to make a file executable
al xcsh 'source ~/.cshrc'        # to execute .cshrc
al xlog 'source ~/.login'        # to execture .login
al whereis \
'find / -name \!* -print'        # locate a Unix file
al dc \
'ls -a \!* | pr -5 -t'            # print all files in 5 cols.
al dsub \
'ls -l \!* | grep "^d"'           # list subdirectories
ctrl-d
%
```

A Session

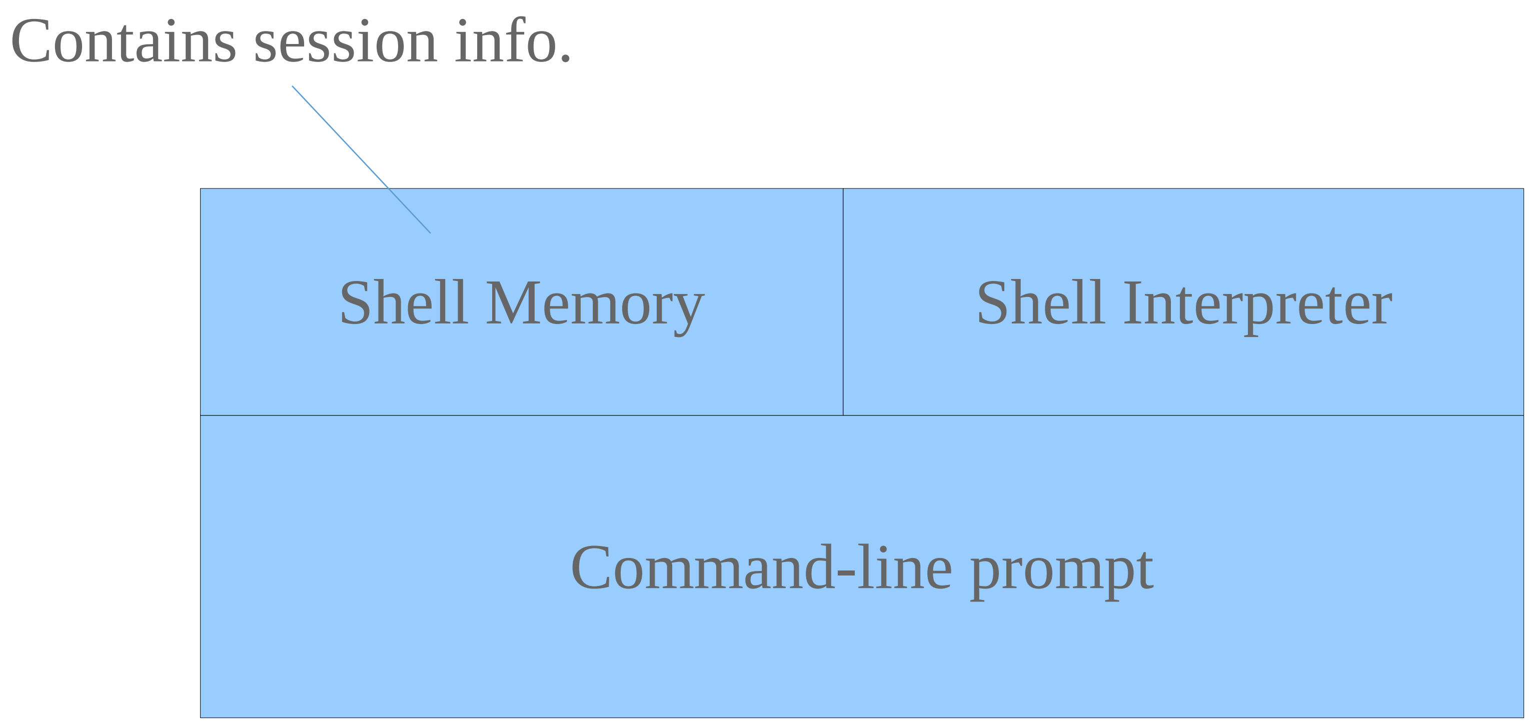


Sessions are stacked and independent from one another, however they may share the same hard drive.

The Importance of Passwords

- All resources are tagged with your username
  - Eg: ls –l
- If anyone gets access to your user name then they become you!
  - Eg: root user controls the entire system
  - Stolen identities!
- Good password strategy?
  - Take a sentence: I love my dog Raoul
  - Mix initialize it: iLmDr
  - Add symbols: iLm!Dr1
  - Easy to remember but hard to guess

The Shell



A Session comprises the run-time environment available to the user, called the Shell Environment.

Environment Session Information

```
LOGNAME=jvybihal
HOME=/home/user/jvybihal
PATH=/bin:/usr/bin:/usr/local/bin
MAIL=/var/mail/jvybihal
SHELL=tcsh
SSH_CONNECTION=132.206.51.226 2444 132.206.3.142 22
SSH_TTY=/dev/pts/6
TERM=xterm
HOSTTYPE=i386-linux
VENDOR=intel
OSTYPE=linux
MACHTYPE=i386
SHLVL=1
PWD=/home/user/jvybihal
GROUP=unknown
```

This can be found in the shell memory. Use the command SET to see the shell memory.

Session Related Commands

Session Related Commands

- WHOAMI
  - Reports on your user name
  - Syntax: whoami
- WHO
  - Tells you who is logged into the server
  - Syntax: who
- FINGER
  - Find detailed information about a user
  - Syntax: finger
- PWD
  - Displays the directory you are currently within
  - Syntax: pwd

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Session Related Commands

- LOGOUT
  - Terminates the connection to the server
  - Syntax: logout
- EXIT
  - Closes the shell and keeps you logged in if there is another shell in the stack, otherwise it logs you out
  - Syntax: exit

Session Related Commands

- SSH
  - Secure SHell remote login
  - Syntax: ssh username@url
  - Demo…
- SFTP
  - It is an interactive Secure File Transfer Protocol to copy files from one computer to another
  - Syntax: sftp username@url
  - Demo…

System Resources

- date [options]
  - report the current date and time
- du [options] [directory or file]
  - report amount of disk space in use
- Hostname   or   uname
  - display or set the name of the current machine
- script file
  - records everything that appears on the screen to file until ctrl-D
- which command
  - reports the path to the command or the shell alias in use

Part 2

Test 1

Part 3

Introduction to C

Readings: chapter 3, https://www.tutorialspoint.com/cprogramming/ or http://www.w3schools.in/c-tutorial/intro/

History of C

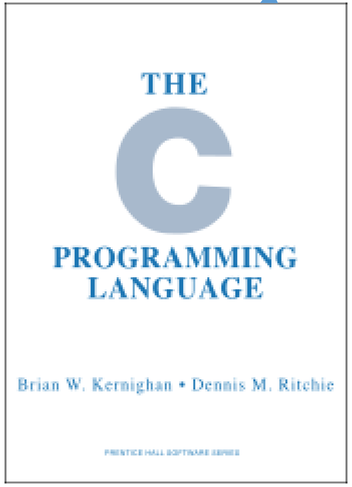
Denis Ritchie
1941 – 2011

Algol	• International Group
BCPL	• Martin Richards
B	• Ken Thomson
Traditional C	• Dennis Ritchie
K&R C	• kernighan & Ritchie
ANSI C	• ANSI Commitee
ANSI/ISO C	• ISO Commitee
C99	• Standerd Commitee

The B language:
- Interpreted C
- Very slow

1972 AT&T Bell Labs

The C language:
- Compiled C
- Created to build the Unix OS



THE C PROGRAMMING LANGUAGE

Brian W. Kernighan • Dennis M. Ritchie

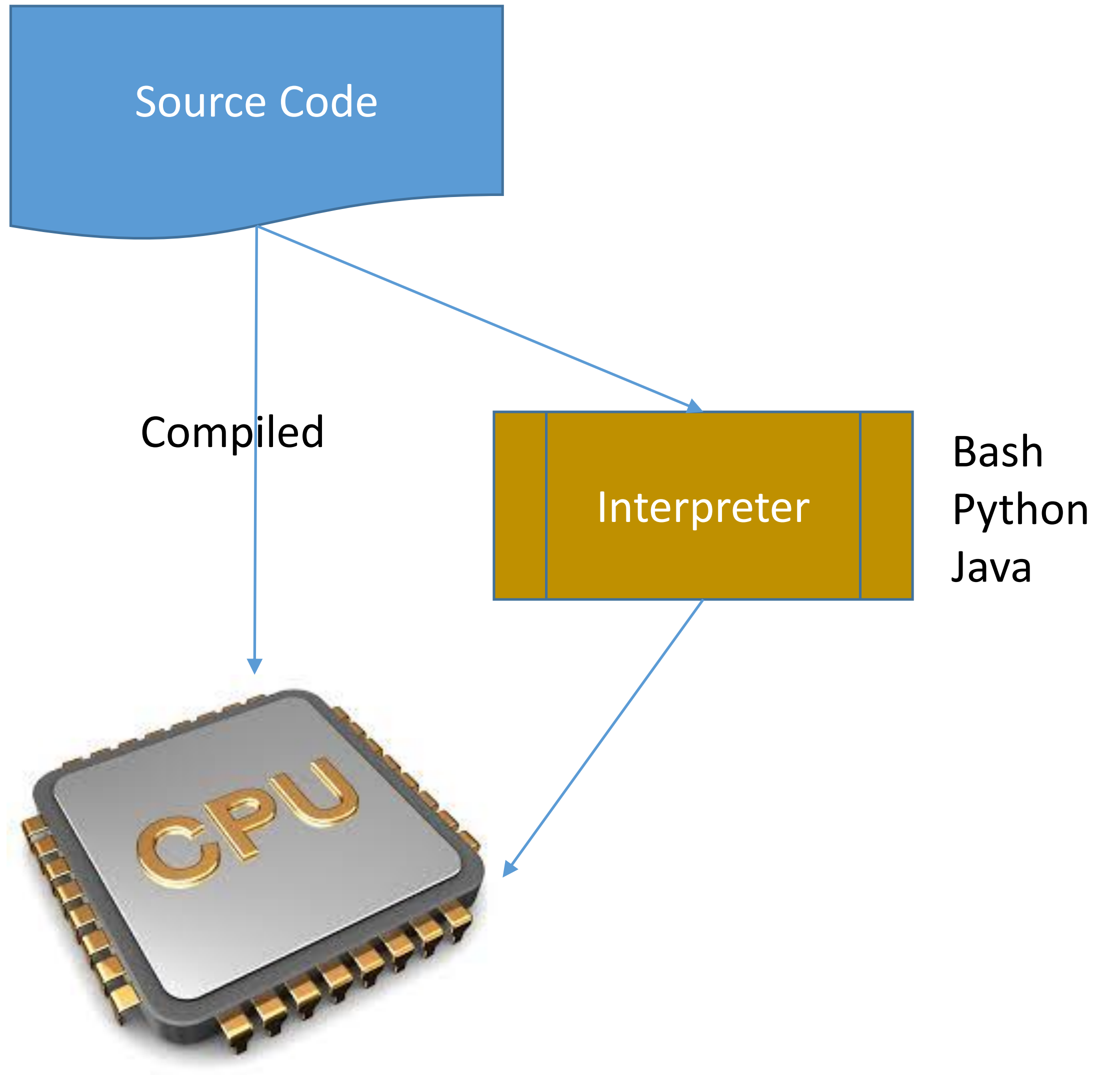
PRENTICE HALL SOFTWARE SERIES

1978

Compilers vs Interpreters

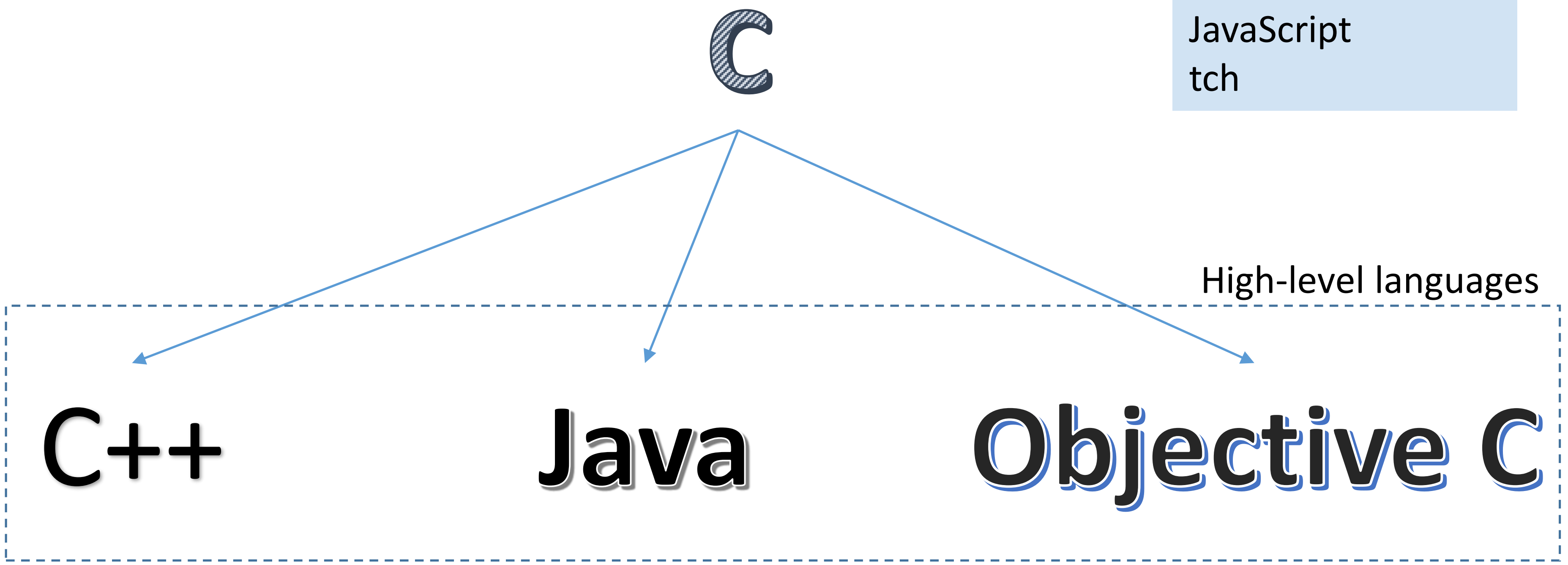
Notice how a compiled program can speak directly with the CPU.

This gives it additional speed and low-level connectivity.



The children of C

JavaScript
tch



Compiled.
Object oriented.
Preserves low-level features.

Highly optimized interpreter.
Object oriented.
No low-level features.
Cross-platform.

1980's improvement of C used by Apple to build their OS X operating system.
Like Small-talk and C++.

Why C?

Because we need an “easy” language that can talk to the hardware and the human.

- Operating systems
- Hardware drivers: printers, mice, etc.
- Specialty machine connectivity: lab machines, robots, VR, etc.

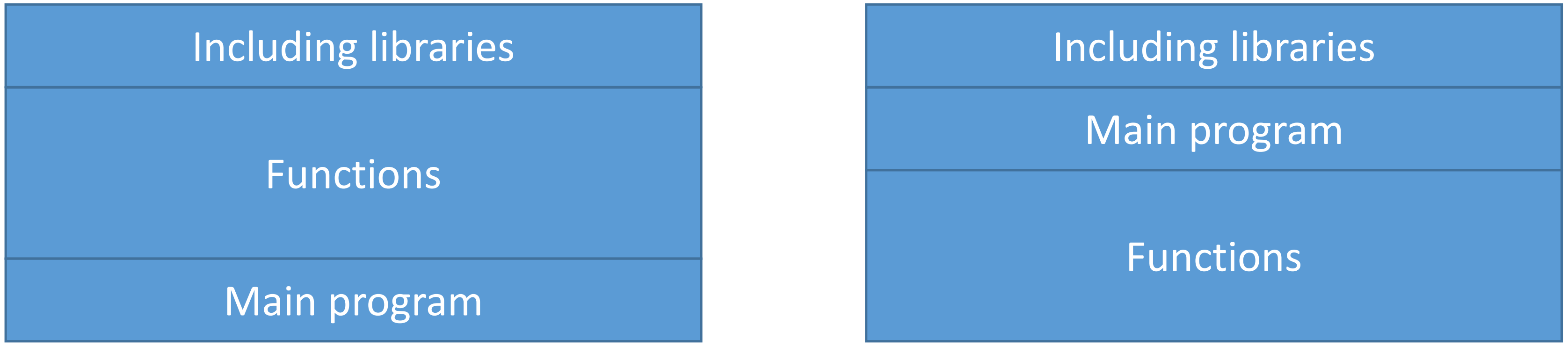
Assembler (COMP 273) is much better but also much harder to write programs.

Basic Structure of a C Program

```
#include <stdio.h>
int main(void)
{
  puts("Hello World\n");

   return 0;
}
```

Library

Main program



Recommended layout

Archaic layout

How to compile and run a C program

```
Bash-prompt $ vi helloworld.c
Bash-prompt $ gcc helloworld.c
Bash-prompt $ ./a.out
```

- We use VI to create out programs
- The GCC compiler is a powerful tool to convert text files into binary machine-code files
- The a.out file is the default binary machine code file name
  - Also known as the Executable file
  - Executable files speak directly with the CPU
- Notice that we execute a.out the same way we executed Bash files, using the ./

Demo

Intel Assembly

```
main:
pushl %ebp
movl %esp, %ebp
subl $8, %esp
andl $-16, %esp
movl $0, %eax
subl %eax, %esp
subl $12, %esp
pushl $.LC0
call puts
addl $16, %esp
movl $0, %eax
leave
ret
```

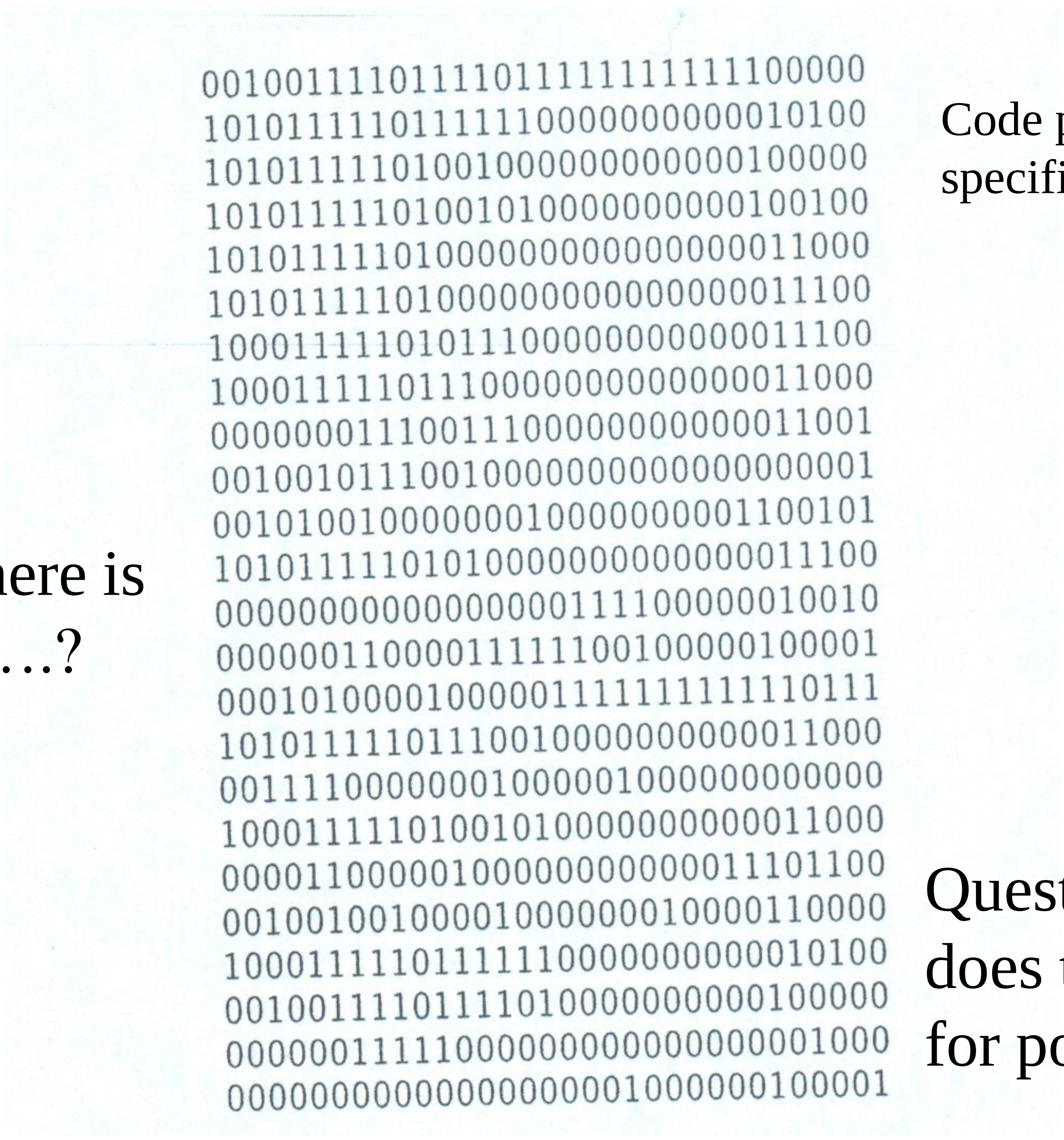
```
#include <stdio.h>
int main(void)
{
   puts("Hello World\n");
    return 0;
}
```

Library call

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Machine Code



```
00100111101111011111111111100000
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
10101111101000000000000000011000
10101111101000000000000000011100
10001111101011100000000000011100
10001111101110000000000000011000
00000001110011100000000000011001
00100101110010000000000000000001
00101001000000010000000001100101
10101111101010000000000000011100
00000000000000000111100000010010
00000011000011111100100000100001
00010100001000001111111111110111
10101111101110010000000000011000
00111100000001000001000000000000
10001111101001010000000000011000
00001100000100000000000011101100
00100100100001000000010000110000
10001111101111110000000000010100
00100111101111010000000000100000
00000011111000000000000000001000
00000000000000000001000000100001
```

Code pattern is specific to CPU

Hmm, where is my error….?

Question: what does this mean for portability?

Basic Structure of a C Program

```
#include <stdio.h>

int main(void)

{

   char c;

    puts("Gender: ");
    c = getc(stdin);

    if (c == 'F' || c == 'f')
     puts("Welcome\n");
    else
     puts("Sorry, try again.\n");

    return 0;

}
```

STDIO.H is the standard input/output library.
- Function puts() writes strings.
- Function getc() reads a character.

Declaring and using variables.

Returning error codes like Bash.

puts

Library: stdio.h

Syntax: int puts(constant_string);

Returns: Error code

- >= 0 if no error

Purpose: To print a string to standard out

Usage:

```
puts("Hello World");     // without new line
puts("Hello World\n");     // with new line
```

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Escape Characters

\n	-	New line
\r	-	Carriage return
\t	-	Tab
\\	-	Backslash
\a	-	Bell
\b	-	Backspace (without delete)

Others…

getc

Library: stdio.h

Syntax: int getc(stdin);

Returns: ASCII code

Purpose: To read a character from standard in

Usage:

```
c = getc(stdin);
```

Notice that this functions does not actually return the character but the ASCII code for that character as an integer number.

This is a low-level feature.

gcc

GNU C Compiler

- gcc SWITCHES FILES

Switches

- Without a switch the default activity is to merge all the FILES into a single a.out executable file.
- -o Replace the default a.out file name with your own
  - gcc –o hello helloworld.c

```
Bash-prompt $ gcc –o hello helloworld.c
Bash-prompt $ ./hello
```

GCC and Errors

Errors are displayed to the screen and can be lost as the screen scrolls.

Solution: gcc helloworld.c > textfilename

All output from gcc will be stored in the textfilename file. You can then use vi, more, or cat to view the contents.

Demo

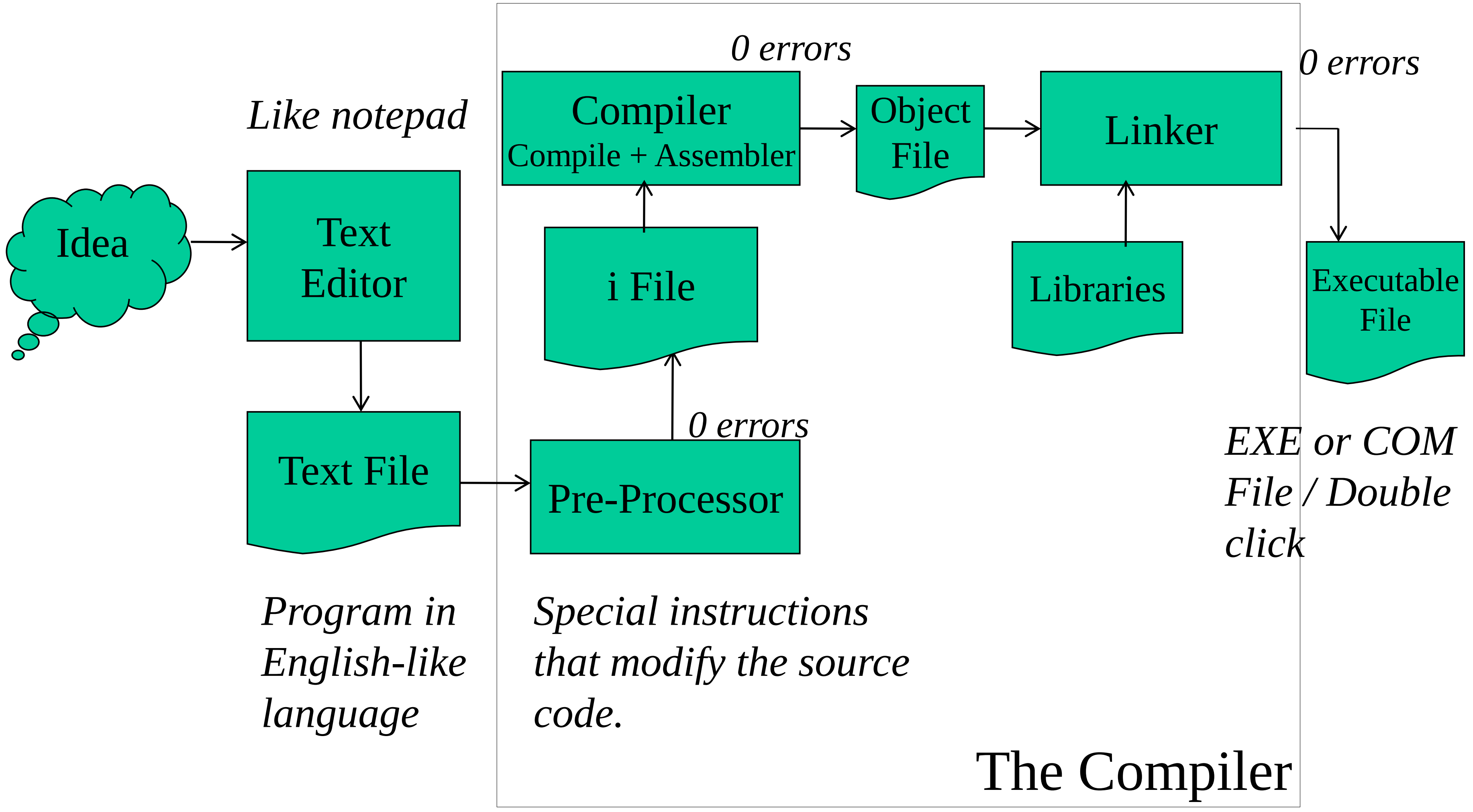
How compiler errors work

A compiler attempts to convert your source code to machine code.

When it finds an error it marks it as an error BUT then makes an assumption and continues compiling.

All other errors are based on the assumption. Trust only the first couple of errors.

The C Compilation Process



Note: The compiler does not compile the Text File you entered but the i File, it has been changed by the Pre-Processor.

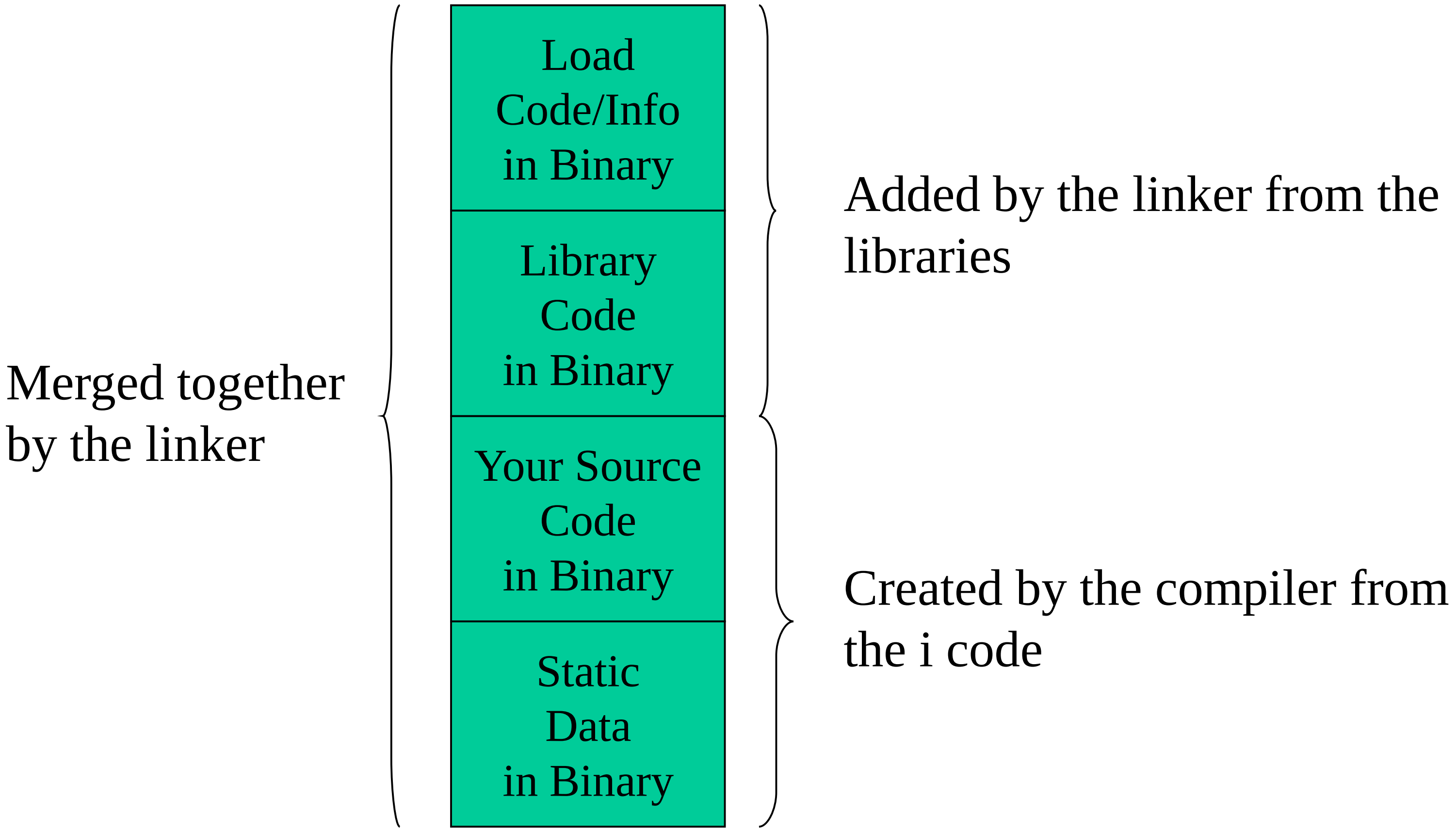
C Files

- Source Files
  - FILENAME.c .... the program
  - FILENAME.h .... header file (shared code)
- Pre-processed File
  - FILENAME.i
- Object Files and Assembler Files
  - FILENAME.o
  - FILENAME.s
- Executable Files
  - FILENAME .... Using the –o switch
  - a.out .... the default executable name

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

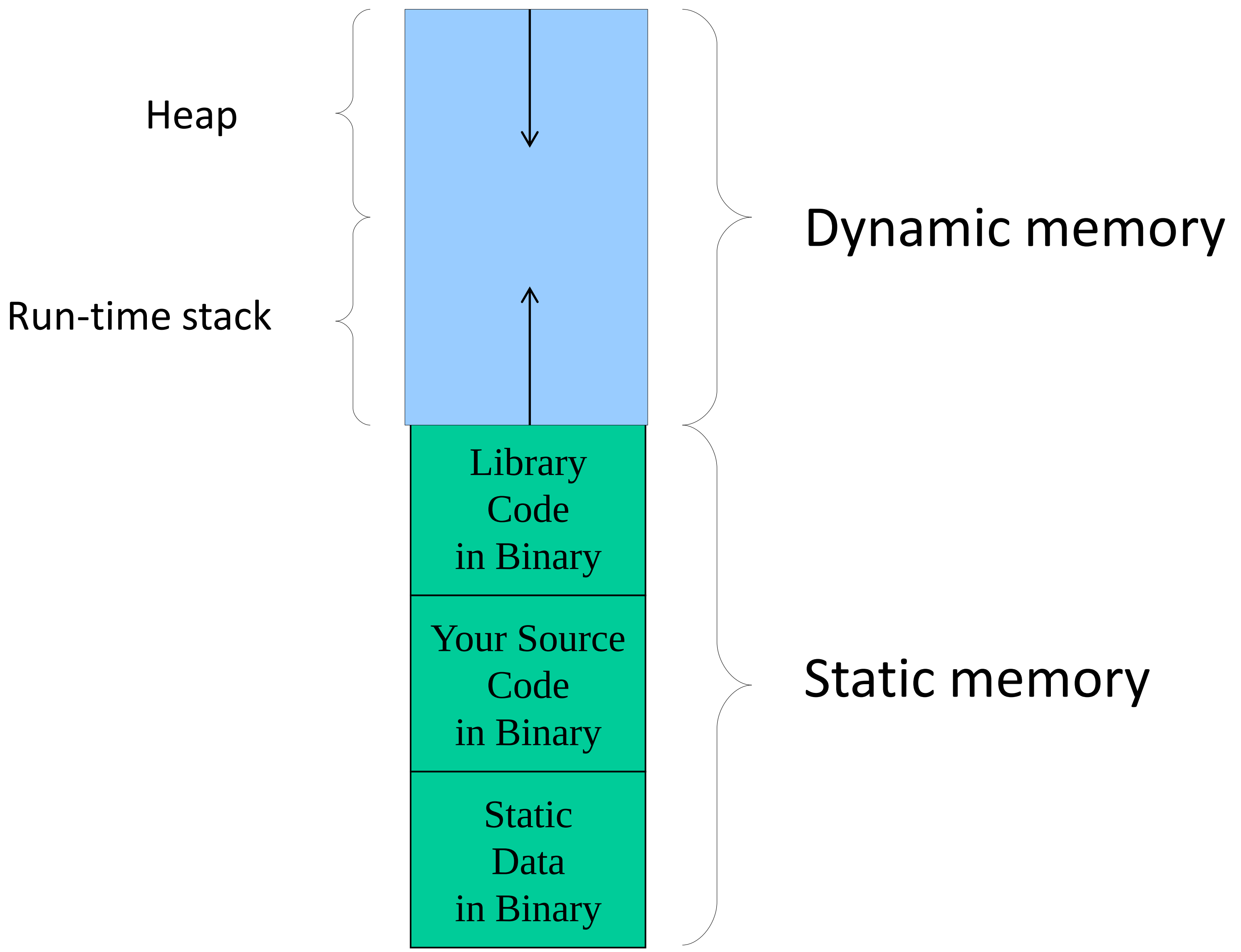
Structure of a Compiled File


Load Code/Info in Binary
Library Code in Binary
Your Source Code in Binary
Static Data in Binary
Merged together by the linker
Added by the linker from the libraries
Created by the compiler from the i code

Unix
Bash
C
GNU
Systems

COMP 206 – Joseph Vybihal
Software Systems

Structure of a Process



The pre-processor

More on this later, but…

```
#include <stdio.h>
```

Is a pre-processor command.