Comp 310
Computer Systems and Organization

Lecture #4

Building A Command Interpreter

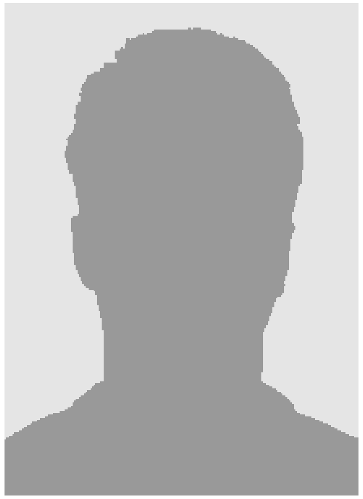
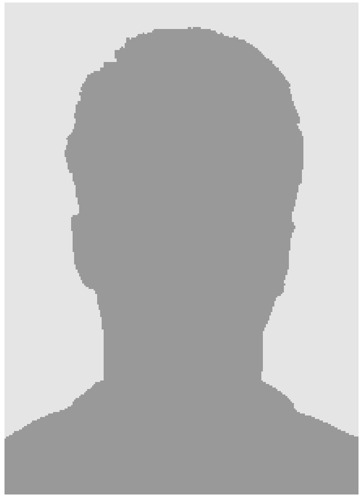
(The OS user interface)

Prof. Joseph Vybihal

Announcements

- Assignment #1 on Web CT Monday
- Tutorials:
  - Unix, scripts and editting C programs: TBA
  - Advanced: offered by SOCS (check web site)

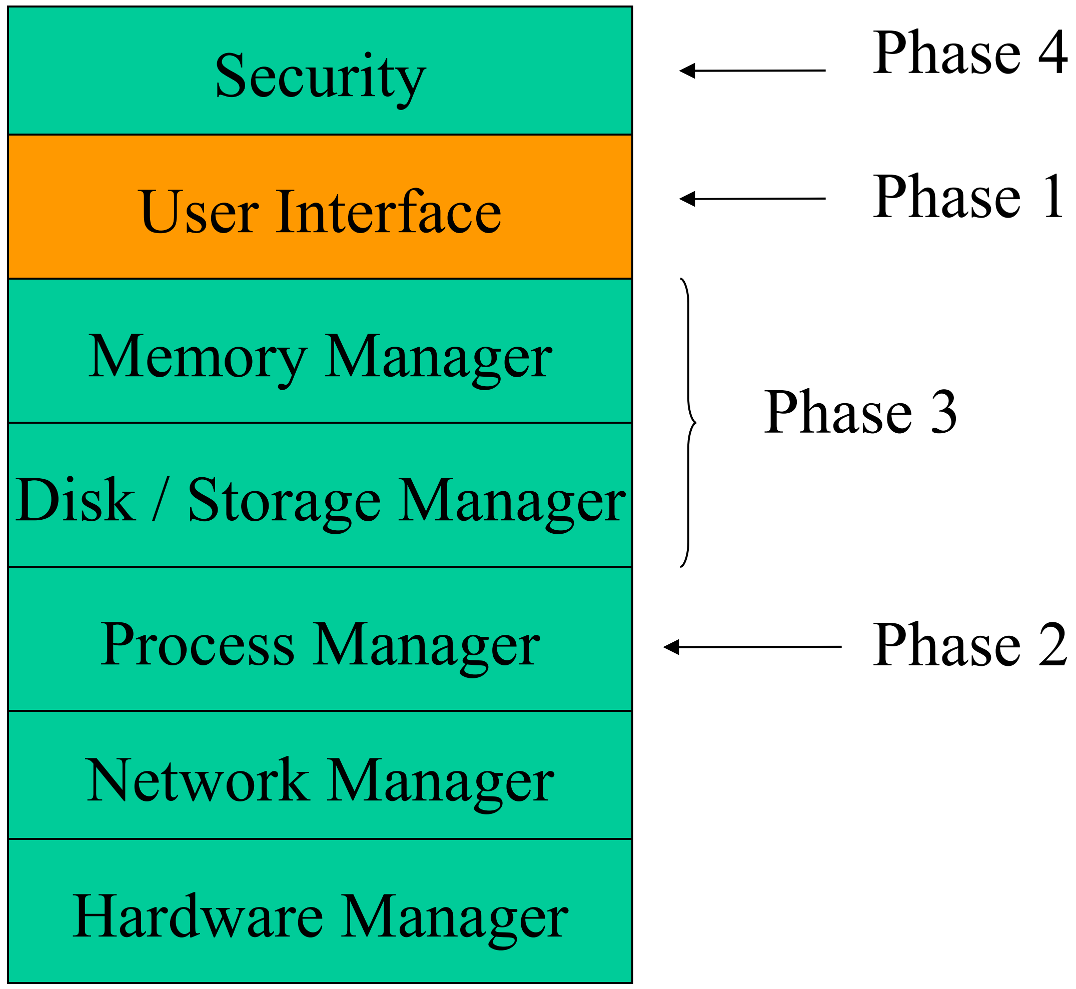
- TA Info:

Theresa Deerng
Office: Web TA
Unix Lab Tutor

Basic OS Architecture

(Course Table of Contents)



Part 1

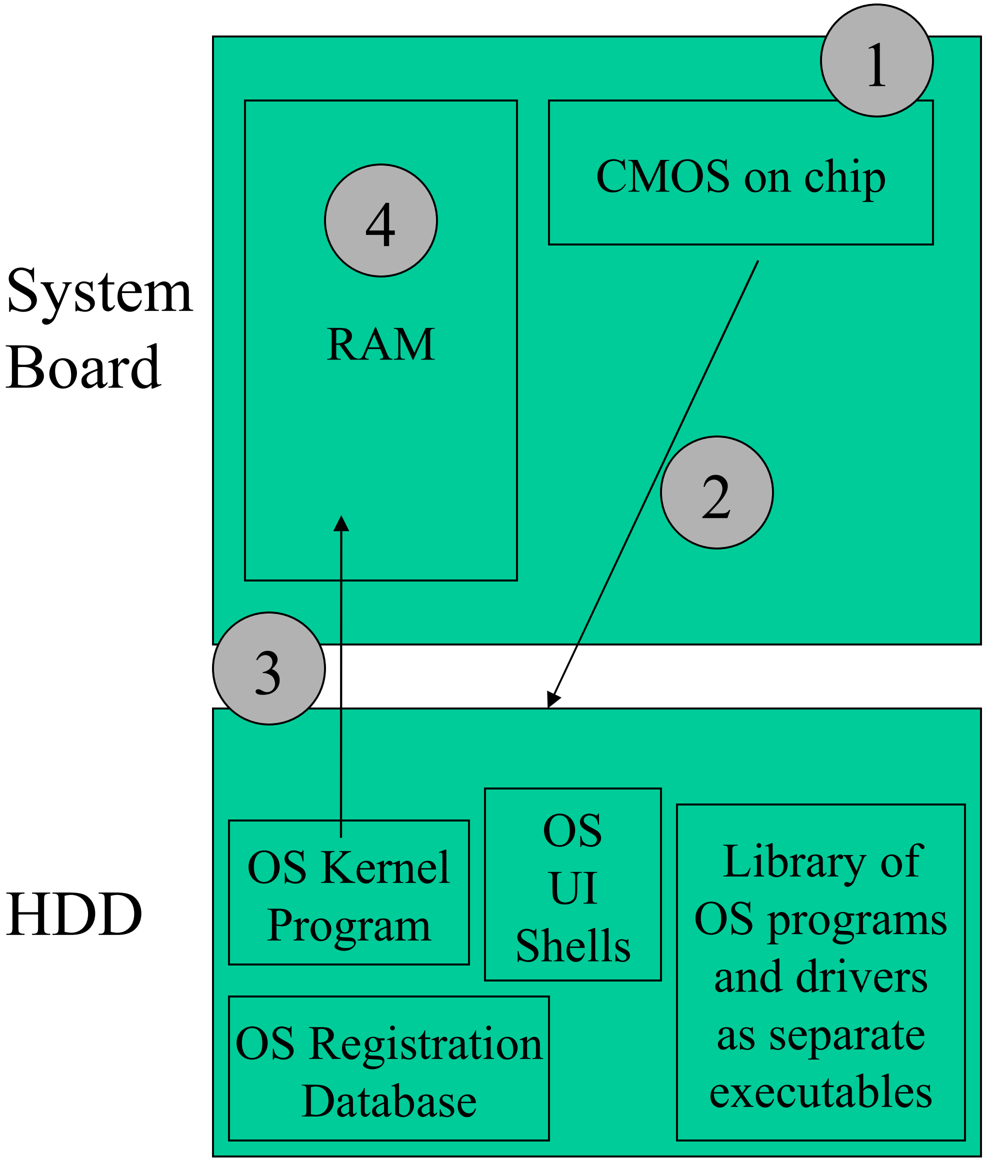
OS Overview

What kind of UI?

- Command-line
- GUI
- Software only
  - Job control language
  - Bash programming

Command-line and GUI can encapsulate Software Only

OS Layout

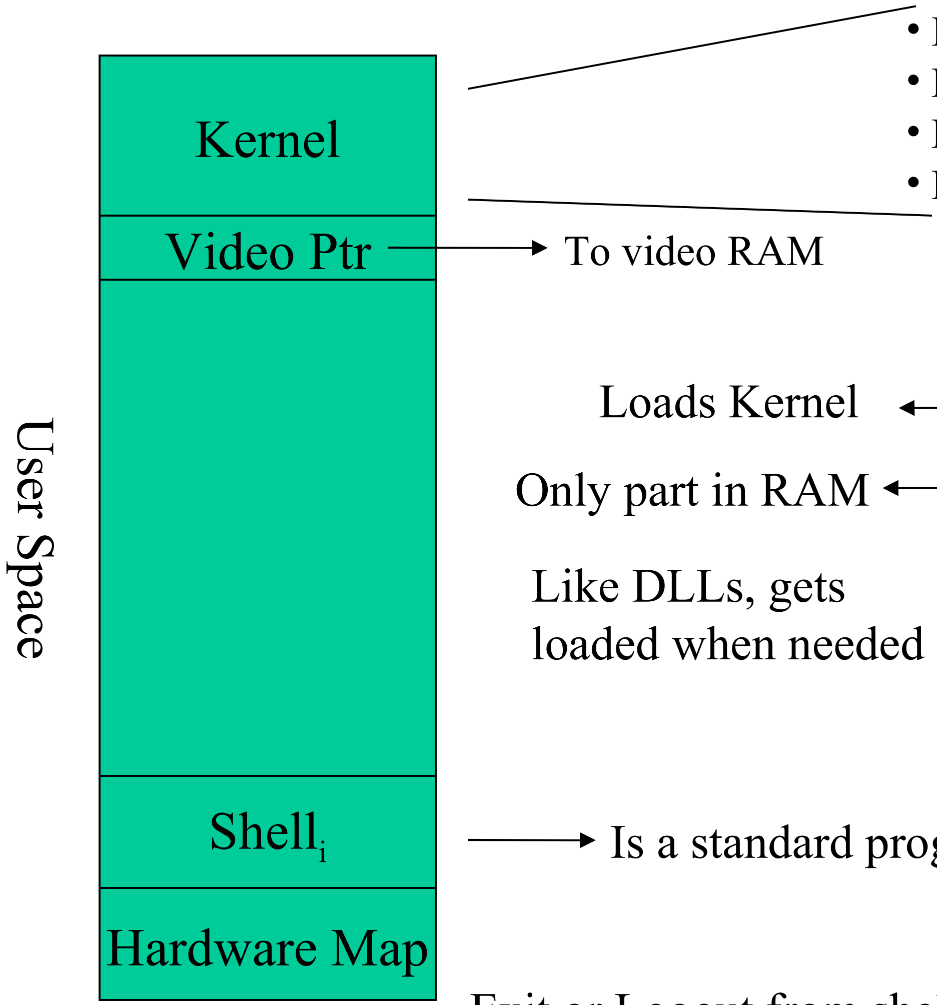


1	CMOS hardware check
2	Launch program at address zero
3	Kernel loaded into RAM
4	RAM is formatted by kernel
5	Kernel's UI function called. (5) Is the default LOGIN shell.

The Boot-Up Sequence

1. Turn on PC
- CMOS checks hardware (RAM, Ports, Video)
  If damage then stop PC with error message
- CMOS goes to "0000" address of HDD loads software (hopefully it is OS, maybe Virus)
4. OS Loader puts only OS "Kernel" at top of RAM and gives CPU to Kernel (designates "privileged" execution status)
5. Kernel deletes loader and formats RAM
6. Kernel checks OS software components & verifies installed hardware drivers (optional)
- If error displays error menu on screen (blue/black)
  If no errors displays login screen

The OS Kernel & Shell Relationship



- Login/out fns → launch default shell
- Process and Memory Managers (in assem)
- Partial support: I/O and Network
- Remainder OS on HDD as DLL or EXE

<u>The OS Structure</u>:

- Loader ← Loads Kernel
- Kernel ← Only part in RAM
- Drivers (libraries on disk)
- Mini programs on disk

Like DLLs, gets loaded when needed

Most of OS here!

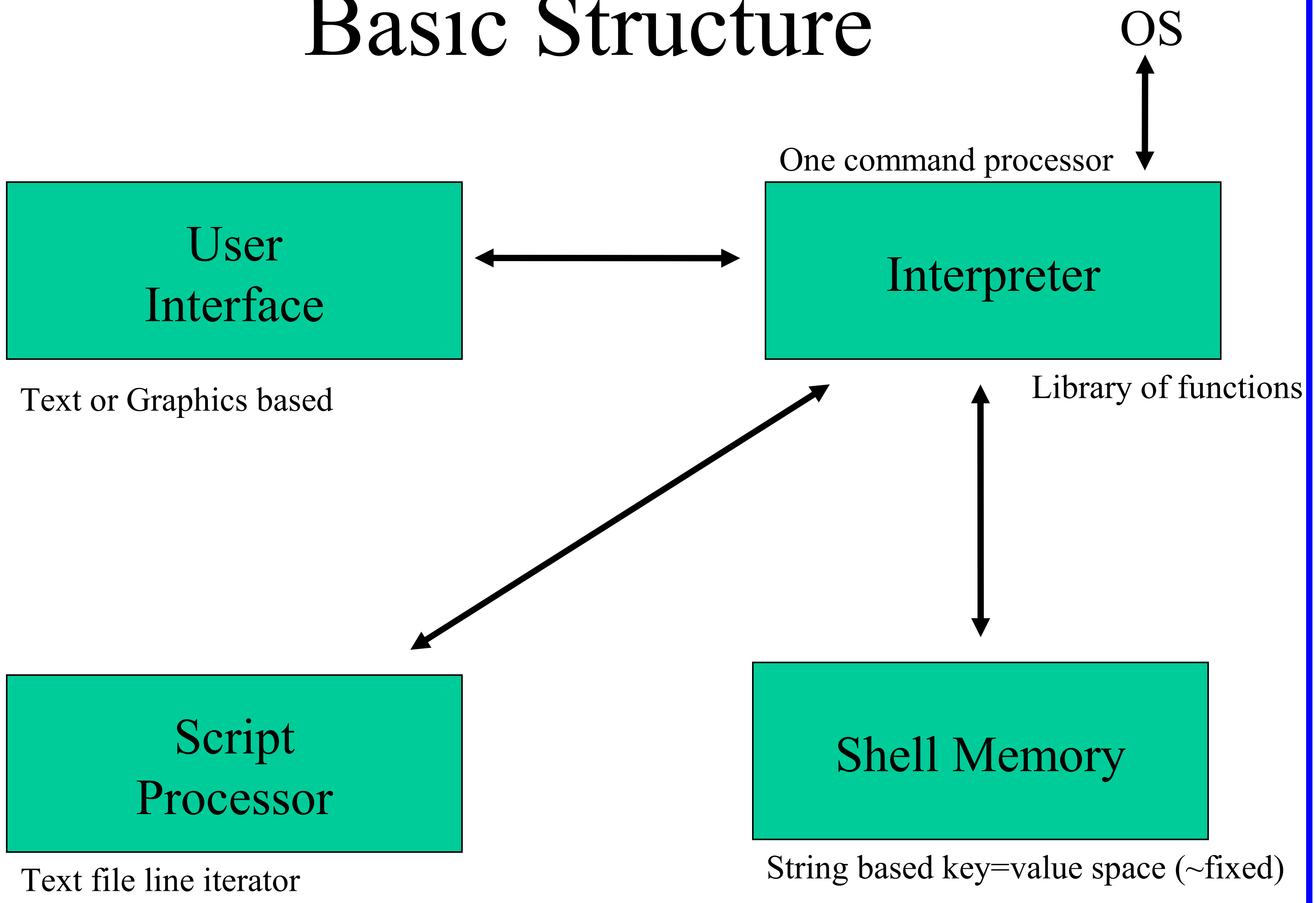
Shell$_i$ → Is a standard program launched by the kernel (sh.exe)

Exit or Logout from shell? What happens?
→Last exit? No more processes on list → auto logout

Part 2

The OS Shell

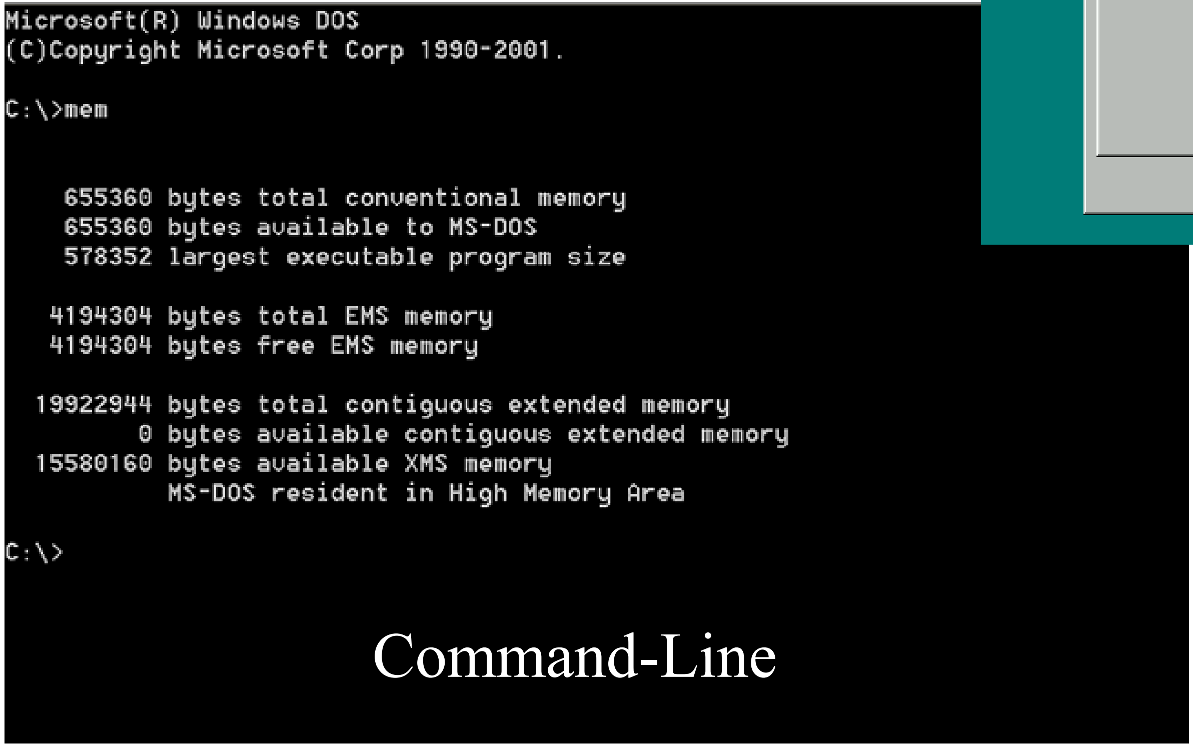
Basic Structure



Shells

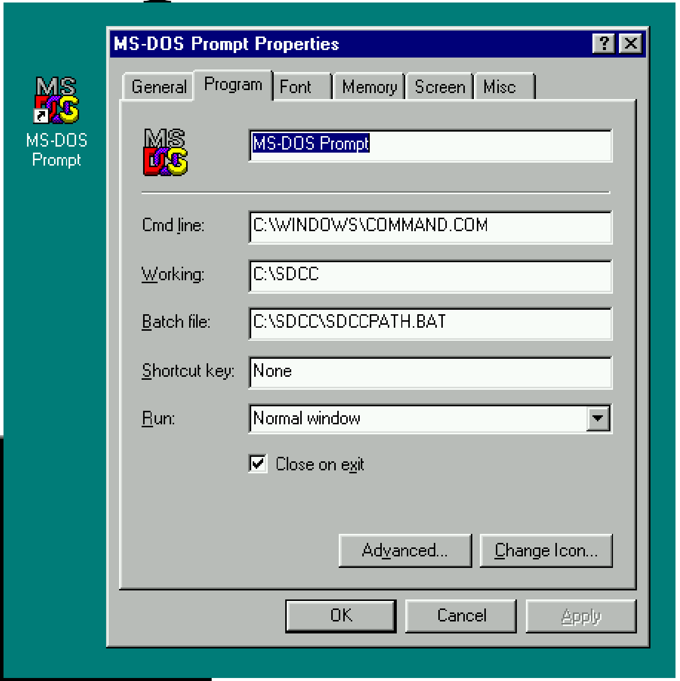
- Features:
  - Command-line Interface
    - Expression syntax
    - Provides access to programs on disk
      - User and OS
    - Provides access to public kernel functions
  - Generates errors with unknown commands
  - Scripting Interface
    - Expression syntax
    - Executable on shell’s command-line or from a script
  - Shell “global” memory sharable under multi-processing
  - Provides for shell switch at launch-time
  - GUI Interface

The Command Interpreters



```
Microsoft(R) Windows DOS
(C)Copyright Microsoft Corp 1990-2001.

C:\>mem


    655360 bytes total conventional memory
    655360 bytes available to MS-DOS
    578352 largest executable program size

   4194304 bytes total EMS memory
   4194304 bytes free EMS memory

  19922944 bytes total contiguous extended memory
         0 bytes available contiguous extended memory
  15580160 bytes available XMS memory
           MS-DOS resident in High Memory Area

C:\>
```

Command-Line



MS-DOS Prompt

Cmd line: C:\WINDOWS\COMMAND.COM

Working: C:\SDCC

Batch file: C:\SDCC\SDCCPATH.BAT

Shortcut key: None

Run: Normal window

[✓] Close on exit

Window

Part 3

Programming the OS Shell

It is just a program!

Sh.exe or win.exe

Parameter passing from the OS or from a launch request from a previous shell

```
int main (int argc, char* argv[]) …
```

In Unix this is programmed in C!

Returned error code:
0 → no error
N → an error code

Used to send switches to the shell
E.G.
-H help info
-V verbose mode

At the prompt:

```
$ sh -v
```

The Command Interpreter is a Program

```
int main(int argc, char* argv[])
```

Standard execution sequence:

•Process parameter switches

•Execute standard initialization scripts (E.G. .cshrc)

•Command Line Processor

- Possible branch to Script Interpreter
- Shell Memory Manager

•Execute standard logout scripts (E.G. .logout)

The Command Line Processor

```
while (strcmp("exit",userinput)!=0)
{
   printf("%s",prompt);
   gets(userinput);

   token = tokenize(userinput);
   token2= tokenize(userinput);

   if (strcmp(token,"ls") directory(token2);
   else if (strcmp(token, "man") system("man"+token2);
   else {
      errorcode = system(userinput);
      if (errorcode > 0)
         printf("some error message");
   }
}
```

Some are calls to the Kernel

Some are calls to internal shell functions

Parsing Strings

- Is tokenizing string optimal?
  - How do we code a tokenizer...
  - Are there other ways...

Command-line Switches

```
if (argc > 0) // parameters exist
{
  for (I=0; I<argc; I++)
  {
     if (strcmp(argv[I],"-H") help();
     else if (strncmp(argv[I],"-V") Verbose();
     :
     else
     {
        printf("%s undefined",argv[I]);
        errorcode++;
     }
  }
}
```

Graceful, if it does not crash…

The Script Interpreter

- Similar to the command line processor, but:
  - ➢ No prompt, instead a pointer is set to the script file
  - ➢ Command are fgets'd from the script file and tokenized
  - ➢ Then processed in the same way as the command-line processor
  - ➢ Interpreter ends at EOF or when EXIT or LOGOUT

How can we program this is C?

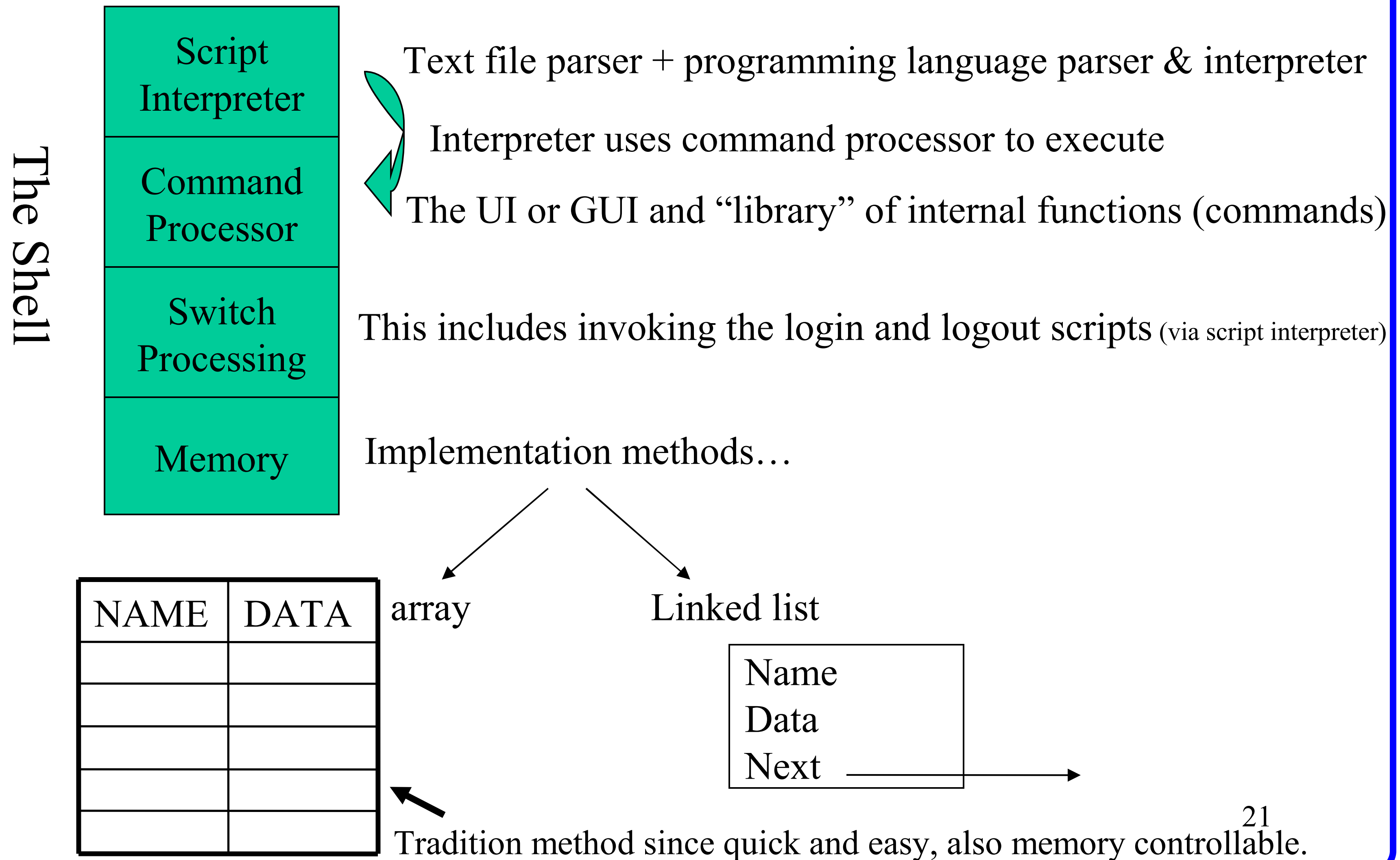
Note: In Unix, all the programming commands can also be input directly through the command-line prompt without the need of a script interpreter.

Obviously there is some reuse of code.

Question

- Can we rework the command-line program so that its switches can accept a command-line command?
  - How would that look like for the user?
  - How would we program this?
  - Are there advantages to this?

The Shell's Memory



Text file parser + programming language parser & interpreter

Interpreter uses command processor to execute

The UI or GUI and "library" of internal functions (commands)

This includes invoking the login and logout scripts (via script interpreter)

Implementation methods…

array

NAME	DATA

Linked list

Tradition method since quick and easy, also memory controllable.

Question

- How do we write code to implement an array form of memory? What would the command-line command look like?

Part 4

GUI Architecture

Windowed Architectures

- Interrupt processing
  - What is an interrupt?
  - The interrupt table.
- Message passing architecture
  - What is a message?
  - The message queue
  - The messaging/process loop
  - How does that fit with interrupts?

Part 5

At Home

Things to try out

- Learn some of the command-line commands on your personal computer:
  - Create a folder
  - Create a text file in that folder
  - List the contents of the folder
  - Delete that file
  - List the contents of the folder again
- Try this out on Unix/Linux