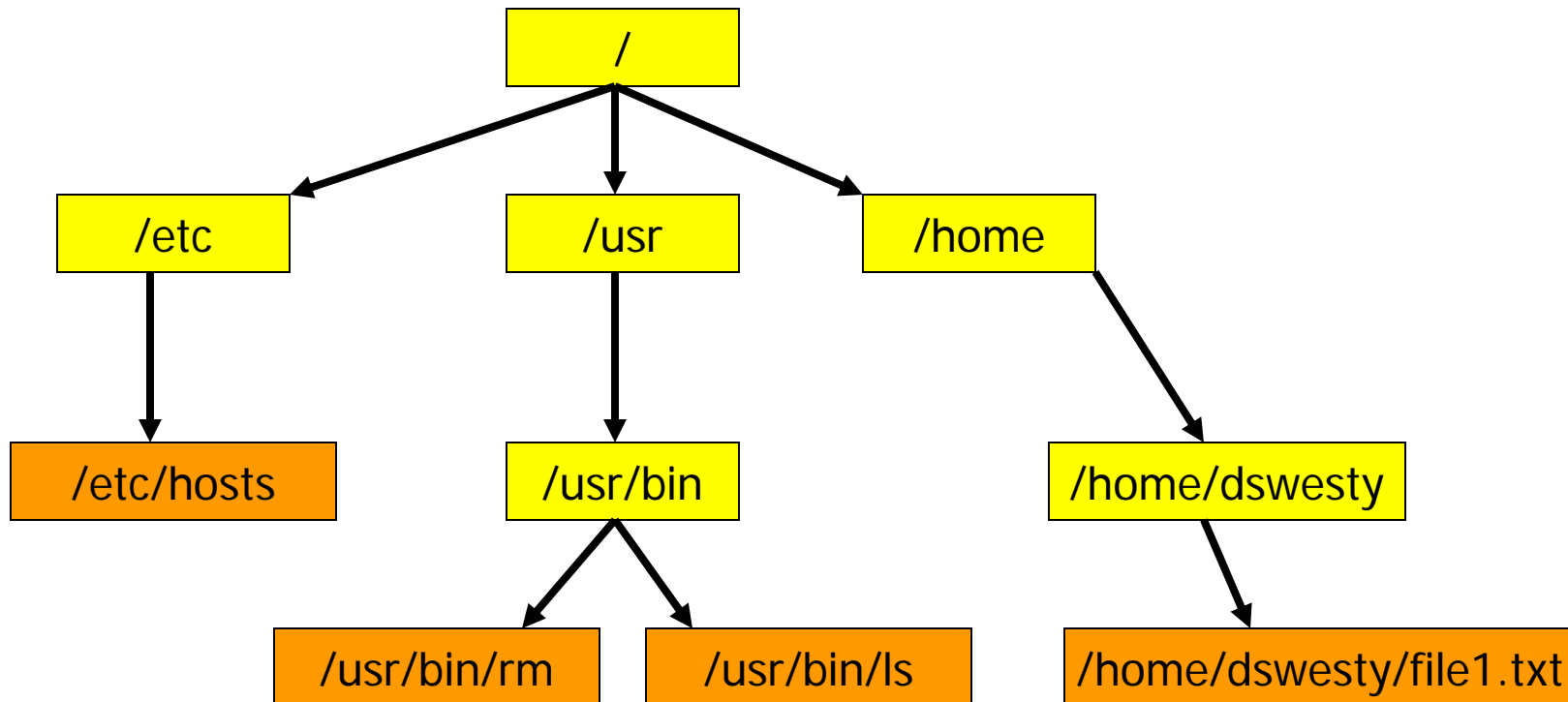


# Goals for This Lecture:

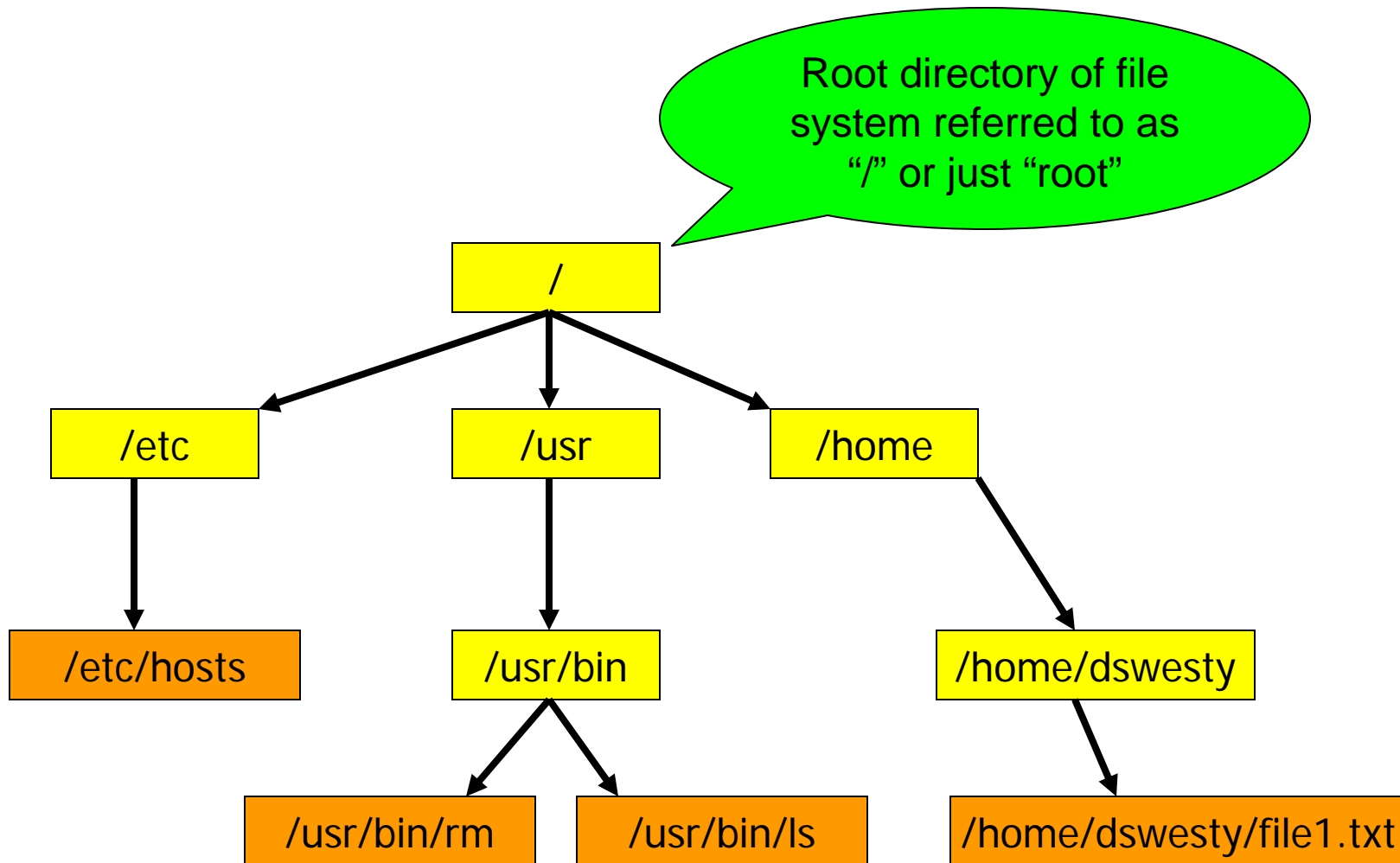
- Understand the Un\*x file system
  - Learn how to navigate around the disk
- Understand what a compiled computer language is
  - A brief history of computer languages
  - What is a compiler?
- Learn how to invoke the compiler
- Write your first program
  - Create a program that says “Hello World!”

# The Un\*x Filesystem

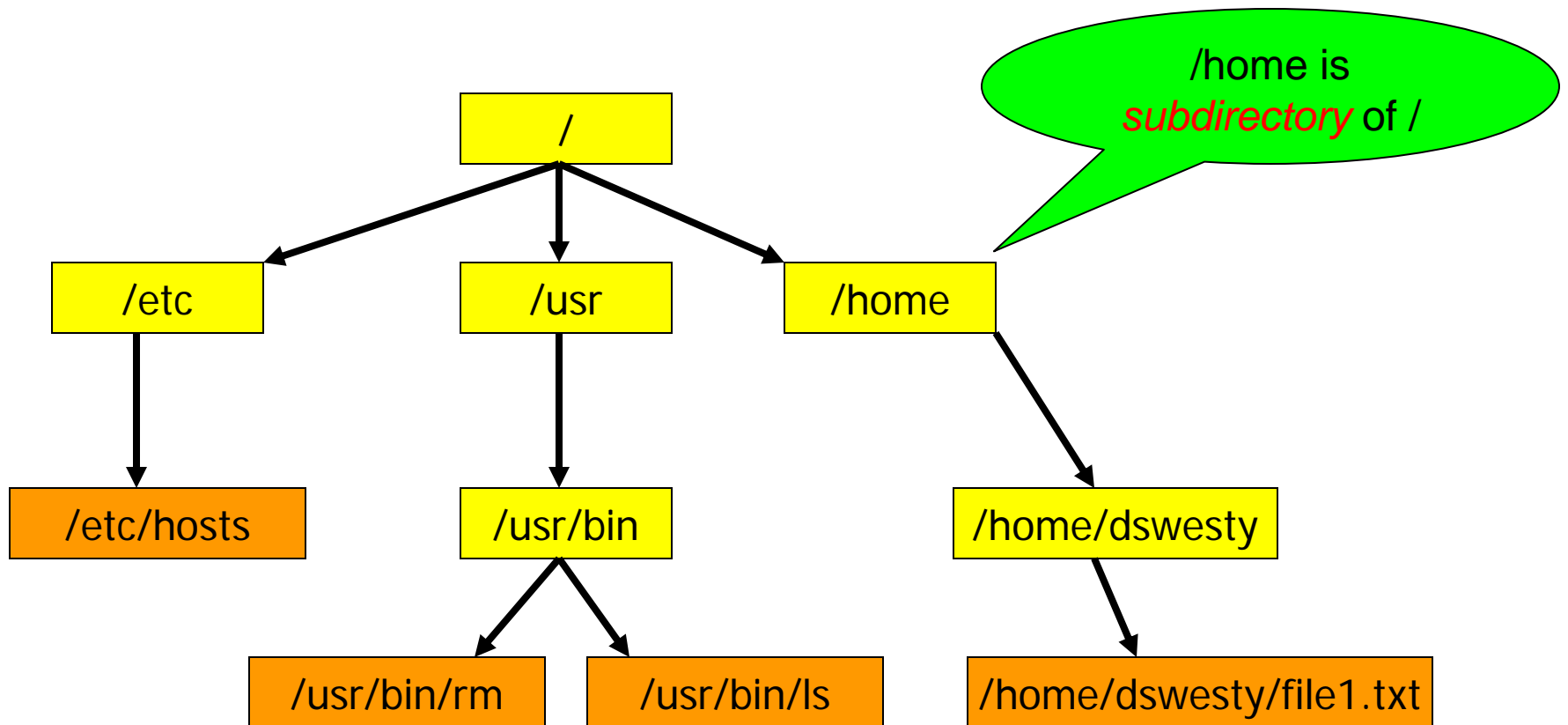
- A disk (or part of a disk called a *partition*) is organized into *directories*
  - Directories are called folders on MS windows systems
- Directories have a hierarchical organizational structure called a *tree*



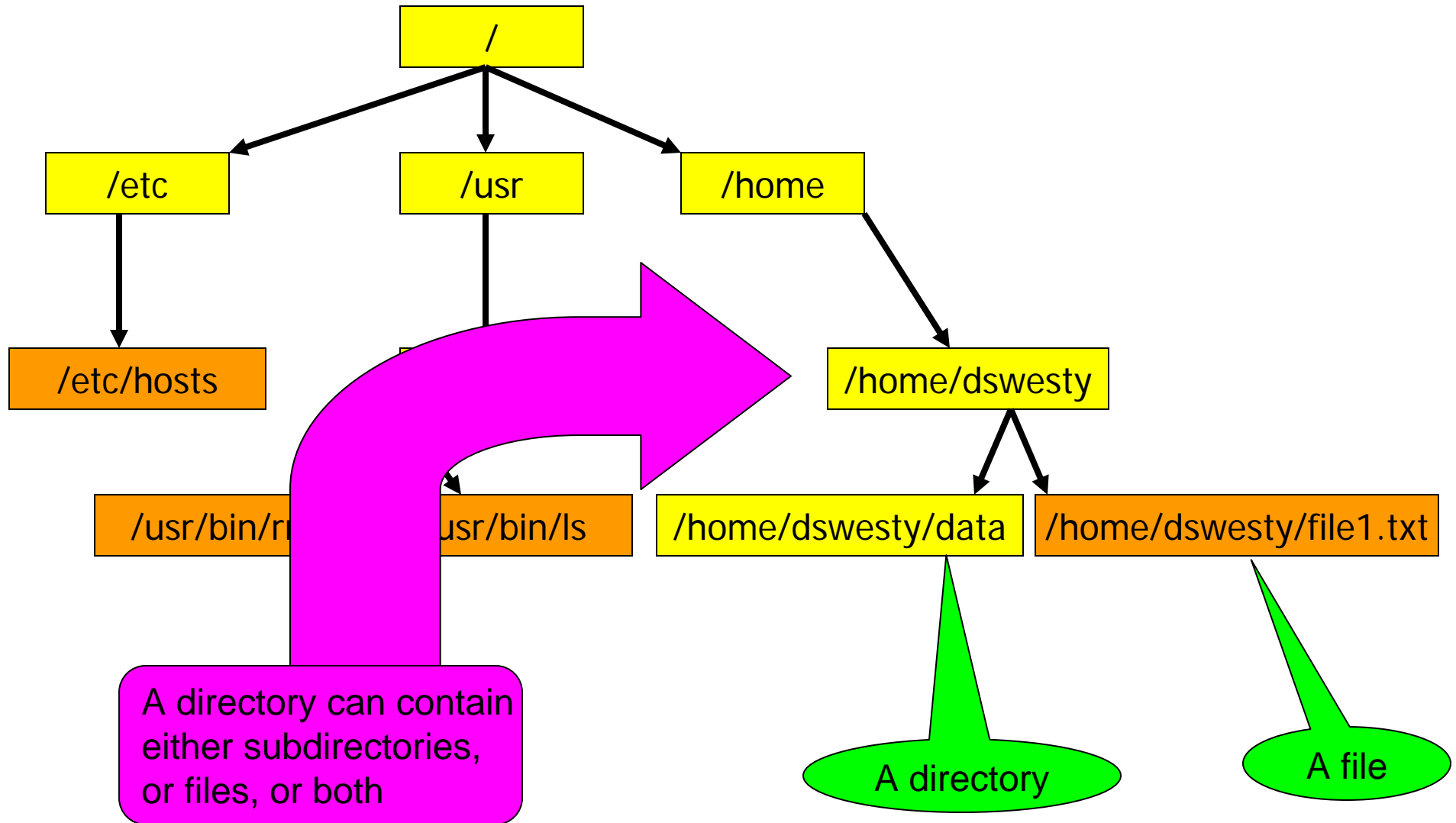
# The Un\*x Filesystem



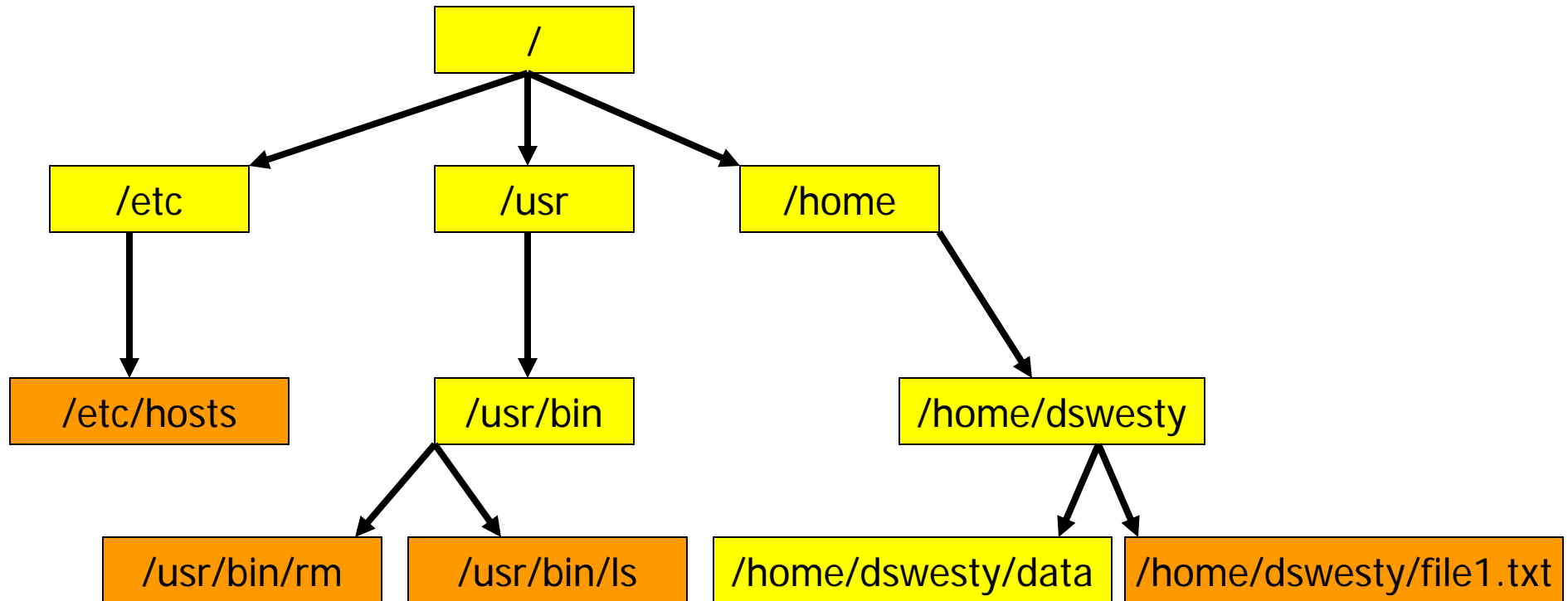
# The Un\*x Filesystem



# The Un\*x Filesystem



# The Un\*x Filesystem

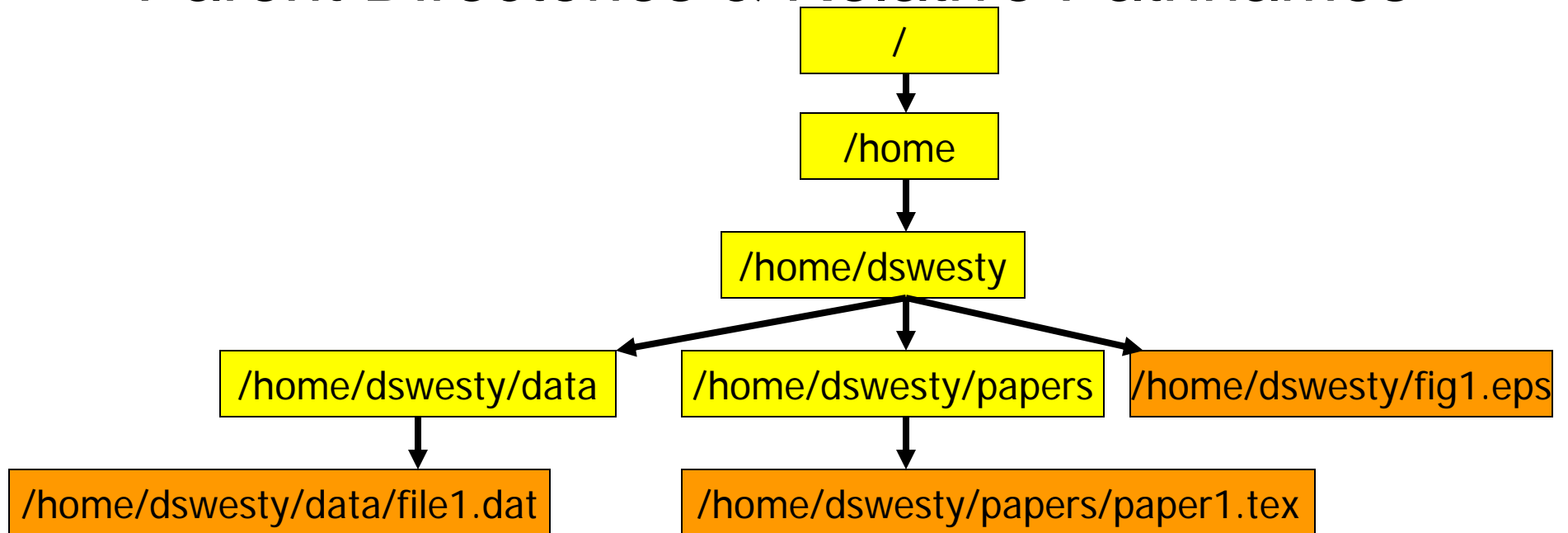


- The location of a file on the disk can be specified by its *absolute pathname* which is a complete listing of its position in the directory tree
- E.g., `/home/dswesty/file1.txt` or `/usr/bin/lis`
- The absolute path to the file named “file1.txt” is `/home/dswesty`
- Slashes, i.e. “/” are used to separate directory names in the path
- The root directory itself is referred to as “/”
- Don’t whitespaces in file or directory names in order to avoid confusing yourself or the shell

# Current Working Directory

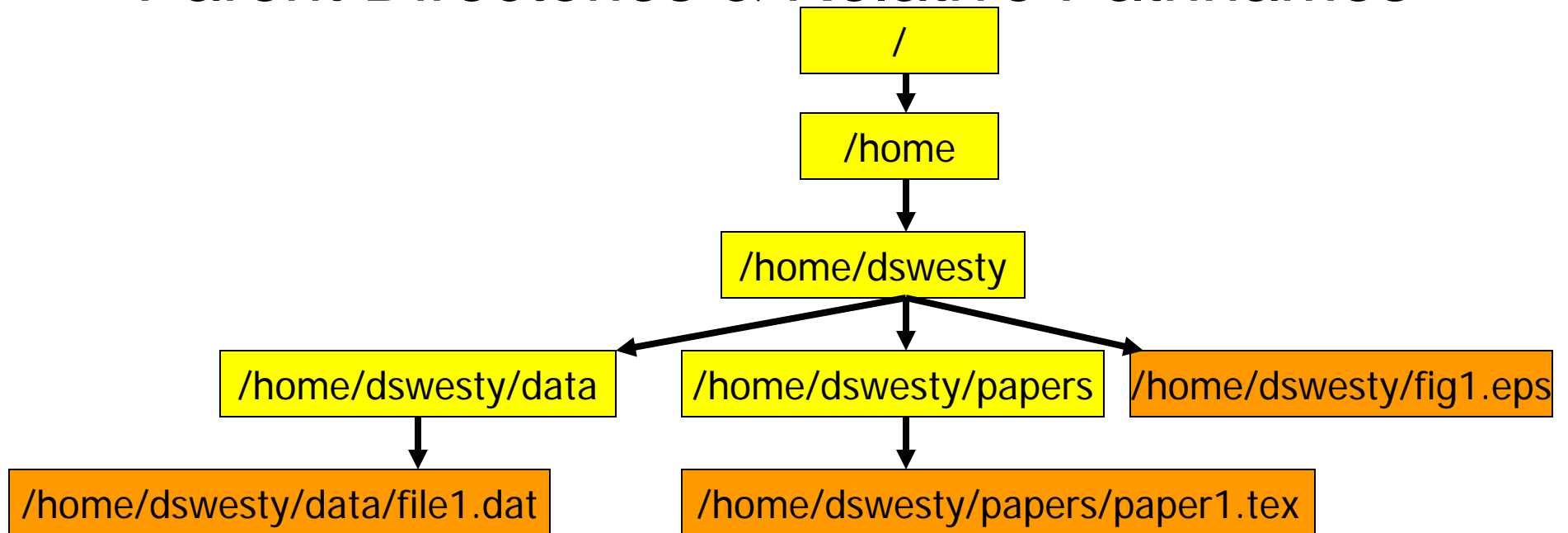
- Absolute pathnames of files can be long & cumbersome
  - Fortunately there is often an easier way of specifying the location of a file!
- Un\*x shells support the notion of a “current working directory”
- When you first log in your *current working directory* is set to your *home directory*
- If I execute the pwd command (which lists the current or “present” working directory) I find:
  - > pwd
  - /home/dswesty*
- I can also specify the location of a file by its path relative to my current working directory.
- For example, if my current working directory is set to */home/dswesty* I can refer to the file */home/dswesty/file1.txt* as just *file1.txt*

# Parent Directories & Relative Pathnames



- From my current working directory `/home/dswesty` I could refer to `/home/dswesty/data/file1.dat` as `data/file1.dat`
- This is called the *relative pathname* of the file
- Suppose I change my current working directory to `/home/dswesty/papers` by executing the command  
    `> cd papers`
- Then I can refer to the file `/home/dswesty/paper1.txt` as just `paper1.txt`
- But how could I refer to `/home/dswesty/fig1.eps` without specifying absolute path to the file?
- I can refer to the directory above my current working directory, i.e. the *parent directory* of my current working directory by the notation `..` (two periods)
- Thus I could refer to `/home/dswesty/fig1.eps` as `../fig1.eps`
- Similarly I could refer to `/home/dswesty/data/file1.dat` by the relative pathname `../data/file1.dat`

# Parent Directories & Relative Pathnames



- If the parent directory of the current working directory is known as “..” how can I refer to the current working directory itself?
- Another shortcut: the current working directory can be referred to as “.” (single period)
- This is an important shorthand notation that is often used as a command argument.
- For example, suppose my current working directory is /home/dswesty/papers
- If I execute the command
  - > `cp ../data/file1.dat /home/dswesty/papers`
- I will copy the file file1.dat from the /home/dswesty/data directory to my current working directory
- But the specification of the absolute path as the second argument is awkward
- Alternatively:
  - > `cp ../data/file1.dat .`
- does the trick!

# Summary of Points Regarding the Un\*x Filesystem and File Manipulation Commands

- Become fluent at this as we will be using it all semester
- There are at least two ways of specifying the path to a file or directory and you should use the one that is easier
- The more familiar that you get with the techniques for manipulating the file system the less time you will waste trying to figure out how to do rudimentary tasks
- Practice, practice, practice!
- More tricks in the next lecture!

# A (Very) Brief History of Computer Programming

- First digital computer invented in 1937 by physicists at Iowa State (Atanasoff & Berry)
  - Solved linear systems of equations but not programmable
- First programmable digital computer (ENIAC, Electronic Numerical Integrator and Calculator) invented by Mauchly & Eckart at Univ. of Pennsylvania (1946)
  - Programming consisted of plugging wires into various components to configure the machine for a specific calculation
- A breakthrough by physicist John von Neumann was the realization that the computer could store the program electronically, with electronically held instructions controlling the operation of the computers circuitry.
  - Lead the way to today's programmable computers.
- The Mauchly/Eckart/von Neumann design resulted in the development of EDVAC (Electronic Discrete Variable Automatic Computer) in 1947

# Programming – The early days

- Early programmable machines were programmed by means of explicitly entering the instructions, called operation codes or op codes for short, and the data into the machine in some fashion.
- Computers are dumb and need to be told how to do almost everything.
- To add two numbers held in memory you might have to:
  - Move the data from memory location 16153 to register 1 in the ALU
  - Move the data from memory location 17224 to register 2 in the ALU
  - Have the ALU to add register 1 to register 2 and store the result in register 3
  - Move the data in register 3 to location 21553 in memory
- This was all done by entering data and instructions into the computer by hand
- These operations can be encoded in an *assembly language* where the numerical values for op codes are replaced by code words which are then translated by the assembler program to numerical values
- Somewhat easier but extremely tedious for everything but the most simplistic problems.

# FORTRAN – The first high level programming language

- Between 1954 & 1957 a team of IBM employees at IBM lead by John Backus developed a language to simplify the programming of numerical calculations
- Language called FORTRAN released in 1957
  - (FORTRAN = FORMula TRANslator)
- Looks almost nothing like todays FORTRAN 95
- Very small number of commands in the language
- Allowed programming of IBM-704
- Allowed floating point arithmetic operations
- Commands were translated by the compiler into numerical values of machine operation codes
- FORTRAN-II released in 1958
- FORTRAN-IV, a.k.a. FORTRAN-66, released in 1962 and adopted as American National Standards Institute (ANSI) standard in 1966
  - starts to look like modern FORTRAN
- FORTRAN 77 next major release
  - lots of powerful modular features in language
- FORTRAN 90 standardized in 1992
  - Struggle over what language should contain
  - Some features from C++ added to language
- FORTRAN 95; a minor update of FORTRAN 90 standard
- FORTRAN 2003 standardized at the end of 2004
  - Includes most features found in C++
  - Compilers not widely available yet

# What's the big deal about FORTRAN?

- FORTRAN today is not your Father's/Mother's FORTRAN!
  - Modern FORTRAN 95 does not look like FORTRAN IV
- What's good about FORTRAN?
  - FORTRAN is very natural language for expressing numerical computations
  - FORTRAN handles arrays really well (extremely important for numerical computing)
    - Think of arrays as enumerated lists of data until we learn about them in a few weeks.
    - Natural syntax for handling subsets or parts of a an array of data
  - Produces extremely fast code.
    - The language has a very restricted set of operations that allows the compiler to find, in many cases, an optimal order in which to execute those operations
    - Languages such as C, C++ allow use of pointers (references to data by it's address in memory) which make finding an optimal order of execution difficult in many cases.
    - C++ code can sometimes be made to run as fast as FORTRAN but it can take some work on the part of the programmer
    - Usually the language recommended by most vendors of supercomputing systems
  - Highly standardized language
    - Makes porting code to a new computer easy
- What's bad about FORTRAN?
  - Not flexible for input/output of data compared to C++
    - Fixed in FORTRAN 2003
  - Not good for system level programming
    - Can be done in some cases but it is not easy
    - Don't try to write an editor in FORTRAN!
  - Expressing modern programming concepts can be difficult in earlier versions of FORTRAN
    - FORTRAN 90 lessens this problem
    - FORTRAN 2003 fully resolves this problem

# A “Hello World” FORTRAN program

```
program hello_world  
write(*,*) “Hello World!”  
stop  
end program hello_world
```



Four lines of  
FORTRAN code

- Each line of code is called a statement
- Order of statements in program matters

# Top-down execution of program

```
program hi_bye  
write(*,*) "Hello World!"  
write(*,*) "Goodbye Cruel World!"  
stop  
end program hi_bye
```



Five lines of  
FORTRAN code

- Some statements are executable statements “write” statement & “stop” statement
  - Executed by the computer
- Other statements are non-executable
  - “program” statement & “end” statement
  - Provide information to the compiler
- Executable statements execute in order from the top of program to the bottom of the program

# Assignments

- Read Sections 1.1-1.6, 2.1-2.3 of Chapman (FORTRAN 90/95)
- Work through University of Utah Un\*x tutorial
  - Link on course web page
- Work through University of Surrey Tutorial 1
  - Link on course web page