Introduction to Systems Programming

COMP2100

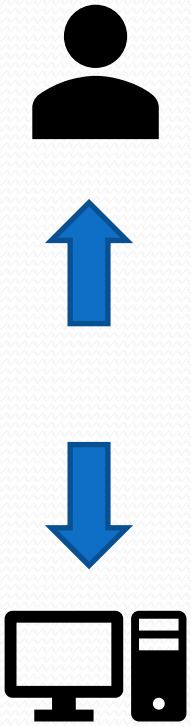
Week 1

Richard Han/**Young** Choon Lee

4 Research Park Drive

richard.han@mq.edu.au/young.lee@mq.edu.au

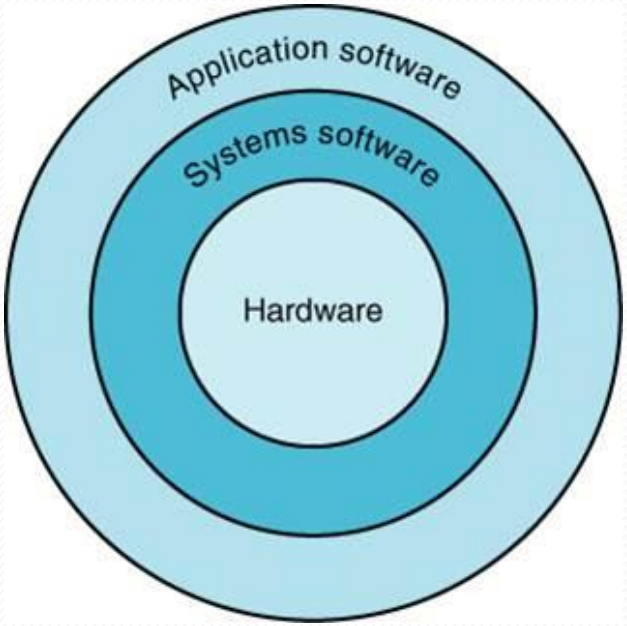
Systems programming vs Application programming

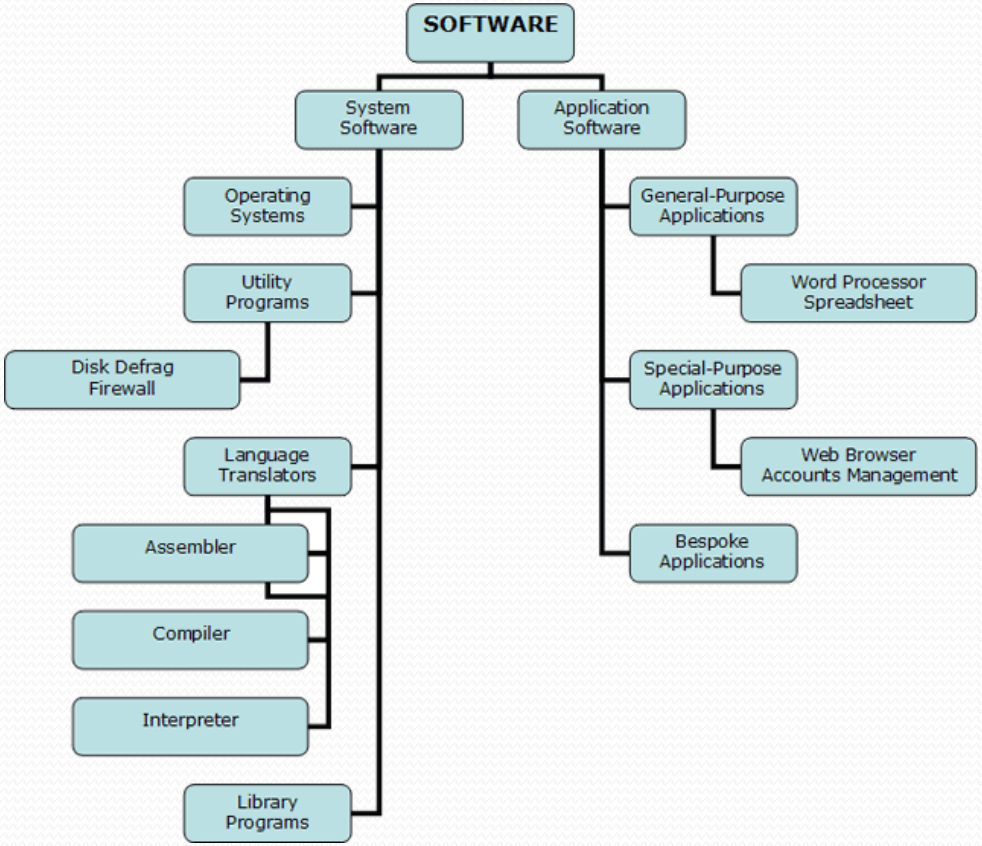


Application programming: facing (end-)users

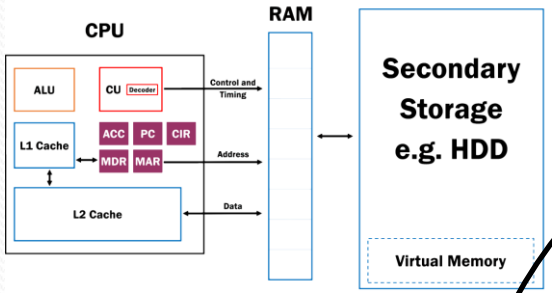
Systems programming: facing systems

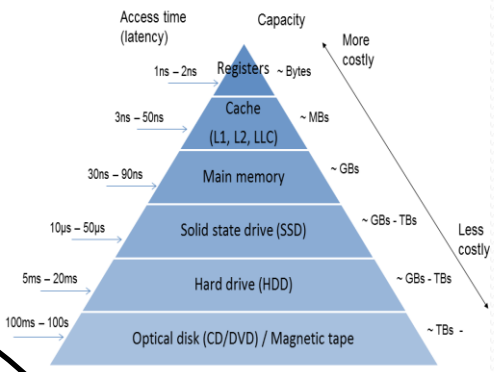
System software

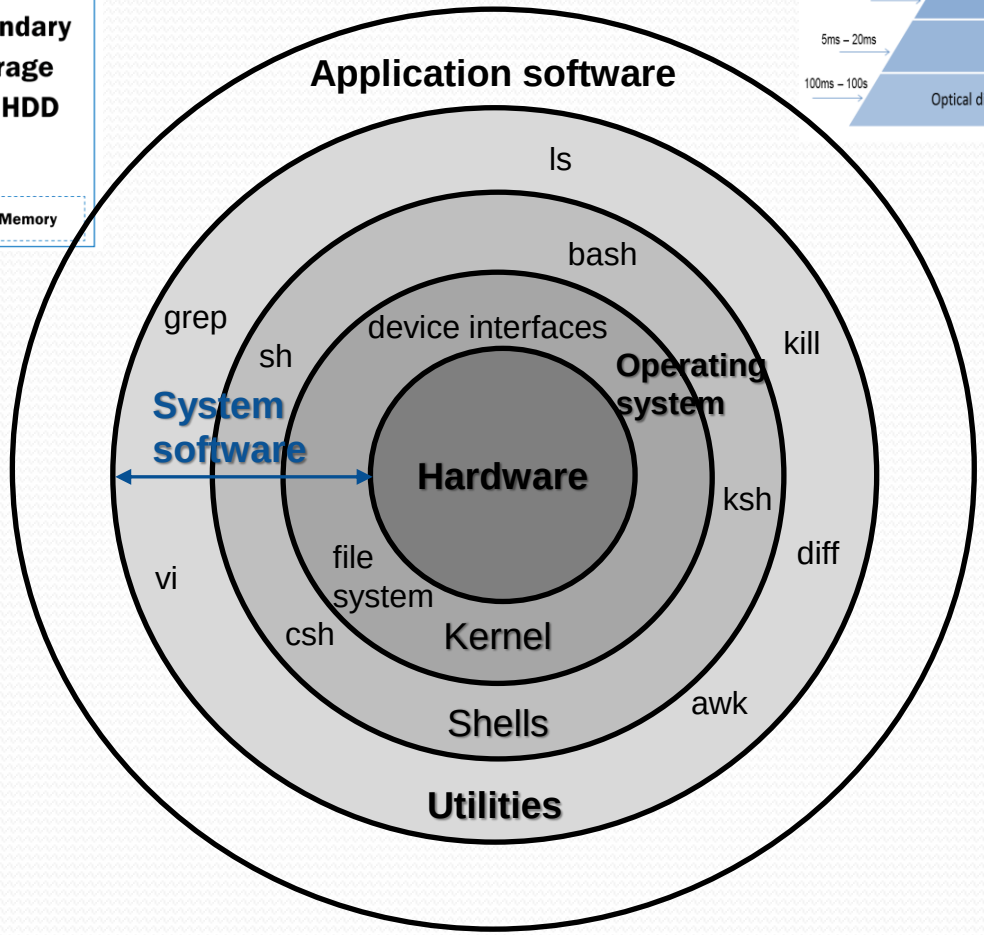


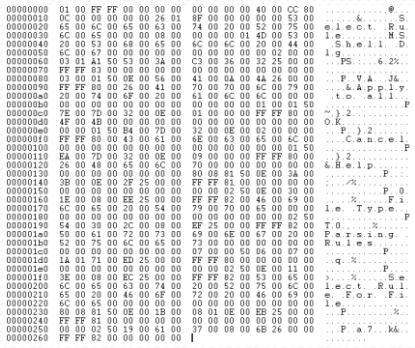


Know the system



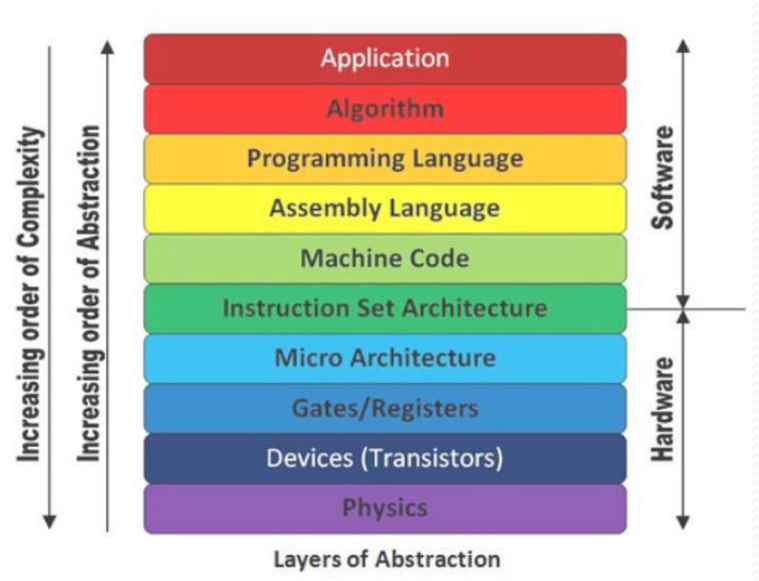






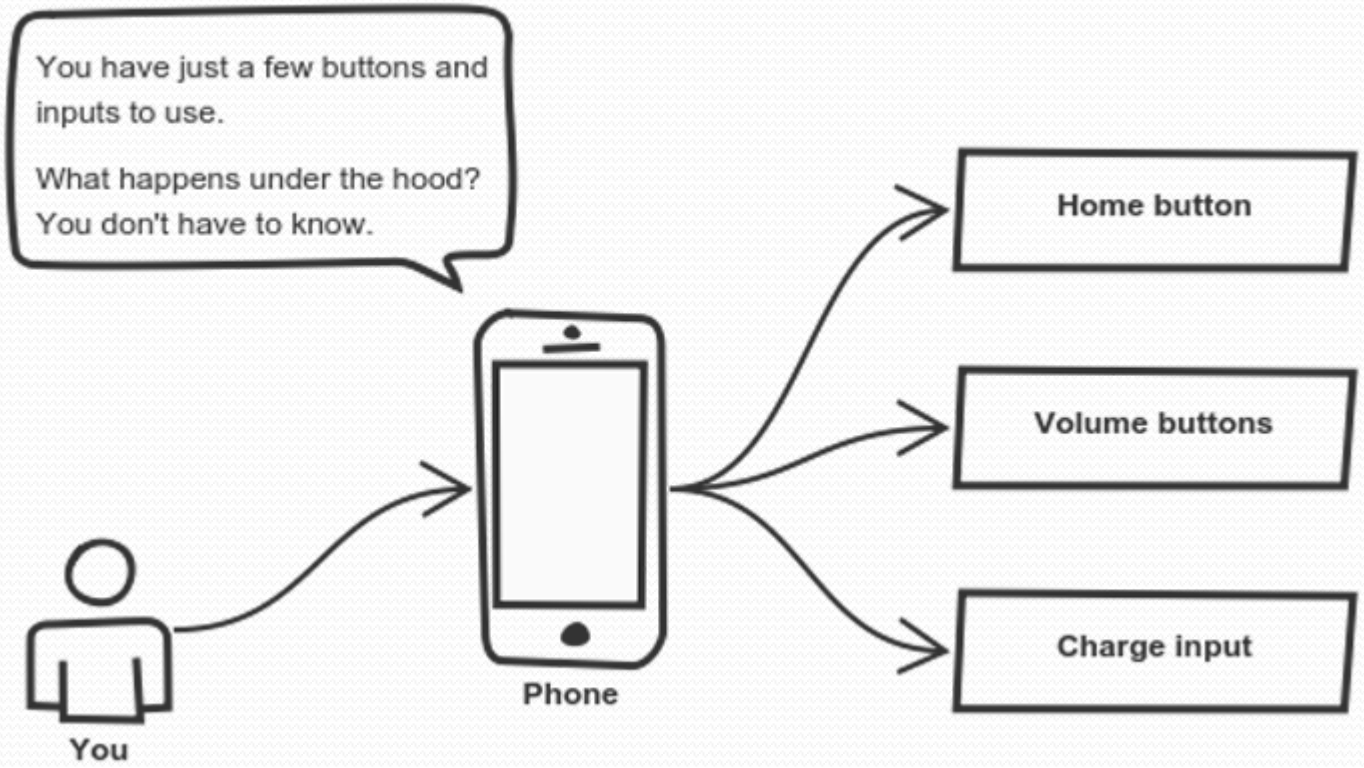
Abstraction

- Simplification of complex system via interface (modelling)
  - Machine language
  - Assembly: an abstraction of the machine language
  - Many languages are abstraction of assembly language, e.g., Java, Python and C

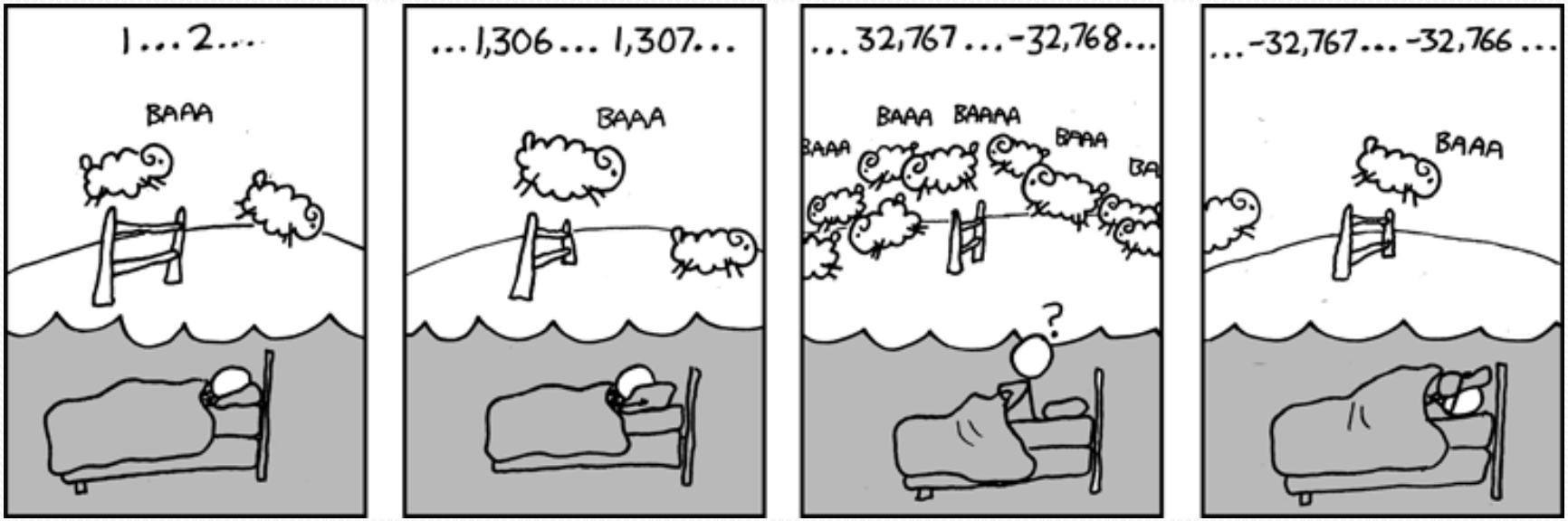


** But they still require you to think in terms of the **structure of the computer** rather than the **structure of the problem***

Abstraction

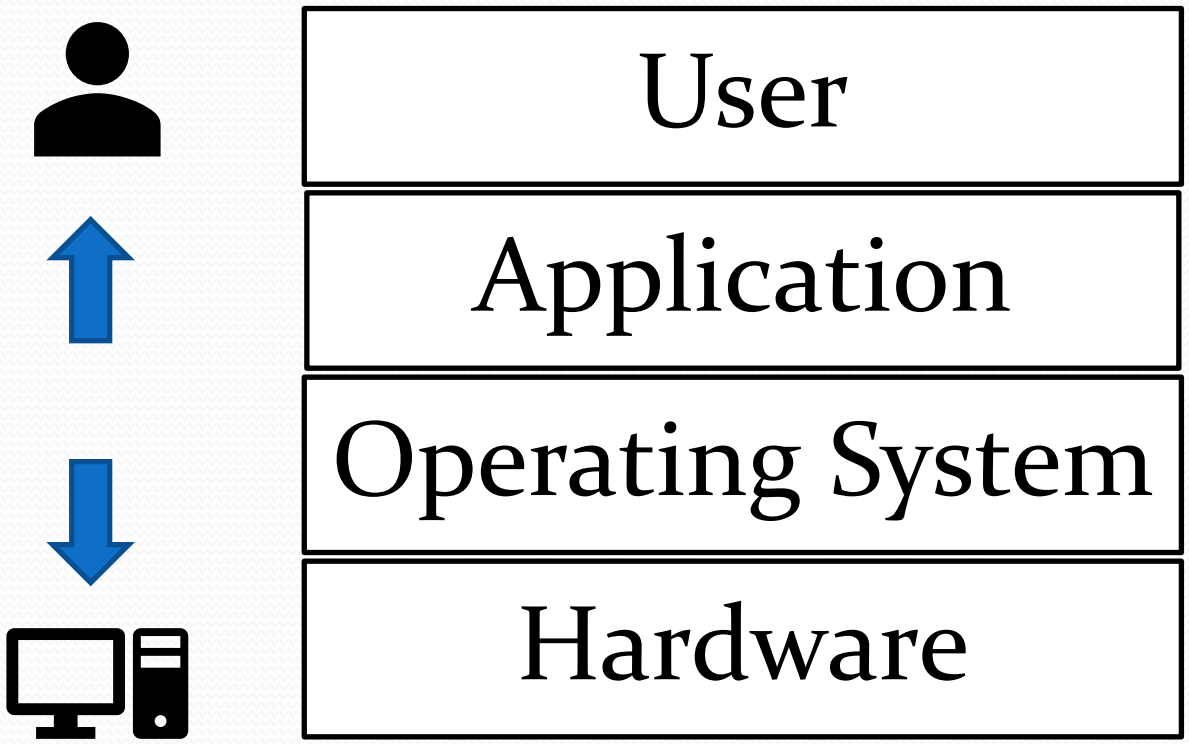


Reality vs Abstraction



Systems programming and C programming language

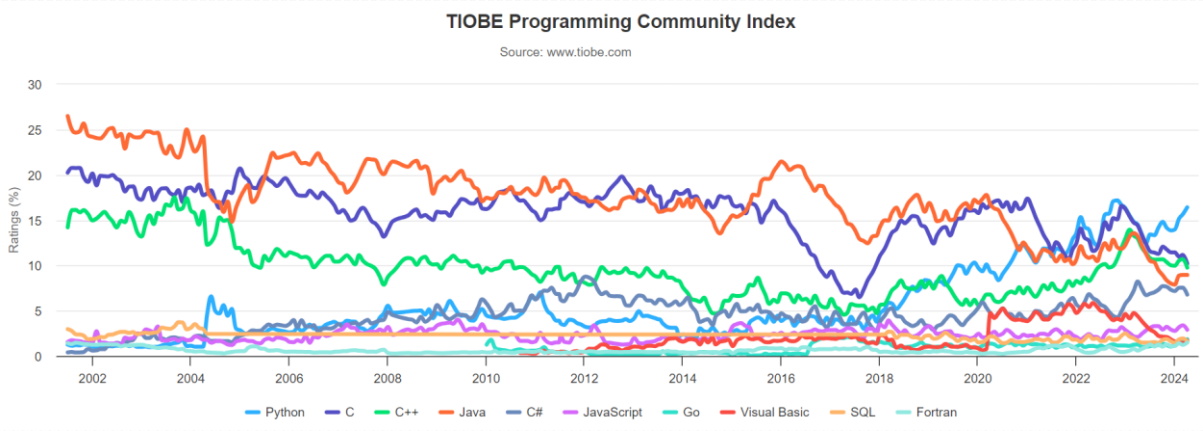
- C is a native language for many operating systems

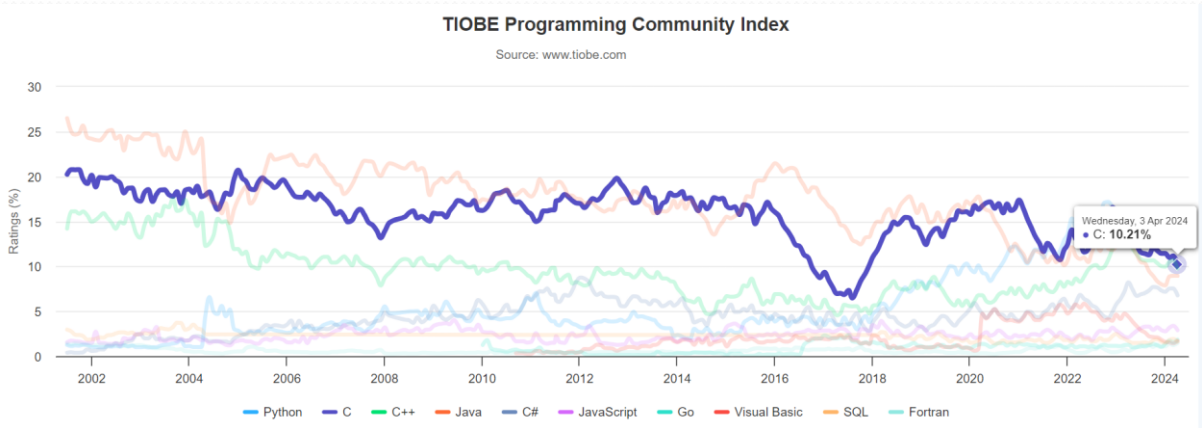


Software Systems written in C and the like

- Operating systems
  - Unix/Linux-like: mostly C and C++
  - Windows: C, C++, C# and some assembly code
  - OSX/iOS: C, C++, Objective-C
  - Android: C, C++, Java
- Other application software
  - MS Office: C → C++
  - Adobe Photoshop: C++
  - Web browsers inc. Google Chrome, Firefox, Opera: C++

C: old == obsolete?



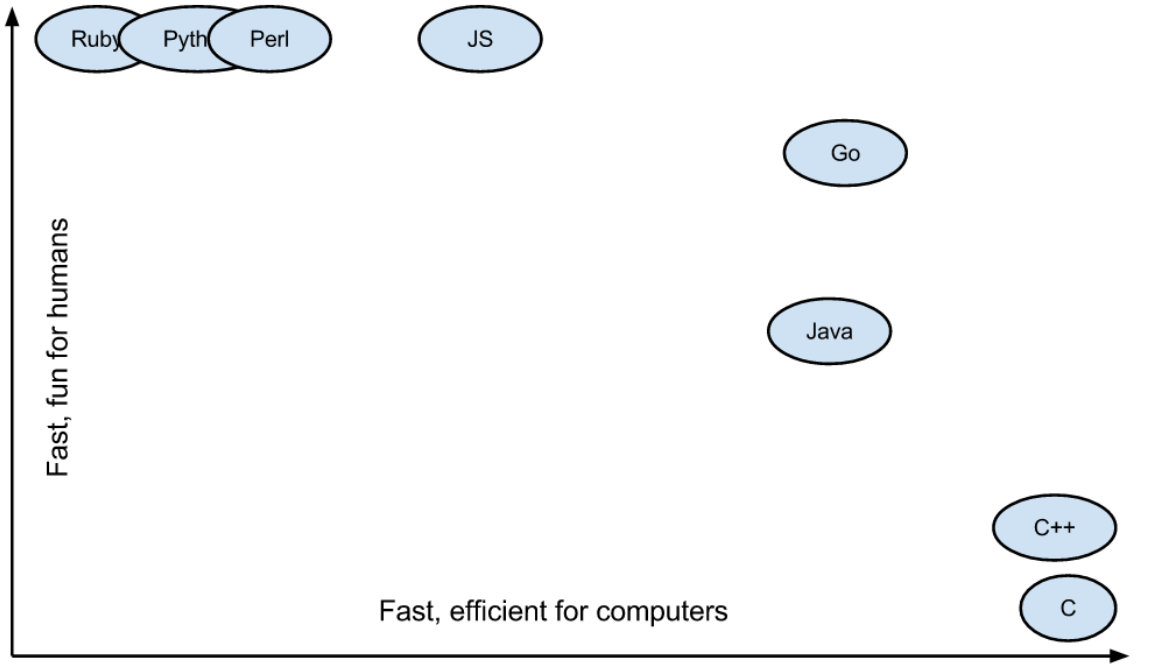


C for Java programmers

- Java is mid-90s high-level OO language
- C is early-70s *procedural* language
- C advantages:
  - Direct access to OS primitives (system calls)
  - Fewer library issues – just execute
- (More) C disadvantages:
  - language is portable, APIs are not (e.g., unistd.h)
  - memory and “handle” leaks
  - preprocessor can lead to obscure errors

Java vs C

"Although C is often criticized for its accidental complexity, unsafe programming, and lack of features. Also, C is platform-dependent, i.e., C code is not portable. But **if you want to make the most use of your hardware, then C/C++ or Rust is your only option**." *



* https://towardsdatascience.com/top-10-in-demand-programming-languages-to-learn-in-2020-4462eb7d8d3e

Operators in C: C == Java?

- Arithmetic + - * / unary –
- Bit-wise ~ & | ^ << >>
- Relational < <= == != >= >
- Logical ! && ||

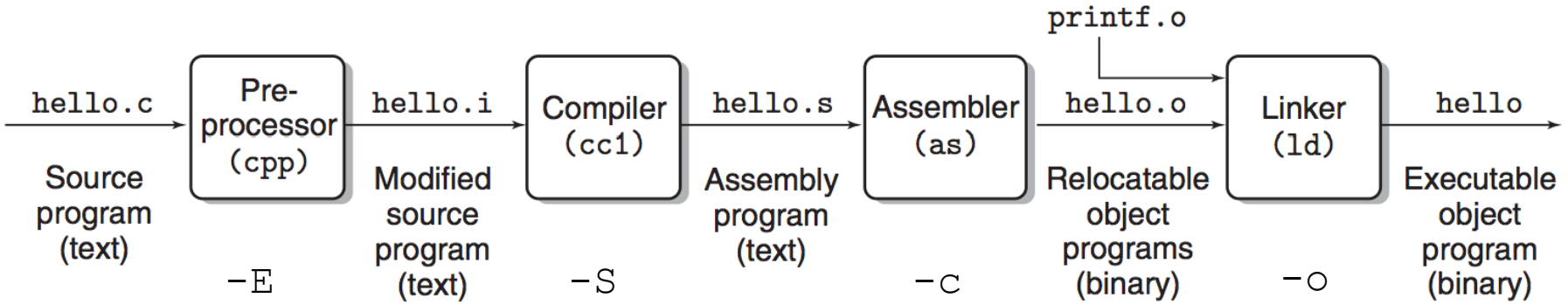
C vs Java

```
public class hello
{
   public static void main (String args []) {
      System.out.println(“Hello world”);
   }
}
```

```
#include <stdio.h>
int main(int argc, char *argv[])
{
  printf(“Hello, World\n”);
  return 0;
}
```

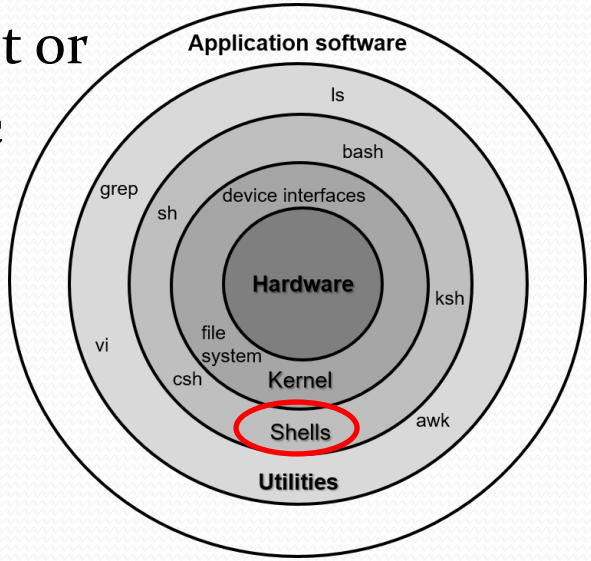
Executing C programs

- Java programs semi-interpreted:
  - javac converts `.java` into `.class`
  - not machine-specific
  - *byte codes* are then interpreted by JVM
- C programs are normally compiled and linked:
  - gcc converts `.c` into `a.out`
  - `a.out` is executed by OS and hardware



Command Line Interface (CLI)

- A (UNIX) command is a program which interacts with the OS kernel to provide the environment and perform the functions called for by the user.
- A command can be:
  - a built-in shell command, e.g., `ls`, `cp` and `whoami`
  - an executable shell file, aka a shell script or
  - a source compiled, object code file, `gcc`
- The shell is a command line interpreter



CLI: Man Pages

- The **man** command allows you to access the MANual pages for a UNIX command.
- To get additional help on any of the commands listed below, you can always type **man name_of_command** at the command prompt.
- Examples:
  - **man ssh**
  - **man passwd**

CLI: Commands

- **ls :** lists the contents of a directory
  - l : long directory listing
  - a : lists all files, including files which are normally hidden
  - F : distinguishes between directories and regular files
  - h : ? Look it up using **man**
- **pwd** : prints the current working directory
- **cd** : changes directories
  - The difference between relative and absolute paths.
  - Special characters ., .., and ~.
- **mkdir** : creates a directory
- **rmdir** : removes a directory (assuming it is empty)
  - If you get an error that the directory isn't empty even though it looks empty, check for hidden files.

CLI: Commands

- **touch** : creates an empty file with the specified name, or if the file already exists it modifies the timestamp.
- **rm** : removes a file.
  - f : force deletion
  - r : recursive deletion
- **mv** - moves a file, or renames a file
  - f : forces overwrite, if the destination file exists
- **cp** - copies a file, leaving the original intact
  - f : forces overwrite, if the destination file exists
  - r : recursive copying of directories

CLI: The UNIX Pipe (|)

- The pipe (|) creates a channel from one command to another. Think of the pipe as a way of connecting the output from one command to the input of another command.
- The pipe can be used to link commands together to perform more complex tasks that would otherwise take multiple steps (and possibly writing information to disk).
- Example: count the number of users logged onto the current system.
  - **who | wc –l**

CLI: Redirection (<, >, >>)

- <: redirects input
  - command < *file*
  - command takes its input from *file*
- >: redirects output (overwrite)
  - command > *file*
  - command (over)writes its output from *file*
- >>: redirects output (append)
  - command >> *file*
  - command writes/appends its output from *file*

CLI: Quotes

- ' (single quote)
  - Enclosing characters in single quotes (') preserves the literal value of each character within the quotes.
- " (double quote)
  - Enclosing characters in double quotes (") preserves the literal value of all characters within the quotes, with the exception of $, `, \, and, when history expansion is enabled, !.

  ** Anything which is enclosed between single and double quote can be treated as single string.*
- ` (back quote)
  - Command substitution
- \ (back slash)
  - Backslash tells shell to "ignore next character."

CLI: Quotes

```
unit=comp2100
```

#	Expression	Result	Comments
1	`"$unit"`	comp2100	variables are expanded inside `""`
2	`'$unit'`	$unit	variables are not expanded inside `''`
3	`"'$unit'"`	'comp2100'	`''` has no special meaning inside `""`
4	`'"$unit"'`	"$unit"	`""` is treated literally inside `''`
5	`'\"'`	\"	`\` has no special meaning inside `''`
6	`"\'"`	\'	`\'` is interpreted inside `""` but has no significance for `` ` ``
7	`"\""`	"	`\"` is interpreted inside `""`
8	`"*"`	*	glob does not work inside `""` or `''`
9	`"`echo hi`"`	hi	`` ` ` `` and `$()` are evaluated inside `""`
10	`'`echo hi`'`	`echo hi`	`` ` ` `` and `$()` are not evaluated inside `''`
11	`'!gcc'`	!gcc	history expansion character `'!'` is ignored inside `''`
12	`"!gcc"`	gcc args	expands to the most recent command matching `"cmd"`

Basic C Data Types

Type Name	Size in bits	Literal	Literal
char	8	`'A'`	`65`
short int	16	`25000`	`-25000`
int	32	`2000000000`	`-2000000000`
long int	64	`90000000000`	`-900000000000000`
unsigned char	8	`255`	`0xff`
unsigned short	16	`50000`	`0xface`
unsigned int	32	`4000000000`	`0x12345678`
unsigned long	64	`90000000000`	`0x1234567812345678`
float	32	`1.34f`	
double	64	`3.1415926535`	`1.234e+5`
Pointer	64	`N/A`	

Number systems

Binary Digit	Octal Digit	Decimal Digit	Hexadecimal Digit
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F

Data types: Java vs C

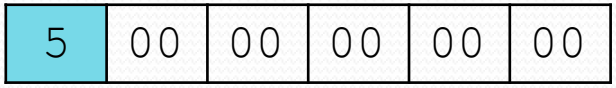
Type	Java	C
char	16 bits	8 bits
short	16 bits	16 bits
int	32 bits	16, 32 or 64 bits
long	64 bits	32 or 64 bits
float	32 bits	32 bits
double	64 bits	64 bits
boolean	1 bit	(use int)
byte	8 bits	(use char)
long long		64 bits (unofficial)
long double		80, 96 or 128 bits

Data in C

- Arrays: no automatic initialisation in C
  - C: `int array[5];`

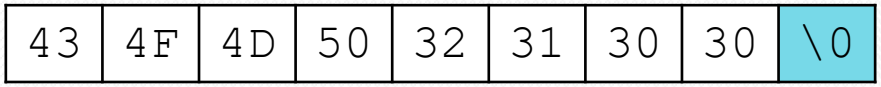
`sizeof()`

  - Java: `int[] array = new int[5];`

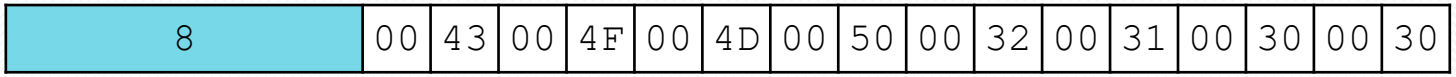


`array.length`

- Characters and strings: string bounded by a '`\0`' (`null`)
  - Example: the string (char array) "COMP2100"
    - C (ASCII)



    - Java (Unicode)



- struct vs object

Formatted Output: printf

```
int printf(char *format, arg1, arg2, ...);
```

- `printf` converts, formats, and prints its arguments on the standard output under control of the `format`. It returns the number of characters printed.
- `format` contains
  - Ordinary characters
  - Conversion specifications
    - Syntax: %*[Flag][Width][Prec][Length]Specifier*
    - e.g., `%c`, `%d`, `%s`, `%x`

Basic conversion specifications

Character (format specifier)	Argument type; printed as
`d, i`	int; decimal number
`o`	int; unsigned octal number (without a leading zero)
`x, X`	int; unsigned hexadecimal number (without a leading ox or oX), using abcdef or ABCDEF for 10, ...,15.
`u`	int; unsigned decimal number
`c`	int; single character
`s`	char *; print characters from the string until a '\o' or the number of characters given by the precision.
`f`	double; [-]m.dddddd, where the number of d's is given by the precision (default 6).
`e, E`	double; [-]m.ddddde+/-xx or [-]m.ddddddE+/-xx, where the number of d's is given by the precision (default 6).
`g, G`	double; use %e or %E if the exponent is less than -4 or greater than or equal to the precision; otherwise use %f. Trailing zeros and a trailing decimal point are not printed.
`p`	void *; pointer (implementation-dependent representation).
`%`	no argument is converted; print a %

printf: examples

```
#include <stdio.h>
int main()
{
    char ch = '@';
    char str[30] = "COMP2100 Systems Programming";
    float pi = 3.14;
    int unit = 2100;
    double lpi = 3.141592;
    printf("Character is %c \n", ch);
    printf("String is %s \n" , str);
    printf("Float value is %f \n", pi);
    printf("Integer value is %d\n" , unit);
    printf("Double value is %lf \n", lpi);
    printf("Octal value is %o \n", unit);
    printf("Hexadecimal value is %x \n", unit);
    return 0;
}
```

Functions

```
int sum(int a, int b);

int main(int argc, char **argv) {
    …
    j = sum(3, 5);
    …
}

int sum(int x, int y) {
    return x + y;
}
```

Prototype

Definition