Programmazione C++ per la Fisica

Emiliano Mocchiutti
INFN Sezione di Trieste

Università degli Studi di Trieste, Laurea Magistrale in Fisica
II Semestre A.A. 2016/2017

Outline 2017/03/03

- Introduction
- Preliminary outline of this course
- Basic programming knowledge: hardware, software, Linux, editors, basic shell commands, shell scripting…
- C++ very brief history
- Executables, compiling and running programs
- Basic grammar:
  - “Hello world” explained
  - Programming style, comments

References for today:

learncpp.com : 0.6 (part), 1.2, 4.1, 4.2, 5.1

http://geosoft.no/development/cppstyle.html

Emiliano Mocchiutti

- INFN Researcher – works on astroparticle and nuclear physics:
  - CAPRICE98 http://ida1.physik.uni-siegen.de/caprice2.html
  - PAMELA http://pamela.roma2.infn.it
  - GAMMA-400 http://gamma-400.roma2.infn.it
  - FAMU http://webint.ts.infn.it/ricerca/exp/famu.html
- Email: **Emiliano.Mocchiutti (at) ts.infn.it**
- Skype: **emocchiutti**
- Office: INFN Labs c/o Area Science Park

  L3 building, Room 225 (2nd floor)

  Padriciano 99, I-34149, Trieste, Italy
- Phone: **+39 040 375 6231**
- URL: http://emocchiutti.altervista.org

Preliminary Outline of this Course

- Introduction and basic proamming knowledge: hardware, software, linux, bash, editors
- Executables, libraries, environment, compiling (GCC) and running programs
- Basic grammar:
  - "Hello world"
  - Programming style, language, comments
  - Expressions, types, declarations, statements
  - Casting, functions, pointers, references, arrays
  - Structures, templates, namespaces
  - I/O

Preliminary Outline of this Course

- C++:
  - objects, Objects Oriented (OO) structure, classes
  - class structure: private, protected, public
  - constructors and destructors, “new” and “delete”
  - getters and setters
  - inheritance
  - overloading
  - polymorphism
- C/C++ standard template library (STL)
- Making compilation easier (make and CMake)

Preliminary Outline of this Course

- The physics data handling tool: ROOT
  - ROOT basics: structure of ROOT, CINT
  - data storage (TFile, rootples,…)
  - graphics, histograms, data analysis, formulas, fitting
- ROOT 6 vs. ROOT 5
- **GEANT4**
  - basics: structure of GEANT4
  - geometry definition
  - output
  - interfaces (GGS?)

Preliminary Outline of this Course

And much more hidden here and there:

- When things are not going as they should…
  - warnings and errors
  - debugging (gdb/ddd, s/ltrace, valgrind,…)
  - optimization (memory, speed, storage)
- Versioning systems (CVS, git)
- Documentation (doxygen)

Preliminary Outline of this Course

examples

C++

1/3

data analysis of simulated data

ROOT
GEANT4

2/3

References (1/2)

- Slides + source code on MOODLE or

http://bit.ly/EM-CPP (points to http://www.ts.infn.it/~mocchiut/C++ from there click on "PROGRAMMAZIONE C++ PER LA FISICA AA 2016/2017")

- Web:

textbooks

http://www.learncpp.com **http://www.cplusplus.com**

http://root.cern.ch **http://geant4.web.cern.ch/geant4/**

- Apps:

Android, C++ Reference (free)
https://play.google.com/store/apps/details?id=com.divergentsl.cpptutorial

References (2/2)

- "*The C++ programming language*" Bjarne Stroustrup, Addison-Wesley Professional, 3 edition (1997), ISBN: 978-0201889543 , http://www.amazon.com/C-Programming-Language-3rd/dp/0201889544/ref=sr_1_4?s=books&ie=UTF8&qid=1349104542&sr=1-4

- "*Scientific and Engineering C++: An Introduction with Advanced Techniques and Examples*", John J. Barton, Lee R. Nackam, Addison Wesley (1994), ISBN: 978-0201533934
  http://www.amazon.com/Scientific-Engineering-Introduction-Advanced-Techniques/dp/0201533936

- other links when needed

Timetable and final examination

- **Place**: here, accounts: http://www.infis.univ.ts.it/index.html
- **Timetable**:

each Friday from 14.00 (sharp) to ~17.00

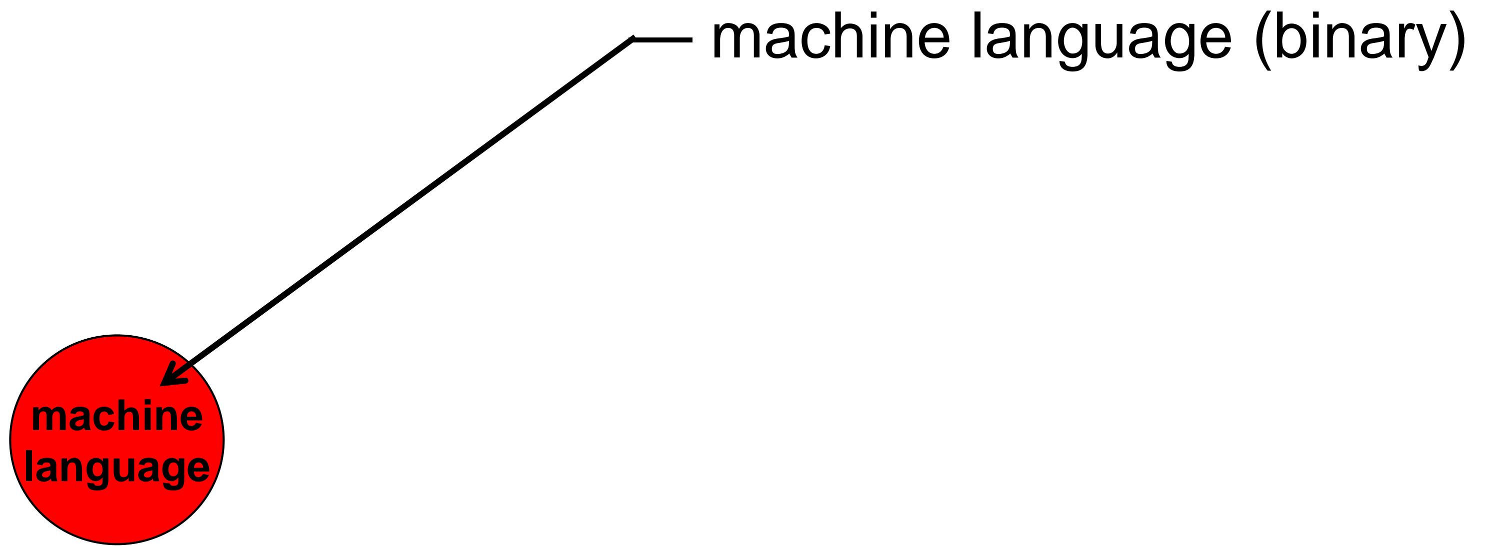
Lectures structure: about 2 hours theory, 1 hour programming

- **Examination**, three steps:
  - written, very short test on C++ (first part of course, three/four questions)
  - written, coding an analysis program
  - oral, running and discussion of the code

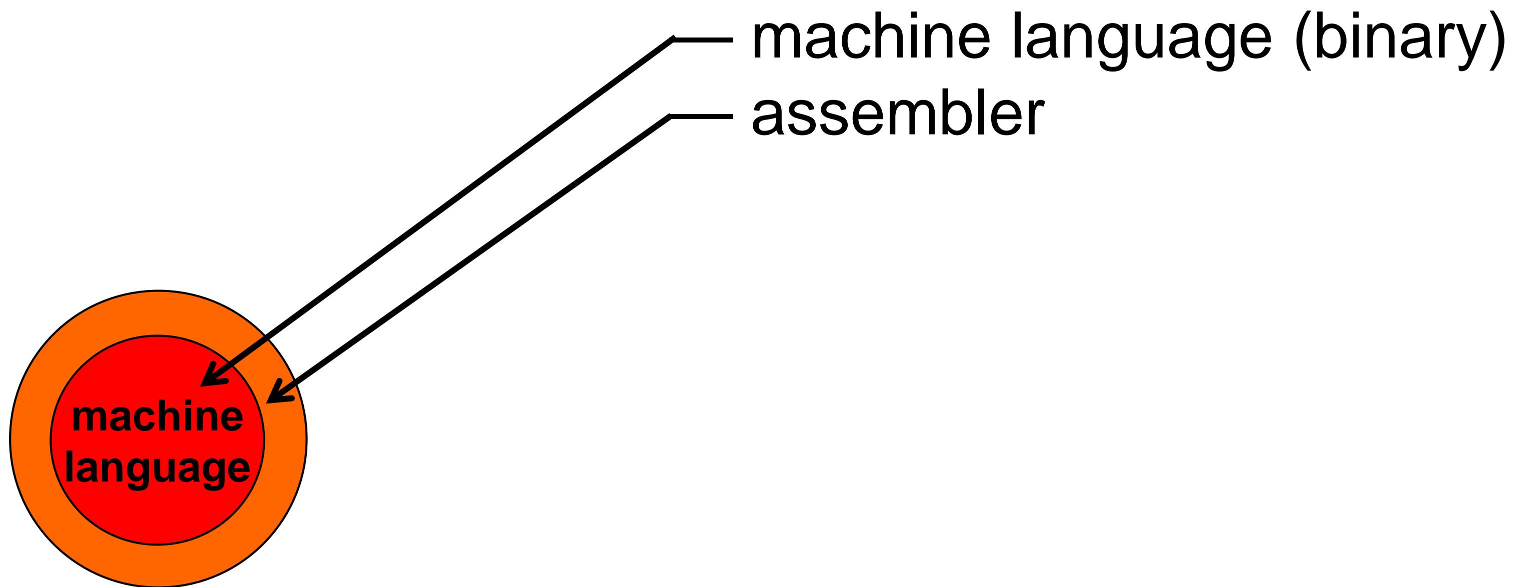
C++ programming in Physics

- Basic programming knowledge: hardware, software, Linux, editors
  http://www.sheffield.ac.uk/polopoly_fs/1.13425!/file/IntroLinux.pdf
- basic shell commands, shell scripting
  http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html (chapters 2 and 3)

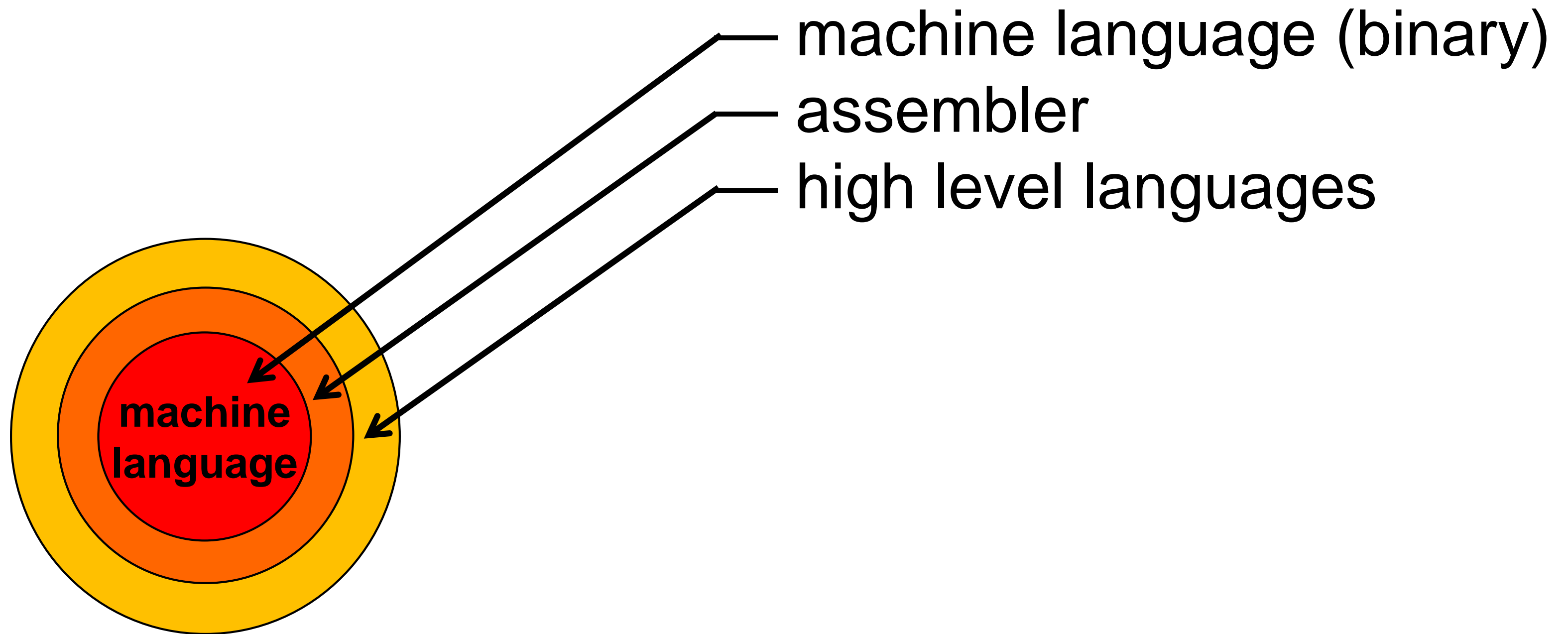
Computers interfaces…



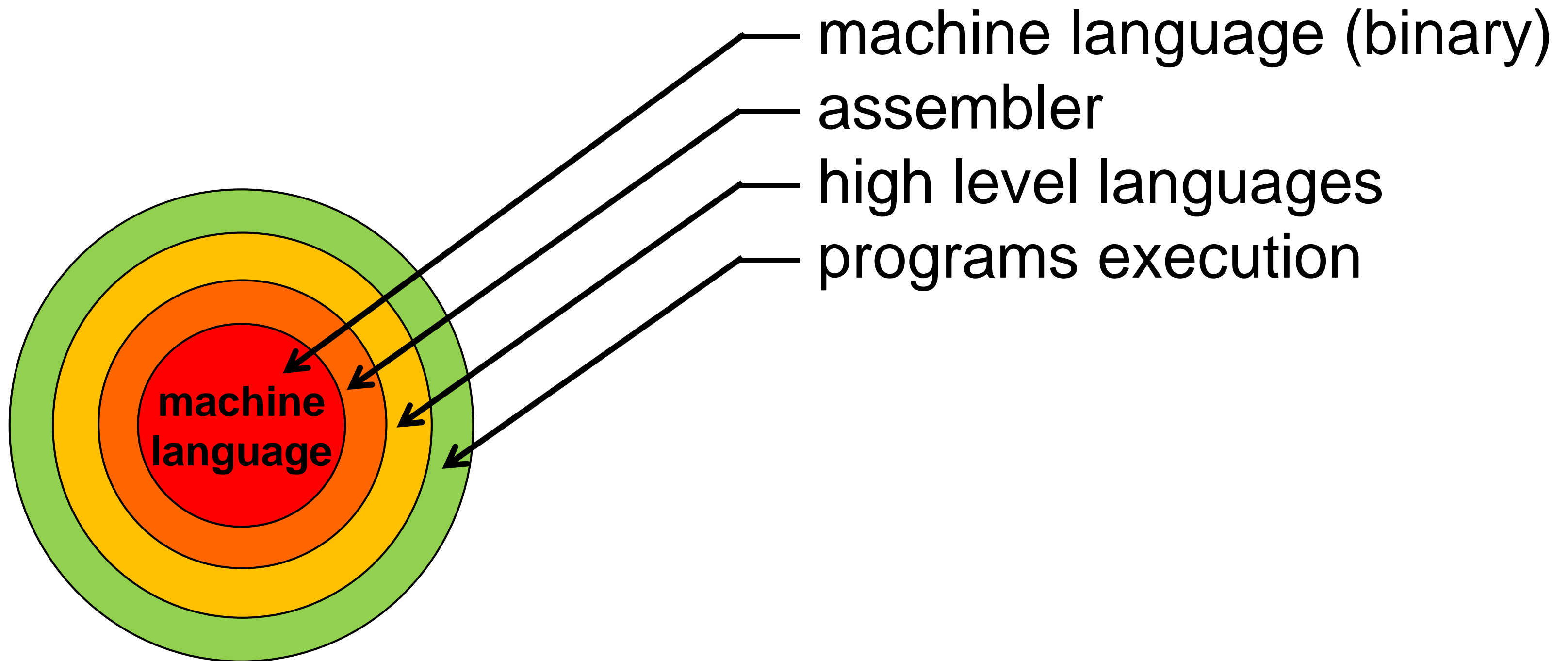
Computers interfaces…



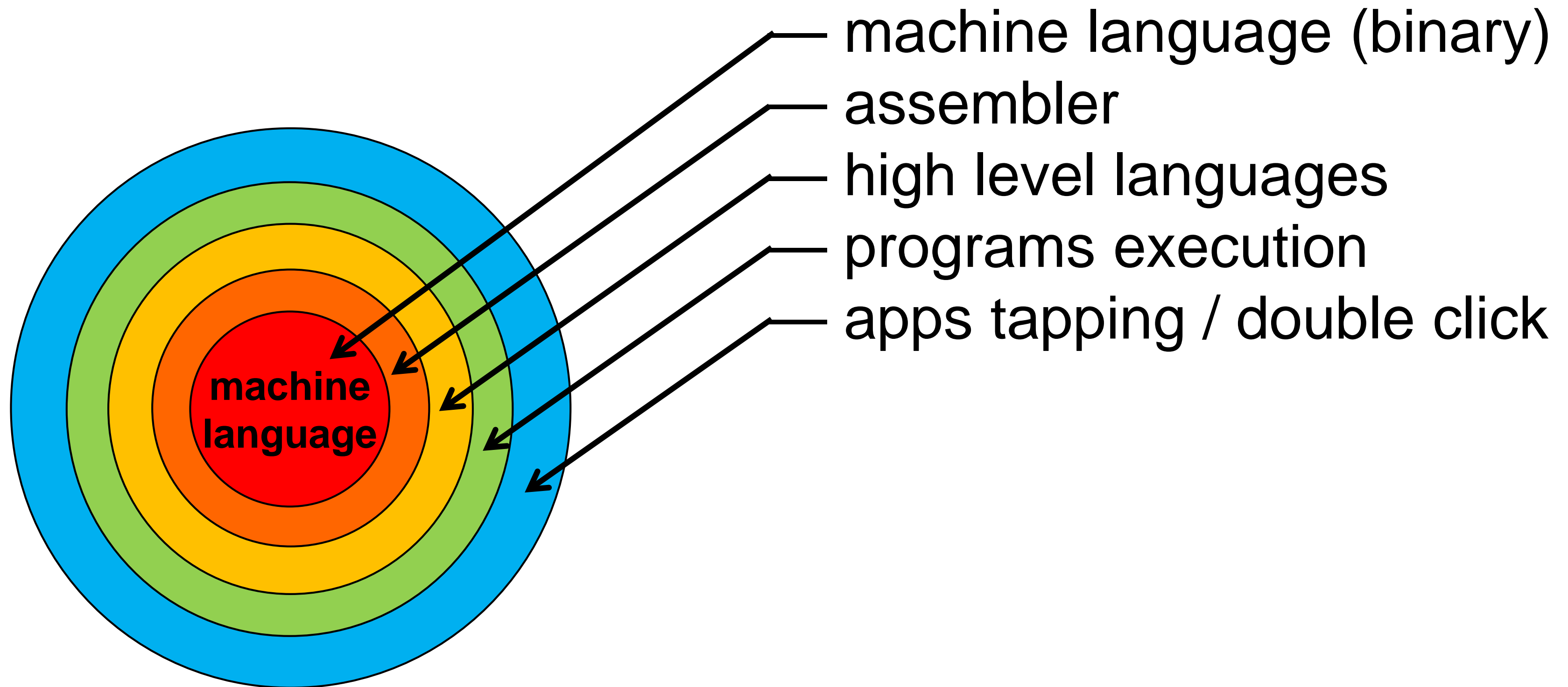
Computers interfaces…



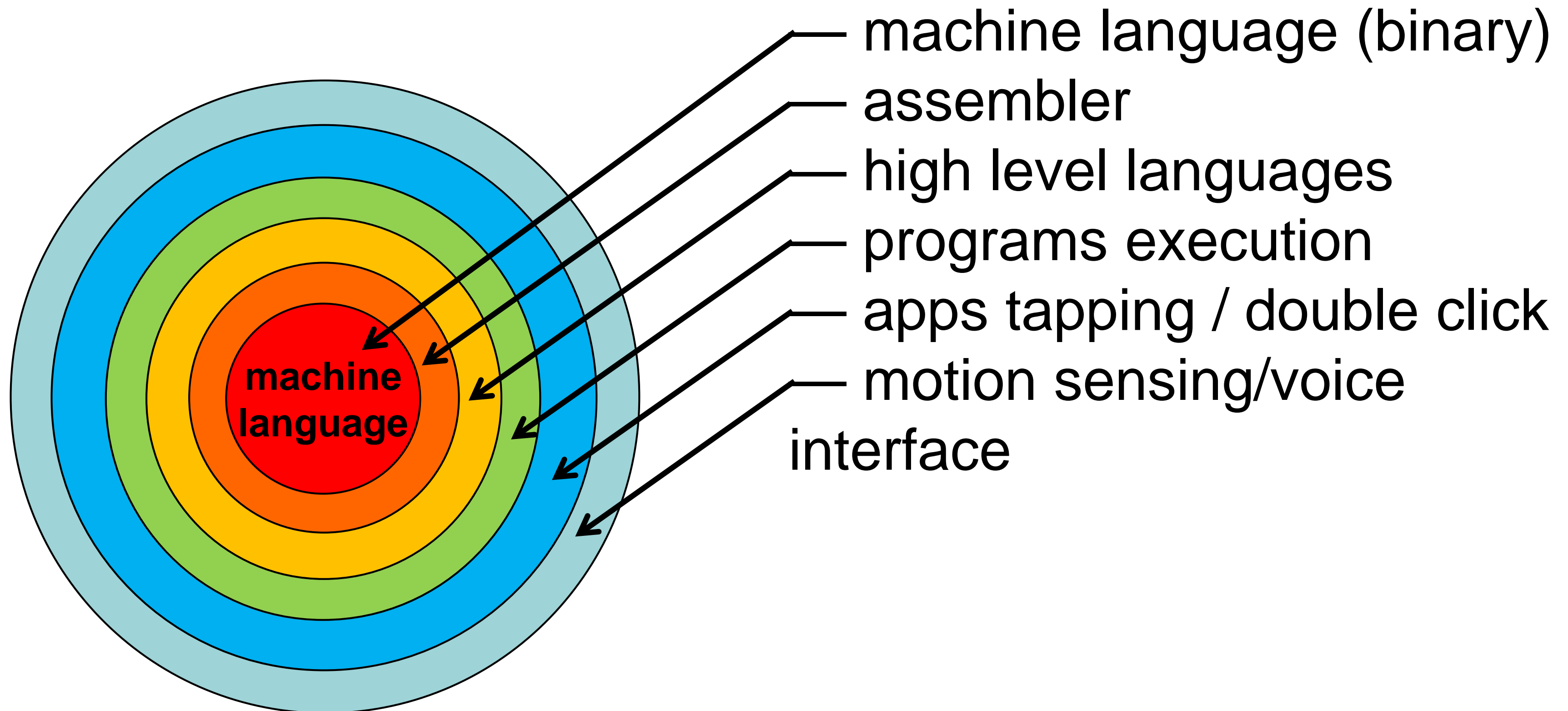
Computers interfaces…



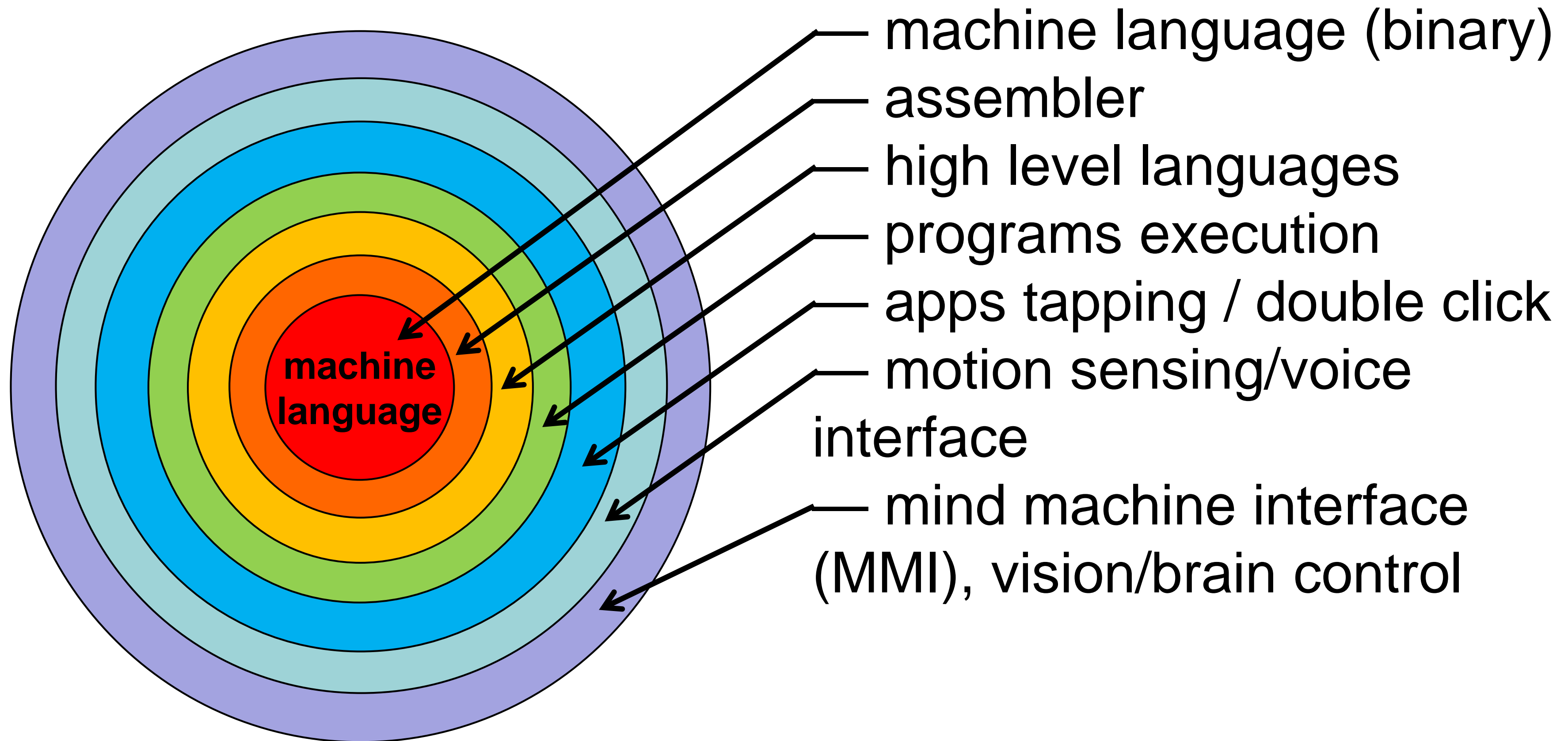
Computers interfaces…



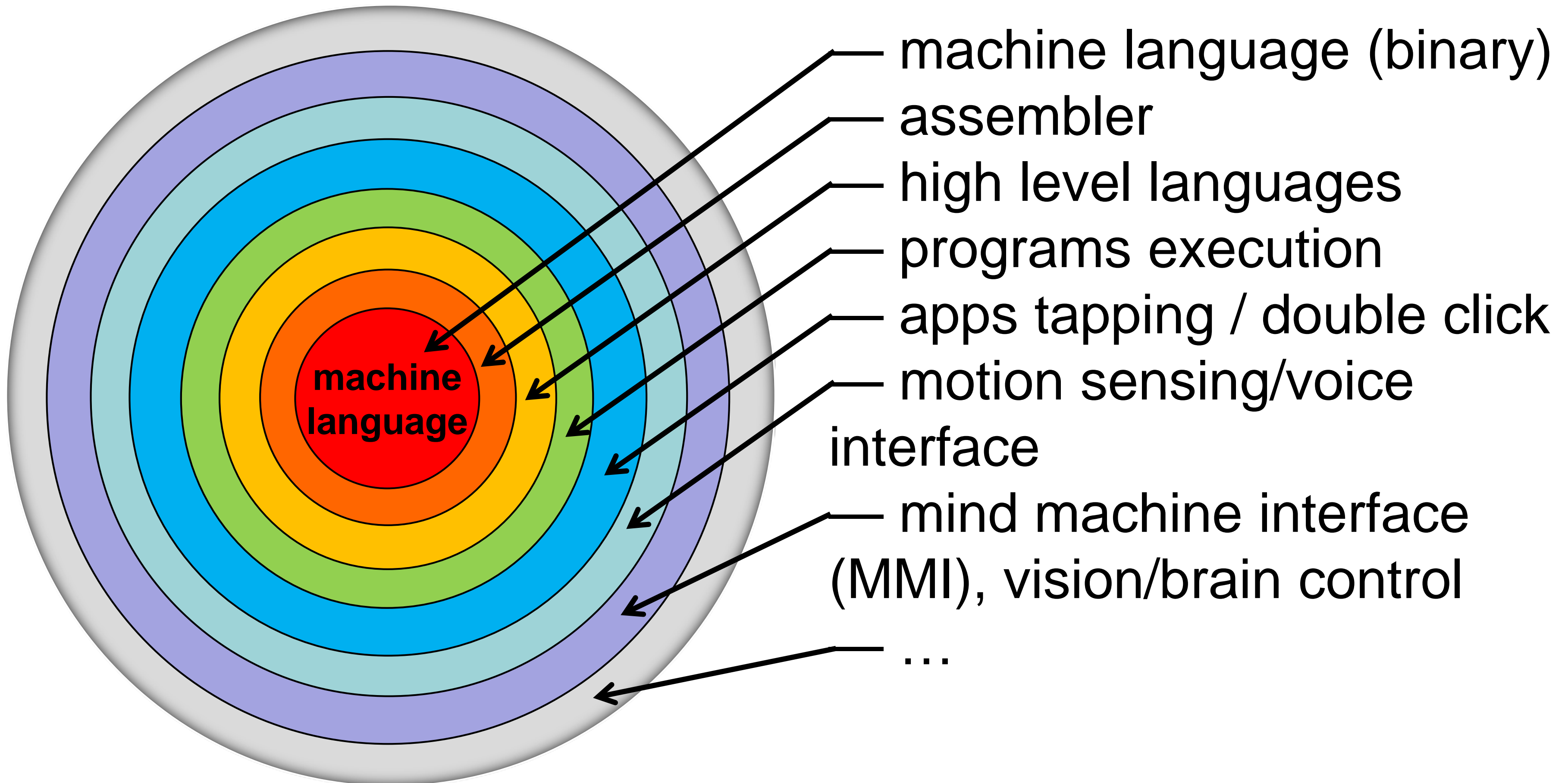
Computers interfaces…



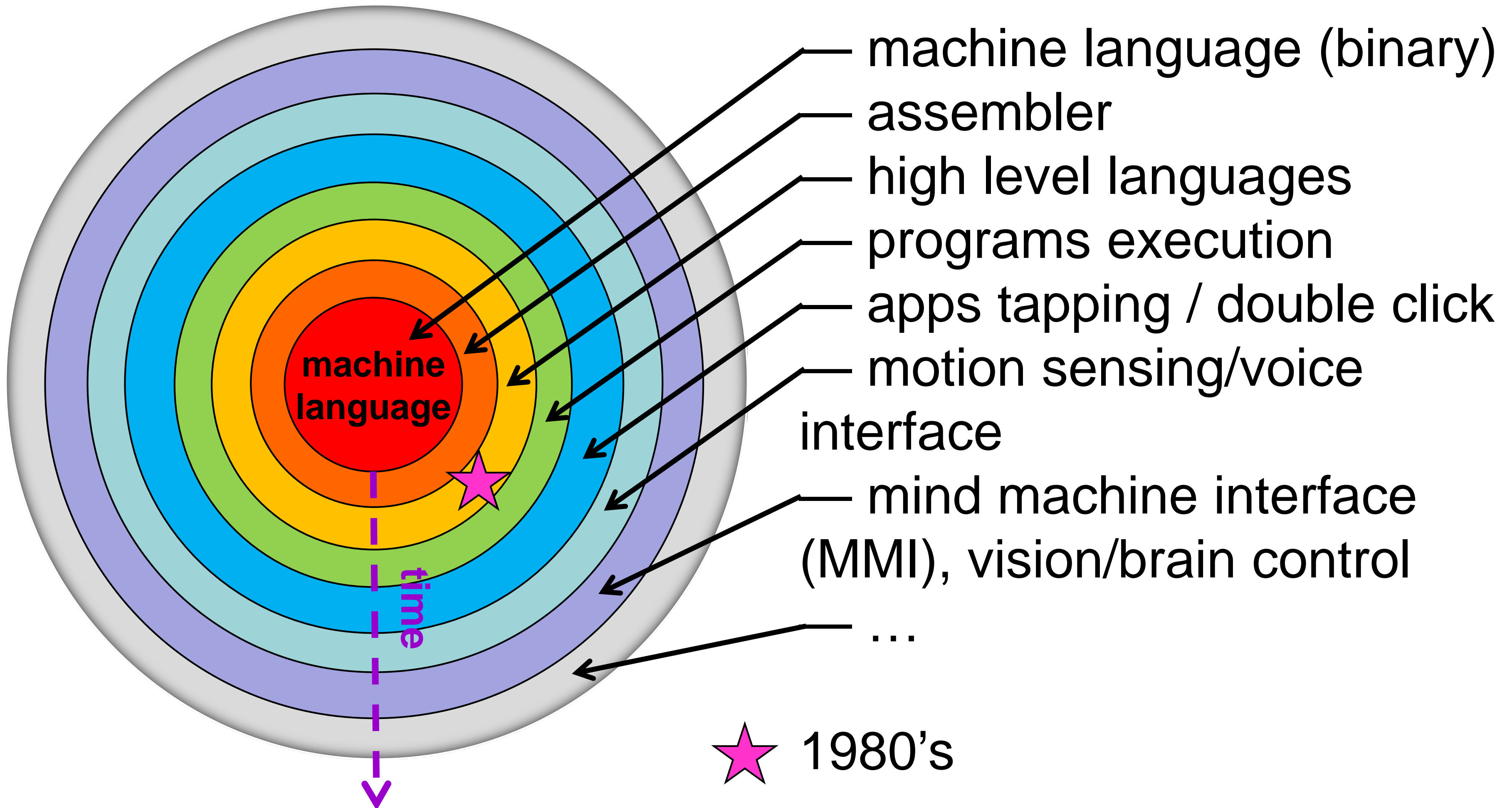
Computers interfaces…



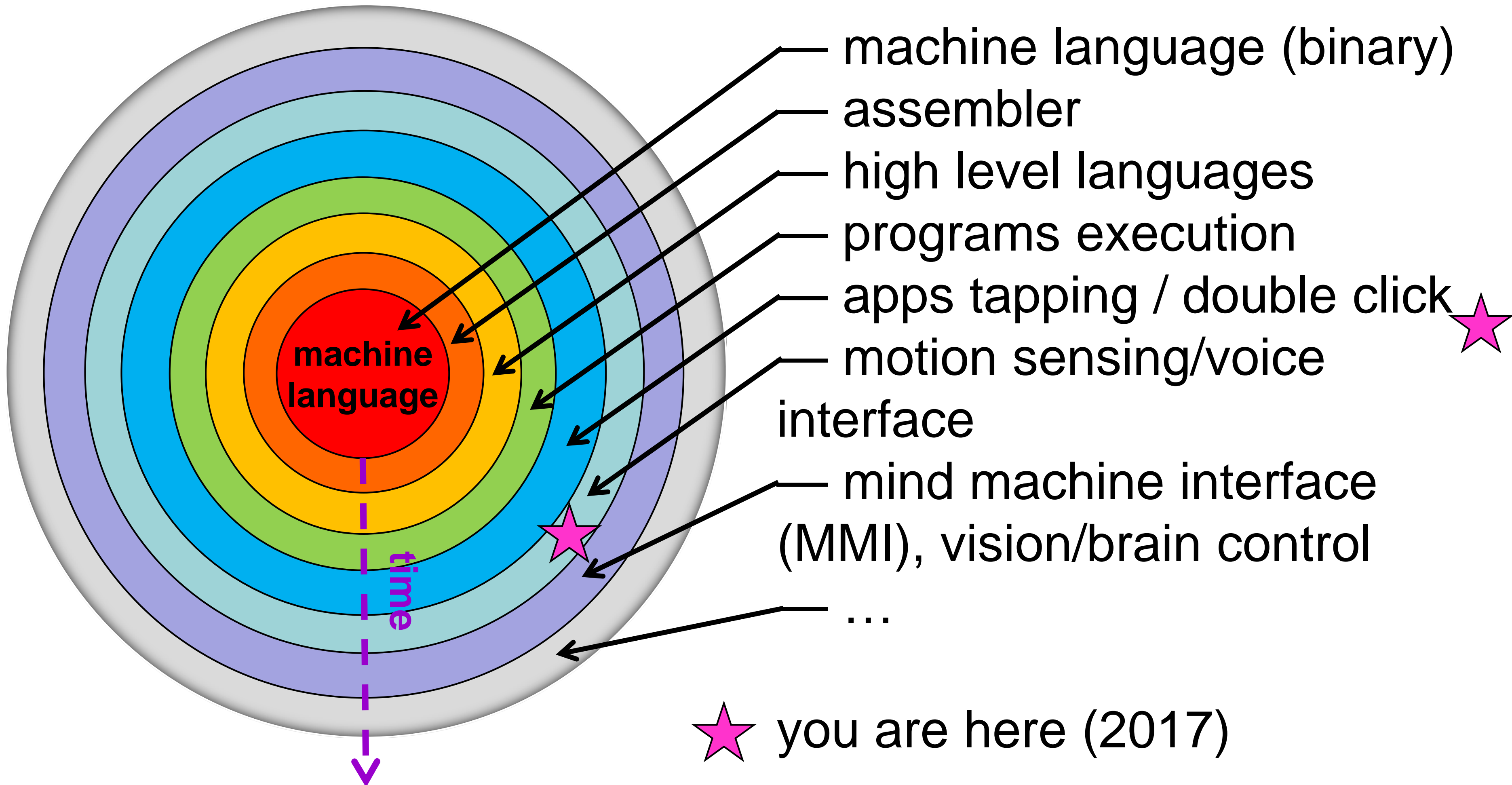
Computers interfaces…



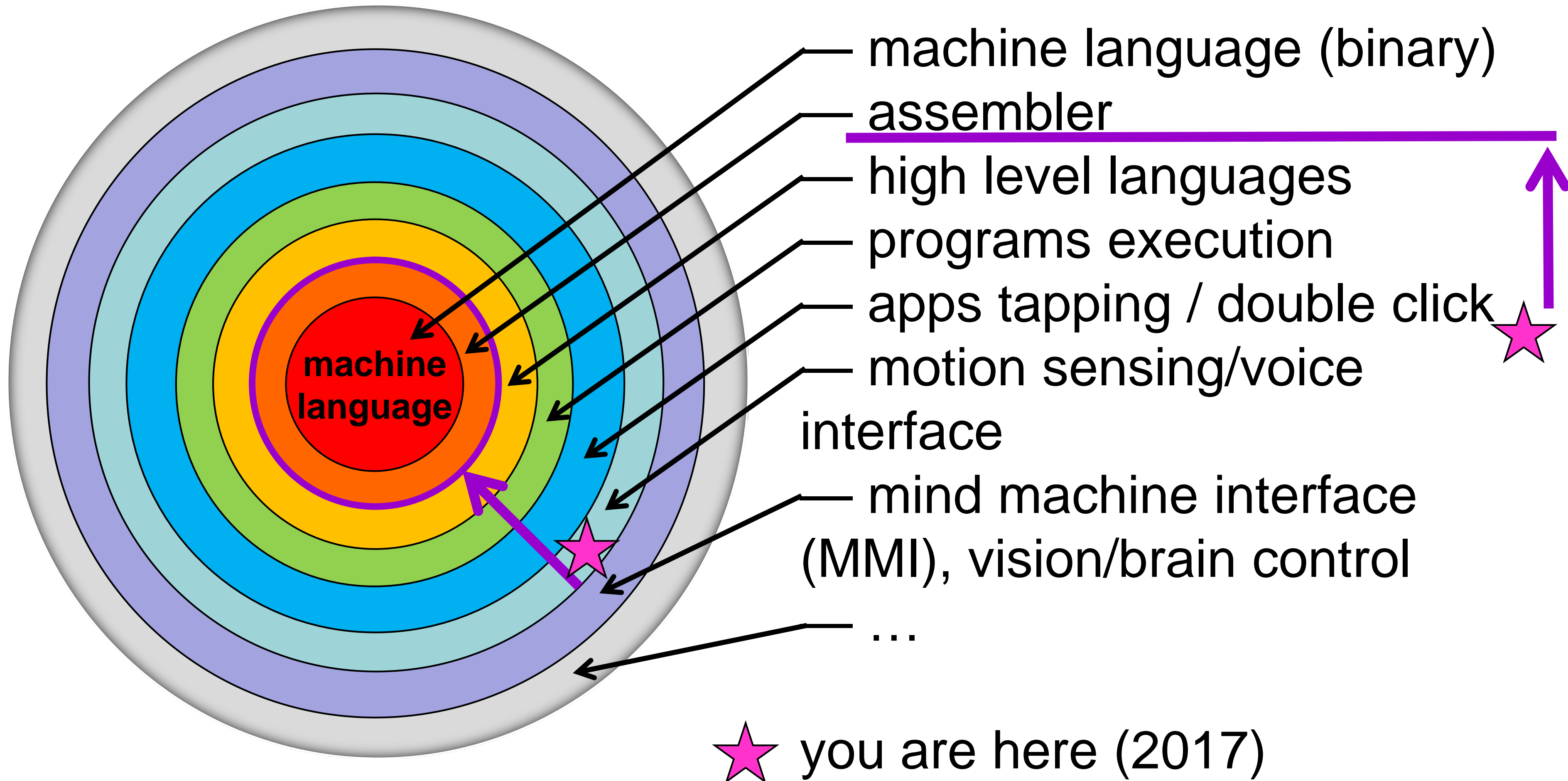
Computers interfaces…



Computers interfaces…



Computers interfaces…



Computers interfaces…

```
3:unixts.ts.infn.it - unixts - SSH Secure Shell
File  Edit  View  Window  Help
Quick Connect   Profiles
total 3.0M
drwxr-x--- 13 mocchiut users 4.0K Sep 19 13:03 scripts/
drwxr-xr-x  5 mocchiut users 4.0K Sep 19 09:50 pamela/
-rwxr-x---  1 mocchiut users 7.7K Aug 22  2013 test*
-rw-r-----  1 mocchiut users 2.0K Aug 22  2013 test.o
-rwxr-x---  1 mocchiut users 6.7K Aug 22  2013 libprova.so*
-rw-r-----  1 mocchiut users 1.4K Aug 22  2013 prova.o
-rw-r-----  1 mocchiut users  139 Aug 22  2013 test.cpp
-rw-r-----  1 mocchiut users  140 Aug 22  2013 prova.h
-rw-r-----  1 mocchiut users  107 Aug 22  2013 prova.cpp
drwxr-xr-x  2 mocchiut users 4.0K Mar 28  2013 Exercise2/
-rw-r-----  1 mocchiut users 1.7M Mar 28  2013 CS_output.root
lrwxrwxrwx  1 mocchiut users   27 Mar 28  2013 pamelasimu.root -> pame
la/data/pamelasimu.root
-rwxr-x---  1 mocchiut users  14K Mar 28  2013 EM_main*
-rw-r--r--  1 mocchiut users  99K Mar 28  2013 Verbale_Cumulativo_Appello_Progra
mmazione_C++_per_la_Fisica_Lotto_31056-28.3.2013.pdf
drwxr-xr-x  2 mocchiut users 4.0K Mar 28  2013 Exercise3/
-rw-r--r--  1 mocchiut users 2.1K Mar 28  2013 one.cpp
drwxr-xr-x  2 mocchiut users 4.0K Mar 25  2013 Desktop/
drwxr-x---  2 mocchiut users 4.0K Mar 25  2013 tar/
drwxr-x---  2 mocchiut users 4.0K Jan 10  2013 bin/
-rw-r-----  1 mocchiut users  606 Nov 29  2012 treeExample2.C
-rw-r--r--  1 mocchiut users 116K Nov 21  2012 TObject_Inh.png
|Emi@marte ~>
Connected to unixts.ts.infn.it    SSH2 - aes128-cbc - hmac-n  80x25
```

Computers in Physics

- Remote control, slow control

- Data acquisition
- Data storage
- Data reduction (from raw data to observables)
- Data analysis
- Detectors simulation

- Data and informations exchange
- Infos research
- Publications

Computers in Physics

- Remote control, slow control

- Data acquisition
- Data storage
- Data reduction (from raw data to observables)
- Data analysis
- Detectors simulation

- Data and informations exchange
- Infos research
- Publications

Open source

- Free Redistribution: the software can be freely given away or sold. (This was intended to expand sharing and use of the software on a legal basis.)
- Source Code: the source code must either be included or freely obtainable. (Without source code, making changes or modifications can be impossible.)
- Derived Works: redistribution of modifications must be allowed. (To allow legal sharing and to permit new features or repairs.)
- Integrity of The Author's Source Code: licenses may require that modifications are redistributed only as patches.
- No Discrimination Against Persons or Groups: no one can be locked out.

Free software

- Open source does not mean free software
- Linux and GNU most famous open source AND free software
- 1989 R. Stalman (Emacs inventor and founder of the GNU) invented the GNU General Public License (GPL)
- since then, the GPL gives to a program the status of "free software" which is preserved even when the program is modified and/or added to other programs:
  - it is an example of "copyleft" (opposite to the "copyright")
  - you can freely use the program but you must provide the source code
  - software is free to be used but you cannot sell it (even if projects based on the software can be sold under the condition of propagating the GPL).

http://www.gnu.org/gnu/thegnuproject.en.html

http://fsfe.org/about/basics/freesoftware.en.html

Free software

- GNU? Gnu is Not Unix! founded in 1984 to develop a POSIX compliant FREE Unix-like operating system
- GNU is not Linux as well, it supports the linux kernel and fundamental applications but does not distribute the system
- Linux and GNU most famous open source AND free software
- 1992 Linus Torvald use the GNU General Public License (GPL) for the Linux kernel
- the open source free software orbit around the GNU products, which are based on the GNU compiler: GCC

http://gcc.gnu.org

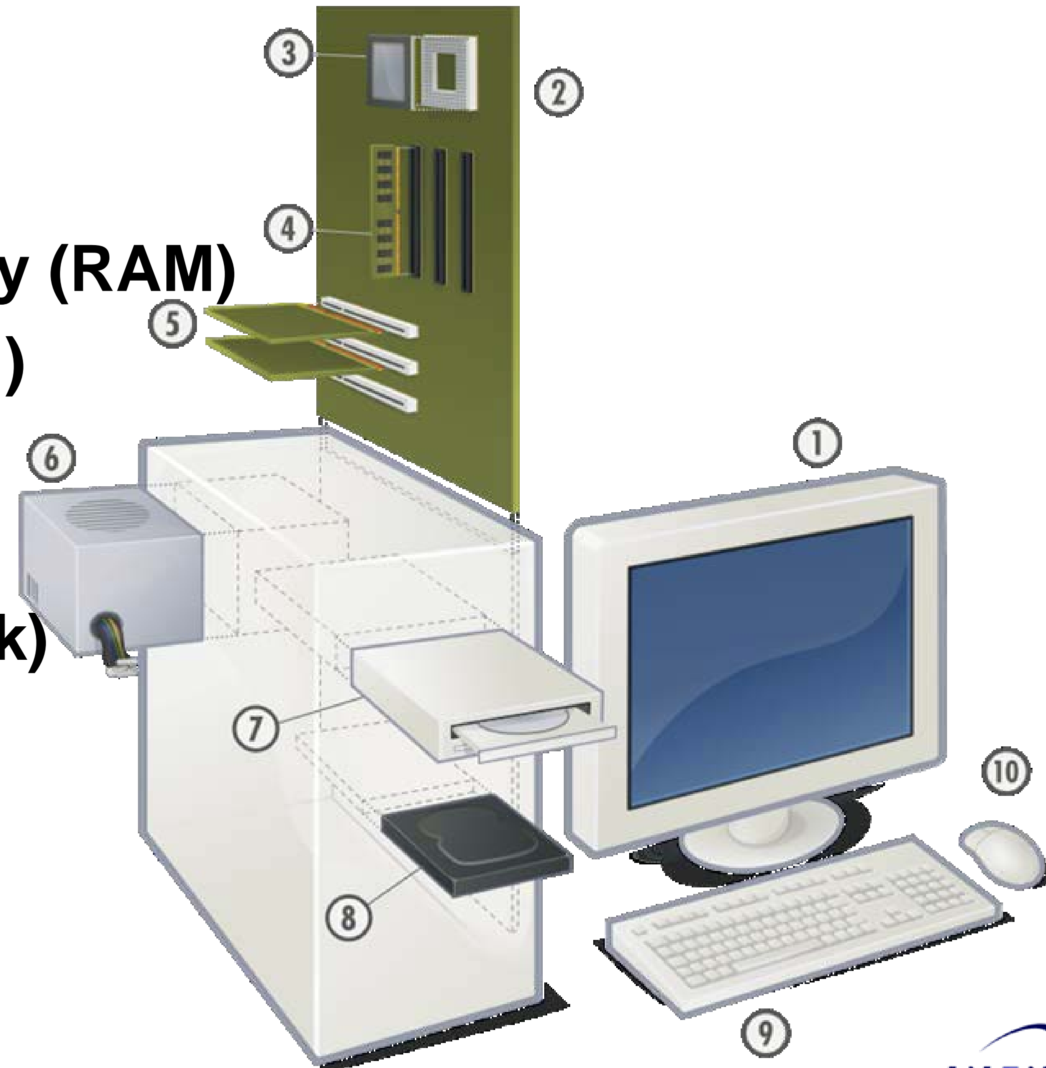
Free software

Other examples of every-day free open source software:

- APACHE, web server and more www.apache.org
- MySQL, data base www.mysql.org
- Postgresql, data base www.postgresql.org
- Asterisk, PBX system www.asterisk.org
- Openoffice, Office suite www.openoffice.org
- Wiki, Collaborative web pages (open-editing) www.wiki.org, www.wikipedia.org
- Gimp, Image manipulation www.gimp.org
- ...

The Personal Computer (PC)

1. Monitor
2. CPU
3. Motherboard
4. Random Access Memory (RAM)
5. Expansion slots (PCI, …)
6. Power
7. Optical support
8. Mass memory (Hard disk)
9. Keyboard
10. Mouse



Random Access Memory (RAM)

- The random access memory (RAM) is a set of electronic circuits (transistors) which are used to store unordered data.
- RAM is volatile. It is deleted when PC is powered off.
- Thanks to its "randomness" any data stored is returned (when needed) at the same time independently from the memory location or from the previously written or read data.
- CPU has its own “small” integrated RAM, called “cache”, used to have a faster access to a portion of the data.

Data

- Data? computers use binary format to store data, any information is converted to a string of bits, i.e. numbers in binary format.
- Bit? one cifer in binary format, it can assume only two values: 0 or 1.
- Byte? string of 8 bits, it can assume binary values from 00000000 to 11111111 that is decimal values from 0 to $(2^8 - 1) = 255$ that is hexadecimal values from 0 to FF

CPU

- The CPU works on data, performs calculations, controls all other PC components
- The CPU is logical machine able to perform a list of instructions making use of RAM
- A CPU without memory is called DSP (Digital Signal Processor) which execute a single logical operation and returns the result

The computer program

A computer program is a list of actions codified in a list of instructions needed to make the CPU performing a specified task.

The Operating System (OS)

- The Operating System is a software which handles PC resources and gives humans the interfaces to access the resources. It controls Input/Output (I/O) requests and it allocates and controls tasks and services.
- The OS controls and distributes the RAM, decides priorities, controls network and filesystem.

Linux

- Linux is an Operating System.
  - Linux is the kernel code
  - Linux is POSIX (Portable Operating System Interface) compliant, is a Unix standardization

- Other OS are: Windows, OS-X, Android,...

Linux distributions / flavours

- Linux kernel + additional softwares to interface to humans for any needed task
- Many flavours including:
  - Debian
  - Slackware
  - RedHat (Fedora Core, Enterprise)
  - Suse
  - Mandrake
  - Gentoo
  - Ubuntu
  - Scientific Linux

X-interface

- XFree86 is the open source X-windows manager in most (all?) distributions...
- ... then we need a “windows manager”
  - twm
  - fwm
  - ...
- ... or even better a “desktop manager”
  - ICE
  - KDE
  - GNOME
  - ...

Basic interface, the shell

OK, with X windows we can do many things but the basic interface to the OS is the shell.

- **The shell is a command line interpreter, it reads the user input and execute the given command(s). The "command prompt" is the line where the user writes command.**
- The shell (usually) runs inside a terminal window
- Most common shells:
  - sh: Bourne Shell
  - csh: C Shell
  - ksh: Korn Shell
  - tcsh: Enhanced C Shell
  - bash: Bourne Again Shell ⇐

The filesystem

- The filesystem represents the way informations are stored on the mass memory.
- Where are data? in a hierarchical organization of directories and files, a “tree”
- The hierchical tree develop from a “root”, the name of the “root” is a single character: /
- Directories are files which contain other files and directories; directories are files which contain the infos of their content, the “filenames”.
- The “filenames” are the names of the files in a directory. “/” is not allowed as character in the filenames.

The Linux filesystem

```
/ -
  |- /bin           -- all basics executables
  |- /boot          -- files needed to boot the system
  |- /home          -- users home directories
  |- /usr           -- everything needed by a user
  |- /usr/local        binaries, libraries, include files etc. etc. etc.
  |- /usr/bin
  |- /usr/lib
  |- /usr/include
  |- /include       -- system headers files
  |- /lib           -- system libraries and driver modules
  |- /etc           -- system configuration files
  |-……….
```

How to...

- ... move in the filesystem? command “cd”
- ... list directory content? command “ls”

./ means “this directory”

../ means “the upper directory”

- get help for a command: usually

command –h

or

man command (“man” stands for manual)

- ... look inside a file? commands: cat, more, less

  or by editing the file

File (text) editors

- xemacs
- emacs
- eclipse
- vi/vim
- nano/pico
- office
- word … …

(xemacs, emacs, eclipse: more than simple editors)

http://en.wikipedia.org/wiki/Comparison_of_text_editors

in principle any editor is good but...

File (text) editors

choose an editor which is good for coding *(nedit or gedit ok)*

- A typical editor designed for coding has a few features that make programming much easier, including:

1. Line numbering. Line numbering is useful when the compiler gives us an error. A typical compiler error will state “error, line 64″. Without an editor that shows line numbers, finding line 64 can be a real hassle.
2. Syntax highlighting and coloring. Syntax highlighting and coloring changes the color of various parts of your program to make it easier to see the overall structure of your program.
3. An unambiguous font. Non-programming fonts often make it hard to distinguish between the number 0 and the letter O, or between the number 1, the letter l (lower case L), and the letter I (upper case i). A good programming font will differentiate these symbols in order to ensure one isn’t accidentally used in place of the other.
4. Identation capabilities. C/C++ do not care about spaces and code text formatting, but humans and source code management programs do!!

Basics bash commands

http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html

|Emi@ws1 ~> ls –lrth /home/mocchiut > file_list.txt

prompt | command | options | I/O instructions

Under unix any files can be:

- r – read
- w – written
- x – executed

by:

- o – others, anybody, the world
- g – group
- u – user only, owner of the file

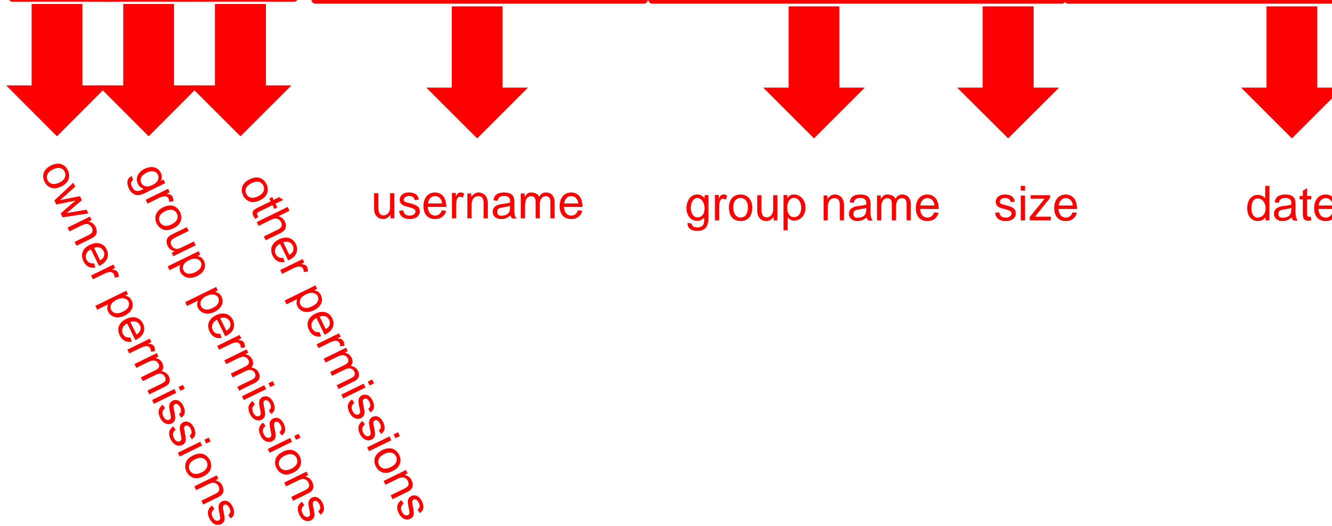
Basics bash commands

|Emi@ws1 ~> ls –lrth /home/mocchiut/.bashrc

-rw-r--r-- 1 mocchiut users 2k 2012-10-03 11:20 .bashrc

Basics bash commands

|Emi@ws1 ~> ls –lrth /home/mocchiut/.bashrc



How to change permissions and ownership:

- chmod ugo+rwx filename
- chown username.groupname filename

Basics bash I/O (redirection)

There are three types of file descriptors:

1. standard input: stdin
2. standard output: stdout (1)
3. standard error: stderr (2)

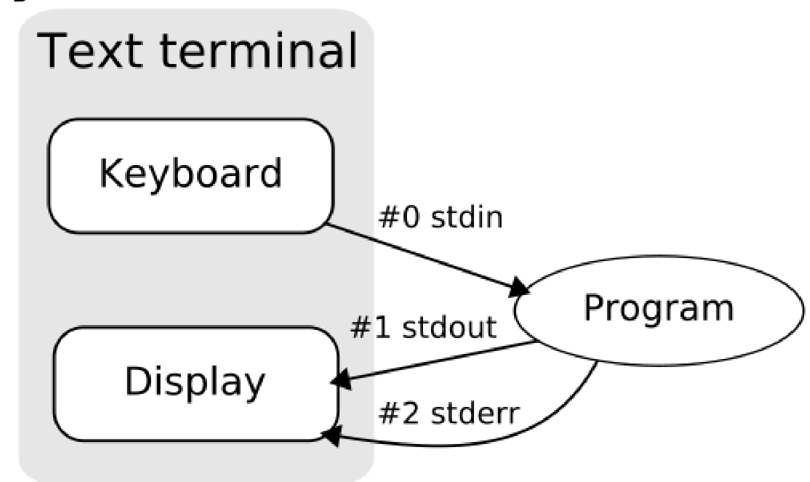
Redirection types: <, >, >>

STDIN from keyboard:

cat –n

STDIN from file:

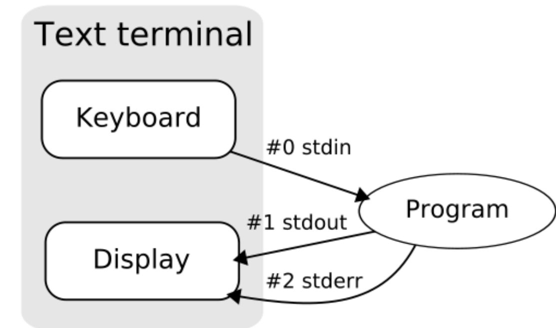
cat –n < .bashrc



Basics bash I/O (redirection)

STDOUT on display:

```
cat -n .bashrc
```



STDOUT on a file:

```
cat -n .bashrc 1> cat_output.txt
cat -n .bashrc > cat_output.txt
```

STDERR on display:

```
cat -n .bashrc ghost
```

STDERR on a file:

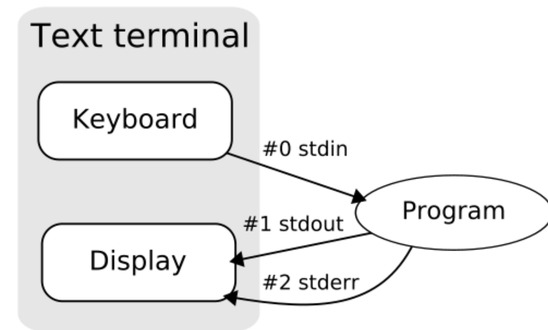
```
cat -n .bashrc ghost 2> cat_output.txt
```

Basics bash I/O (redirection)

what if

```
cat -n .bashrc ghost > cat_output.txt
```

?



STDERR to STDOUT on display:

```
cat -n .bashrc ghost 2>&1
```

STDERR to STDOUT on file:

```
(cat -n .bashrc ghost 2>&1) > cat_output.txt
cat -n .bashrc ghost >& cat_output.txt
```

“>>” is like “>” but it adds output to the file without overriding the old existing one

Executables

Any executables is run calling its name:

```
/home/mocchiut/test.exe
/usr/local/bin/mozilla
./my_print
```

Mmm, so why do we call "`ls`, `cat`, etc." and not "`/bin/ls`, `/bin/cat`, /bin/etc."?

bash uses ENVIRONMENTAL VARIABLES to make life easier

Bash environmental variables

If full path is not given, any executables is searched in the directories listed in the environmental variable called “PATH”.

how to print an env variable?

```
Emi@venere ~>echo $PATH
```

/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/local/sbin

directories are separated by colons “:”

how to add a directory?

```
export PATH=/home/mocchiut/bin:$PATH
```

Bash environmental variables

- Libraries contain code and data that provide services to independent programs. This encourages the sharing and changing of code and data in a modular fashion, and eases the distribution of the code and data. Library files are not executable programs.
- A shared library or shared object is a file that is intended to be shared by executable files and further shared objects files. Modules used by a program are loaded from individual shared objects into memory at load time load or run time.

Bash environmental variables

Shared libraries are searched at load time or run time in the directories listed in the environmental variable called “LD_LIBRARY_PATH”.

how to print an env variable?

```
Emi@venere ~>echo $LD_LIBRARY_PATH
```

directories are separated by colons “:”

how to add a directory?

```
export LD_LIBRARY_PATH=/home/mocchiut/lib:$LD_LIBRARY_PATH
```

Bash environmental variables

- The prompt

```
Emi@marte ~>
```

also can be changed with the env variable “PS1”

```
Emi@marte ~>echo $PS1
\[\e[28;1m\]Emi@\[\e[32;1m\]\h \[\e[35;1m\]\W\[\e[0m\]>
```

- **bash settings are read when you start the terminal form the file: $HOME/.bashrc**

http://www.cyberciti.biz/tips/howto-linux-unix-bash-shell-setup-prompt.html

http://www.cyberciti.biz/faq/bash-shell-change-the-color-of-my-shell-prompt-under-linux-or-unix

http://maketecheasier.com/8-useful-and-interesting-bash-prompts/2009/09/04

Bash scripting

- bash script: list of bash commands written in a text file usually having suffix ".sh"

- scripts can be run with:
  - `prompt>source script.sh`
  - `prompt>. script.sh`

  within current shell session

  - `prompt>./script.sh` (but first you must make script.sh executable) it opens a new session

Bash scripting

- minimal bash script:
- edit a new file, let's say test.sh and write:

```
#!/bin/bash
echo “This is test file!”
```

run it with in the three possible ways.

Bash scripting

- now change it to:

```
#!/bin/bash
cd /tmp
pwd
echo “This is test file!”
```

run it with in the three possible ways.

Bash scripting, env test

create $HOME/test1/bin/test.sh

```
#!/bin/bash
echo “This is test1 file!”
```

create $HOME/test2/bin/test.sh

```
#!/bin/bash
echo “This is test2 file!”
```

```
export PATH=$HOME/test1/bin:$PATH
prompt>test.sh
```

Bash scripting, wildcards

`?` any character (one and only one character)

```
prompt>ls .b?shrc
prompt>ls .b?shr?c
```

`*` any character, even none

```
prompt>ls .b*
prompt>ls .b*shrc
prompt>ls .b*rc
prompt>ls .b*r*c
```

Bash scripting, pipe

A pipe is "|" character. Pipes let you use the output of a program as the input of another one

```
|prompt>cat .bashrc | grep "interactive"
```

Bash, some useful commands

ctrl-a Move cursor to beginning of line

ctrl-e Move cursor to end of line

meta-b Move cursor back one word

meta-f Move cursor forward one word

ctrl-w Cut the last word

ctrl-u Cut everything before the cursor

ctrl-k Cut everything after the cursor

ctrl-y Paste the last thing to be cut

ctrl-_ Undo

ctrl-r Reverse search in the command history

NOTE: ctrl- = hold control, meta- = hold meta (where meta is usually the alt or escape key).

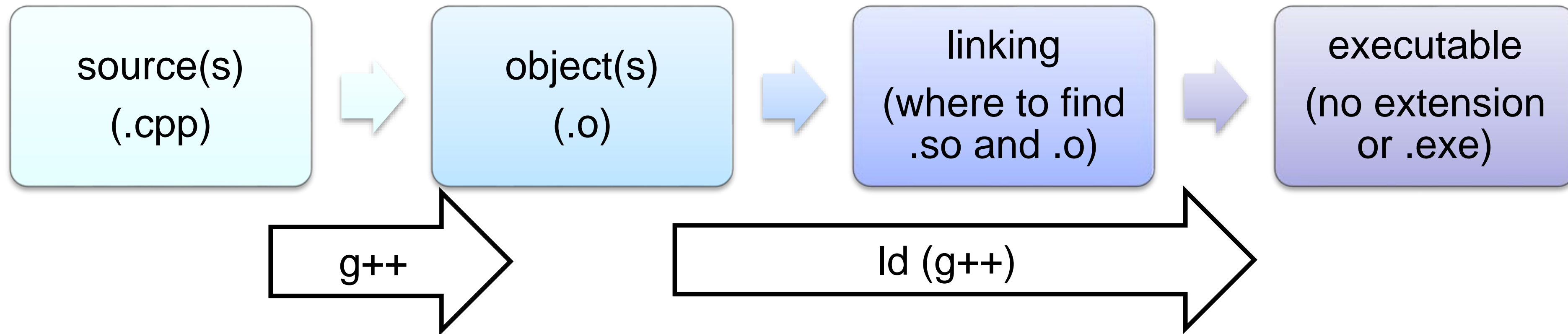
C++ history

- Bjarne Stroustrup developed C++ in the 80s at the AT&T Bell Labs (where Unix was created)
- C++ is (almost) a superset of C
- GNU GCC is the compiler package we are going to use
- in this course C++ and C differences will NOT be pointed out

C++ history

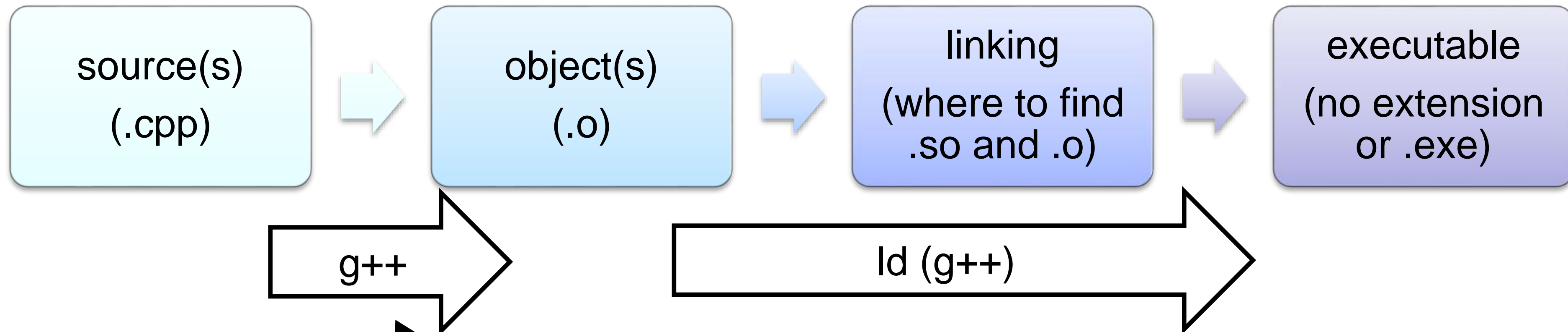
- C++ advantages and claims
  - old **code** can be easily **re-used**
  - **memory management** under C++ **easier** and more **transparent**
  - programs would be less bug-prone, as C++ uses a **stricter syntax** and **type checking**
  - “**data hiding**”, the usage of data by one program part while other program parts cannot access the data, would be easier to implement

Executables, compiling and running



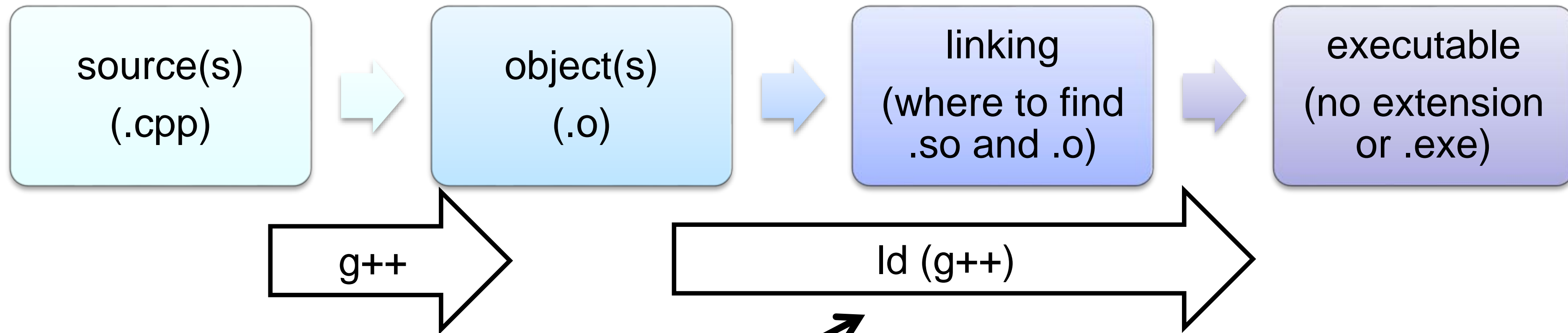

```
prompt> g++ -c hello.cpp
prompt> g++ -o hello hello.o
prompt> ls -lr hello*
-rwxr-xr-x 2 mocchiut users 5k 2012-10-12 11:15 hello.cpp
-rwxr-xr-x 2 mocchiut users 5k 2012-10-12 11:16 hello.o
-rwxr-xr-x 2 mocchiut users 5k 2012-10-12 11:17 hello
```

Executables, compiling and running



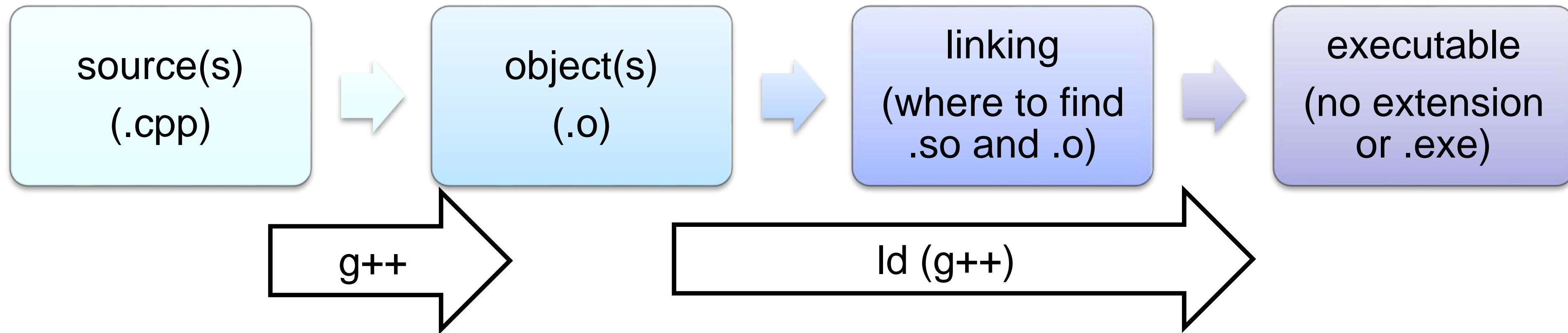

```
prompt> g++ -c hello.cpp
prompt> g++ -o hello hello.o
prompt> ls -lr hello*
-rwxr-xr-x 2 mocchiut users 5k 2012-10-12 11:15 hello.cpp
-rwxr-xr-x 2 mocchiut users 5k 2012-10-12 11:16 hello.o
-rwxr-xr-x 2 mocchiut users 5k 2012-10-12 11:17 hello
```

Executables, compiling and running




```
prompt> g++ -c hello.cpp
prompt> g++ -o hello hello.o
prompt> ls -lr hello*
-rwxr-xr-x 2 mocchiut users 5k 2012-10-12 11:15 hello.cpp
-rwxr-xr-x 2 mocchiut users 5k 2012-10-12 11:16 hello.o
-rwxr-xr-x 2 mocchiut users 5k 2012-10-12 11:17 hello
```

Executables, compiling and running




```
prompt> g++ -c hello.cpp
prompt> g++ -o hello hello.o
prompt> ls -lr hello*
-rwxr-xr-x 2 mocchiut users 5k 2012-10-12 11:15 hello.cpp
-rwxr-xr-x 2 mocchiut users 5k 2012-10-12 11:16 hello.o
-rwxr-xr-x 2 mocchiut users 5k 2012-10-12 11:17 hello
prompt> g++ -o hello hello.cpp
```

“Hello world” cpp program

Create with an editor a file called hello.cpp and write inside:

```
#include <iostream>
using namespace std;

int main(){
  cout << “Hello world! \n”;
  return 0;
}
```

Compile it with:

```
prompt> g++ hello.cpp -o hello
```

Execute the program with:

```
prompt> ./hello
```

"Hello world" cpp program

```
#include <iostream>
using namespace std;

int main(){
  cout << "Hello world! \n";
  return 0;
}
```

pre-processor instruction

“Hello world” cpp program

```
#include <iostream>
using namespace std;

int main(){
  cout << “Hello world! \n”;
  return 0;
}
```

“iostream” header file to be included

“Hello world” cpp program

```
#include <iostream>
using namespace std;

int main(){
  cout << “Hello world! \n”;
  return 0;
}
```

global namespace directive

“Hello world” cpp program

GLOBAL

```
#include <iostream>
using namespace std;

int main(){
  cout << “Hello world! \n”;
  return 0;
}
```

"Hello world" cpp program

```
#include <iostream>
using namespace std;
int main(){
  cout << "Hello world! \n";
  return 0;
}
```

function called "main" returning an integer

“Hello world” cpp program

```
#include <iostream>
using namespace std;

int main(){
   cout << “Hello world! \n”;
   return 0;
}
```

block: determine a scope

scope (skōpe), n. 1. the extent or range of one’s understanding. 2. the area of extent covered by something 3. opportunity or freedom for movement or activity

“Hello world” cpp program

```
#include <iostream>
using namespace std;

int main(){
  cout << “Hello world! \n”;
  return 0;
}
```

instruction, print on the STDOUT

“Hello world” cpp program

```
#include <iostream>
using namespace std;

int main(){
  cout << “Hello world! \n”;
  return 0;
}
```

return statement, return an integer (zero) to the prompt

“Hello world” cpp program

```
#include <iostream>
using namespace std;

int main(){
  cout << “Hello world! \n”;
  return 0;
}
```

ALL C++ lines must end with a semi-colon “;”
it is usually not needed after “{” and “}” – specific case a part (classes definition)

“Hello world” cpp program

```
// hello.cpp, print on STDOUT “Hello world!”
// Author: Emiliano Mocchiutti
// Email : Emiliano.Mocchiutti@ts.infn.it
// 2012/10/12 comments added
// 2012/10/10 hello.cpp created
#include <iostream>
using namespace std;

/* Print on STDOUT “Hello world!” using, as an
example, the iostream library */
int main(){
cout << “Hello world! \n”; // here we want to see
// a STDOUT
return 0; // no errors, exit with zero
}
```

"Hello world" cpp program

```
// hello.cpp, print on STDOUT "Hello world!"
// Author: Emiliano Mocchiutti
// Email : Emiliano.Mocchiutti@ts.infn.it
// 2012/10/12 comments added
// 2012/10/10 hello.cpp created
#include <iostream>
using namespace std;

/* Print on STDOUT "Hello world!" using, as an
   example, the iostream library */
int main(){
  cout << "Hello world! \n"; // here we want to see
                              // a STDOUT
  return 0; // no errors, exit with zero
}
```

“Hello world” cpp program

```
// hello.cpp, print on STDOUT "Hello world!"
// Author: Emiliano Mocchiutti
// Email : Emiliano.Mocchiutti@ts.infn.it
// 2012/10/12 comments added
// 2012/10/10 hello.cpp created
#include <iostream>
using namespace std;

/* Print on STDOUT "Hello world!" using, as an
   example, the iostream library */
int main(){
  cout << "Hello world! \n"; // here we want to see
                             // a STDOUT
  return 0; // no errors, exit with zero
}
```

WHAT

HOW

WHY

Proper use of comments - *what*

- Typically, comments should be used for three things. At the library, program, or function level, comments should be used to describe *what* the library, program, or function, does. For example:

```
// This program calculate the student's final grade
// based on his test and homework scores.
```

```
// This function uses newton's method to
// approximate the root of a given equation.
```

```
// The following lines generate a random item based
// on rarity, level, and a weight factor.
```

All of these comments give the reader a good idea of what the program is trying to accomplish without having to look at the actual code. The user (possibly someone else, or you if you're trying to reuse code you've already written in the future) can tell at a glance whether the code is relevant to what he or she is trying to accomplish. This is **particularly important when working as part of a team**, where not everybody will be familiar with all of the code.

Proper use of comments - *how*

- Second, within the library, program, or function described above, comments should be used to describe *how* the code is going to accomplish it's goal.

```
/* To calculate the final grade, we sum all the weighted
   midterm and homework scores and then divide by the number of
   scores to assign a percentage. This percentage is used to
   calculate a letter grade. */


// To generate a random item, we're going to do the following:
// 1) Put all of the items of the desired rarity on a list
// 2) Calculate a probability for each item based on level and
//    weight factor
// 3) Choose a random number
// 4) Figure out which item that random number corresponds to
// 5) Return the appropriate item
```

These comments give the user an idea of how the code is going to accomplish it's goal without going into too much detail.

Proper use of comments - *why*

- At the statement level, comments should be used to describe *why* the code is doing something. A bad statement comment explains *what* the code is doing. If you ever write code that is so complex that needs a comment to explain *what* a statement is doing, you probably need to rewrite your code, not comment it.

Bad comment:

```
// Set sight range to 0
sight = 0;
```

(yes, we already can see that sight is being set to 0 by looking at the statement)

Good comment:

```
// The player just drank a potion of blindness and
   can not see anything
sight = 0;
```

(now we know WHY the player's sight is being set to 0)

Proper use of comments

Bad comment:

```
// Calculate the cost of the items
cost = items / 2 * storePrice;
```

(yes, we can see that this is a cost calculation, but why is items divided by 2?)

Good comment:

```
// We need to divide items by 2 here because
  they are bought in pairs
cost = items / 2 * storePrice;
```

(now we know!)

Proper use of comments

- Programmers often have to make a tough decision between solving a problem one way, or solving it another way. Comments are a great way to remind yourself (or tell somebody else) the reason you made a one decision instead of another.

Good comments:

```
// We decided to use a linked list instead of an array because
// arrays do insertion too slowly.

// We're going to use newton's method to find the root of a
// number because there is no deterministic way to solve these
// equations.
```

Proper use of comments

- Finally, comment should be written in a way that makes sense to someone who has no idea what the code does. It is often the case that a programmer will say "It's obvious what this does! There's no way I'll forget about this". Guess what? It's not obvious, and you will be amazed how quickly you forget. :) You (or someone else) will thank you later for writing down the what, how, and why of your code in human language. Reading individual lines of code is easy. Understanding what goal they are meant to accomplish is not.

http://www.learncpp.com/cpp-tutorial/12-comments/

To summarize:

- **At the library, program, or function level, describe *what***
- **Inside the library, program, or function, describe *how***
- **At the statement level, describe *why*.**

Programming style

http://geosoft.no/development/cppstyle.html

- Variable names must be in mixed case starting with lower case:

  ```
  line, savingsAccount
  ```

  Common practice in the C++ development community. Makes variables easy to distinguish from types, and effectively resolves potential naming collision as in the declaration Line line.

- Names representing methods or functions must be verbs and written in mixed case starting with lower case.

  ```
  getName(), computeTotalWidth()
  ```

  Common practice in the C++ development community. This is identical to variable names, but functions in C++ are already distinguishable from variables by their specific form.

Programming style

- Abbreviations and acronyms must not be uppercase when used as name:

```
exportHtmlSource(); // NOT: exportHTMLSource();
openDvdPlayer(); // NOT: openDVDPlayer();
```

Using all uppercase for the base name will give conflicts with the naming conventions given above. A variable of this type would have to be named dVD, hTML etc. which obviously is not very readable. Another problem is illustrated in the examples above: when the name is connected to another, the readability is seriously reduced; the word following the abbreviation does not stand out as it should.

Programming style

- Private class variables and methods should have underscore prefix.

```
class SomeClass {
  private:
  int _length;
}
```

Apart from its name and its type, the scope of a variable is its most important feature. Indicating class scope by using underscore makes it easy to distinguish class variables from local scratch variables. This is important because class variables are considered to have higher significance than method variables, and should be treated with special care by the programmer. A side effect of the underscore naming convention is that it nicely resolves the problem of finding reasonable variable names for setter methods and constructors:

```
void setDepth (int depth){ _depth = depth; }
```

An issue is whether the underscore should be added as a prefix or as a suffix. Both practices are commonly used (`_depth` and `depth_`).

Programming style

- All names should be written in English.

```
fileName; // NOT: filNamn
```

English is the preferred language for international development.

- Variables with a large scope can have long names, variables with a small scope should have short names.

Scratch variables used for temporary storage or indices are best kept short. A programmer reading such variables should be able to assume that its value is not used outside of a few lines of code. Common scratch variables for integers are *i, j, k, m, n* and for characters *c* and *d.*

- The name of the object is implicit, and should be avoided in a method name.

```
line.getLength(); // NOT: line.getLineLength();
```

The latter seems natural in the class declaration, but proves superfluous in use, as shown in the example.

Programming style

- Plural form should be used on names representing a collection of objects:

```
vector<Point> points;
int values[];
```

  Enhances readability since the name gives the user an immediate clue of the type of the variable and the operations that can be performed on its elements.

- The prefix *n* should be used for variables representing a number of objects*:*

```
nPoints, nLines
```

  The notation is taken from mathematics where it is an established convention for indicating a number of objects

Programming style

- The prefix *is* should be used for boolean variables and methods:

  `isSet, isVisible, isFinished, isFound, isOpen`

  Common practice in the C++ development community and partially enforced in Java. Using the *is* prefix solves a common problem of choosing bad boolean names like status or flag. isStatus or isFlag simply doesn't fit, and the programmer is forced to choose more meaningful names. There are a few alternatives to the *is* prefix that fit better in some situations. These are the *has, can* and *should.*

- Functions (methods returning something) should be named after what they return and procedures (*void* methods) after what they do.

  Increase readability. Makes it clear what the unit should do and especially all the things it is not supposed to do. This again makes it easier to keep the code clean of side effects.

Programming style

- The parts of a class must be sorted *public, protected and private*. All sections must be identified explicitly. Not applicable sections should be left out.

  The ordering is *"most public first" so people who only wish to use the class can stop reading when they reach the* protected/private sections.
- Type conversions must always be done explicitly. Never rely on implicit type conversion:

```
floatValue = static_cast<float>(intValue);
                              // NOT: floatValue = intValue;
```

  By this, the programmer indicates that he is aware of the different types involved and that the mix is intentional.

Programming style

- variable names can contain numbers, letters and underscores "_"
- variable names can NOT start with numbers
- variable names can NOT be any of the reserved words, the list of all the C++ reserved keywords is:

asm auto bool break case catch char class const const_cast continue default delete do double dynamic_cast else enum explicit export extern false float for friend goto if inline int long mutable namespace new operator private protected public register reinterpret_cast return short signed sizeof static static_cast struct switch template this throw true try typedef typeid typename union unsigned using virtual void volatile wchar_t while

2017/03/03 – take home message

- feel free to contact me if you need help!
- we want to be programmers not only apps users…
- choose a “good” text editor for programming
- learn some basic bash commands and I/O redirection

2017/03/03 – exercise

Learning bash:

- create a new directory (mkdir)
- enter the directory (cd)
- use a text editor (vi, nano, emacs…) to create a script that will print out on the STDOUT a sentence (any)
- run the script and redirect the STDOUT to a file
- run again and redirect to a file with a different name
- change the filenames adding the string "_test.txt" (pippo.txt -> pippo.txt_text.txt , …) using a loop in a single line command (bash commands: for, do, done, mv, basename…)