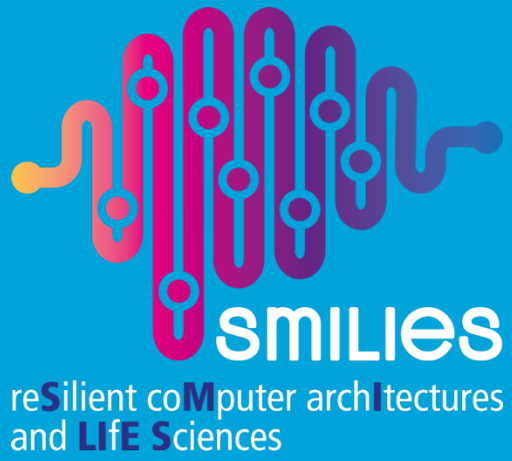




Politecnico di Torino

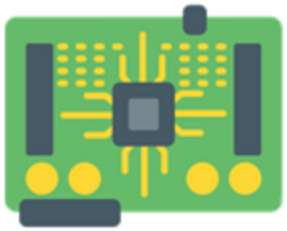
Department of Control and Computer Engineering

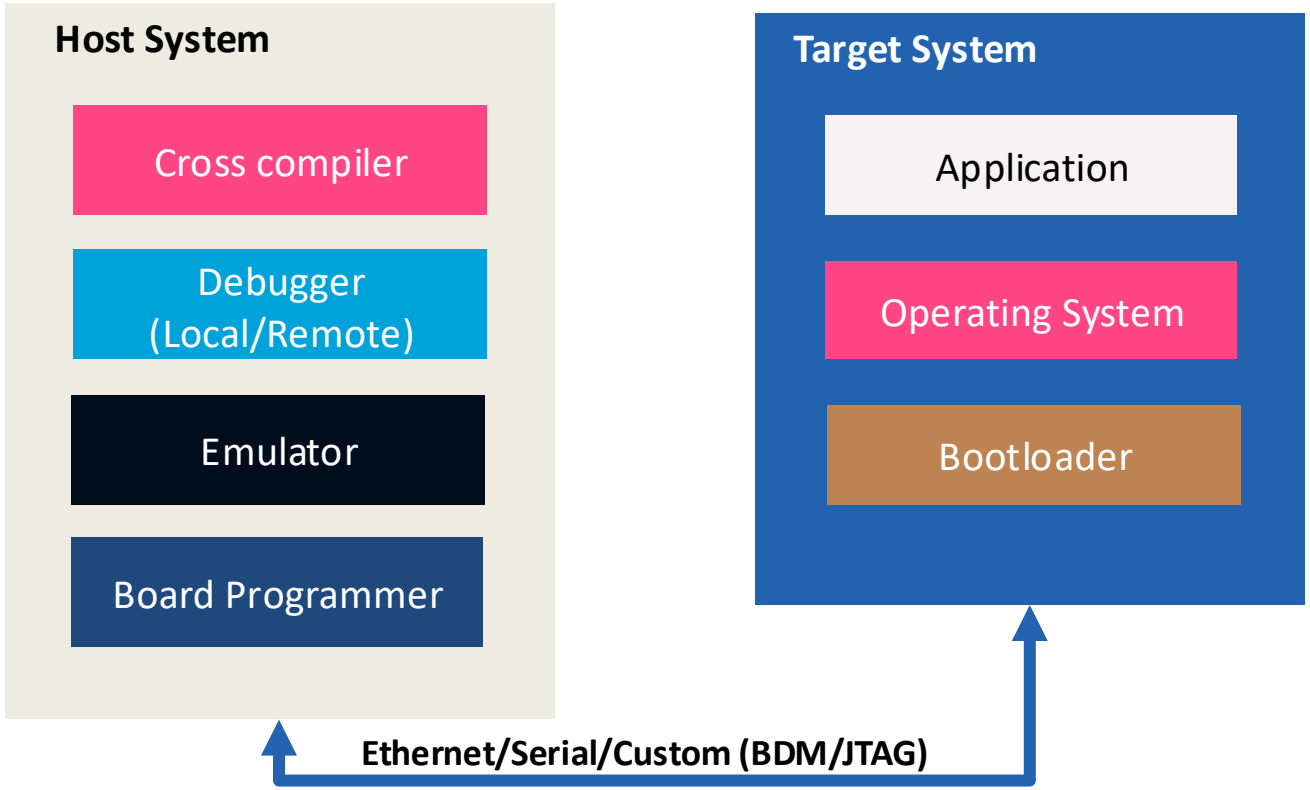

# CROSS PLATFORM DEVELOPMENT

STEFANO DI CARLO

# WHAT IS CROSS DEVELOPMENT

- Cross development is the process of developing code on one machine – **the host**, to run on **another machine** – the target
- The host is a **normal, powerful machine** running an operating system
- The target is often a **single board** computer that may have no or limited software and hardware resources





# FEATURES OF CROSS DEVELOPMENT

- The code created can’t run on the host system
  - Sometimes an emulator is used
- Special tools are required
  - The standard PC compiler won’t do!
- The code must be moved from the host to the board

# WHY CROSS DEVELOP?

- Code is cross developed for several reasons
  - Frequently the development can’t be done on the board
    - The board has no disc, compiler, screen etc
  - The development environment is very powerful and fast
    - Games developers do this
  - Often the development might be done by a team on networked machines

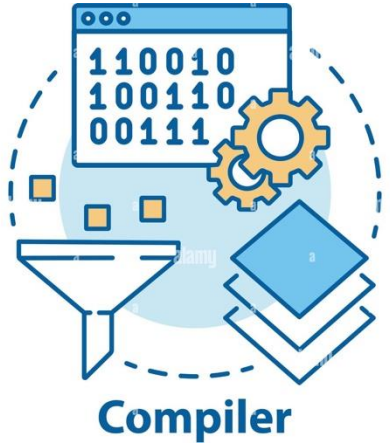
# CROSS DEVELOPMENT TOOLS

- To do cross development you need special tool kits, sometimes called **toolchains**. These consist of
  - Cross compiler, assembler and linkers
  - Programming and conversion software
  - Remote debuggers
  - Useful utilities
    - Files to dump or strip binary files, all in binutils.

# COMPILER

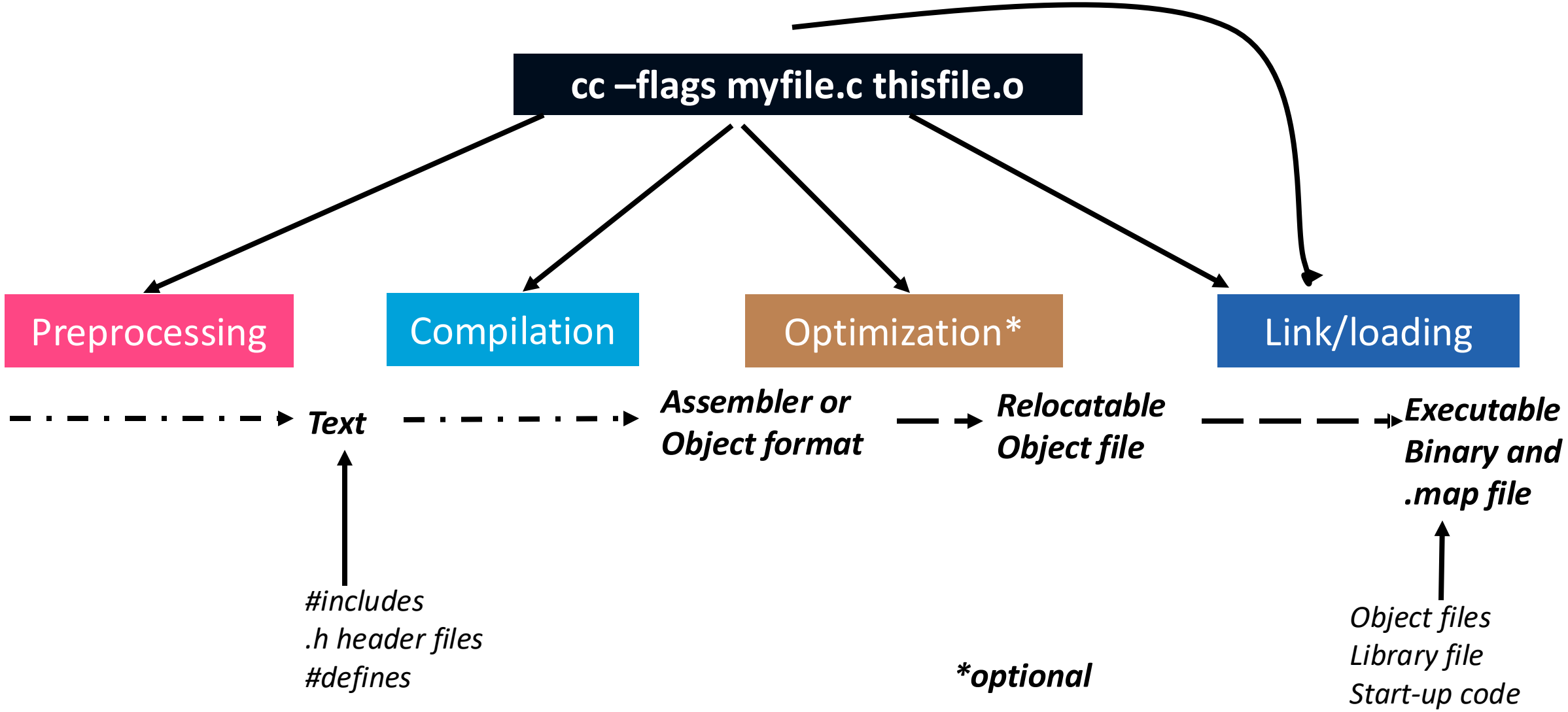
- Before we understand what cross compilers do it might be worth reviewing what compilers do
  - They take an input text source program file and output an executable binary output file (or some error messages!)

Source Program file




Executable Binary File

# THE COMPILATION PROCESS IN DETAIL



# EXECUTABLE FORMATS

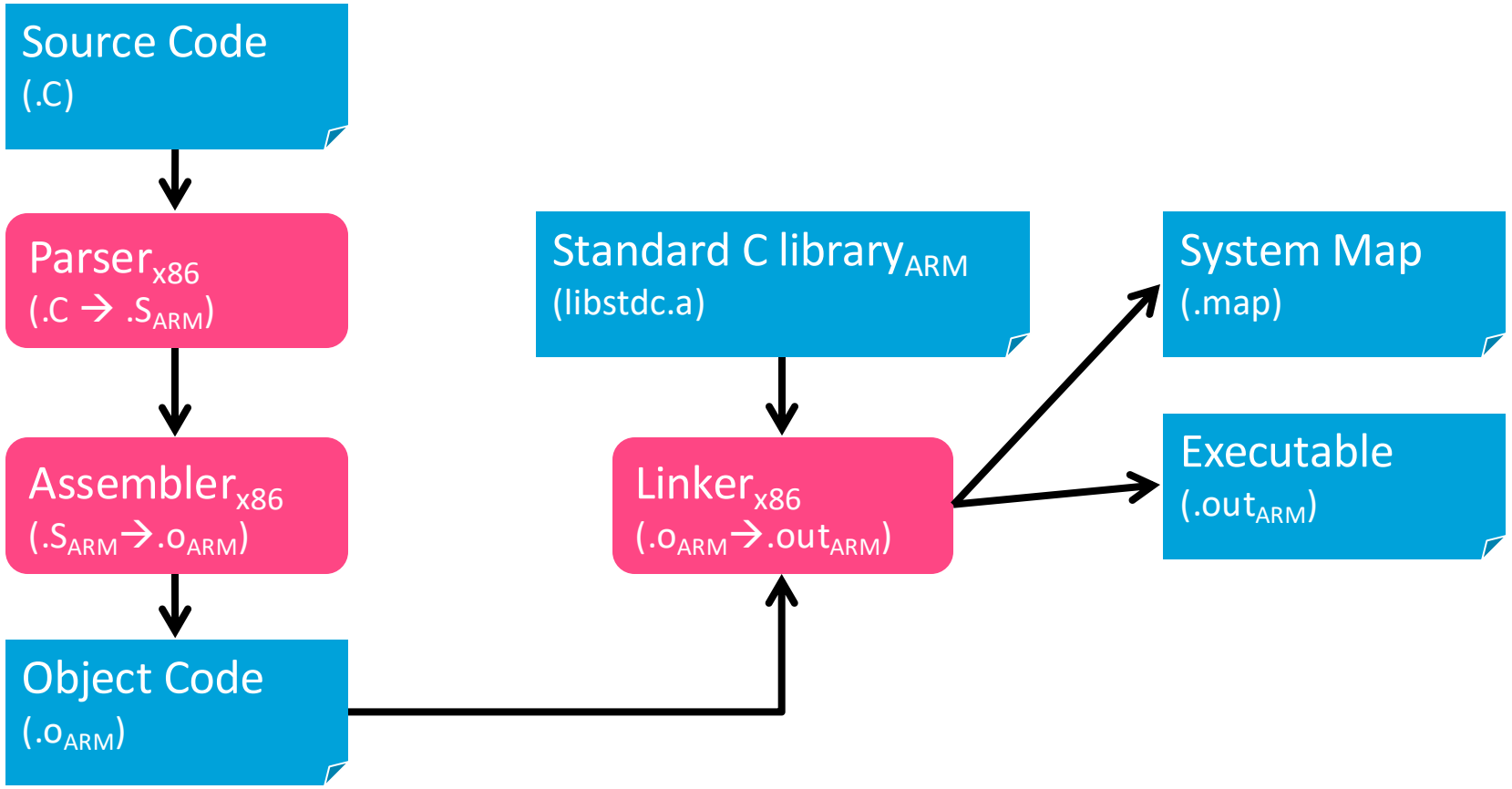
- There are a number of different exectuable formats
  - A.out is the traditional UNIX format
  - **Elf – Executable and  Linking Format**
  - COFF – Common Object File Format
  - Plus lots of others

# MAP FILE

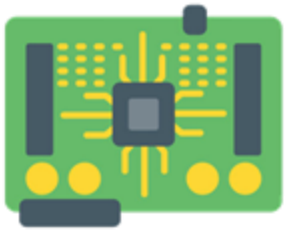
- A **.map** file is a detailed report generated by the linker during the build process of a program.
- It provides valuable information about the memory layout and symbol addresses in the final executable
  - Memory layout (see later)
  - Symbol Table (the addresses of all global variables, functions, and objects)
  - Section Mapping (see later)
  - Stack and Heap Usage (if available)
  - Function Sizes: the size of each function and its location in memory
- This file is useful for debugging, optimizing memory usage, and understanding how the linker has allocated sections of code and data in memory.

# CROSS COMPILER

- Target system (e.g., ARM) != Host system (e.g., x86)



# WHAT IS CROSS DEVELOPMENT



**Host System**

Source Code (.C)

Cross compiler

Executable (.out)

Debugger

Board Programmer

Target system

RAM Memory (DRAM/SRAM)

CPU

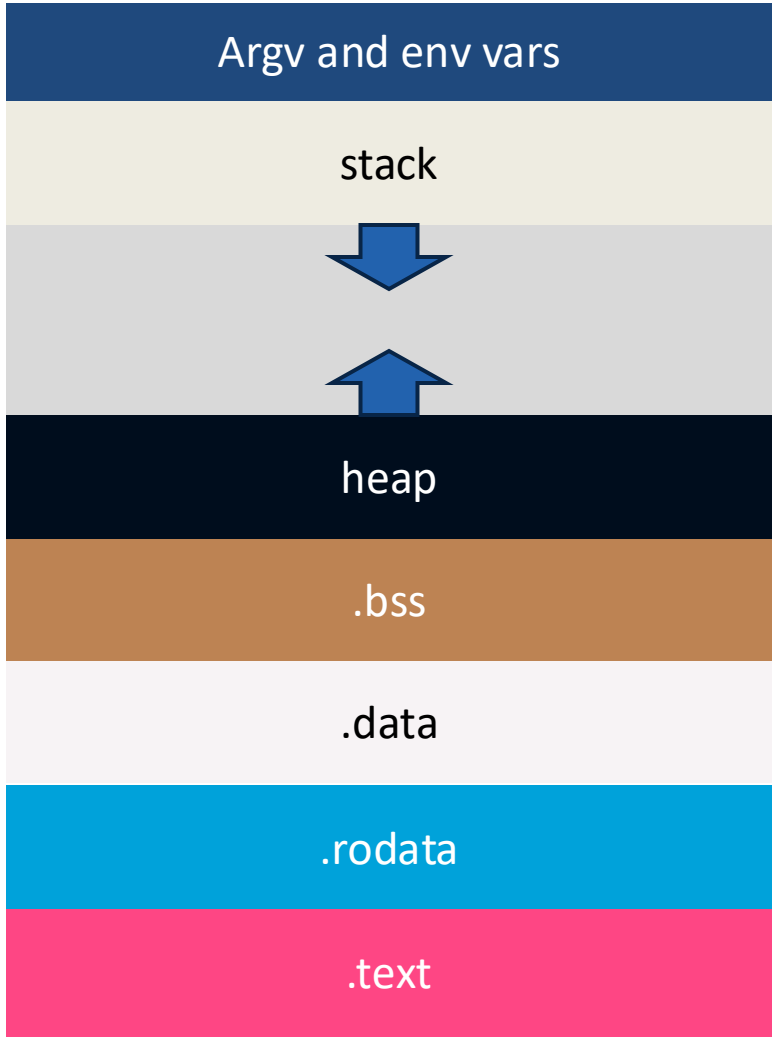
I/O

Firmware Flash (NOR)

Mass Memory Flash (NAND)

**Ethernet/Serial/Custom (BDM/JTAG)**

# PROGRAMS IN MEMORY




| Memory Area | Section Name | Section Type | Write Operation | Initial Value | Contents |
|---|---|---|---|---|---|
| Program | .text | Code | No | Yes | Stores machine code |
| Constants | .rodata | Data | Yes | Yes | Constant data. This section may not be produced. |
| Initialized data | .data | Data | Yes | Yes | Initialized global and static data |
| Uninitialized data | .bss | Data | Yes | No | Global data whose initial value is not specified (zero initialized). BSS stands for Block Started by Symbol |
| Heap | -- | -- | Yes | No | Dynamic area allocation used by library functions (malloc, realloc, ...) |
| Stack | -- | -- | Yes | No | Dynamic area required for program execution |
| Command line | -- | Data | Yes | Yes | Area where command line arguments are stored. It is the initial part of the stack |

# PROGRAMS IN MEMORY

```
const int alfa=25;
int beta = 44;
char tmp;

int foo(int X)
{
  char *ptr;
  ptr = (char *)malloc(X);
}

int main(int argc, char ** argv)
{

   int a = foo (3);
}
```

| Memory Area | Section Name | Section Type | Write Operation | Initial Value | Contents |
|---|---|---|---|---|---|
| Program | .text | Code | No | Yes | Stores machine code |
| Constants | .rodata | Data | Yes | Yes | Constant data. This section may not be produced. |
| Initialized data | .data | Data | Yes | Yes | Initialized global and static data |
| Uninitialized data | .bss | Data | Yes | No | Global data whose initial value is not specified (zero initialized). BSS stands for Block Started by Symbol |
| Heap | -- | -- | Yes | No | Dynamic area allocation used by library functions (malloc, realloc, ...) |
| Stack | -- | -- | Yes | No | Dynamic area required for program execution |
| Command line | -- | Data | Yes | Yes | Area where command line arguments are stored. It is the initial part of the stack |

# PROGRAMS IN MEMORY

```
const int alfa=25;
int beta = 44;
char tmp;

int foo(int X)
{
  char *ptr;
  ptr = (char *)malloc(X);
}

int main(int argc, char ** argv)
{

   int a = foo (3);
}
```

| Memory Area | Section Name | Section Type | Write Operation | Initial Value | Contents |
|---|---|---|---|---|---|
| Program | .text | Code | No | Yes | Stores machine code |
| Constants | .rodata | Data | Yes | Yes | Constant data. This section may not be produced. |
| Initialized data | .data | Data | Yes | Yes | Initialized global and static data |
| Uninitialized data | .bss | Data | Yes | No | Global data whose initial value is not specified (zero initialized). BSS stands for Block Started by Symbol |
| Heap | -- | -- | Yes | No | Dynamic area allocation used by library functions (malloc, realloc, ...) |
| Stack | -- | -- | Yes | No | Dynamic area required for program execution |
| Command line | -- | Data | Yes | Yes | Area where command line arguments are stored. It is the initial part of the stack |

# PROGRAMS IN MEMORY

```
const int alfa=25;
int beta = 44;
char tmp;

int foo(int X)
{
  char *ptr;
  ptr = (char *)malloc(X);
}

int main(int argc, char ** argv)
{

   int a = foo (3);
}
```

| Memory Area | Section Name | Section Type | Write Operation | Initial Value | Contents |
|---|---|---|---|---|---|
| Program | .text | Code | No | Yes | Stores machine code |
| Constants | .rodata | Data | Yes | Yes | Constant data. This section may not be produced. |
| Initialized data | .data | Data | Yes | Yes | Initialized global and static data |
| Uninitialized data | .bss | Data | Yes | No | Global data whose initial value is not specified (zero initialized). BSS stands for Block Started by Symbol |
| Heap | -- | -- | Yes | No | Dynamic area allocation used by library functions (malloc, realloc, ...) |
| Stack | -- | -- | Yes | No | Dynamic area required for program execution |
| Command line | -- | Data | Yes | Yes | Area where command line arguments are stored. It is the initial part of the stack |

# PROGRAMS IN MEMORY

```
const int alfa=25;
int beta = 44;
char tmp;

int foo(int X)
{
  char *ptr;
  ptr = (char *)malloc(X);
}

int main(int argc, char ** argv)
{

   int a = foo (3);
}
```

| Memory Area | Section Name | Section Type | Write Operation | Initial Value | Contents |
|---|---|---|---|---|---|
| Program | .text | Code | No | Yes | Stores machine code |
| Constants | .rodata | Data | Yes | Yes | Constant data. This section may not be produced. |
| Initialized data | .data | Data | Yes | Yes | Initialized global and static data |
| Uninitialized data | .bss | Data | Yes | No | Global data whose initial value is not specified (zero initialized). BSS stands for Block Started by Symbol |
| Heap | -- | -- | Yes | No | Dynamic area allocation used by library functions (malloc, realloc, ...) |
| Stack | -- | -- | Yes | No | Dynamic area required for program execution |
| Command line | -- | Data | Yes | Yes | Area where command line arguments are stored. It is the initial part of the stack |

# PROGRAMS IN MEMORY

```
const int alfa=25;
int beta = 44;
char tmp;

int foo(int X)
{
  char *ptr;
  ptr = (char *)malloc(X);
}

int main(int argc, char ** argv)
{

   int a = foo (3);
}
```

| Memory Area | Section Name | Section Type | Write Operation | Initial Value | Contents |
|---|---|---|---|---|---|
| Program | .text | Code | No | Yes | Stores machine code |
| Constants | .rodata | Data | Yes | Yes | Constant data. This section may not be produced. |
| Initialized data | .data | Data | Yes | Yes | Initialized global and static data |
| Uninitialized data | .bss | Data | Yes | No | Global data whose initial value is not specified (zero initialized). BSS stands for Block Started by Symbol |
| Heap | -- | -- | Yes | No | Dynamic area allocation used by library functions (malloc, realloc, ...) |
| Stack | -- | -- | Yes | No | Dynamic area required for program execution |
| Command line | -- | Data | Yes | Yes | Area where command line arguments are stored. It is the initial part of the stack |

# PROGRAMS IN MEMORY

```
const int alfa=25;
int beta = 44;
char tmp;

int foo(int X)
{
  char *ptr;
  ptr = (char *)malloc(X);
}

int main(int argc, char ** argv)
{

   int a = foo (3);
}
```

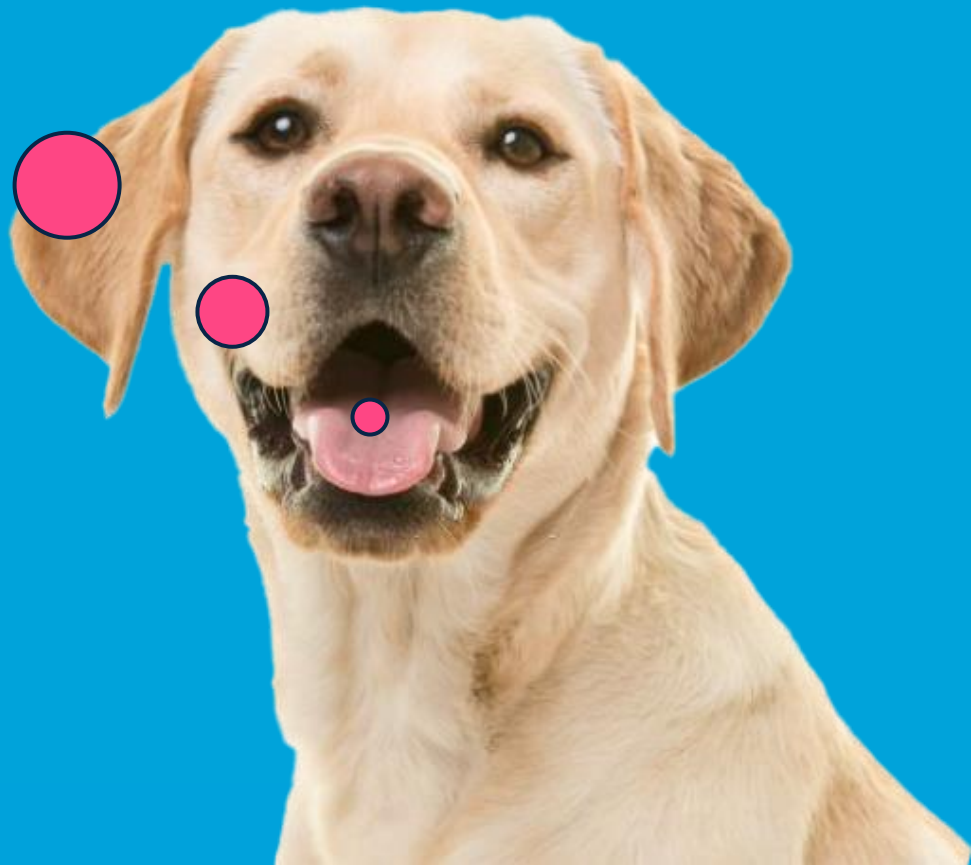
| Memory Area | Section Name | Section Type | Write Operation | Initial Value | Contents |
|---|---|---|---|---|---|
| Program | .text | Code | No | Yes | Stores machine code |
| Constants | .rodata | Data | Yes | Yes | Constant data. This section may not be produced. |
| Initialized data | .data | Data | Yes | Yes | Initialized global and static data |
| Uninitialized data | .bss | Data | Yes | No | Global data whose initial value is not specified (zero initialized). BSS stands for Block Started by Symbol |
| Heap | -- | -- | Yes | No | Dynamic area allocation used by library functions (malloc, realloc, ...) |
| Stack | -- | -- | Yes | No | Dynamic area required for program execution |
| Command line | -- | Data | Yes | Yes | Area where command line arguments are stored. It is the initial part of the stack |

SMILIES
reSilient coMputer archItectures and LIfE Sciences

Politecnico di Torino
Department of Control and Computer Engineering

DAUIN

QUESTIONS?



# THANK YOU!