MINESnancy
ARTEM

Introduction to C++ (and C)

Cédric Zanni

MINESnancy ARTEM

Operating mode

- **Ask questions**
- **Feedback requested !**
  - Speed/Difficulty/Interest (both lectures and practical works)
- **Between lectures**
  - Try to finish/advance exercices
  - CppIsoGuidelines or blog post to read
- **Come back with questions if something is not crystal clear !!!**

MINESnancy
ARTEM

First, a question…

- **Why did you choose to follow a lecture on C++ ?**

MINESnancy ARTEM

Lecture Plan

- **Why C++ (and C) and a bit of history**
- **What type of programming language is C++ (and C)**
- **A small word on computer architecture**
- **Base syntax (C and C++)**
  - Types, variables
  - Control flow
  - Functions
  - Arrays

MINESnancy ARTEM

C strong points

- Open normalized standard know by *¿most?* developers
- Optimization and control of memory allocation
- Limited number of concepts to understand (*¿easy?* handling)
- Relatively close to machine code (no hidden cost)
- Compile directly to machine code

zero overhead principle

MINESnancy ARTEM

C weak points

- **No exception management**
- **No object oriented programming**
- **No special support for memory allocation** (no garbage collector, ...)
- **A lot of bugs if we are not careful**

limited abstractions !

MINESnancy ARTEM

C weak points

- **No exception management**
- **No object oriented programming**
- **No special support for memory allocation** (no garbage collector, ...)
- **A lot of bugs if we are not careful**

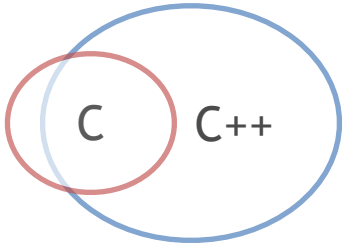
limited abstractions !

Comment : Reason why we will focus mostly on C++

C weak points

- **No exception management**
- **No object oriented programming**
- **No special support for memory allocation** (no garbage collector, ...)
- **A lot of bugs if we are not careful**

limited abstractions !



Comment : Reason why we will focus mostly on C++

Comment : C++ is **nearly** a superset of C

MINESnancy ARTEM

C++ a well criticized language

- **Wikipedia page : Criticism of C++**
  - Large number of listed criticism concern old compilers and c++ version
- **Too hard and too complicated**
- **But …**

C++ a well criticized language

- **Wikipedia page : Criticism of C++**
  - Large number of listed criticism concern old compilers and c++ version
- **Too hard and too complicated**
- **But …**

Brian Kernighan : « C++ has been enormously influential. … Lots of people say C++ is too big and too complicated etc. etc. but in fact it is a very powerful language and pretty much everything that is in there is there for a really sound reason: it is not somebody doing random invention, it is actually people trying to solve real world problems. Now a lot of the programs that we take for granted today, that we just use, are C++ programs. » consideration …

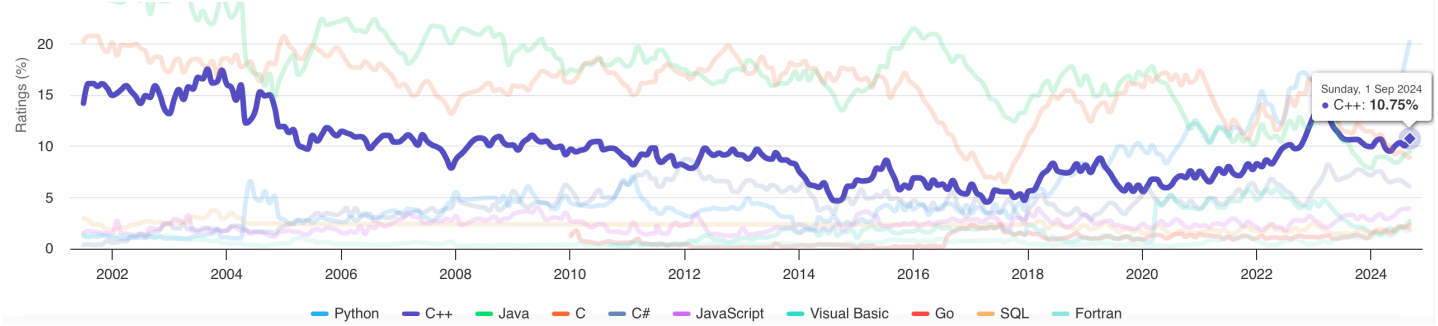
Why C++ when Rust is rising ?

- **Rust is currently safer, but you still have to use *Unsafe Rust* if**
  - you want interoperability with C
  - you need some optimization (advanced data structures, …)

- **What about Carbon** (google), **Hylo** (open-source), cppfont, circle, …

- **Large existing code base are still written in C++**
- **C++ is the main language for GPGPU programming**
- **It's interesting to see how to make a language safer (C -> C++)**
- **C++ is still evolving and moving toward safety profile**

TIOBE index



Learn both depending on your target domain/job !

MINESnancy
ARTEM

Where is C++ used ?

MINESnancy
ARTEM

Where is C++ used ?

AIRBUS SPACEX TESLA

BOEING Mercedes-Benz

NASA Embedded system

BostonDynamics

MINESnancy
ARTEM

Where is C++ used ?

AIRBUS SPACEX TESLA

BOEING Mercedes-Benz

NASA Embedded system

Boston Dynamics

Alphabet

Meta

Office

MINESnancy
ARTEM

Where is C++ used ?

AIRBUS SPACEX TESLA

BOEING Mercedes-Benz

NASA Boston Dynamics

Embedded system

Alphabet

Meta

Office

chrome Firefox

MINESnancy ARTEM

Where is C++ used ?

Embedded system

System programming

Where is C++ used ?

AIRBUS SPACEX TESLA

BOEING Mercedes-Benz

NASA Boston Dynamics

Embedded system

Alphabet

Meta

Office

chrome Firefox

Microsoft

macOS

System programming

Java JVM (Java Virtual Machine)

Parallels

Virtual machine

MINES nancy
ARTEM

Where is C++ used ?

AIRBUS SPACEX TESLA

BOEING Mercedes-Benz

NASA Boston Dynamics

Embedded system

Alphabet

Meta

Office

chrome Firefox

Microsoft

macOS

System programming

Java JVM (Java Virtual Machine)

Parallels

Virtual machine

JPMorgan Chase & Co. Banking

Where is C++ used ?

Embedded system

Databases

System programming

Virtual machine

Banking

Where is C++ used ?

Embedded system

Maths and simulation

Databases

System programming

Virtual machine

Banking

Where is C++ used ?

Image, Geometry, Video editing, CAD, ...

Embedded system

Maths and simulation

Databases

System programming

Virtual machine

Banking

MINESnancy ARTEM

Where is C++ used ?

Unity

UNREAL ENGINE

GODOT Game engine

Video game engine

AIRBUS SPACEX TESLA

DASSAULT SYSTEMES

OpenCV

nTop

Adobe

CGAL

BOEING

Mercedes-Benz

Image, Geometry, Video editing, CAD, …

NASA

Boston Dynamics

Embedded system

MATLAB

Ansys

Wolfram Mathematica

Maths and simulation

Alphabet

Meta

Office

MySQL

mongoDB

Databases

chrome

Firefox

Microsoft

macOS

System programming

Java

JVM (Java Virtual Machine)

Parallels

Virtual machine

JPMorgan Chase & Co.

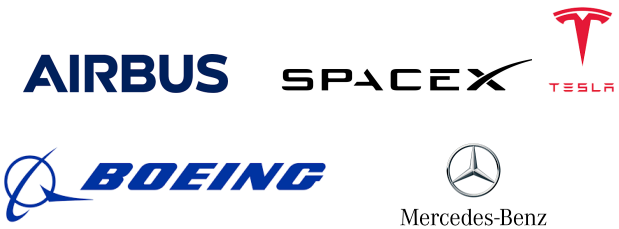
Banking

UNIVERSITÉ DE LORRAINE | Institut Mines-Télécom | ARTEM

MINESnancy ARTEM

Where is C++ used ?

Video game engine



Image, Geometry, Video editing, CAD, ...

Embedded system

Maths and simulation

GPGPU

Databases

System programming

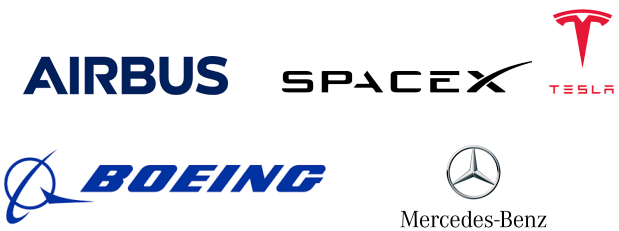
Virtual machine

Banking

MINESnancy ARTEM

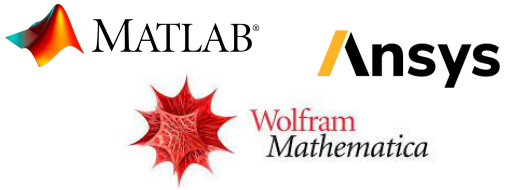
Where is C++ used ?

Video game engine



Embedded system

Image, Geometry, Video editing, CAD, ...



Maths and simulation

GPGPU

Databases

Machine learning

System programming

Virtual machine

Banking

MINESnancy ARTEM

Examples

pybind11

C++ library (binding)

Desktop C++ app

Google Earth

TensorFlow

C++ library

Ported to web using Emscripten
(Compiled to WASM)

emscripten

emscripten

WA

WA

Google V8, JavaScript and WebAssembly engine written in C++

IceSL
modeler and slicer

Desktop C++ app

Examples : C and C++ in Python

- **CPython (Python reference implementation)**
  - is written primarily in C
- **Python standard library**
  - contains many modules implemented in C
- **NumPy, Pandas, SciPy**
  - are mostly written in C with a few C++
- **TensorFlow, PyTorch, OpenCV, VTK, PyQt…**
  - are mostly written in C++

Examples : C and C++ in Python

- **CPython (Python reference implementation)**
  - is written primarily in C
- **Python standard library**
  - contains many modules implemented in C
- **NumPy, Pandas, SciPy**
  - are mostly written in C with a few C++
- **TensorFlow, PyTorch, OpenCV, VTK, PyQt…**
  - are mostly written in C++

Comment : for performance consideration …

Examples : C and C++ in Python

- **CPython (Python reference implementation)**
  - is written primarily in C
- **Python standard library**
  - contains many modules implemented in C
- **NumPy, Pandas, SciPy**
  - are mostly written in C with a few C++
- **TensorFlow, PyTorch, OpenCV, VTK, PyQt…**
  - are mostly written in C++

Comment : for performance consideration …

Comment : … can also find python library mostly written in Rust

MINESnancy ARTEM

Where is C used ?

- **System programming**
  - Operating system (OS) such as Linux
  - Microcontrollers: transports (planes, cars, …)
  - Digital signal processor (DSP) : remote controller
  - Embedded processors (portable electronics)
  - …

MINESnancy ARTEM

Where is C used ?

- **System programming**
  - Operating system (OS) such as Linux
  - Microcontrollers: transports (planes, cars, ...)
  - Digital signal processor (DSP) : remote controller
  - Embedded processors (portable electronics)
  - ...
- **In python...**
  - Reference Python interpreter
  - Large part of python standard library
  - Possibility to extend python using C

MINESnancy ARTEM

A bit of history

- **Evolution of C and C++ over time**
  - 1972 : invention of C by Dennis Ritchie
  - 1978 : The C Programming Language (stabilisation of the language)
  - 1979 : C with Classes (first implementation)
  - 1985 : The C++ Programming Language, 1st edition
  - 1989 : first C norm(C89) by American National Standard Institute
  - 1991 : ISO C++ Committee founded
  - 1998 : first C++ norm (minor update in 2003)
  - 1999 : C99 one of the most commonly used C norm
  - **C11, 18, *23 (underway)***
  - **C++11, 14, 17, 20, 23, *26 (underway)* -> « Modern » C++**

MINESnancy
ARTEM

Object-oriented programming and modeling

Key concept

Object-oriented programming and modeling

Key concept

- **Encapsulation**
  - hide implementation details
  - guarantee a class invariant
- **Aggregation**
  - define composite objects from several simpler objects
- **Extension / Inheritance**
  - define a class from another existing class, e.g. derive
  - class hierarchy

Object-oriented programming and modeling

Key concept

- **Encapsulation**
  - hide implementation details
  - guarantee a class invariant
- **Aggregation**
  - define composite objects from several simpler objects
- **Extension / Inheritance**
  - define a class from another existing class, e.g. derive
  - class hierarchy

Strength : Allow local reasoning !

What type of programming language is C++ (and C)

	Type checking	Type safety	Type semantics	Execution Strategy	Memory management	Object oriented	Functional
Python	dynamic	weak	reference	Interpreted (*)	Garbage collector	YES	PARTIAL
Java	static + dynamic	strong	reference (classes) value (primitive types)	Compiled + Interpreted	Garbage collector	YES	PARTIAL
Rust	static	strong	value + explicit references	Compiled	Borrow Checker	YES	PARTIAL
C	static	weak	value (*) + explicit pointers	Compiled	Manual	NO	NO(*)
C++	static + dynamic	strong	value (*) + explicit pointers/ references	Compiled	Semi-automatic (RAII)	YES	PARTIAL

MINESnancy ARTEM

What is "Modern C++"

- **Static type safety**
  - Well-specified interfaces
- **Resources safety**
  - Constructors/Destructors, RAII
- **Abstraction**
  - Often zero overhead
- **Encapsulation, invariants**
  - Classes
- **Generic programming**
  - Templates
- **Simplicity for most developers**
  - Complexity hidden in libraries
- **Multiparadigm programming**
  - Object oriented, functional, concurrent, ...

MINESnancy
ARTEM

What is "Modern C++"

- **Static type safety**
  - Well-specified interfaces
- **Resources safety**
  - Constructors/Destructors, RAII
- **Abstraction**
  - Often zero overhead
- **Encapsulation, invariants**
  - Classes
- **Generic programming**
  - Templates
- **Simplicity for most developers**
  - Complexity hidden in libraries
- **Multiparadigm programming**
  - Object oriented, functional, concurrent, …

MINESnancy ARTEM

Multiparadigm

- **Example :** traversing statically typed container of objects of polymorphic type

```
template<class C>
void drawAll(C& drawable_container)
{
    for_each(drawable_container.begin(), drawable_container.end(),
             [](Shape*) { p->draw(); });
}
```

Multiparadigm

- **Example :** traversing statically typed container of objects of polymorphic type

Static type used to generate optimized iteration on container

```
template<class C>
void drawAll(C& drawable_container)
{
    for_each(drawable_container.begin(), drawable_container.end(),
             [](Shape*) { p->draw(); });
}
```

Multiparadigm

- **Example :** traversing statically typed container of objects of polymorphic type

Static type used to generate optimized iteration on container

```
template<class C>
void drawAll(C& drawable_container)
{
    for_each(drawable_container.begin(), drawable_container.end(),
             [](Shape*) { p->draw(); });
}
```

Function passed as parameter of algorithm

MINESnancy ARTEM

Multiparadigm

- **Example :** traversing statically typed container of objects of polymorphic type

Static type used to generate optimized iteration on container

```
template<class C>
void drawAll(C& drawable_container)
{
    for_each(drawable_container.begin(), drawable_container.end(),
             [](Shape*) { p->draw(); });
}
```

Dynamic polymorphic call (**inheritance**)

Function passed as parameter of algorithm

C(++) : A compiled language

- **From source code to execution**

Source code

```
int main () {
    int a = 0;
    int b = a+1;
    return b;
}
```

Compilation

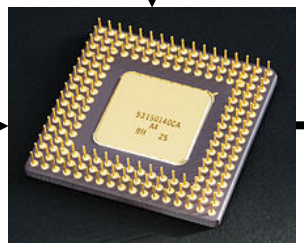
Assembly language

```
MOV EAX, EBX
ADD EBX, ECX
…
```

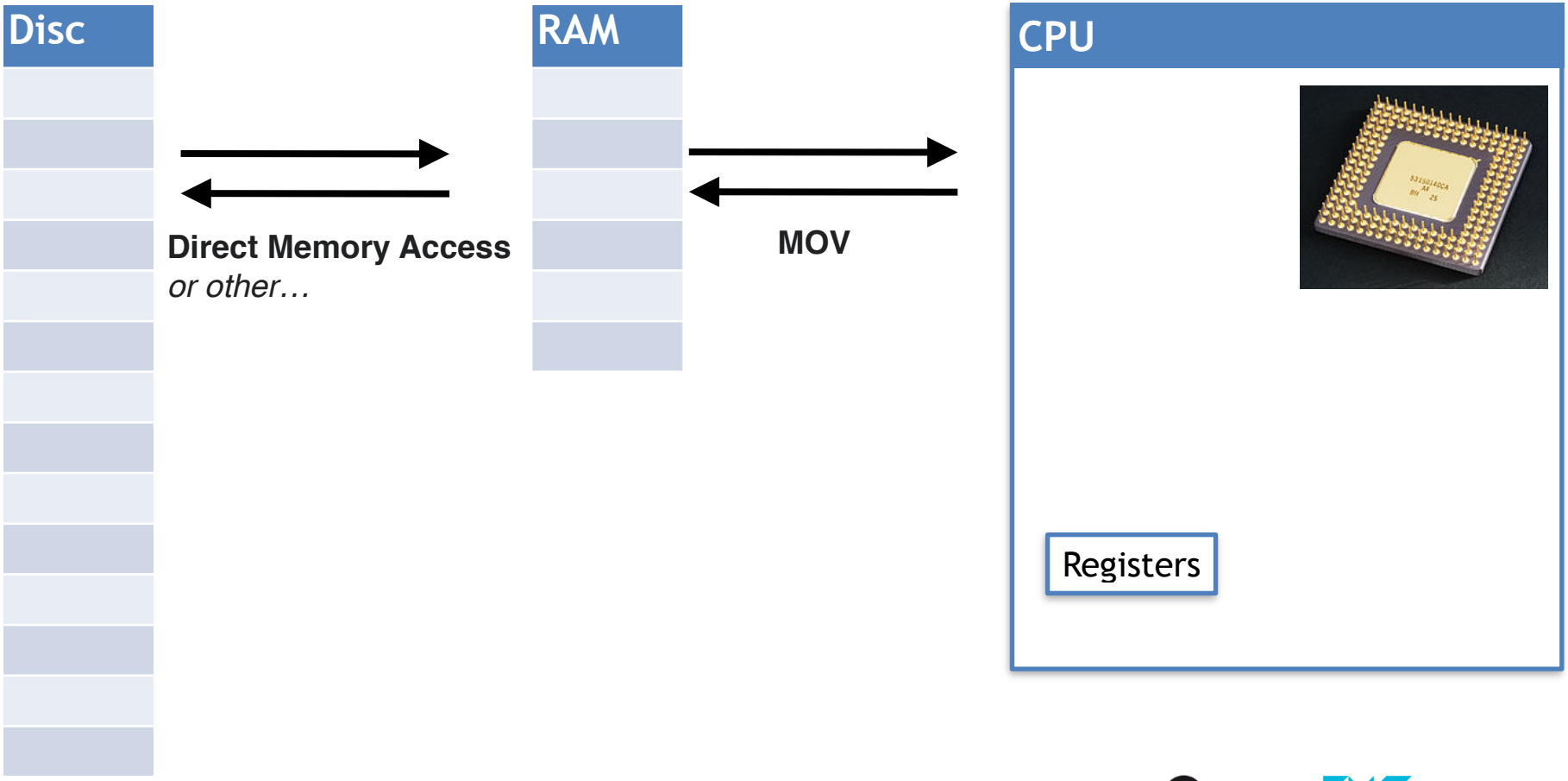
Assembly

Machine code (binary)

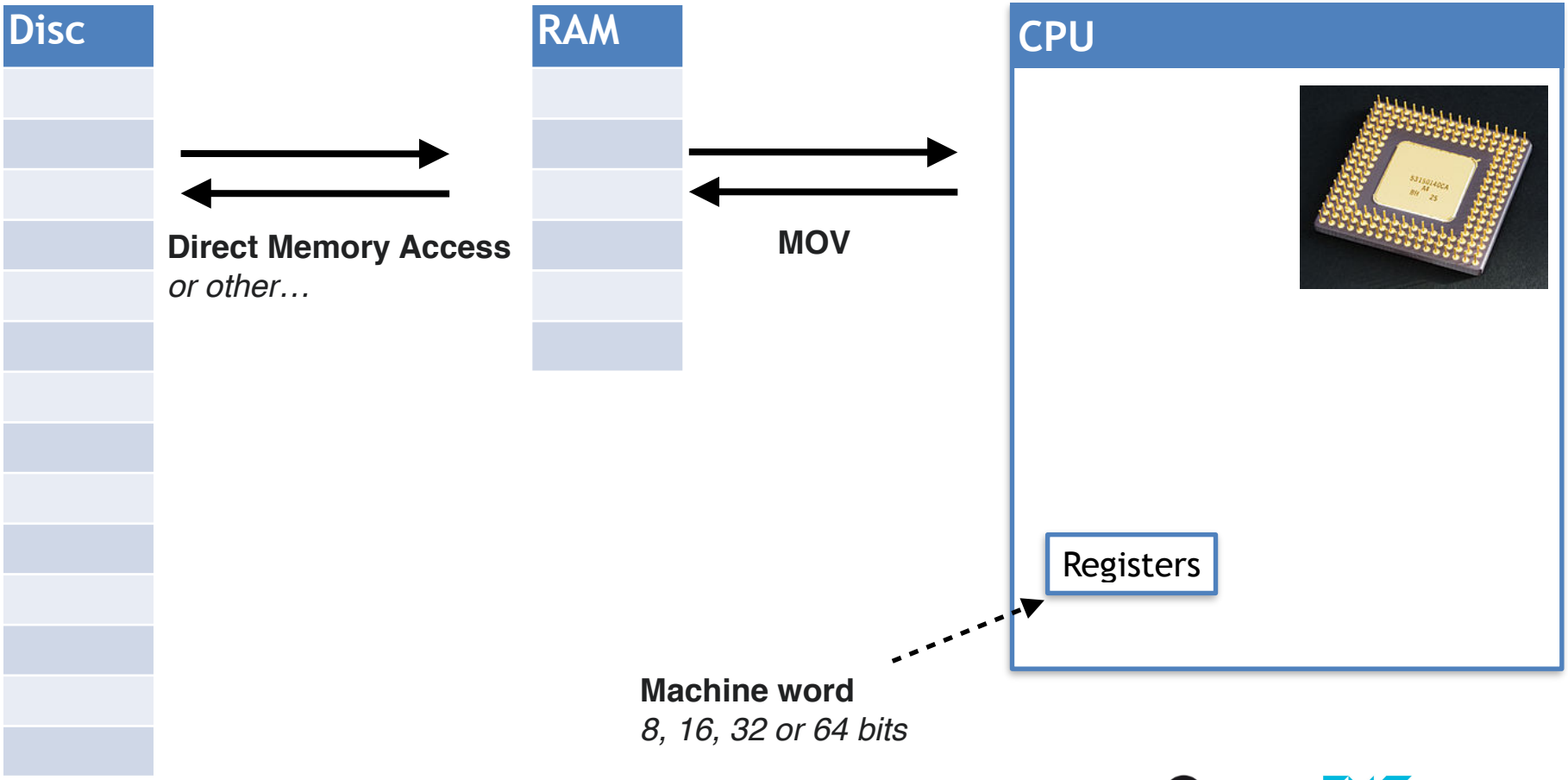
```
01001100110001100
11100110011001100
01110011110001110
01110000110010011
…
```

Input data →  → Output data

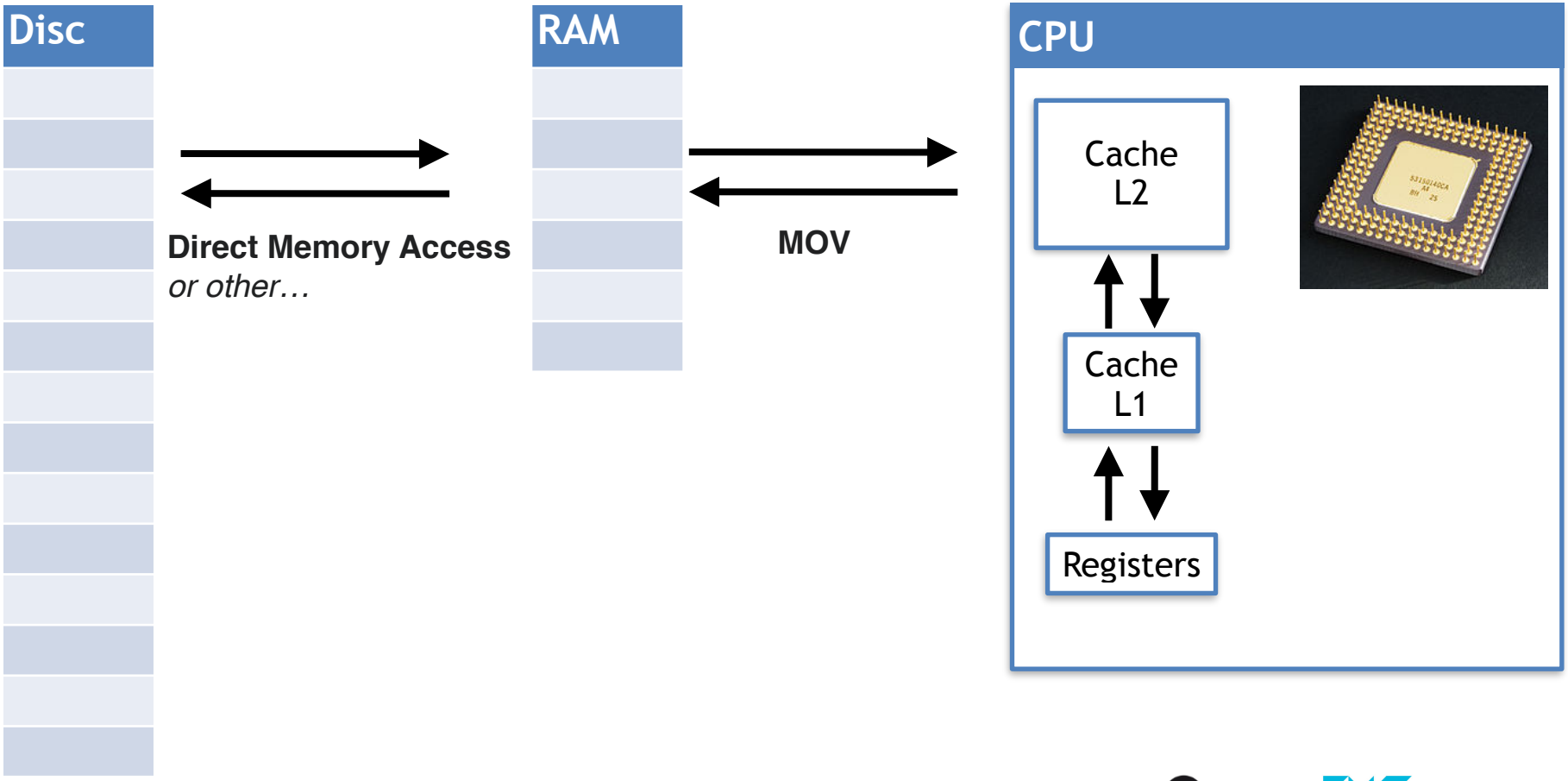
Architecture *(oversimplified)*



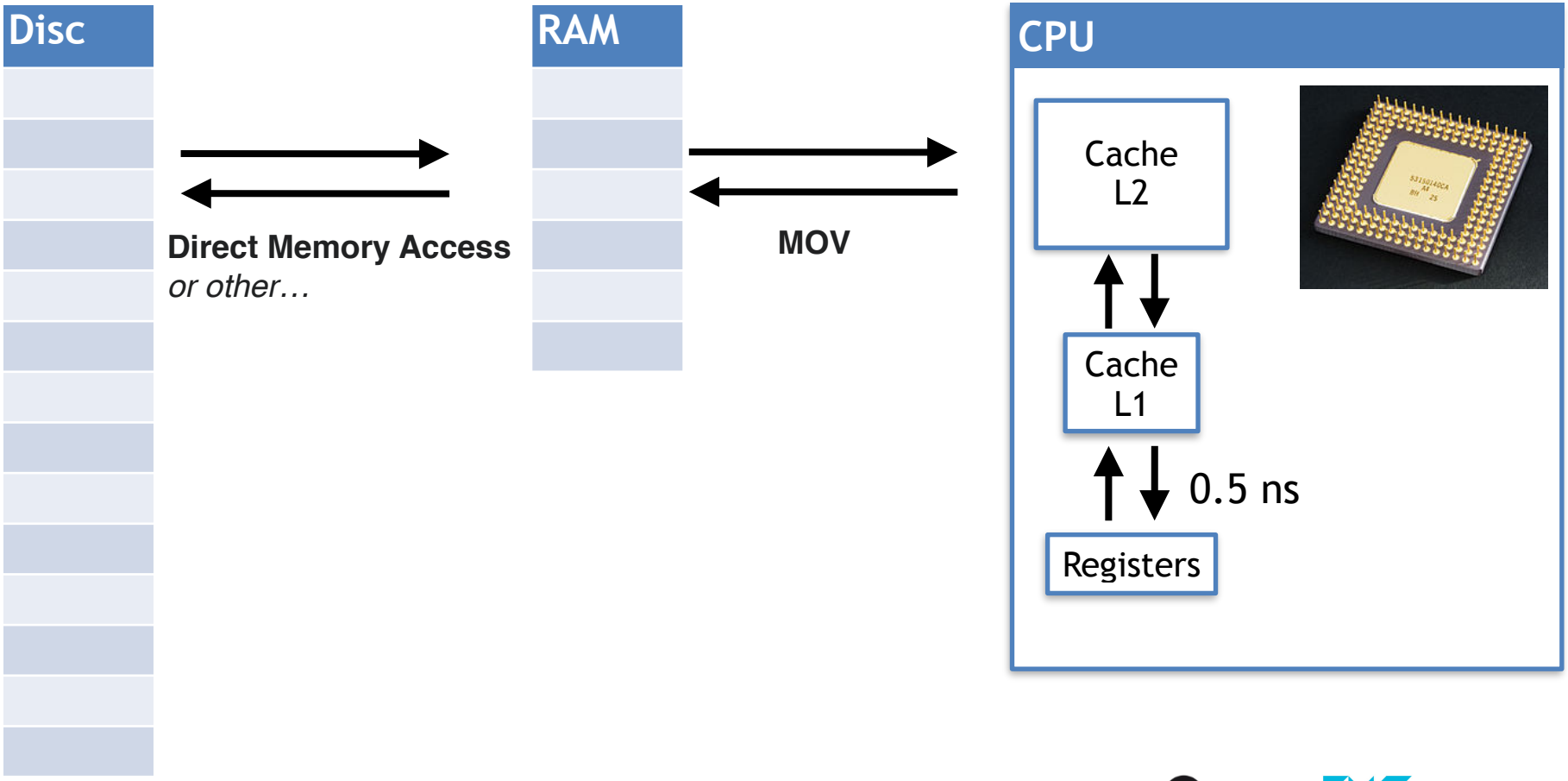
Architecture *(oversimplified)*



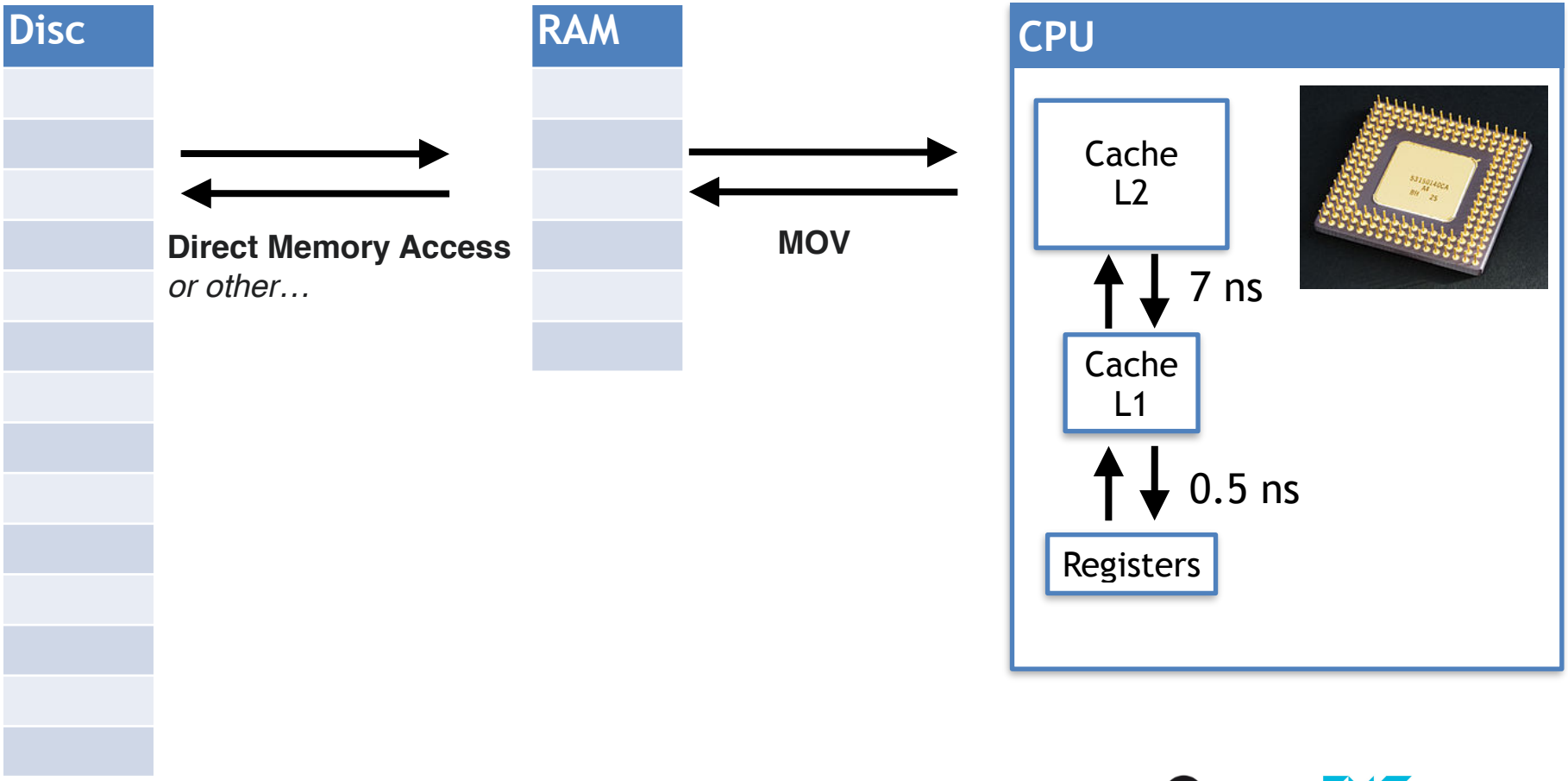
Architecture *(oversimplified)*



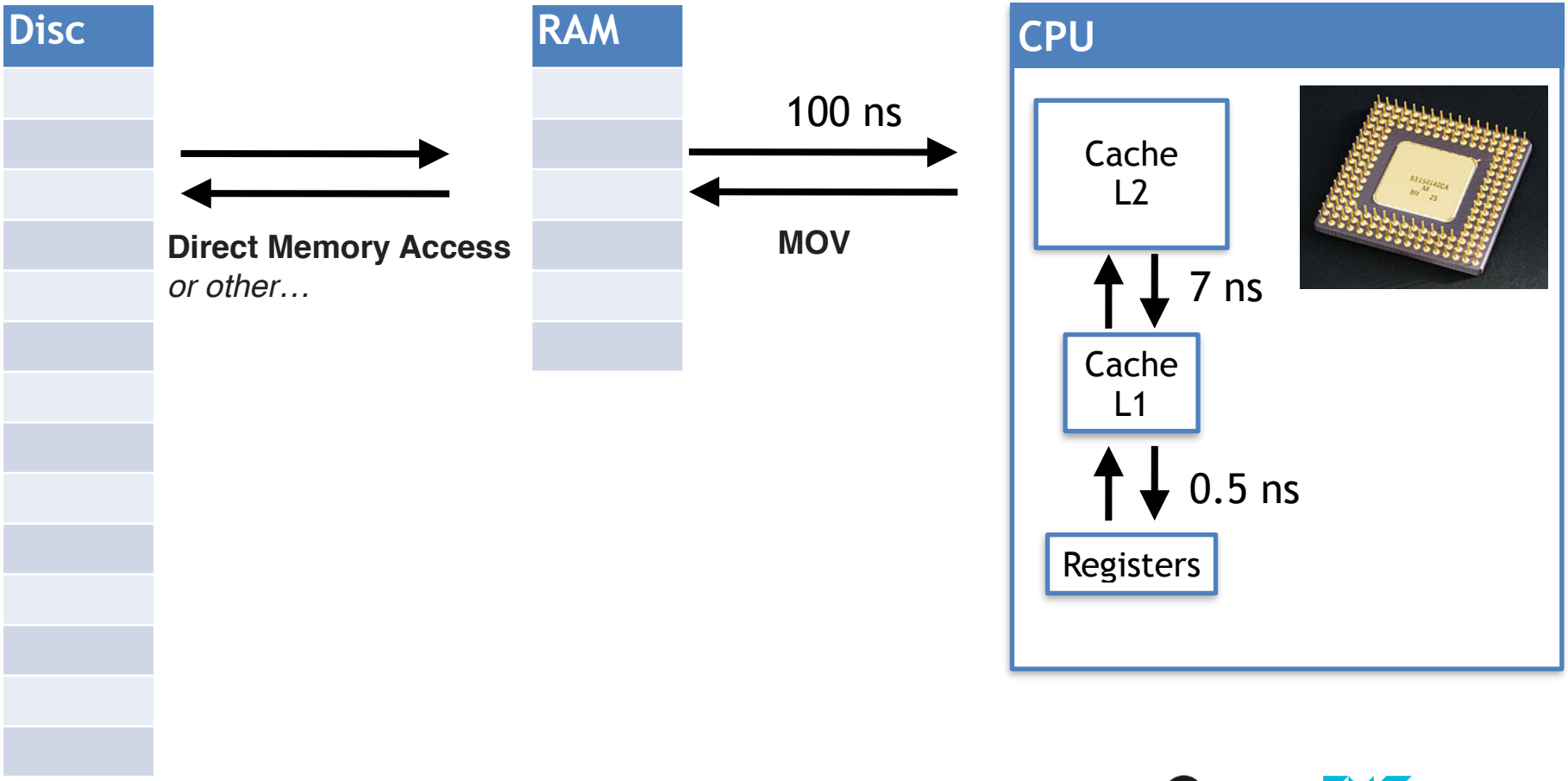
Architecture *(oversimplified)*



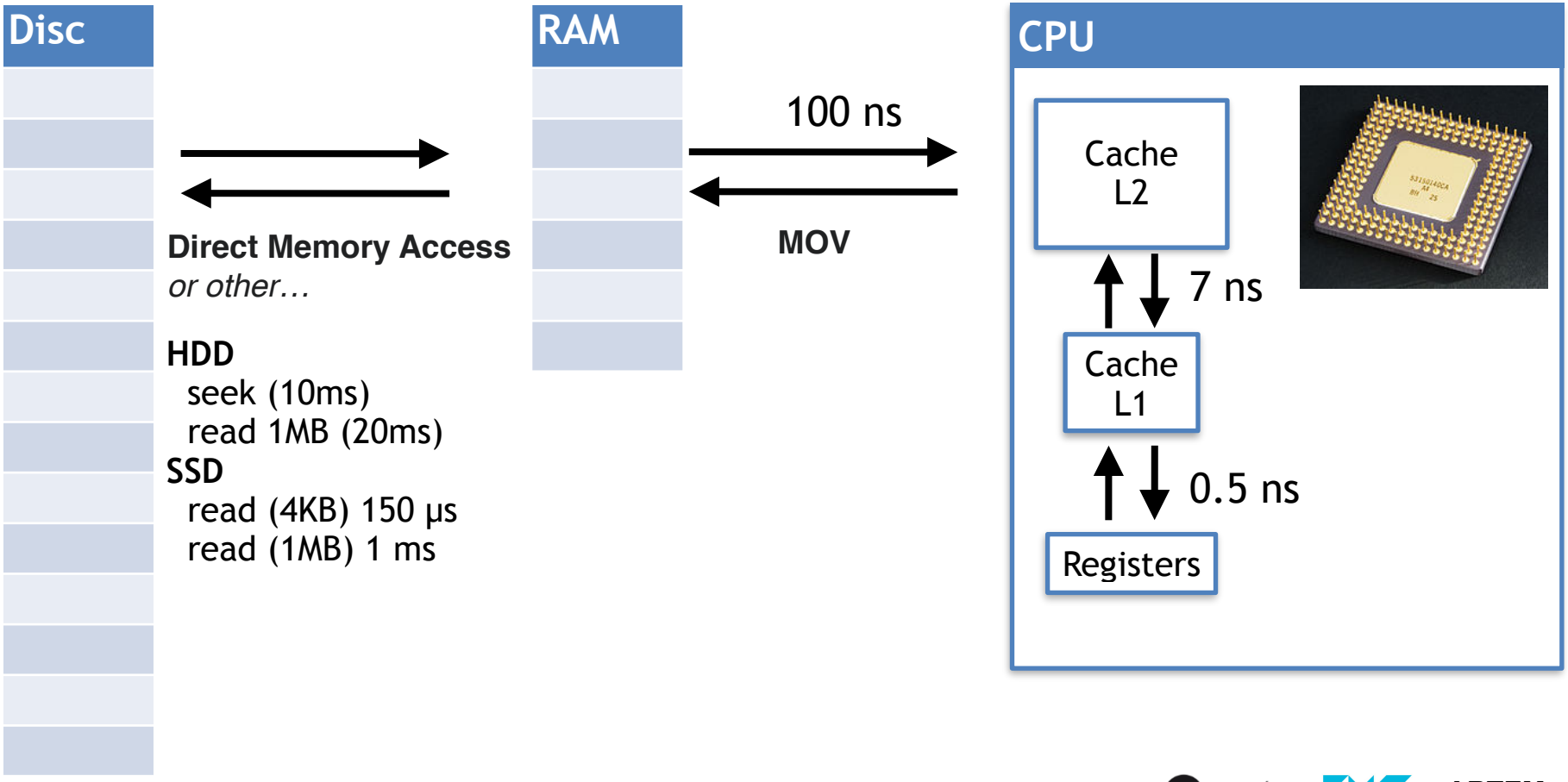
Architecture *(oversimplified)*



Architecture *(oversimplified)*



Architecture *(oversimplified)*



Syntax - Declaration and instruction (C/C++)

- **Declare variables**

```
int capacity;
float height;
```

- **Declare and initialize variables (advised!)**

```
int capacity = 1;
float height = 0.5F;
char letter = 'A';
```

- **Instructions**

```
letter = 'A';
height = 0.5F* height + 1.0F;
++capacity;
```

Syntax - Declaration and instruction (C/C++)

- **Declare variables**

```
int capacity;
float height;
```

- **Declare and initialize variables (advised!)**

```
int capacity = 1;
float height = 0.5F;
char letter = 'A';
```

- **Instructions**

```
letter = 'A';
height = 0.5F* height + 1.0F;
++capacity;
```

All variables have a static type !

Syntax - Declaration and instruction (C/C++)

- **Declare variables**

```
int capacity;
float height;
```

- **Declare and initialize variables (advised!)**

```
int capacity = 1;
float height = 0.5F;
char letter = 'A';
```

- **Instructions**

```
letter = 'A';
height = 0.5F* height + 1.0F;
++capacity;
```

All variables have a static type !

Delimiter for expressions is ';'

Base types (C/C++)

- The **Type** of a variable determine :
  - The set of possible values for a variable
  - The set of operations that can be applied to a variable
  - The size in memory required to store the variable

Typename	Minimal value	Maximal value
unsigned char	0	255
short	-32768	32767
int	-2147483648	2147483647
float	≈ -10^38	≈ 10^38
double	≈ -10^308	≈ 10^308

Base types (C/C++)

- The **Type** of a variable determine :
  - The set of possible values for a variable
  - The set of operations that can be applied to a variable
  - The size in memory required to store the variable

Typename	Minimal value	Maximal value	sizeof()
unsigned char	0	255	1
short	-32768	32767	4
int	-2147483648	2147483647	8
float	≈ -10^38	≈ 10^38	4
double	≈ -10^308	≈ 10^308	8

MINESnancy ARTEM

Base types (C/C++)

- The **Type** of a variable determine :
  - The set of possible values for a variable
  - The set of operations that can be applied to a variable
  - The size in memory required to store the variable

Typename	Minimal value	Maximal value	sizeof()
unsigned char	0	255	1
short	-32768	32767	4
int	-2147483648	2147483647	8
float	≈ -10^38	≈ 10^38	4
double	≈ -10^308	≈ 10^308	8

Warning : not signed, if incremented after maximal value, fall back to zero !

MINESnancy ARTEM

Base types (C/C++)

- The **Type** of a variable determine :
  - The set of possible values for a variable
  - The set of operations that can be applied to a variable
  - The size in memory required to store the variable

Typename	Minimal value	Maximal value	sizeof()
unsigned char	0	255	1
short	-32768	32767	4
int	-2147483648	2147483647	8
float	≈ -10^38	≈ 10^38	4
double	≈ -10^308	≈ 10^308	8

Warning : not signed, if incremented after maximal value, fall back to zero !

Warning : remember that floating point precision is not uniform !

Expressions (C/C++)

- **Expression**
  - Combination of values, variables, operators and function calls
  - Delimited by « ; »

- **Variable**
  - Mutable « value » identified by a name
  - Linked to a memory address and identified by a type to interpret memory content

- **Operator**
  - **Arithmetic:** +, -, *, /, %
  - **Relational:** >, >=, <=, ==, !=
  - **Logical :** &&, ||, !
  - **Increments :** ++, --
  - Bitwise (will be discussed in other lectures)

Expressions (C/C++)

- **Expression**
  - Combination of values, variables, operators and function calls
  - Delimited by « ; »

- **Variable**
  - Mutable « value » identified by a name
  - Linked to a memory address and identified by a type to interpret memory content

- **Operator**
  - **Arithmetic:** +, -, *, /, %
  - **Relational:** >, >=, <=, ==, !=
  - **Logical :** &&, ||, !
  - **Increments :** ++, --
  - Bitwise (will be discussed in other lectures)

Comment : Memory address of a variable can be seen as an abstraction of RAM memory address

Syntax - Explicit type conversion (cast)

- **Compute a ratio from integer numbers**

C

```c
int i = 2;
int j = 10;

double q = ((double) i) / j ;
```

C++

```cpp
int i = 2;
int j = 10;

double q = static_cast<double>(i) / j ;
```

Implicit type conversion (C++)

- **Widening conversions (promotion)**
  - signed, unsigned (not int64) -> double
  - bool, char -> any built-in type
  - short -> int, long, long long
  - int, long -> long long
  - float -> double
- **Narrowing conversions (coercion)**
  - floating point to integral types
  - int -> unsigned int
  - ...

MINESnancy ARTEM

Implicit type conversion (C++)

- **Widening conversions (promotion)**
  - signed, unsigned (not int64) -> double
  - bool, char -> any built-in type
  - short -> int, long, long long
  - int, long -> long long
  - float -> double
- **Narrowing conversions (coercion)**
  - floating point to integral types
  - int -> unsigned int
  - …

Comment : Narrowing conversions raise compiler warnings!

Syntax - Control flow

- **Conditions**

```
if (valeur==0 /* expression of the boolean condition */) {
  // Instruction 1;
  // Instruction 2;
  // …;

} else if (valeur==0 /* expression of another condition */) {
  // …;
} else { // if none of the above test passes
  // …;
}
```

- **NB: no « ; » after a control flow structure**

Syntax - Scopes

- **Scope of a variable = the block {} where it is defined**

```
{
  int i = 1;
  { // Beginning of Scope
    int j = 2;
    // can use i and j
  } // End of Scope
  // j does not exist anymore
}
```

- **Usual scopes :** control flow structure and functions

Syntax - Scopes

- **Scope of a variable = the block {} where it is defined**

```
{
  int i = 1;
  { // Beginning of Scope
    int j = 2;
    // can use i and j
  } // End of Scope
  // j does not exist anymore
}
```

- **Usual scopes :** control flow structure and functions

Scope = sequence of instructions, of control flow structures and Scopes delimited by a pair of opening and closing brackets {}

Syntax - Scopes

- **Scope of a variable = the block {} where it is defined**

```
{
  int i = 1;
  { // Beginning of Scope
    int j = 2;
    // can use i and j
  } // End of Scope
  // j does not exist anymore
}
```

- **Usual scopes :** control flow structure and functions

Scope = sequence of instructions, of control flow structures and Scopes delimited by a pair of opening and closing brackets {}

Important : a variable cannot be used outside its scope of definition !

Syntax - Control flow

- **while loop**

```
int compteur = 0;
while (compteur < 10) {
  // Instructions…
  ++compteur;
}
```

- **for loop**

```
for (int compteur = 0; compteur < 10; ++compteur) {
  // Instructions…
}
```

Syntax - Control flow

```
for (int compteur = 0; compteur < 10; ++compteur) {
  // Instructions…
}
```

- **for loop, general case**

```
for (Decl. and init. ; End condition; End of iteration action) {
  // Instructions…
}
```

  - Declaration and initialization

```
int a=0, b=10
```

  - End condition

```
a>10 && b<0
```

  - End of iteration action

```
++a, b-=2
```

Syntax - Control flow

```
for (int compteur = 0; compteur < 10; ++compteur) {
  // Instructions…
}
```

- **for loop, general case**

```
for (Decl. and init. ; End condition; End of iteration action) {
  // Instructions…
}
```

  - Declaration and initialization

`int a=0, b=10`

  - End condition

`a>10 && b<0`

  - End of iteration action

`++a, b-=2`

Warning : beware of decrements and strict equality checks !

Syntax - Functions

- **Programme** = set of functions + entry point (main)
- **Minimal program**

C

```c
#include <stdio.h>

int main()
{
  printf("Hello World %!d", 2);
  return 0;
}
```

C++

```cpp
#include <print>

int main()
{
  std::print("Hello World {}!", 2);
  return 0;
}
```

Syntax - Functions

- **Programme** = set of functions + entry point (main)
- **Minimal program**

C

```c
#include <stdio.h>

int main()
{
  printf("Hello World %!d", 2);
  return 0;
}
```

C++

```cpp
#include <print>

int main()
{
  std::print("Hello World {}!", 2);
  return 0;
}
```

Comment : code outside of functions limited to declaration and initialization

Syntax - Function declaration

- **Return type** (or **`void`** if no value is returned)
- Parameter types for inputs
- Parameters are **copied** at call site !

C/C++

```c
#include <stdio.h>

int addition(int nombre1, int nombre2)
{
  return nombre1 + nombre2;
}

int main()
{
  printf("Hello World! »);
  printf("Hello World!");
  return 0;
}
```

Memory organization - the stack

- **Reminder :** variable passed as parameter to a fonction are copied

```
int addOne(int x) {
  int y = x+1;
  return y;
}

int main () {
    int a;
    int b;
    a = 5;
    b = addOne(a);
    return 0;
}
```

IP

Frame	VAR	Pile	Adresse
main	...	...	...
	a		10001
	b		10002
			10003
			10004
			10005
			10006
			10007
			...

IP = Instruction Pointer

Memory organization - the stack

- **Reminder :** variable passed as parameter to a fonction are copied

```
int addOne(int x) {
  int y = x+1;
  return y;
}

int main () {
    int a;
    int b;
    a = 5;
    b = addOne(a);
    return 0;
}
```

IP

Frame	VAR	Pile	Adresse
main	...	...	...
	a	5	10001
	b		10002
			10003
			10004
			10005
			10006
			10007

Memory organization - the stack

- **Reminder :** variable passed as parameter to a fonction are copied

```
int addOne(int x) {
  int y = x+1;
  return y;
}

int main () {
    int a;
    int b;
    a = 5;
    b = addOne(a);
    return 0;
}
```

IP

Frame	VAR	Pile	Adresse
main	...	...	...
	a	5	10001
	b		10002
addOne			10003
			10004
			10005
			10006
			10007
			...

How to handle a function call ?
=> Stack a new « frame »

Memory organization - the stack

- **Reminder :** variable passed as parameter to a fonction are copied

```
int addOne(int x) {
  int y = x+1;
  return y;
}

int main () {
    int a;
    int b;
    a = 5;
    b = addOne(a);
    return 0;
}
```

IP

Frame	VAR	Pile	Adresse
main	...	...	...
	a	5	10001
	b		10002
addOne	x	5	~~10003~~
			10004
			10005
			10006
			10007
			...

Param copy

Content of a « frame »

Memory organization - the stack

- **Reminder :** variable passed as parameter to a fonction are copied

```
int addOne(int x) {
  int y = x+1;
  return y;
}

int main () {
    int a;
    int b;
    a = 5;
    b = addOne(a);
    return 0;
}
```

IP

Frame	VAR	Pile	Adresse
main	...	...	...
	a	5	10001
	b		10002
addOne	x	5	10003
			10004
	y		10005
			10006
			10007
			...

Alloc locals

Content of a « frame »

Memory organization - the stack

- **Reminder :** variable passed as parameter to a fonction are copied

```
int addOne(int x) {
  int y = x+1;
  return y;
}

int main () {
    int a;
    int b;
    a = 5;
    b = addOne(a);
    return 0;
}
```

IP

Frame	VAR	Pile	Adresse
main	...	...	...
	a	5	10001
	b		10002
addOne	x	5	10003
			10004
	y	6	10005
			10006
			10007
			...

Memory organization - the stack

- **Reminder :** variable passed as parameter to a fonction are copied

```
int addOne(int x) {
  int y = x+1;
  return y;
}

int main () {
    int a;
    int b;
    a = 5;
    b = addOne(a);
    return 0;
}
```

IP

Frame	VAR	Pile	Adresse
main	...	...	...
	a	5	10001
	b		10002
addOne	x	5	10003
			10004
	y	6	10005
			10006
			10007
			...

What to do ?

Memory organization - the stack

- **Reminder :** variable passed as parameter to a fonction are copied

```
int addOne(int x) {
  int y = x+1;
  return y;
}

int main () {
    int a;
    int b;
    a = 5;
    b = addOne(a);
    return 0;
}
```

IP

Frame	VAR	Pile	Adresse
main	...	...	...
	a	5	10001
	b		10002
addOne	x	5	10003
	@return	10002	10004
	y	6	10005
			10006
			10007
			...

Save return address at call time !

Memory organization - the stack

- **Reminder :** variable passed as parameter to a fonction are copied

```
int addOne(int x) {
  int y = x+1;
  return y;
}

int main () {
    int a;
    int b;
    a = 5;
    b = addOne(a);
    return 0;
}
```

IP

Frame	VAR	Pile	Adresse
main	...	...	...
	a	5	10001
	b		10002
addOne	x	5	10003
	@return	10002	10004
	y	6	10005
			10006
			10007
			...

Memory organization - the stack

- **Reminder :** variable passed as parameter to a fonction are copied

```
int addOne(int x) {
  int y = x+1;
  return y;
}

int main () {
    int a;
    int b;
    a = 5;
    b = addOne(a);
    return 0;
}
```

IP

Frame	VAR	Pile	Adresse
main	...	...	...
	a	5	10001
	b	6	10002
addOne	x	5	10003
	@return	10002	10004
	y	6	10005
			10006
			10007
			...

Copy

Memory organization - the stack

- **Reminder :** variable passed as parameter to a fonction are copied

```
int addOne(int x) {
  int y = x+1;
  return y;
}

int main () {
    int a;
    int b;
    a = 5;
    b = addOne(a);
    return 0;
}
```

IP

Frame	VAR	Pile	Adresse
main	...	...	...
	a	5	10001
	b	6	10002
			10003
			10004
			10005
			10006
			10007
			...

Unstack the frame when leaving the function

MINESnancy
ARTEM

Memory organization - the stack

MINESnancy ARTEM

Memory organization - the stack

Important : memory of **local** variable is managed automatically !

MINESnancy
ARTEM

Memory organization - the stack

Important : memory of **local** variable is managed automatically !

Comment : can be optimized further by the compiler…

Syntax - Function declaration, definition and usage

```c
#include <stdio.h>

// Declaration of function (necessary before usage
// need to know the « contract »)
int addition(int nombre1, int nombre2);

// Definition of function (not necessary before usage)
int addition(int nombre1, int nombre2)
{
  return nombre1 + nombre2;
}

int main()
{
  printf("Result of 5 + 3 is %d \n", addition(5,3));
  //                                  ^^^^^^^^^^
  //                                  Using the function
 return 0;
}
```

Arrays (additional information on board)

- **Declaration (**allocation de N * sizeof(data_type) octets**)**

```
int tableau[4];
```

- **Initialization**

```
int tableau[4] = {10, 23}; // valeurs insérées : 10, 23, 0, 0
```

- **Usage**

```
tableau[0] = 10;
tableau[1] = tableau[0]+1;
```

Arrays (additional information on board)

- **Declaration (**allocation de N * sizeof(data_type) octets**)**

```
int tableau[4];
```

- **Initialization**

```
int tableau[4] = {10, 23}; // valeurs insérées : 10, 23, 0, 0
```

- **Usage**

```
tableau[0] = 10;
tableau[1] = tableau[0]+1;
```

Important : Beware when passed as parameter to function (pointer semantics)

MINESnancy ARTEM

Libraries

- **Import declaration from functions** (#include)
- **Require linking** (will be rediscussed later)

- **Useful thing in C standard library** : math.h, string.h, …
- **… and a lot more in in C++ standard library…**

MINESnancy ARTEM

Libraries

- **Import declaration from functions** (#include)
- **Require linking** (will be rediscussed later)

- **Useful thing in C standard library** : math.h, string.h, …
- **… and a lot more in in C++ standard library…**

Comment : in C++ new import keyword for modules …

MINESnancy ARTEM

IMPORTANT

- **Read carefully compiler errors**
  - Ask for clarification if required !
- **Consider warning as errors**

- **Two main compilation mode**
  - **Debug** (option -g with gcc and clang)
  - **Release** (option -O2 or -O3 with gcc and clang)

- **the palest of ink is better than the best memory** (Chinese proverb)
  - Use a version control software => Git

MINESnancy
ARTEM

References

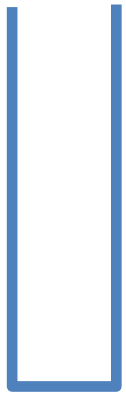
- en.cppreference.com
- learn.microsoft.com/en-us/cpp/cpp-language-reference

Notation polonaise inversé

- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x



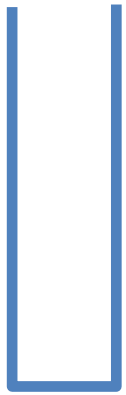
Notation polonaise inversé

- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Constante : insert in LIFO



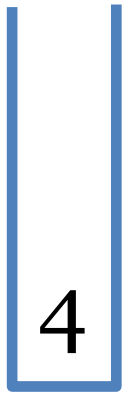
Notation polonaise inversé

- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Constante : insert in LIFO



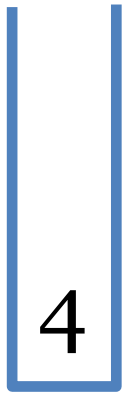
Notation polonaise inversé

- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Constante : insert in LIFO



Notation polonaise inversé

- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Constante : insert in LIFO

Notation polonaise inversé

- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Binary Operation :
- Pop operand 1 from LIFO
- Pop operand 2 from LIFO
- Compute operation
- Insert result in LIFO

Notation polonaise inversé

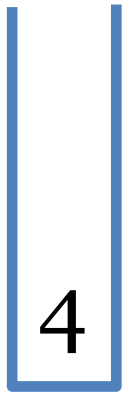
- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Binary Operation :
Pop operand 1 from LIFO
Pop operand 2 from LIFO
Compute operation
Insert result in LIFO

3



Notation polonaise inversé

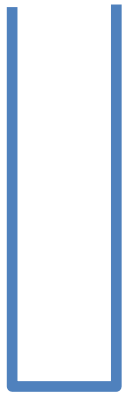
- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Binary Operation :
Pop operand 1 from LIFO
Pop operand 2 from LIFO
Compute operation
Insert result in LIFO

3
4



Notation polonaise inversé

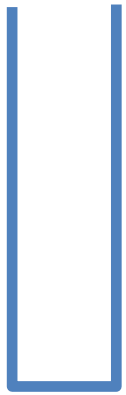
- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Binary Operation :
- Pop operand 1 from LIFO
- Pop operand 2 from LIFO
- Compute operation
- Insert result in LIFO

3
4
7



Notation polonaise inversé

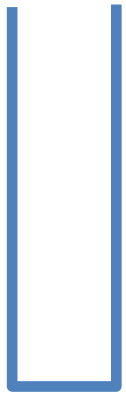
- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Binary Operation :
- Pop operand 1 from LIFO
- Pop operand 2 from LIFO
- Compute operation
- Insert result in LIFO

7



Notation polonaise inversé

- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Binary Operation :
- Pop operand 1 from LIFO
- Pop operand 2 from LIFO
- Compute operation
- Insert result in LIFO

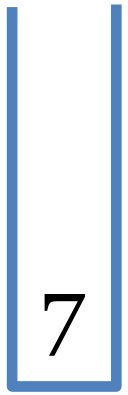
Notation polonaise inversé

- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Constante : insert in LIFO



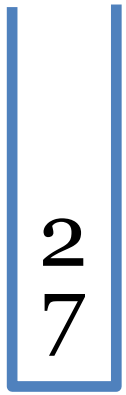
Notation polonaise inversé

- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Constante : insert in LIFO



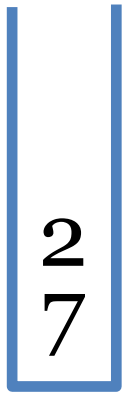
Notation polonaise inversé

- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Binary Operation :
- Pop operand 1 from LIFO
- Pop operand 2 from LIFO
- Compute operation
- Insert result in LIFO



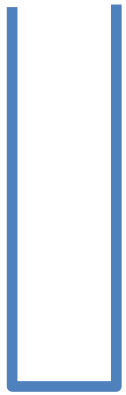
Notation polonaise inversé

- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Binary Operation :
Pop operand 1 from LIFO
Pop operand 2 from LIFO
Compute operation
Insert result in LIFO



Notation polonaise inversé

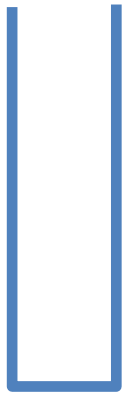
- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Binary Operation :
Pop operand 1 from LIFO
Pop operand 2 from LIFO
Compute operation
Insert result in LIFO

2
7
14



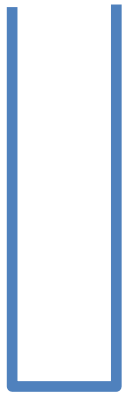
Notation polonaise inversé

- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Binary Operation :
Pop operand 1 from LIFO
Pop operand 2 from LIFO
Compute operation
Insert result in LIFO



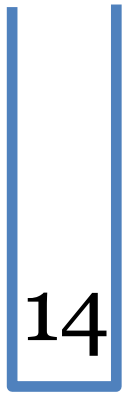
Notation polonaise inversé

- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Binary Operation :
Pop operand 1 from LIFO
Pop operand 2 from LIFO
Compute operation
Insert result in LIFO



Notation polonaise inversé

- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Pop result

Notation polonaise inversé

- **Expression :** 2 x (4 + 3)
- **Peut s'écrire en NPI :** 4 3 + 2 x

- **Evaluation à l'aide d'une pile**

4 3 + 2 x

Pop result 14