Robotics Corner

Robotics Corner

Polymorphism

01 What is Polymorphism

02 Types of Polymorphism

03 Compile Time Polymorphism

04 Run Time Polymorphism

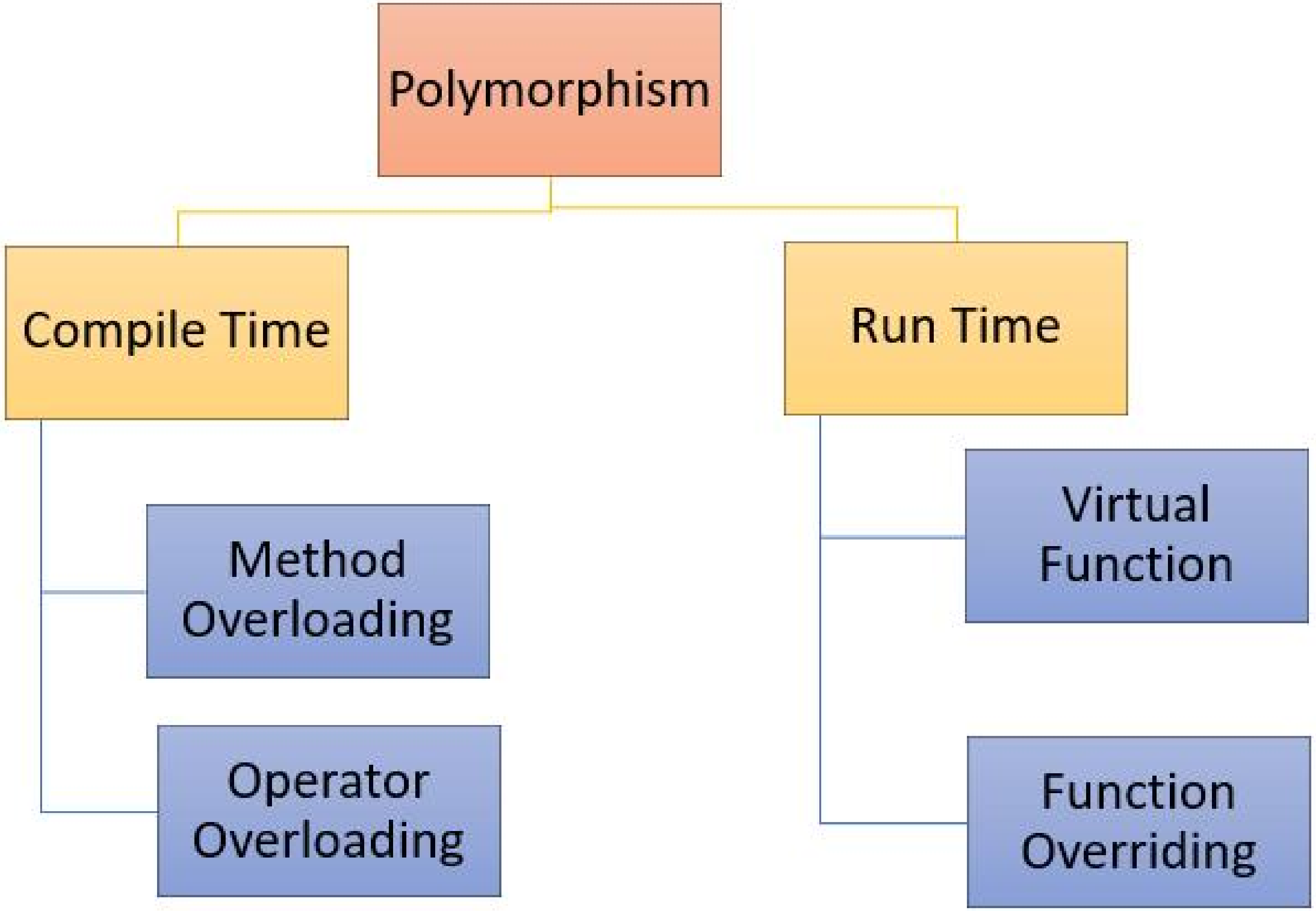
05 Abstract class

01 What is Polymorphism

02

Types of Polymorphism

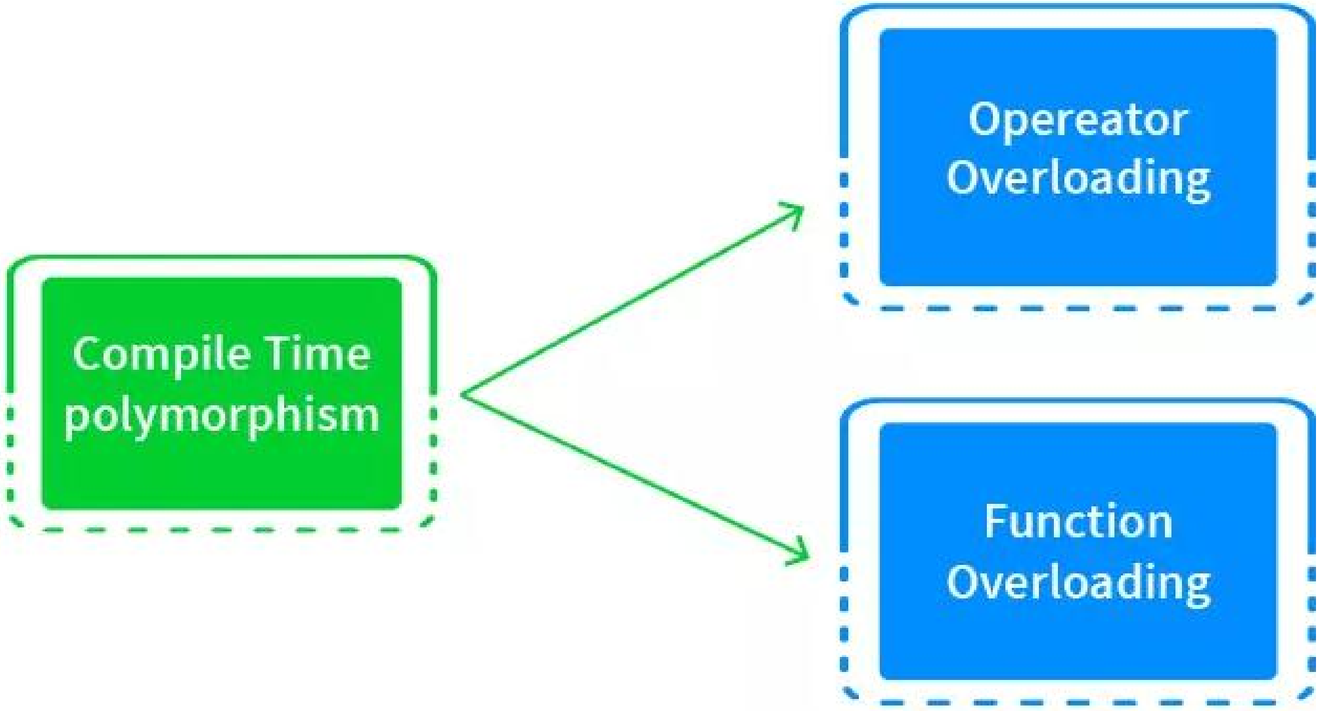
Types of Polymorphism



ROBOTICS CORNER

Static Polymorphism

- This type of polymorphism is also referred to as static binding or early binding. It takes place during compilation.
  We use function overloading and operator overloading to achieve compile-time polymorphism.



Function overloading

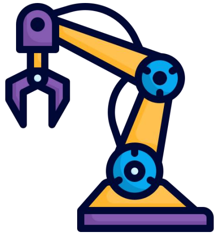
```
int add(int, int);
```

```
int add(int, int, int); //number of parameters different
```

```
double add(double, double); //type of parameters different
```

03

Compile Time Polymorphism



Example code

```
#include <iostream>
using namespace std;
int add(int a, int b) {
return a+b;
}
int add(int a, int b, int c) {
return a+b+c;
}
double add(double a, double b) {
return a+b;
}
int main() {
int x = 3, y = 7, z = 12;
double n1 = 4.56, n2 = 13.479;
cout<<"x+y = "<<add(x,y)<<endl;
cout<<"x+y+z = "<<add(x,y,z)<<endl;
cout<<"n1+n2 = "<<add(n1,n2);
return 0;
}
```

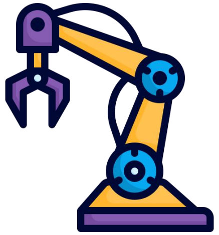
ROBOTICS CORNER

Operator overloading

- We can also overload operators in C++. We can change the behavior of operators for user-defined types like objects and structures.
- For example, the ‘+’ operator, used for addition, can also be used to concatenate two strings of std::string class. Its behavior will depend on the operands.

03

Compile Time Polymorphism



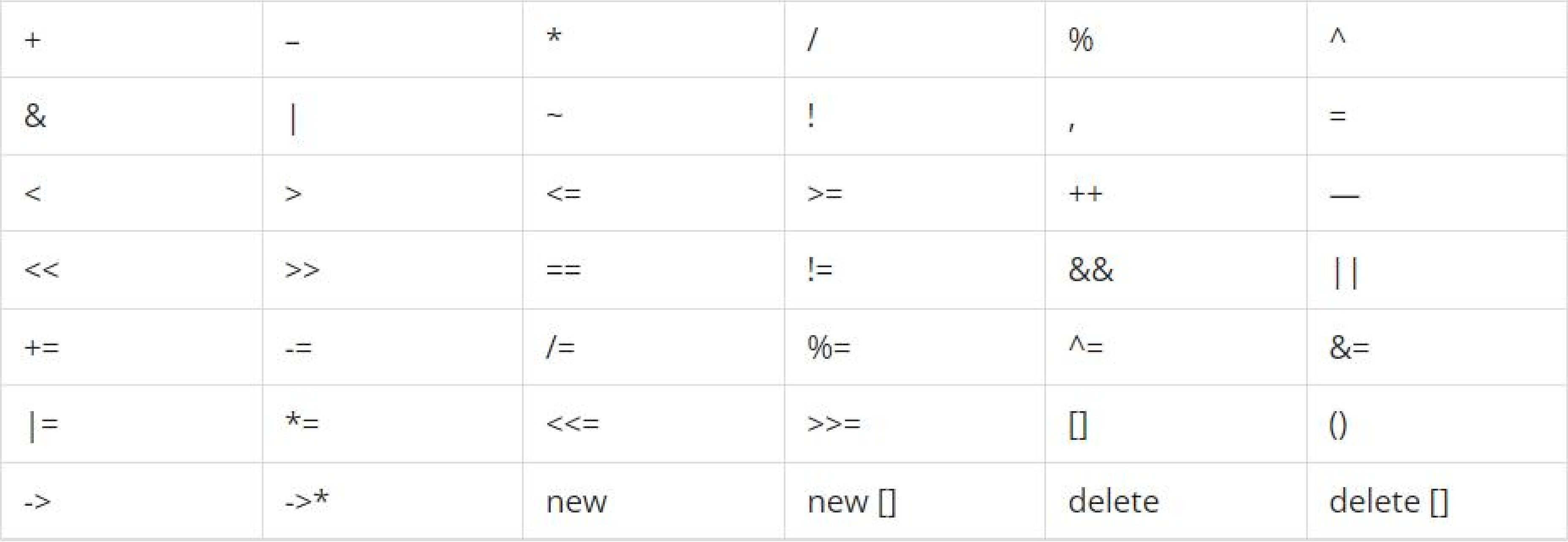
Example Code

```
//Using + operator to add complex
numbers
#include <iostream>
using namespace std;
class complex {
private:
float real, imag;
public:
complex(float r=0, float i=0){
real = r;
imag = i;
}
complex operator + (complex const &obj)
{
complex result;
result.real = real + obj.real;
result.imag = imag + obj.imag;
return result;
}
void display() {
cout<<real<<"+i"<<imag<<endl;
}
};
int main() {
complex c1(12.4,6), c2(7.9,8);
complex c3 = c1 + c2;
c3.display();
return 0;
}
```

ROBOTICS
CORNER

Overloadable Operators

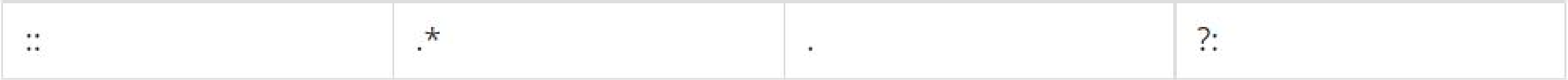
- Any Operator in this table can be overloaded



+	–	*	/	%	^
&	|	~	!	,	=
<	>	<=	>=	++	—
<<	>>	==	!=	&&	||
+=	-=	/=	%=	^=	&=
|=	*=	<<=	>>=	[]	()
->	->*	new	new []	delete	delete []

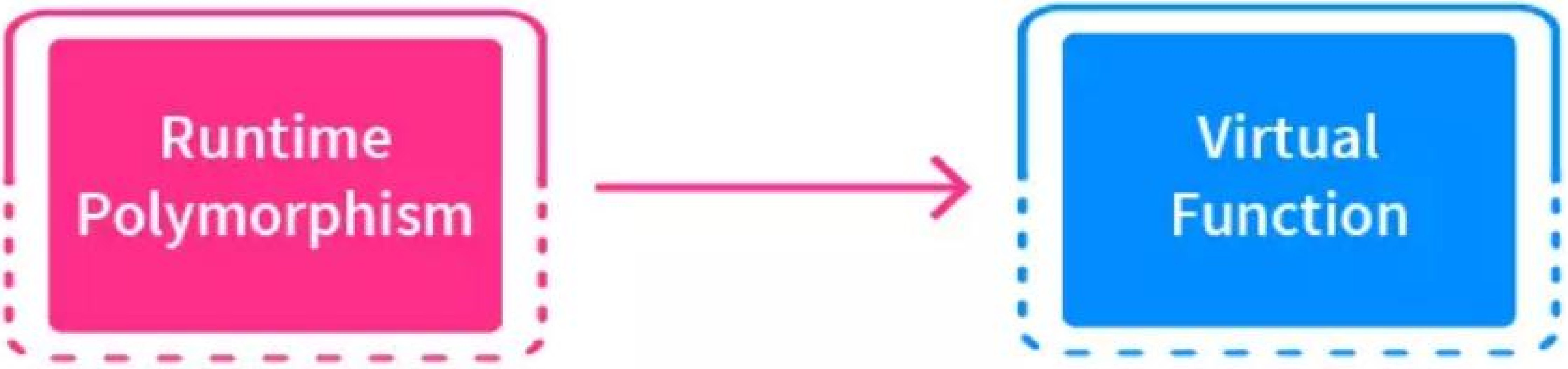
Non Overloadable Operators

Following is the list of operators, which can not be overloaded.



::	.*	.	?:

04 Runtime Polymorphism



Runtime Polymorphism

Virtual Function

ROBOTICS CORNER

04 Runtime Polymorphism

Virtual Function

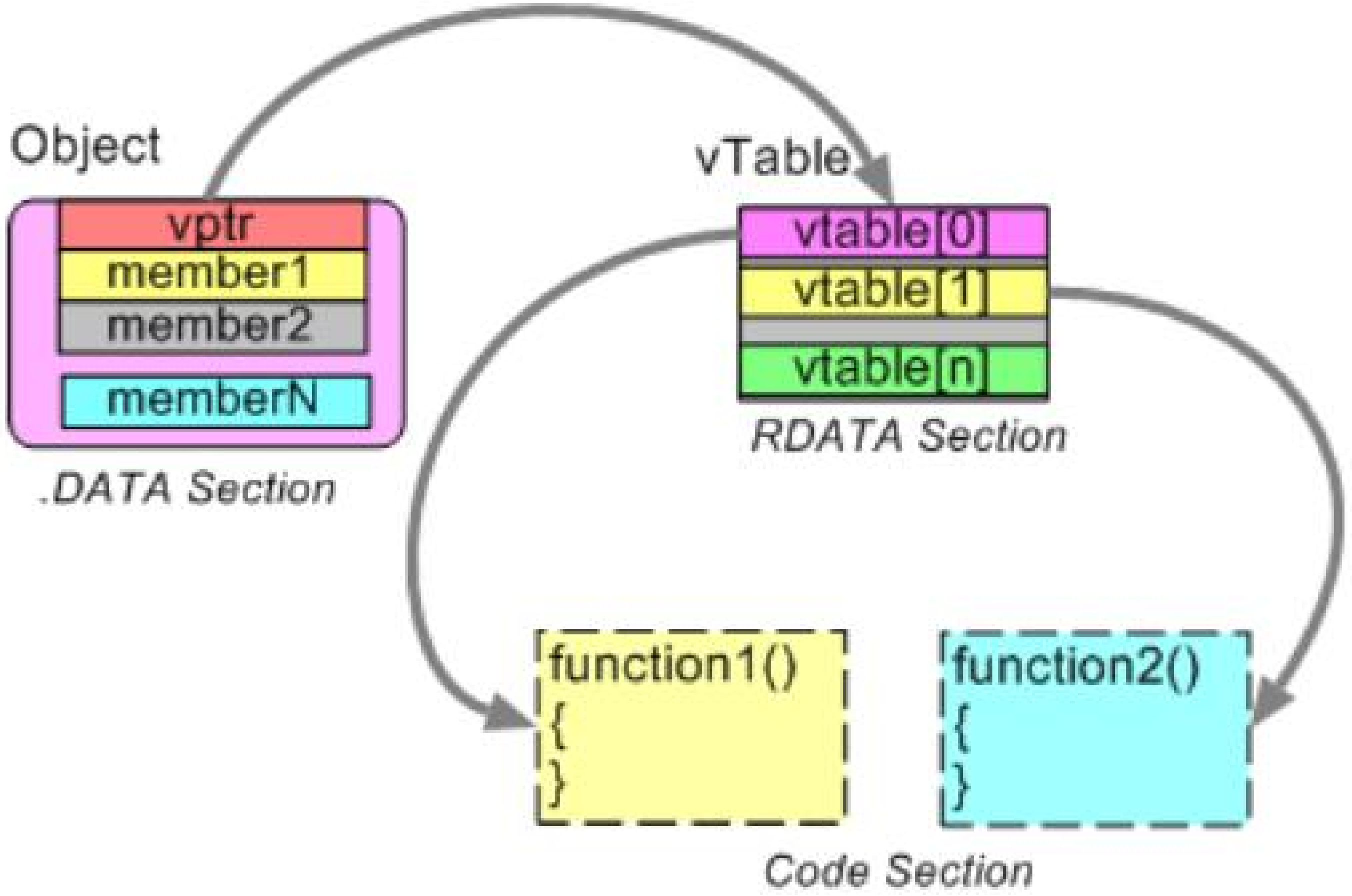
- Run-time polymorphism takes place when functions are invoked during run time. It is also known as dynamic binding or late binding. Function overriding is used to achieve run-time polymorphism.

Virtual Function Example

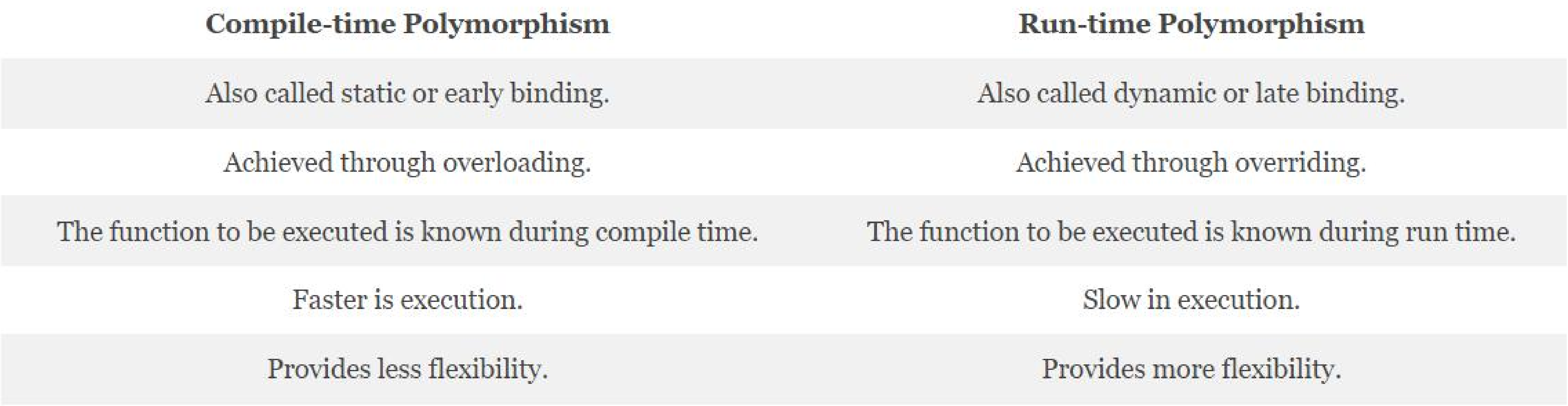
- When a member function of a base class is redefined in its derived class with the same parameters and return type, it is called function overriding in C++. The base class function is said to be overridden.
- The function call is resolved during run time and not by the compiler.

```
#include <iostream>
using namespace std;
class base {
public:
virtual void display() {
cout<<"Function of base class"<<endl;
}
};
class derived : public base {
public:
void display() {
cout<<"Function of derived class"<<endl;
}
};
int main() {
derived d1;
d1.display();
return 0;
}
```

Virtual Table



Summary



Compile-time Polymorphism	Run-time Polymorphism
Also called static or early binding.	Also called dynamic or late binding.
Achieved through overloading.	Achieved through overriding.
The function to be executed is known during compile time.	The function to be executed is known during run time.
Faster is execution.	Slow in execution.
Provides less flexibility.	Provides more flexibility.

```
Virtual void draw() = 0;
```

Pure Virtual Function

Is the function in the base class left without implementation

The real implementation is done in the derived class

ROBOTICS CORNER

Abstract Class

- This is where pure virtual functions are used without implementation then this class is inherited to derived class and the concrete class is the One, we use to implement and invoke functions.

Thank You

Do you have any questions?

ROBOTICS
CORNER

ROBOTICS CORNER

01211626904

www.roboticscorner.tech

Robotics Corner

Robotics Corner

Robotics Corner

Robotics Corner