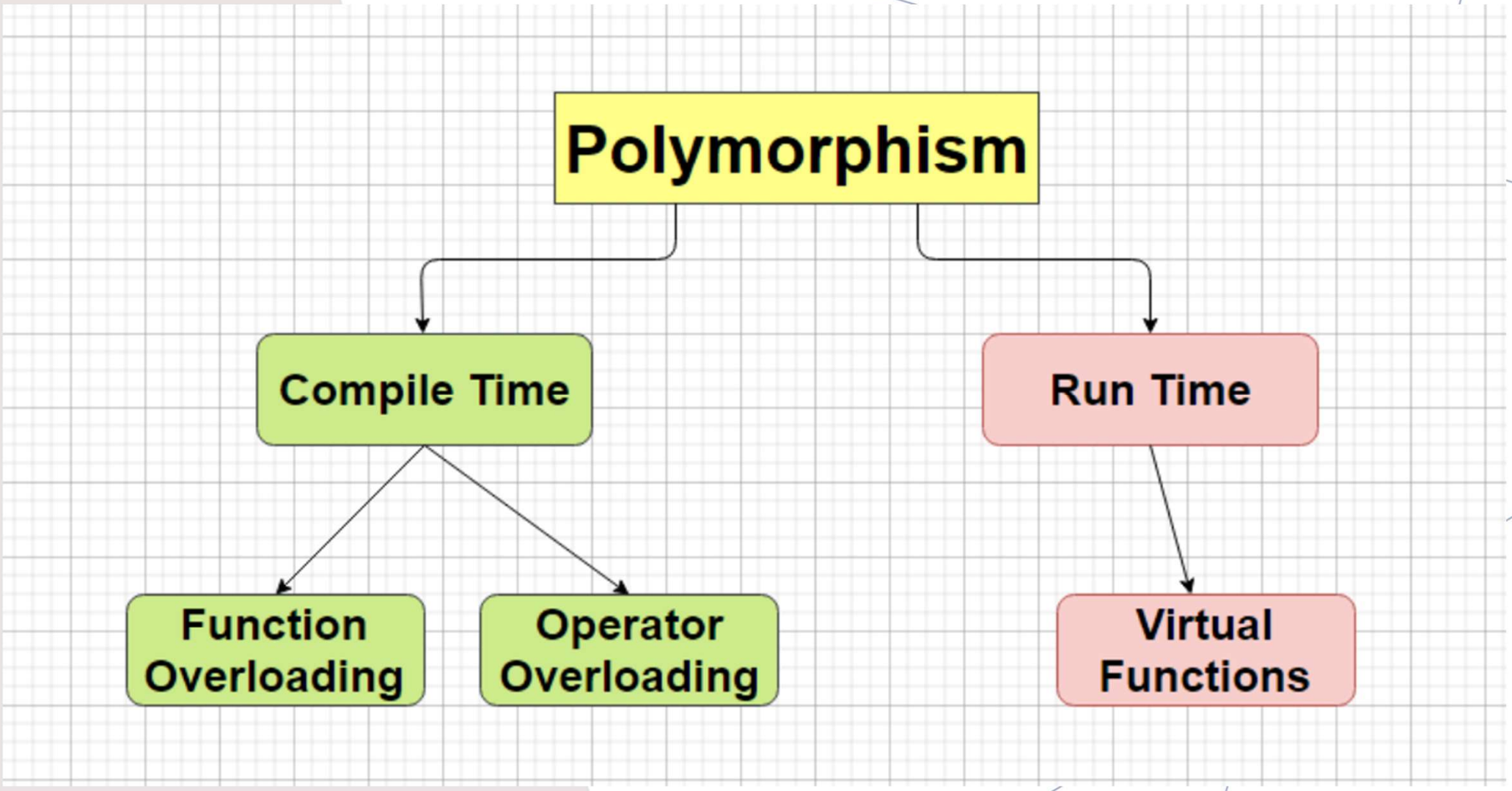
Polymorphism
Compile Time
Run Time
Function Overloading
Operator Overloading
Virtual Functions

VIRTUAL FUNCTIONS

When we use same function name in both the classes – derived and base class. The function in base class is declared as Virtual using the keyword Virtual preceding its normal declaration.

When a function is made Virtual C++ determines which function to use at run time depending on the type of object pointed to by the pointer rather than the type of the pointer

RULES OF VIRTUAL FUNCTION

- Virtual functions must be member of some class
- They cannot be static members
- They are accessed by using object pointers
- A virtual function can be a friend of another class
- A virtual function in the base class must be defined even though it is not used.
- The prototype/signature of the virtual function in the derived class must be same as the functions in the derived classes.
- We cannot have virtual constructors but we can have virtual destructors
- While the base pointer can point to any type of the deived object, the reverse is not true.

VTABLE AND VPTR

- Vtable and VPTR gets created automatically by compiler to track the virtual function calls.
- Vtable : It is a table that contains the memory addresses of all virtual functions of a class in the order in which they are declared in a class. This table is used to resolve function calls in dynamic/late binding manner. Every class that has virtual function will get its own Vtable.
- VPTR : After creating Vtable address of that table gets stored inside a pointer i.e. VPTR (Virtual Pointer). When you create an object of a class which contains virtual function then a hidden pointer gets created automatically in that object by the compiler. That pointer points to a virtual table of that particular class.

VIRTUAL CONSTRUCTORS AND DESTRUCTORS

- Constructors cannot be virtual
  - Creating an object constructor should be same as class which is not possible with virtual implementation
  - At the time of calling the constructor virtual table would not have been created.
- Virtual destructors
  - When a base class pointer is created and made to point to derive class object, delete on the pointer will call the base class destructor.
  - Making the destructor virtual will call the derived class destructor and memory leak will not happen