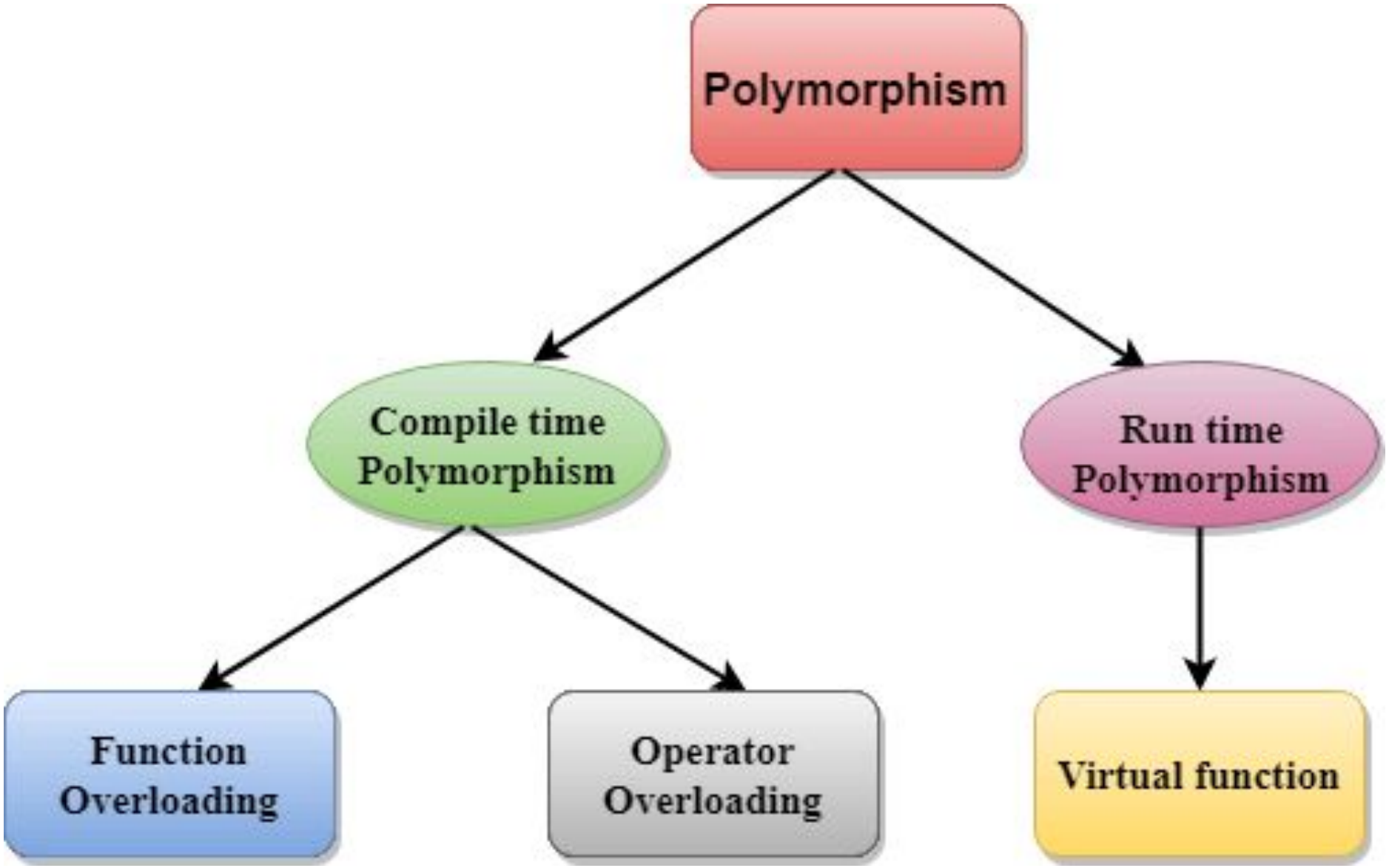
Slide Set 4

Polymorphism: Classification of Polymorphism, Compile time and Run time Polymorphism, Pointers to derived class object, Virtual functions, Pure virtual functions.

Polymorphism

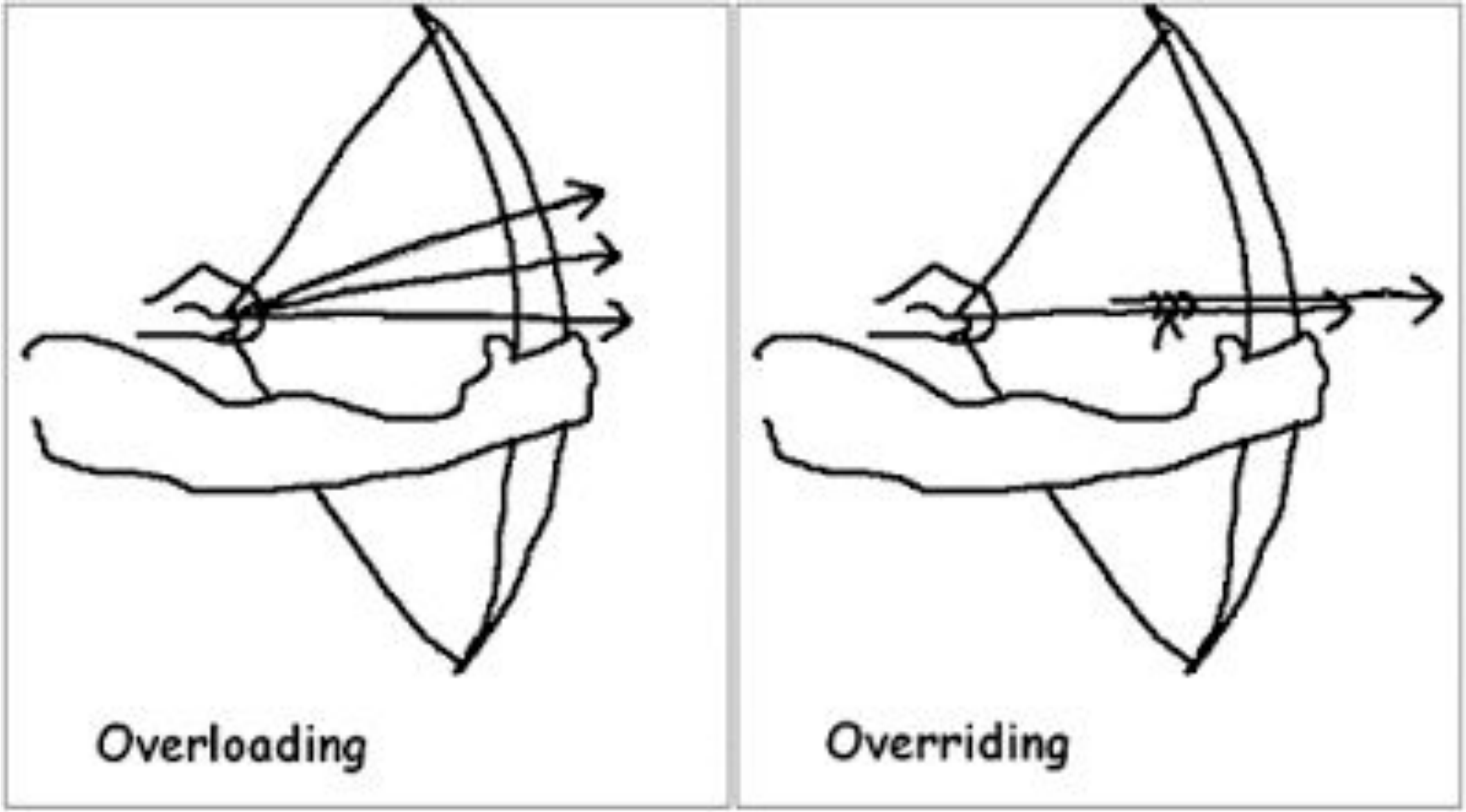
- *Poly* means many and *morph* means form


Polymorphism
Compile time Polymorphism
Run time Polymorphism
Function Overloading
Operator Overloading
Virtual function

Overloading vs overriding

- Function overloading – same function name but different arguments. Functions are defined in the same class.

- Function overriding – same function name and arguments. Defined in different classes.


Overloading
Overriding

Compile time and Run time Polymorphism

- Compile time OR static polymorphism is executed using function overloading.

- Run time polymorphism or dynamic/late binding is done using function overriding and *virtual functions*.


Polymorphism
Compile Time
Overloading
Run Time
Overriding

Virtual function VS pure virtual function

Virtual Function	**Pure Virtual Function**
Has some definition in the class.	NO definition
Syntax: virtual funct_name(parameter_list) {. . . . .}	**Syntax**: virtual funct_name(parameter_list) **= 0;**
Base class can override a virtual function *if required*.	Derived class has to *definitely* override the pure virtual function.

Example – Virtual Function

```
class Base {
public:
virtual void msg() { cout<<" In Base \n";}
};
class Derived: public Base {
public:
void msg() {cout<<"In Derived \n";}
};
int main() {
Derived d; Base *b;
b = &d;  b->msg();      // Output: In Derived
}
```

Example – pure virtual function

```
class Base {
public:
virtual void msg() = 0;
};
class Derived: public Base {
public:
void msg(){ cout << " In derived class"<<endl; }
};
int main() {
Base *b;     Derived obj;
b = &obj;  b->msg(); // Output: In derived class
}
```

Shorter is better (same output for previous programs)

```
int main() {
Base *b;     Derived obj;
b = &obj; b->msg();
}
```

```
 int main() {
Base *b =   new Derived;
b->msg();
delete b;
   }
```