Intermediate/Advanced Computer Programming

Final Exam Review

Unix

Basic OS design

C++/OOP

Unix

Basic commands

- ls
- mkdir
- cat
- Cd

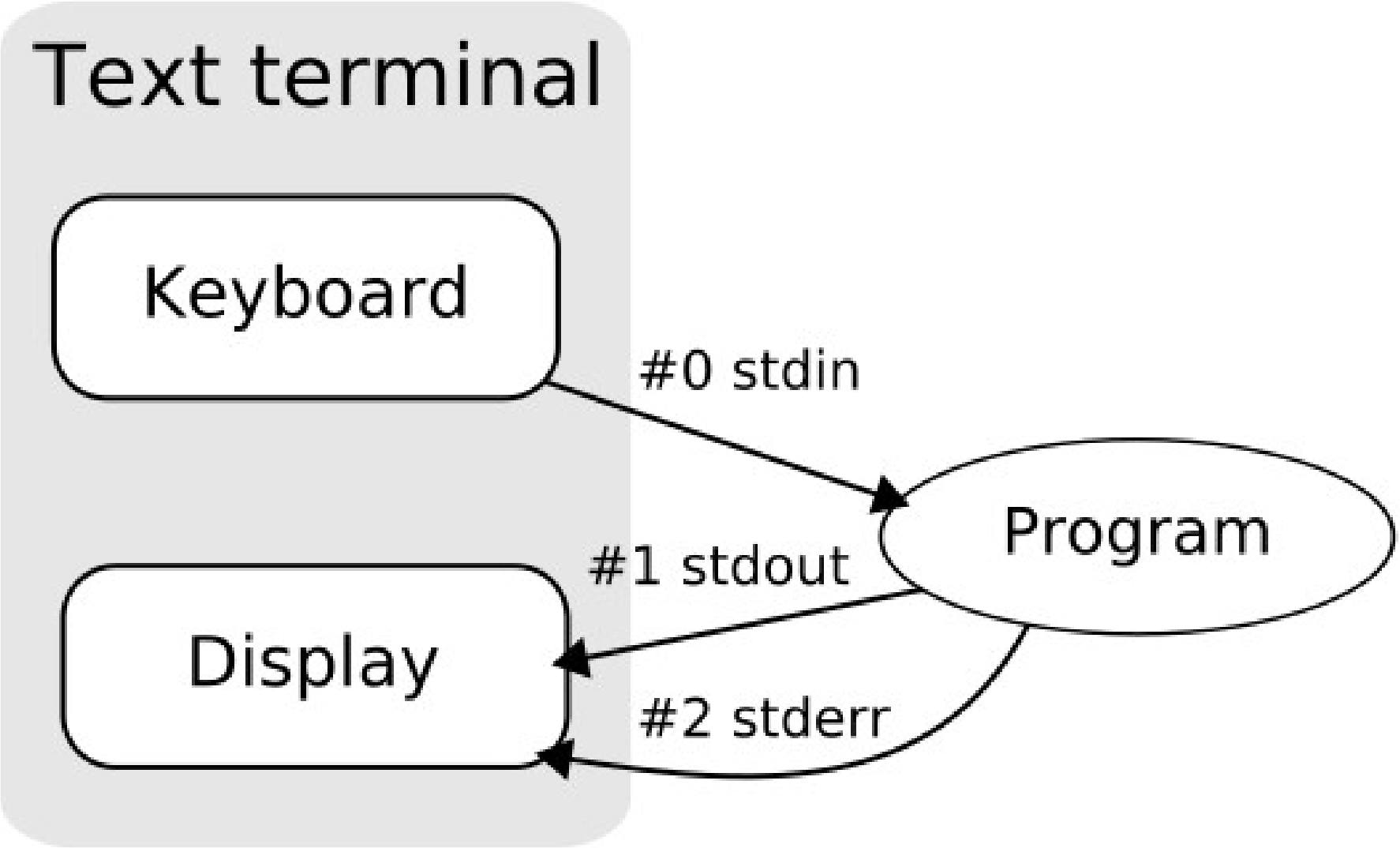
Traversing Filesystem

- Relative/Absolute Paths
- .
- ..
- /

Redirection

- Control standard input/output to/from a program
- > < |

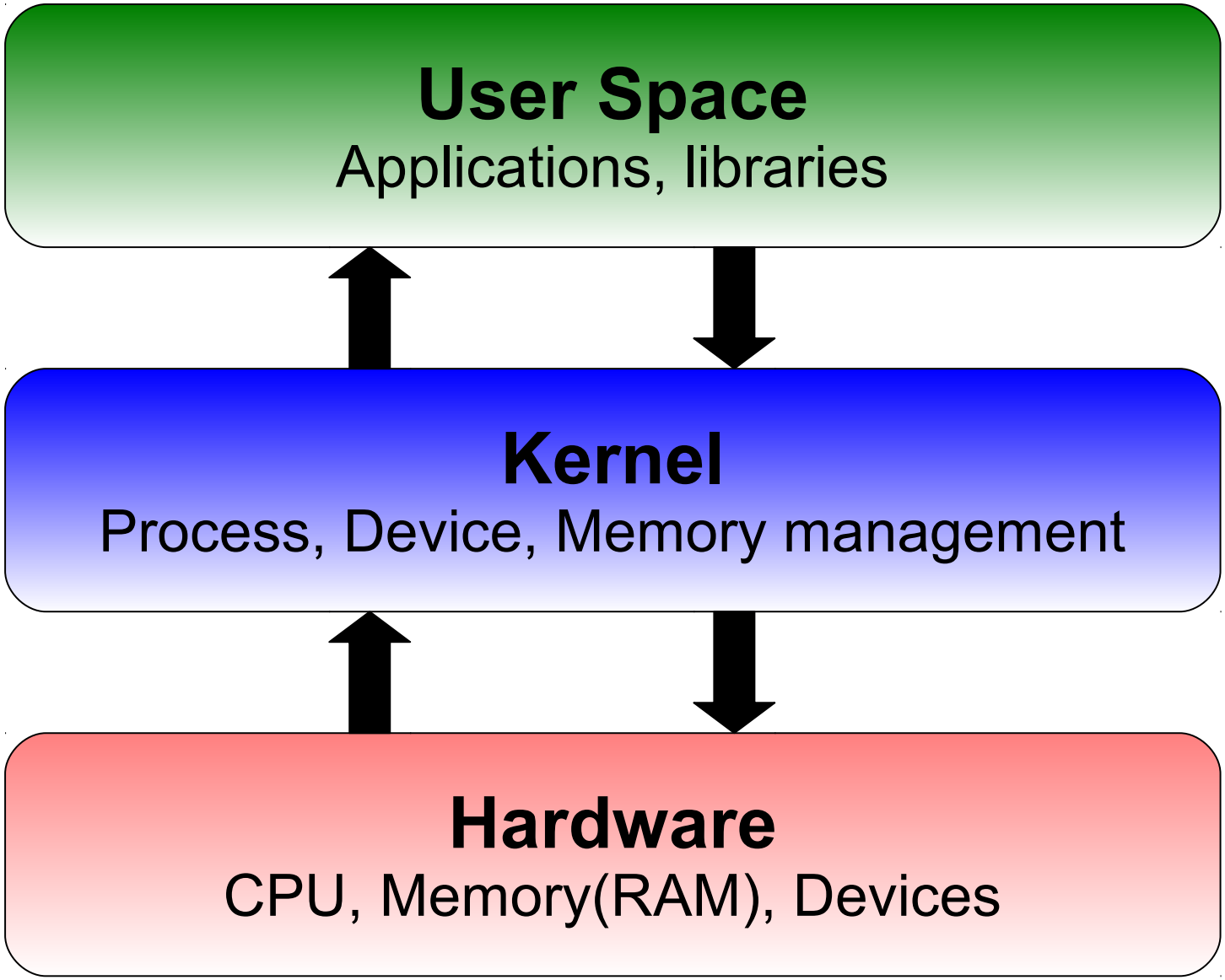
Redirecting/Piping



Unix

- Single UNIX Specification defined by the Open Group
- Standard definition for the Unix system API
  - Specifies requirements for a UNIX system
- Not code
  - Allows different implementations
- Contains POSIX specifications
- Defines system interfaces, headers, commands, and utilities
- http://pubs.opengroup.org/onlinepubs/009695399/

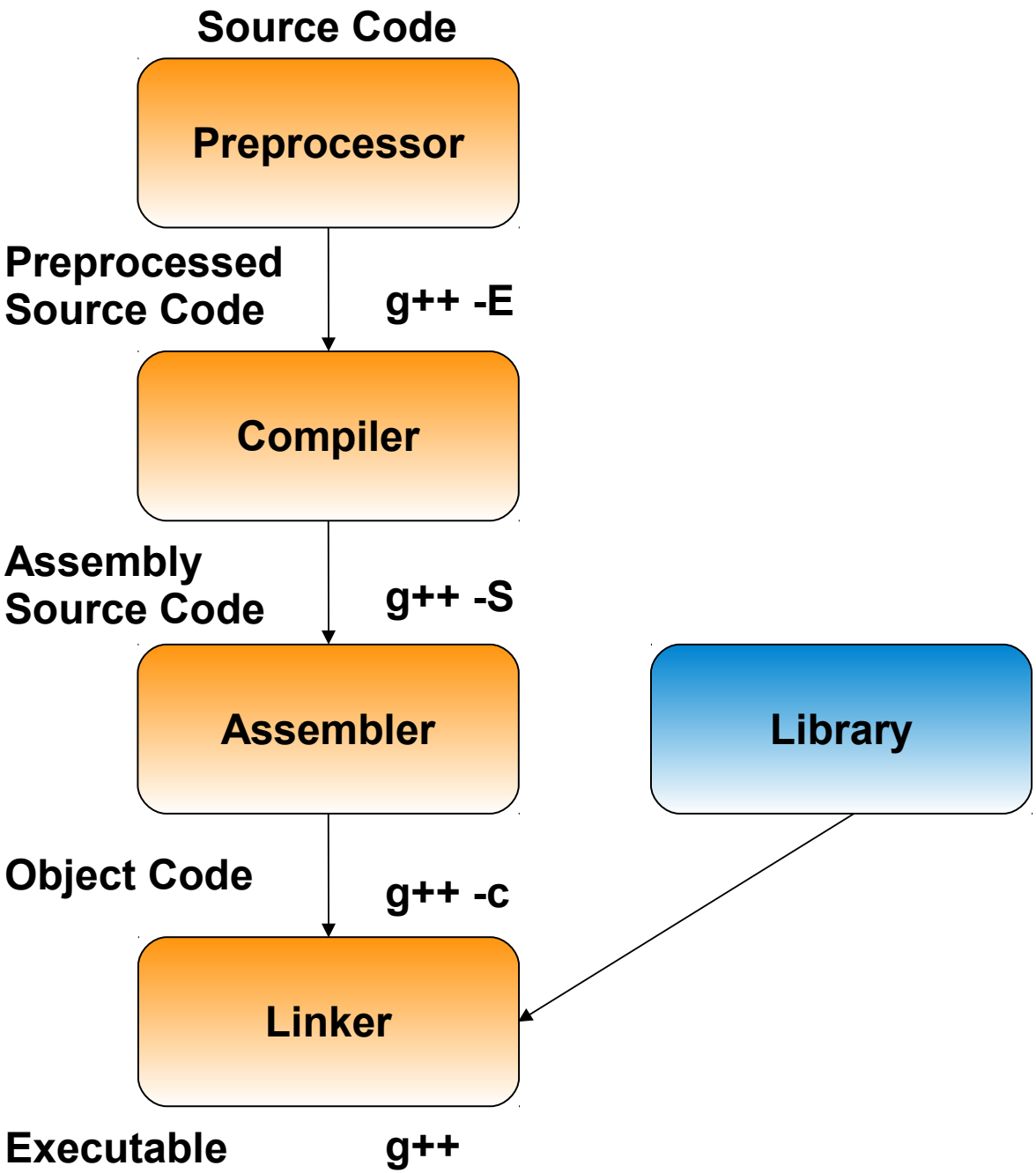
Operating System (OS)



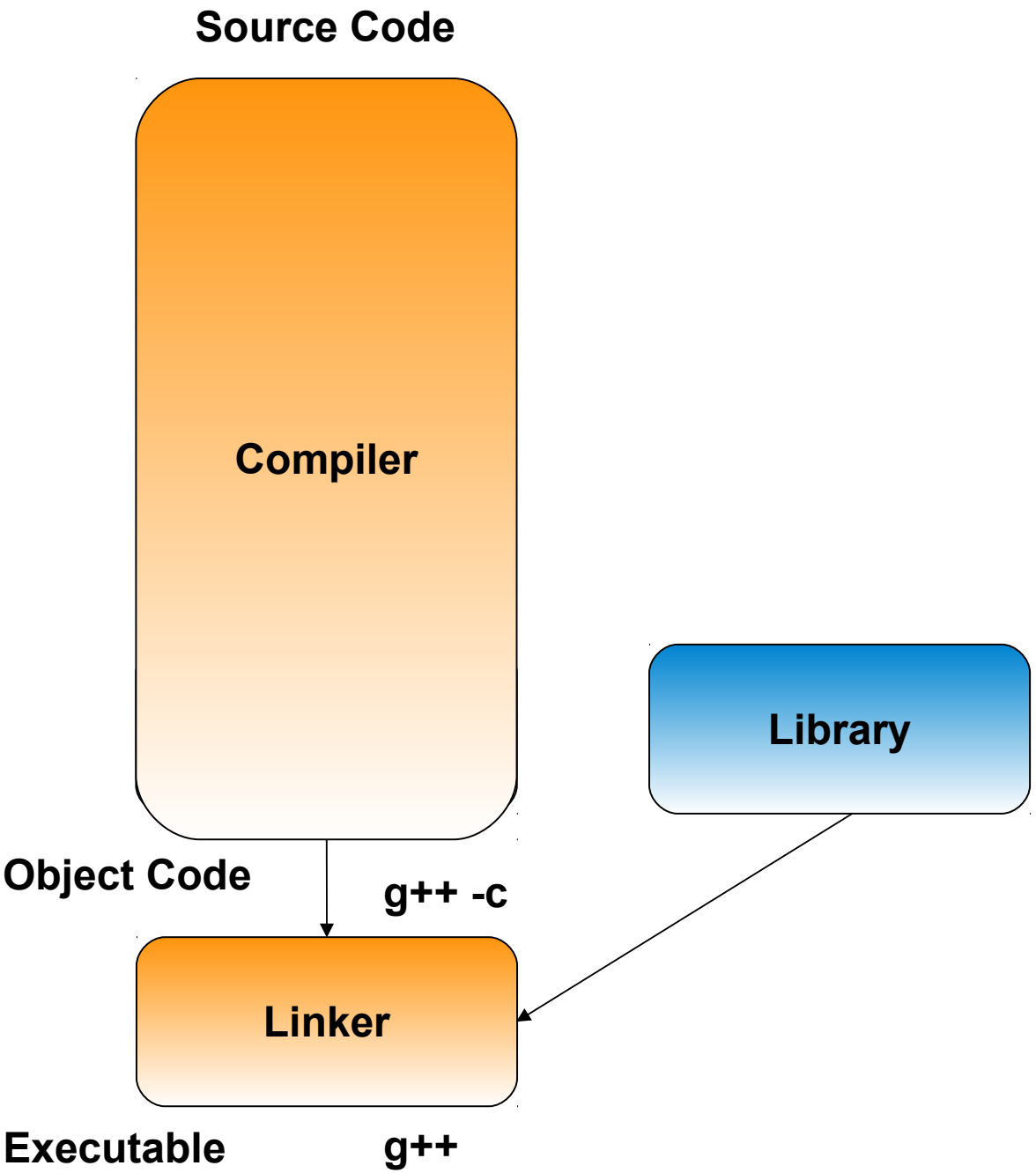
C++/OOP

- Building executables
  - Compiling and linking
- Control structures
  - Loops
  - Conditionals
- Const
- Passing arguments to functions
- Classes and Objects

Building an Executable



Building an Executable



Errors

Compile Time

Link Time

Run Time

Control Structures

Loops

- for, while, do-while
- break
- Continue

Conditionals

- if
- else if
- else

Passing Arguments

Pass by value

Pass by reference

Pass by pointer

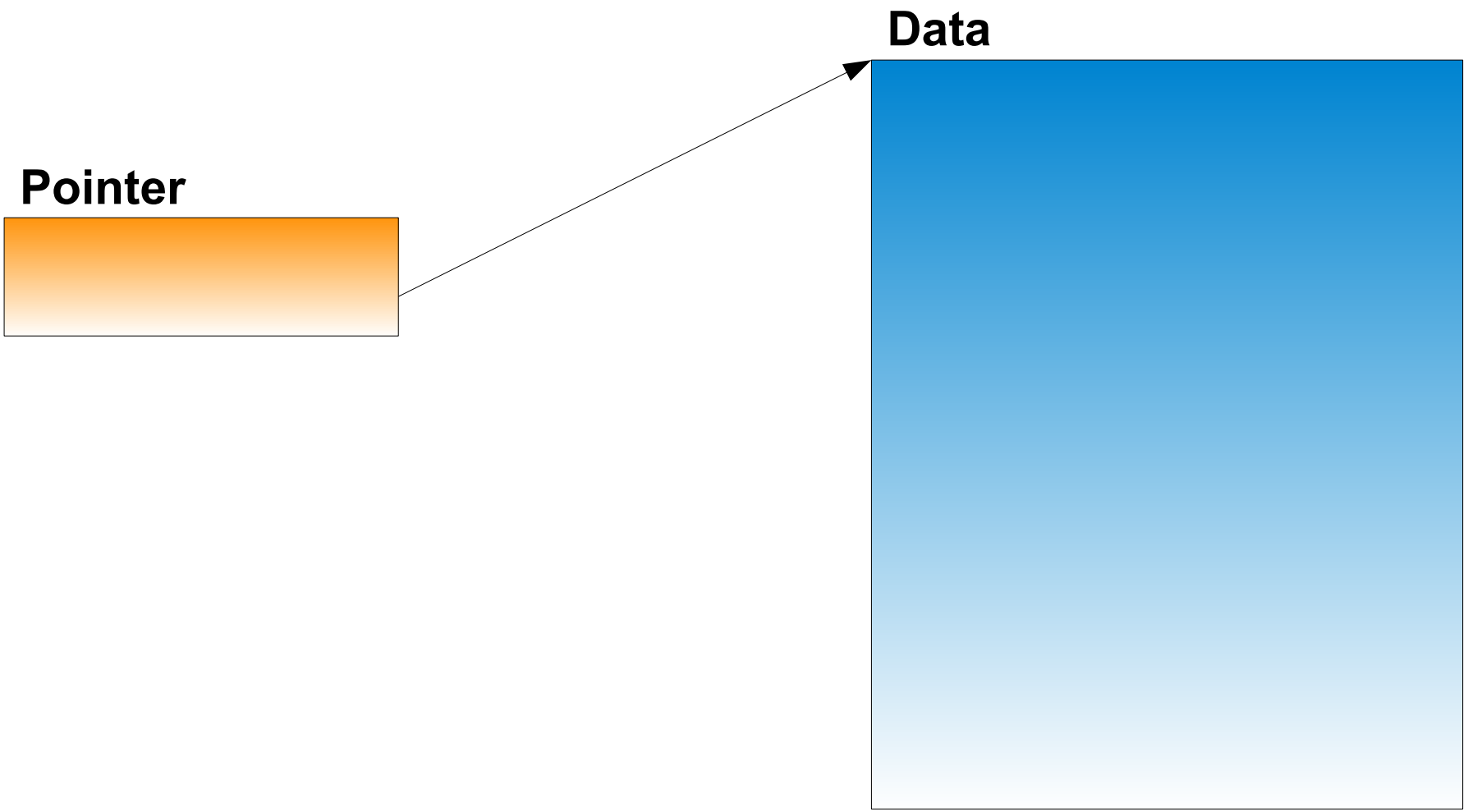
const

Informs compiler that a value/object should not be changed

Must initialize variable when the variable is defined

```
const int x = 5;
//error: uninitialized const 'x'
const int x;
```

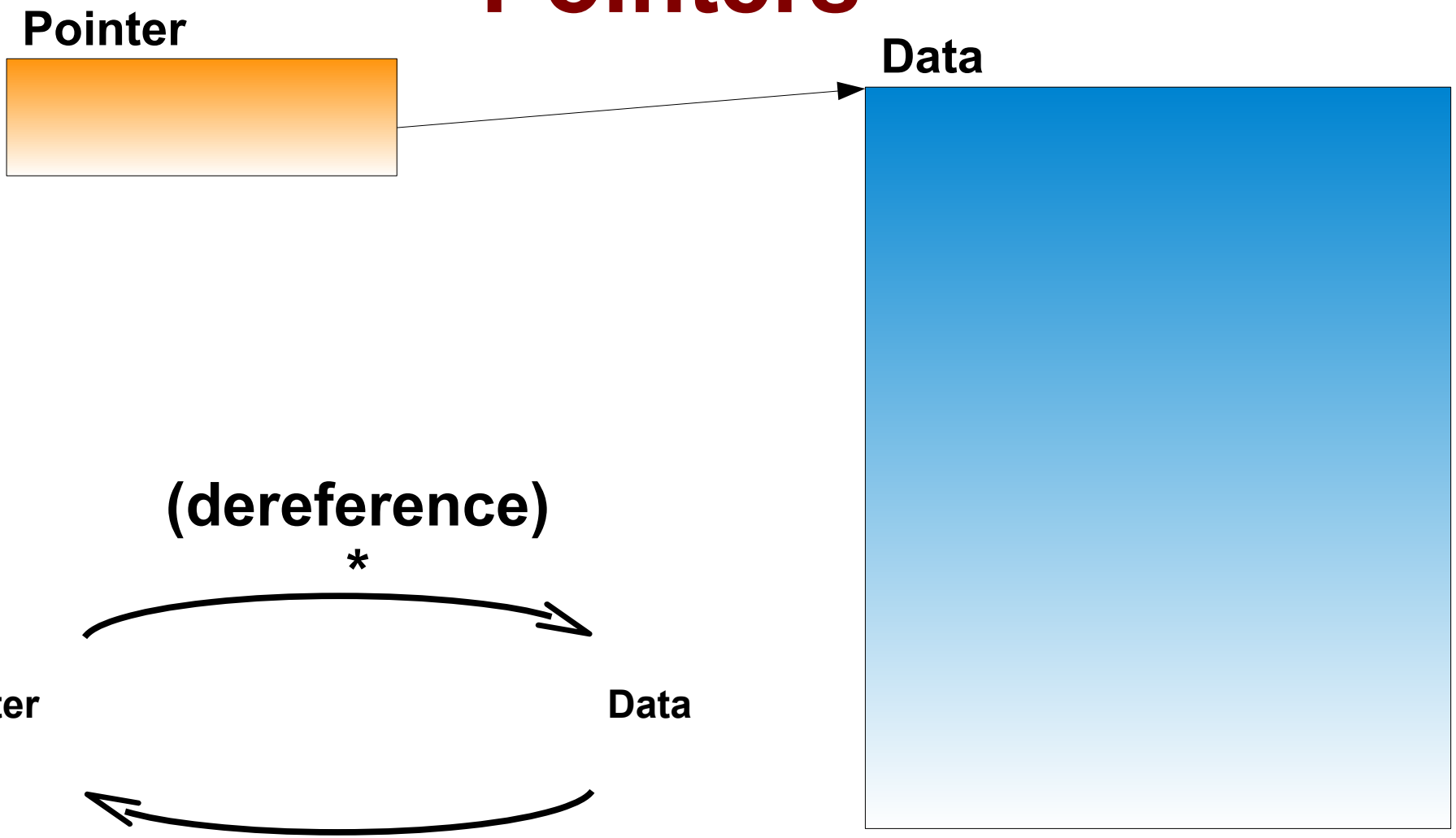
Pointers

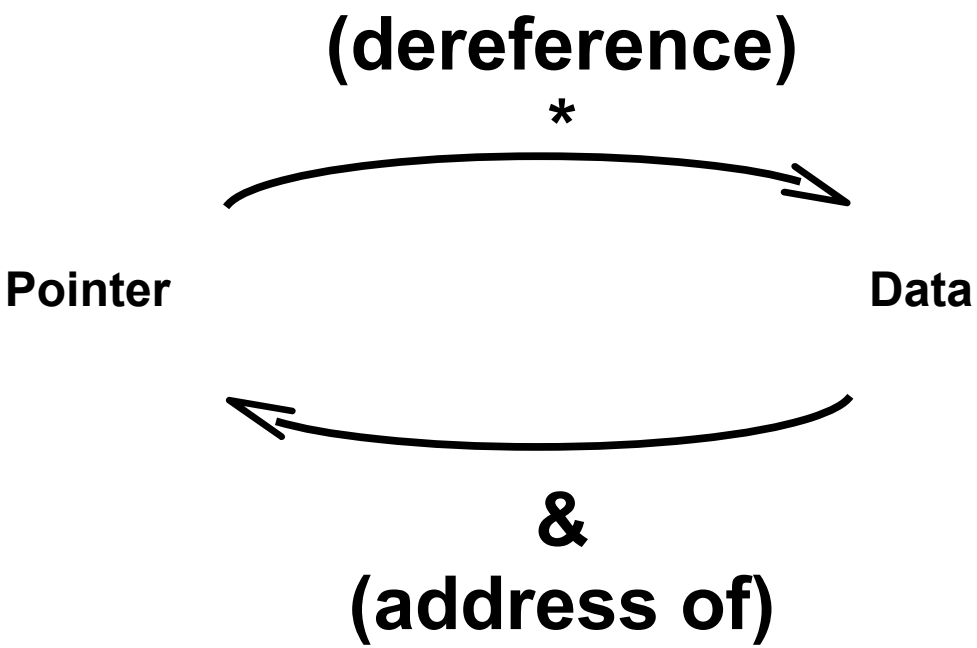


Pointers

```
void add(int a, int b, int *result)
{
    *result = a + b
}
```

Pointers





Pointers

```
int x = 5;
int *x_ptr = &x;

*x = 7;

cout << x << endl;
```

Const with Pointers

```
int x = 5;

[const] int * [const] x_ptr = &x;
```

References

```
void add(int a, int b, int &result)
{
    result = a + b
}
```

OOP

Encapsulation

Inheritance

Polymorphism

OOP

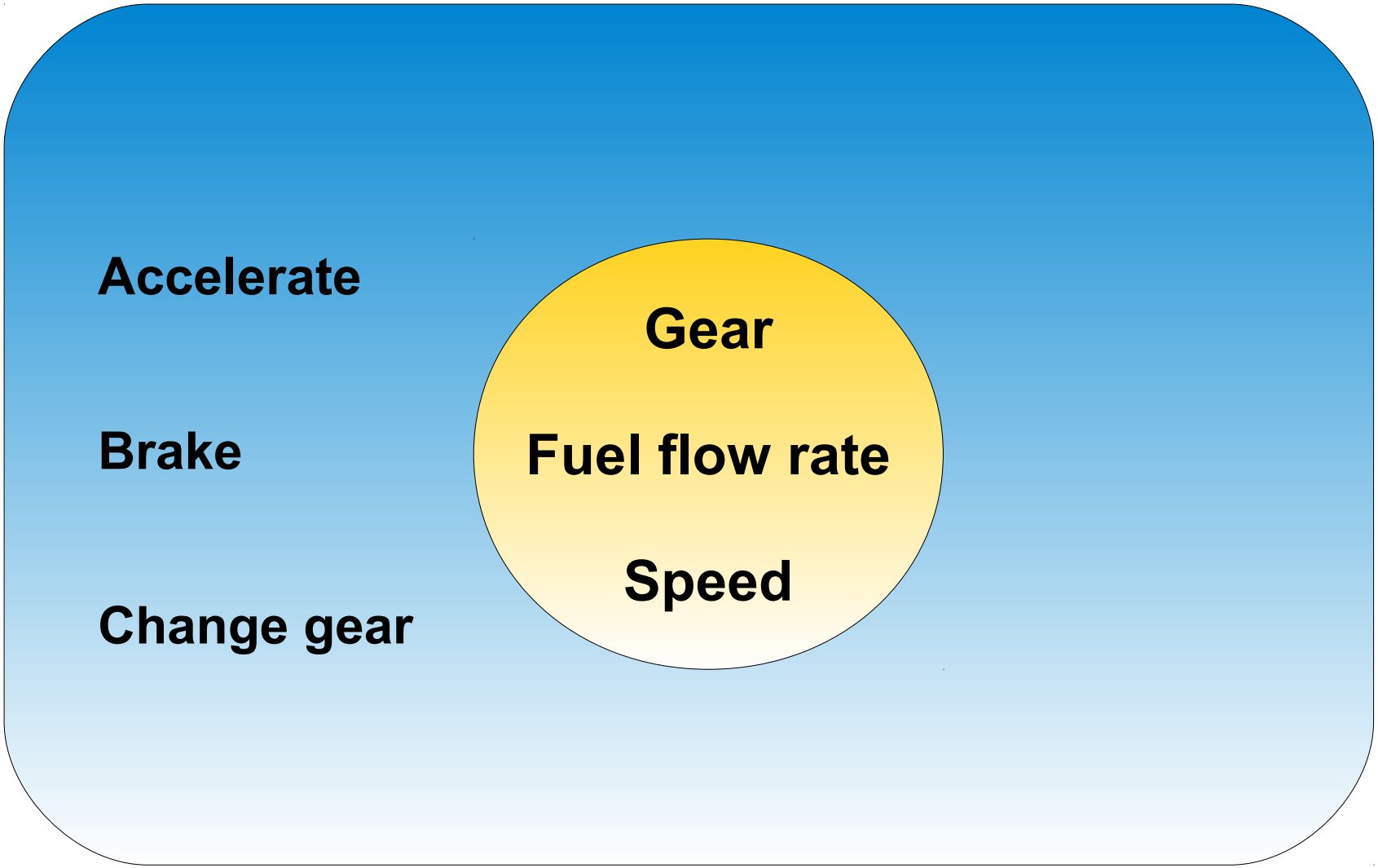
Classes

- “blueprint” for creating an object
- source code describes the class

Object

- Instance of a class
- Have common properties, attributes, operations, behavior with other objects of the same class

Objects



Constructors

Place to ***initialize*** member data and to allocate other resources needed (e.g. memory, files, sockets)

When code in constructor body is called, data members are already allocated

Compiler will generate a default constructor if any constructor is not defined

Constructor

Before calling constructor

Object

data members:

x = 9827432
y = 28374243

Constructor

After calling constructor

Object

data members:

x = 0
y = 0

Types of Constructors

Default

Conversion

Others

Initialization List

Initialize objects/data as they are allocated

Foo:Foo() *:Initialization List* {}

```
engine:engine(): cylinder(8){}
```

Function Overloading

Using the same name for two or more functions

- Different parameter **types**
- Different **number** of parameters
- **Cannot** differ only in the return type
- **Cannot** differ only if one parameter is passed **by-value** and one parameter is passed **by-reference**

```
double sqrt(double x);
double sqrt(double x, int y);
float sqrt(float x);
long double sqrt(long double x);
```

Operator Overloading

```
BigInt a = “55”;
BigInt b = “7”;

BigInt c;

c = a + b;
```

Operator Overloading

Convert operator call into a function call and overload the function call

```
<BigInt> + <BigInt>
```

Operator Overloading

Convert operator call into a function call and overload the function call

```
<BigInt> + <BigInt>

BigInt operator+(BigInt, BigInt);

              - or -

BigInt BigInt::operator+(BigInt);
```

Comparison Operators

Used to compare two objects

```
bool operator==(const X& lhs, const X& rhs)
bool operator!=(const X& lhs, const X& rhs)
bool operator< (const X& lhs, const X& rhs)
bool operator<=(const X& lhs, const X& rhs)
bool operator> (const X& lhs, const X& rhs)
bool operator>=(const X& lhs, const X& rhs)
```

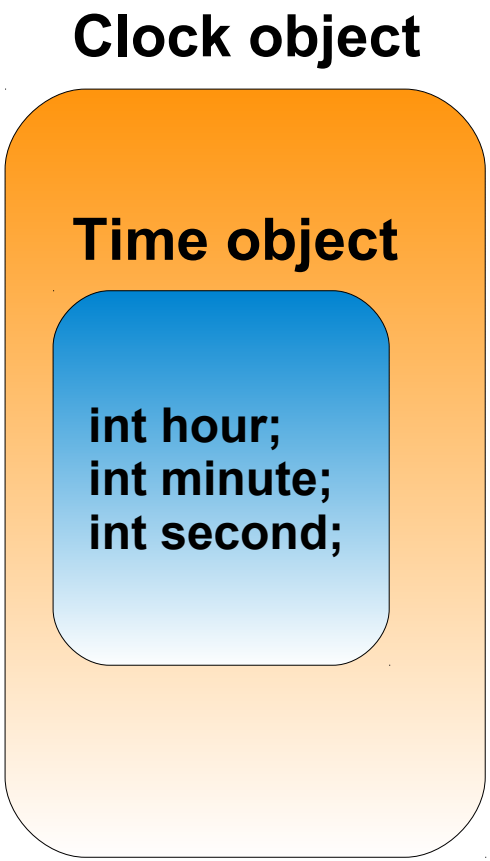
Composition

Classes allow the programmer to define user-defined data types

- User-defined data types allow the programmer to better describe real-world data

Just as objects can contain native data types, objects can also contain other objects

Composition



<u>Time class</u>
IncrSeconds(...);
getTime(...);
GetTime(...);

<u>Clock class</u>
DisplayTime24(...);
DisplayTime12(...);

Inheritance

Allows one to **reuse** and extend existing classes

Base class

- existing class

Derived class

- new class that uses the base class's code

Allows different abstractions

- Is-a-kind-of
- has-a

Inheritance

```
class B : public A
{
    ...
}
```

Protection Levels

Public interface

- For unrelated clients

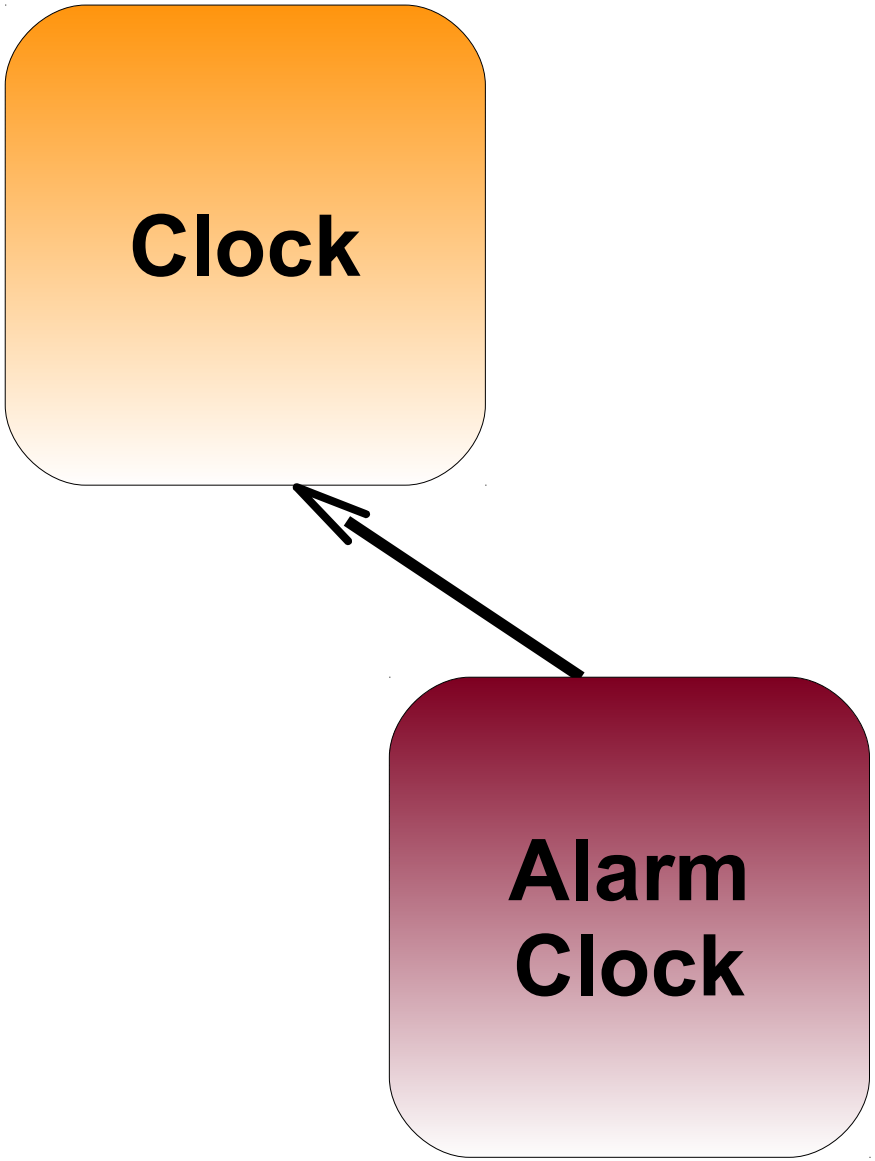
Protected interface

- For clients that inherit from the class

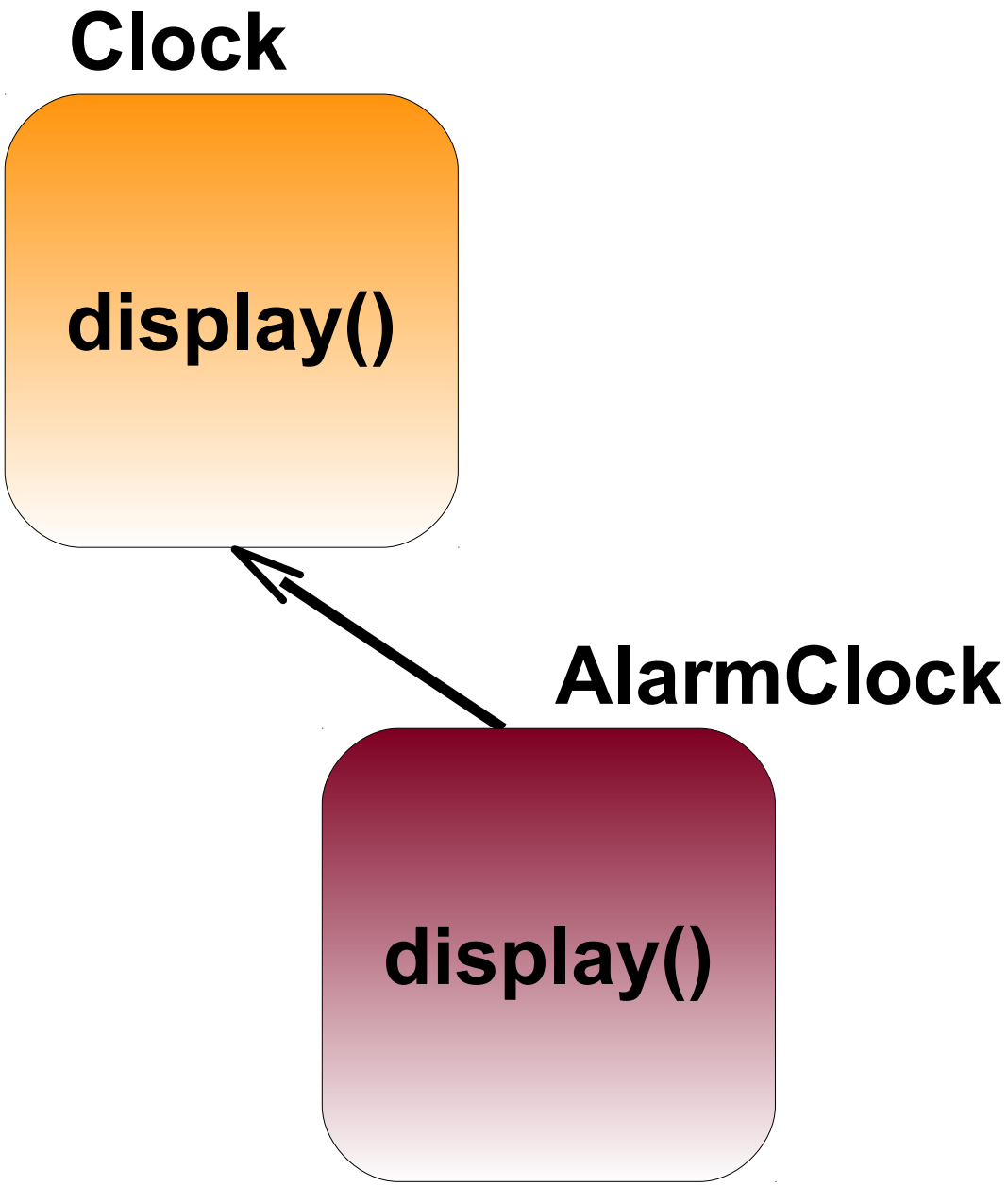
Private interface

- Generally for members/friends of the class

Inheritance



Virtual Functions



Virtual Functions

```
Clock c;
AlarmClock ac;

Clock *myClocks[2];
myClocks[0] = &c;
myClocks[1] = &ac;

myClocks[0].display();
myClocks[1].display();
```

Order of Calling Constructors

Composition

Inheritance

Initialization lists

Calling Base Class Functions

Function()

Base::Function()

Enumeration/Counting

Counting example:

1+2+3+...+n

List all permutations

List number of substrings