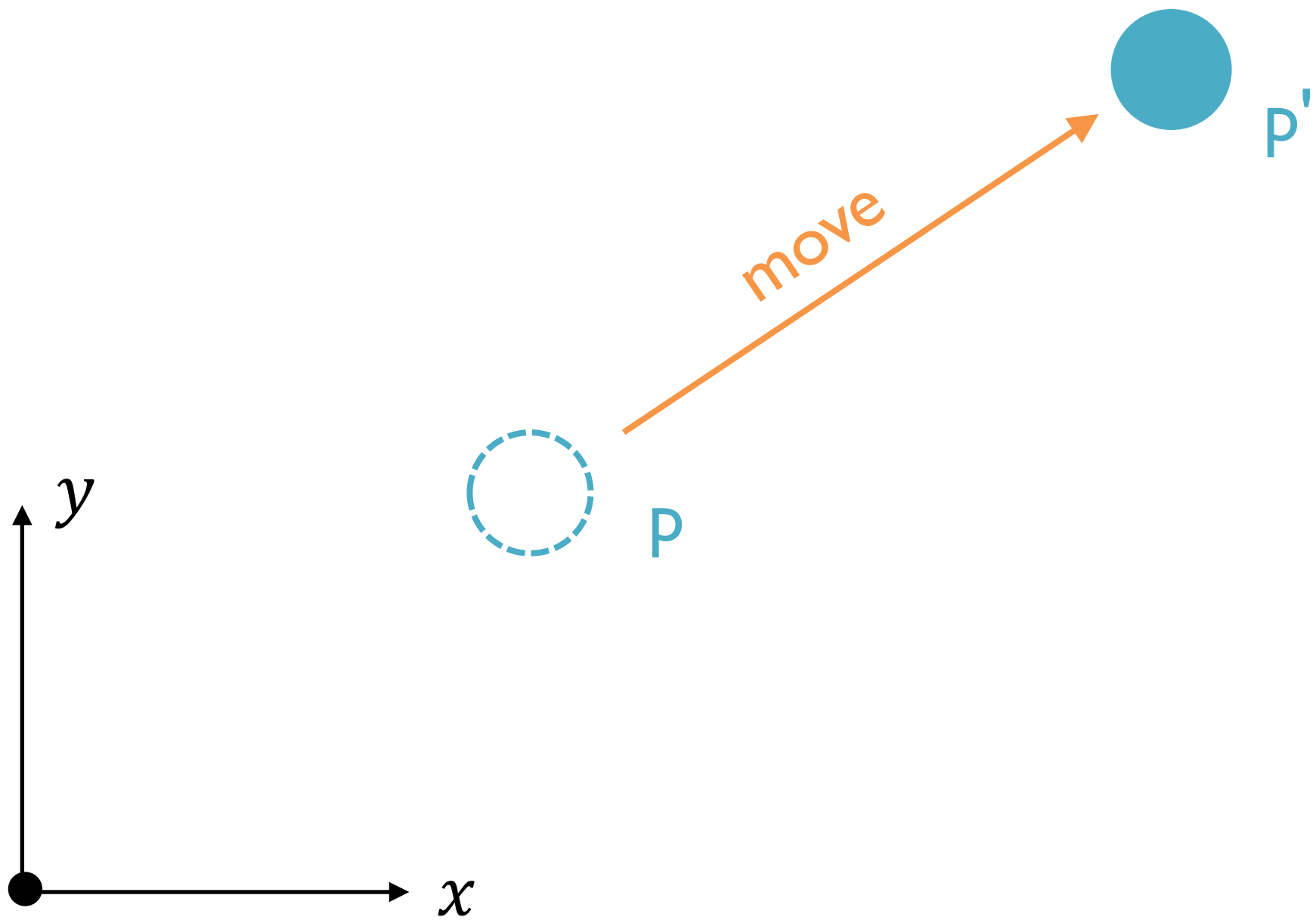


# Class

Object-Oriented Programming with C++



# Point

```
typedef struct point {  
    int x;  
    int y;  
} Point;
```

# Point

```
typedef struct point {  
    int x;  
    int y;  
} Point;
```

```
Point a;
```

```
a.x = 1;a.y = 2;
```

# Point

```
typedef struct point {  
    int x;  
    int y;  
} Point;  
  
Point a;  
a.x = 1;a.y = 2;  
void print(const Point* p) {  
    printf("%d %d\n", p->x, p->y);  
}
```

# Point

```
typedef struct point {  
    int x;  
    int y;  
} Point;  
  
Point a;  
a.x = 1;a.y = 2;  
void print(const Point* p) {  
    printf("%d %d\n", p->x, p->y);  
}  
print(&a);
```

move (dx, dy)?

# move (dx, dy)?

```
void move(Point* p, int dx, int dy) {  
    p->x += dx;  
    p->y += dy;  
}
```



# Prototypes

```
typedef struct point {  
    int x;  
    int y;  
} Point;
```

```
void print(const Point* p);  
void move(Point* p, int dx, int dy);
```

# Usage

```
Point a;  
Point b;  
a.x = b.x = 1; a.y = b.y = 1;  
move(&a, 2, 2);  
print(&a);  
print(&b);
```

# C++ version

```
class Point {  
public:  
    void init(int x, int y);  
    void move(int dx, int dy);  
    void print() const;  
  
private:  
    int x;  
    int y;  
} ;
```

# Implementations

```
void Point::init(int ix, int iy) {  
    x = ix; y = iy;  
}  
void Point::move(int dx, int dy) {  
    x+= dx; y+= dy;  
}  
void Point::print() const {  
    cout << x << ' ' << y << endl;  
}
```

# C vs. C++

```
typedef struct point {
    int x;
    int y;
} Point;

void print(const Point* p);
void move(Point* p,
          int dx,
          int dy);

Point a;
a.x = 1;
a.y = 2;
move(&a, 2, 2);
print(&a);
```

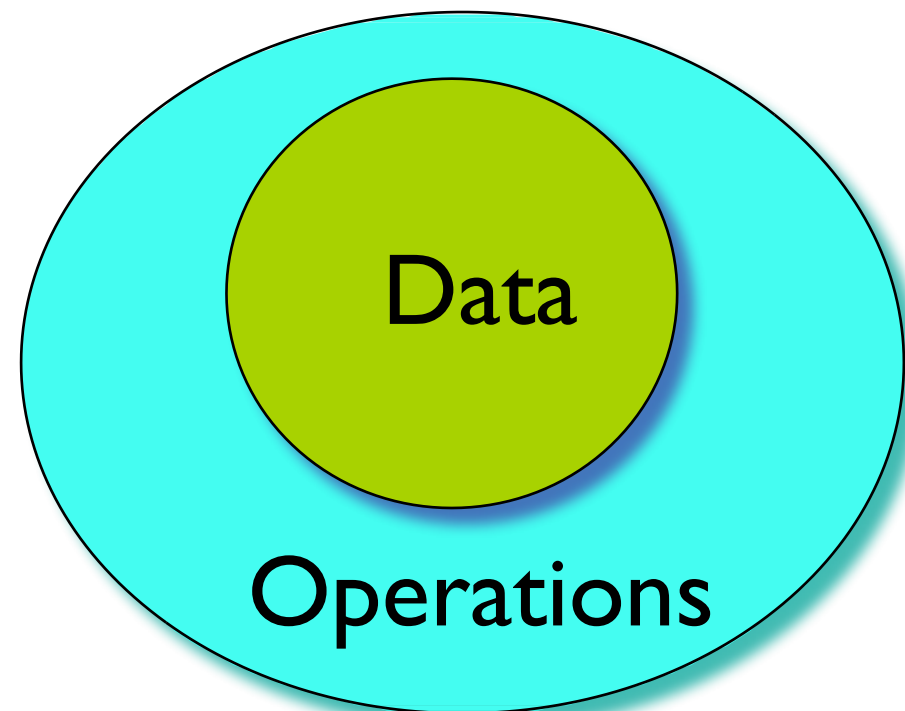
```
class Point {
public:
    void init(int x, int y);
    void print() const;
    void move(int dx, int dy);

private:
    int x;
    int y;
};

Point a;
a.init(1, 2);
a.move(2, 2);
a.print();
```

# Objects = Attributes + Services

- Data: the properties or status
- Operations: the functions



# Ticket Machine

- Ticket machines print a ticket when a customer inserts the correct money for their fare.
- Our ticket machines work by customers' inserting money into them, and then requesting a ticket to be printed. A machine keeps a running total of the amount of money it has collected throughout its operation.



# Something is there





# Something is there

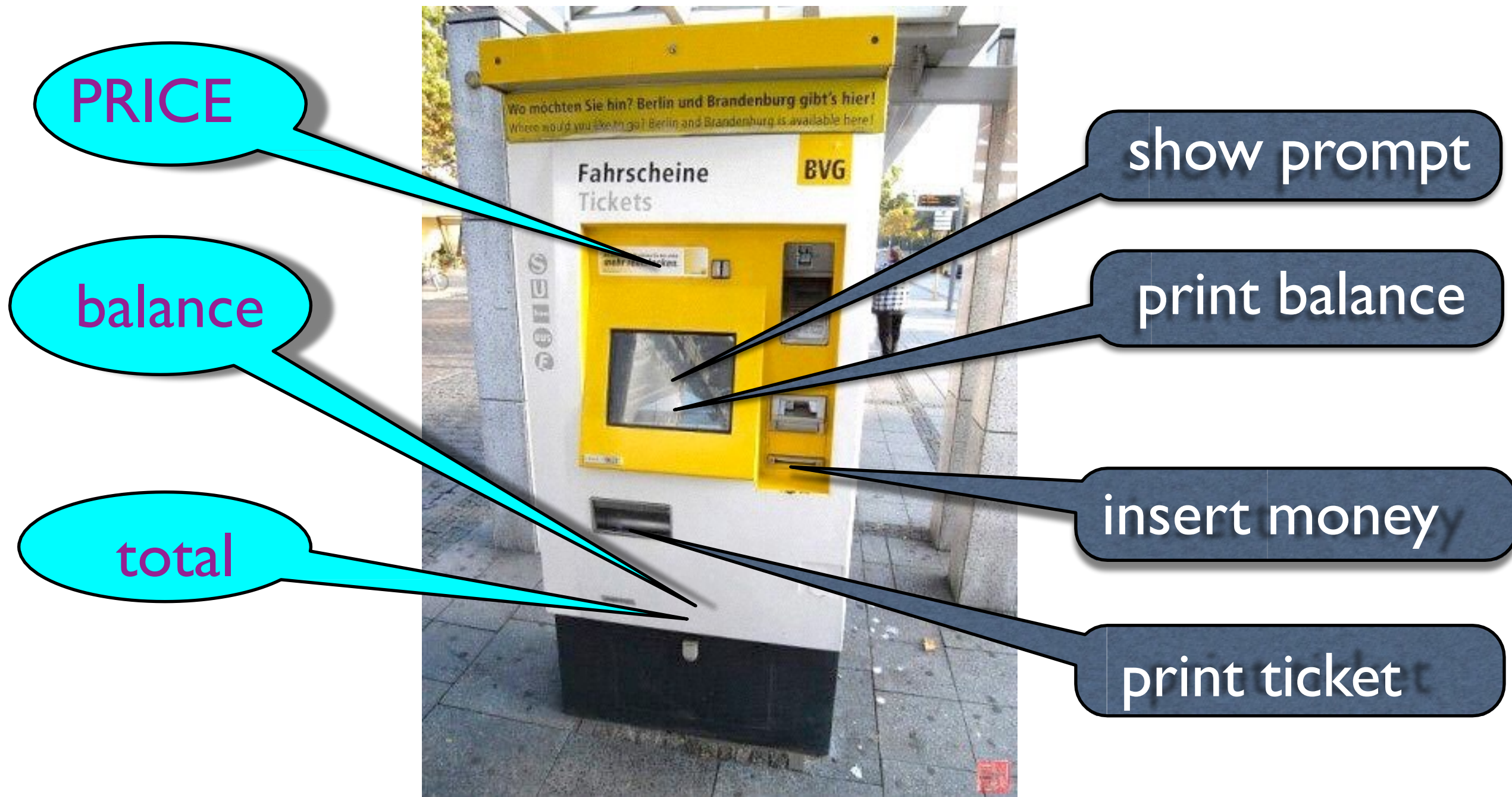
PRICE

balance

total



# Something is there



# Something is there

TicketMachine
PRICE balance total
showPrompt getMoney printTicket showBalance printError

# Something is there

TicketMachine	
PRICE	
balance	
total	
showPrompt	
getMoney	
printTicket	
showBalance	
printError	

ticketMachine 1:  
TicketMachine

PRICE

balance

total



# Turn it into code

TicketMachine

PRICE

```
class TicketMachine {  
    private:  
        const int PRICE;  
        int balance;  
        int total;  
};
```

showBalance  
printError

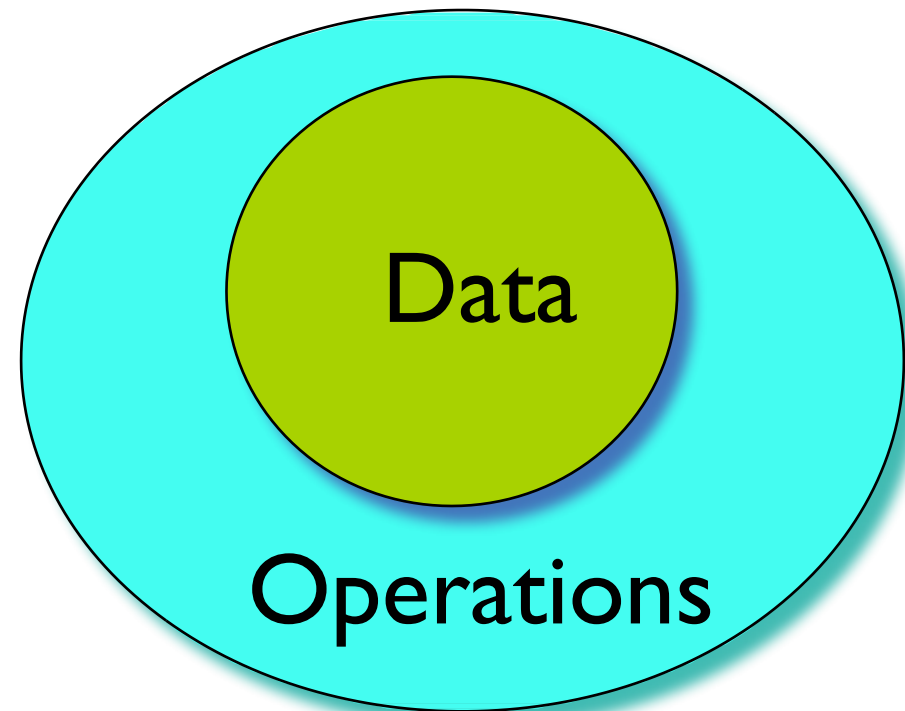
ticketMachine I:

# Turn it into code

```
class TicketMachine {  
public:  
    void showPrompt();  
    void getMoney();  
    void printTicket();  
    void showBalance();  
    void printError();  
private:  
    const int PRICE;  
    int balance;  
    int total;  
};
```

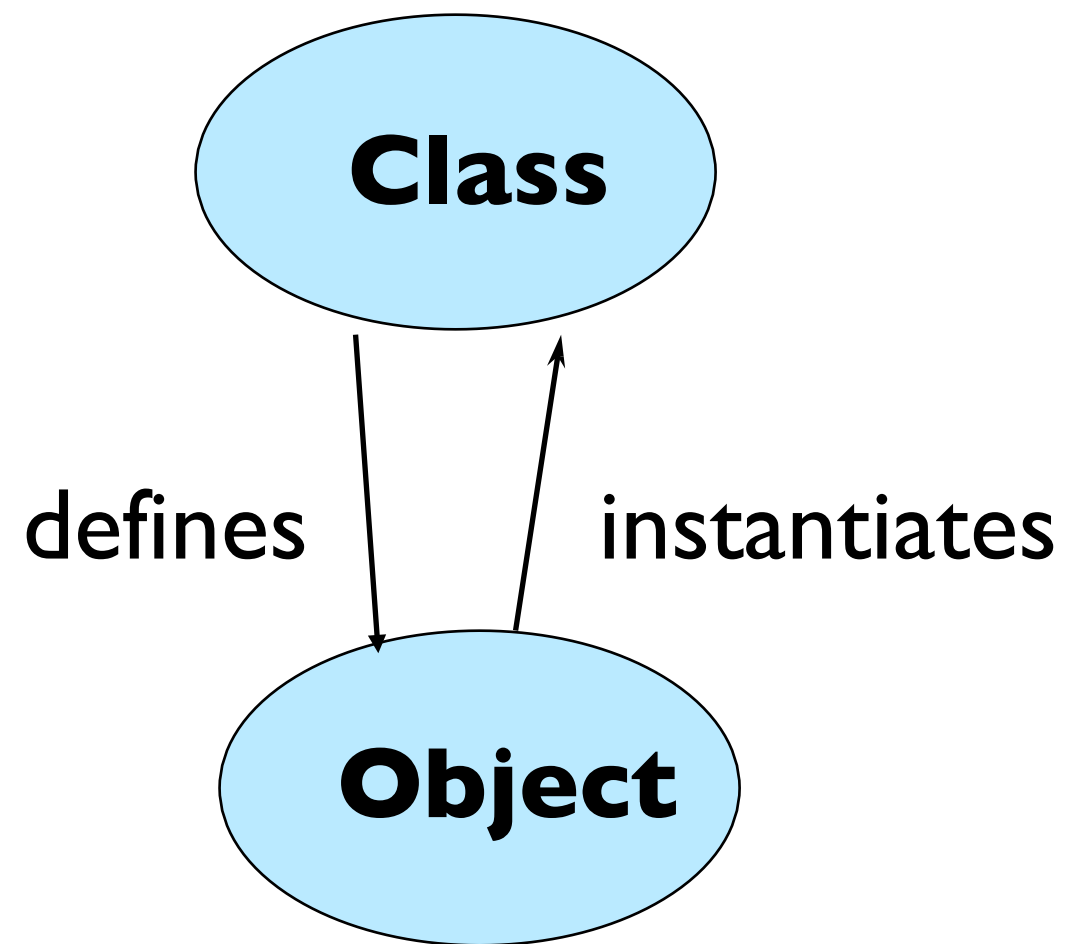
# Objects = Attributes + Services

- Data: the properties or status
- Operations: the functions



# Object vs. Class

- Objects (cat)
  - Represent things, events
  - Respond to messages at run-time
- Classes (cat class)
  - Define properties of instances
  - Act like types in C++





# OOP Characteristics

1. Everything is an object.
2. A program is a bunch of objects telling each other what to do by sending messages.
3. Each object has its own memory made up of other objects.
4. Every object has a type.
5. All objects of a particular type can receive the same messages.

# Definition of a Class

- In C++, separated .h and .cpp files are used to define one class.
- Class declaration and member function prototypes are in the header file (.h).
- All the bodies of these functions are in the source file (.cpp).

# :: resolver

- <Class Name>::<function name>
- ::<function name>

```
void S::f() {  
    ::f(); // Would be recursive otherwise!  
    ::a++; // Select the global 'a'  
    a--;   // The 'a' at class scope  
}
```

# Compilation unit

- The compiler sees only one .cpp file, and generates .obj file
- The linker links all .obj into one executable file
- To provide information about functions in other .cpp files, use .h

# The header files

- If a function is declared in a header file, you *must* include the header file everywhere the function is used and where the function is defined.
- If a class is declared in a header file, you *must* include the header file everywhere the class is used and where class member functions are defined.

# Header = interface

- The header is a contract between you and the user of your code.
- The compiler enforces the contract by requiring you to declare all structures and functions before they are used.

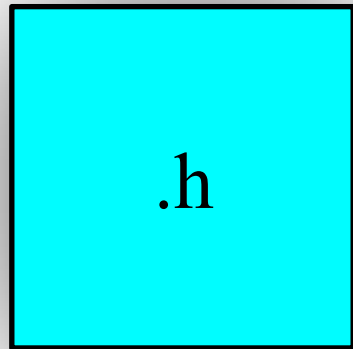
# Structure of C++ program



.h

# Structure of C++ program

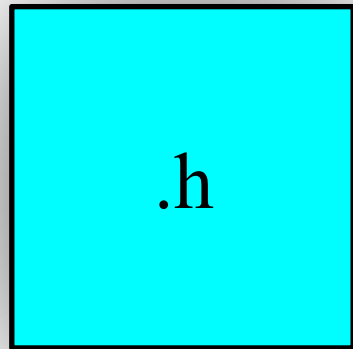
declarations



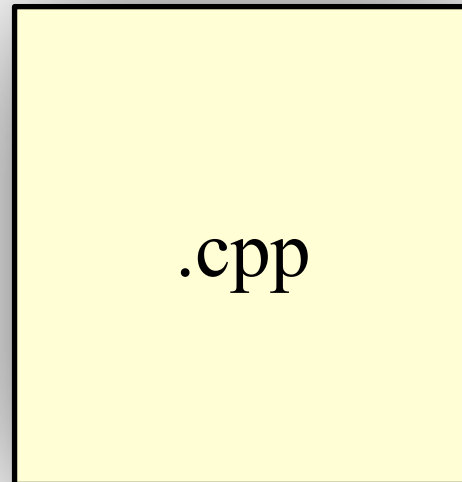


# Structure of C++ program

declarations

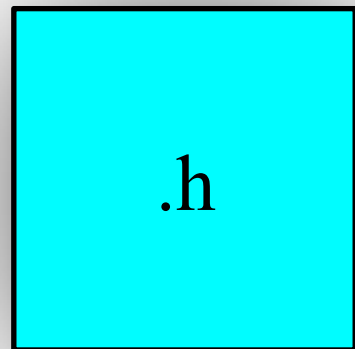


definitions

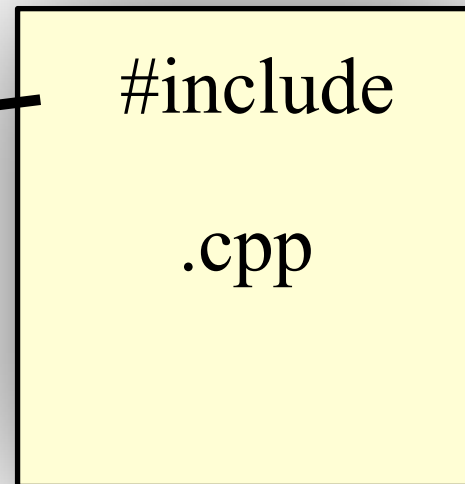


# Structure of C++ program

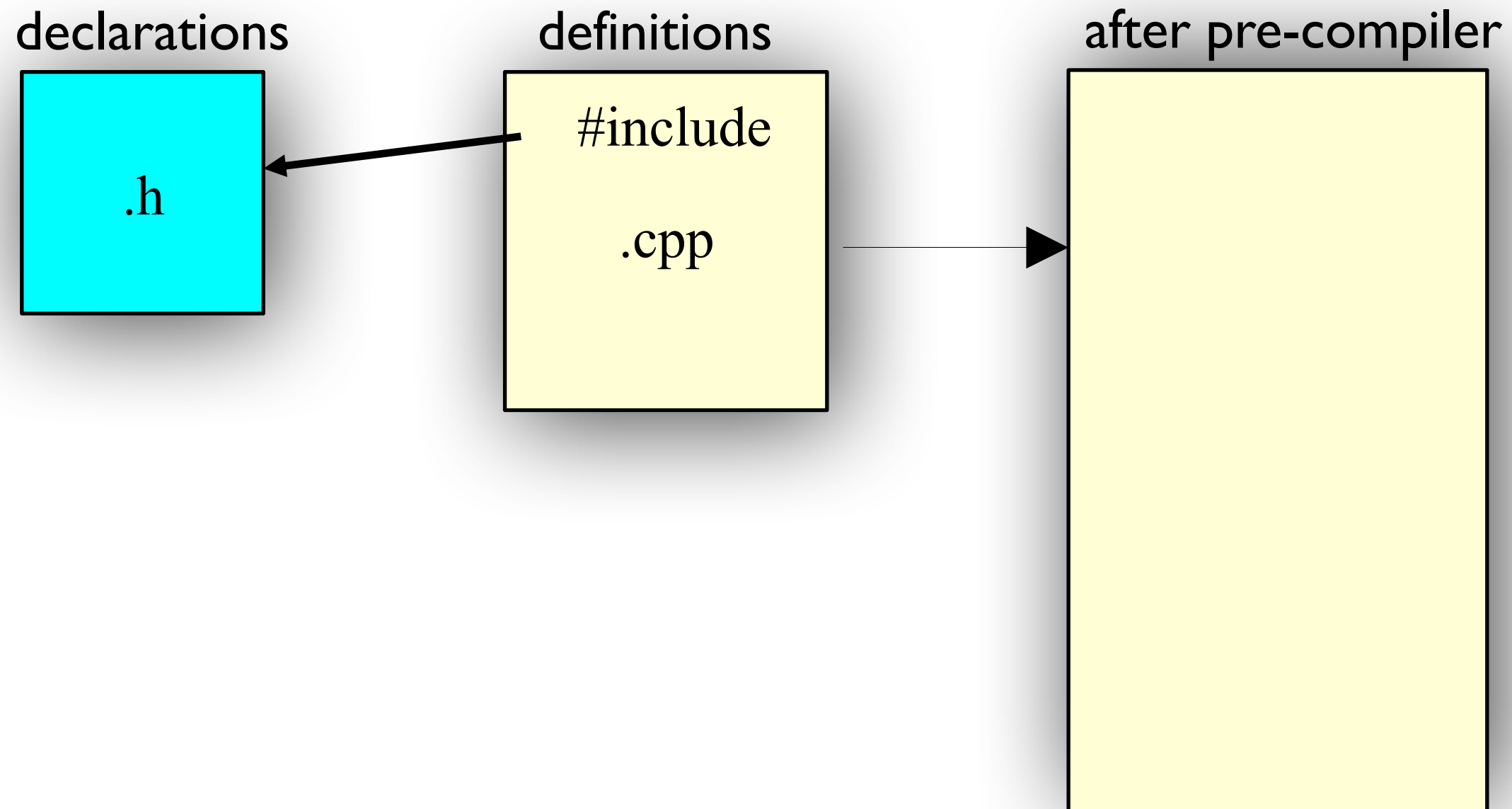
declarations



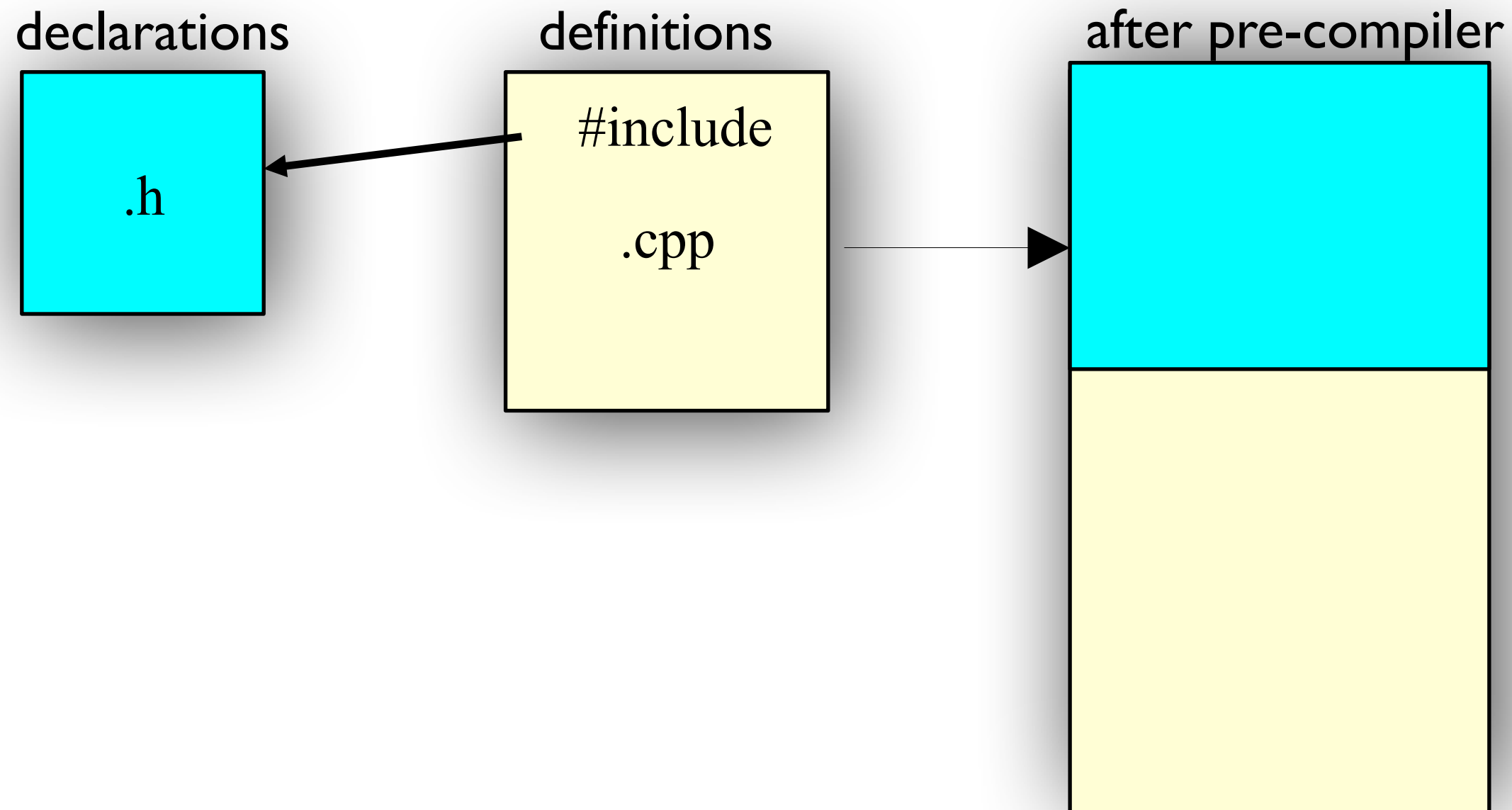
definitions



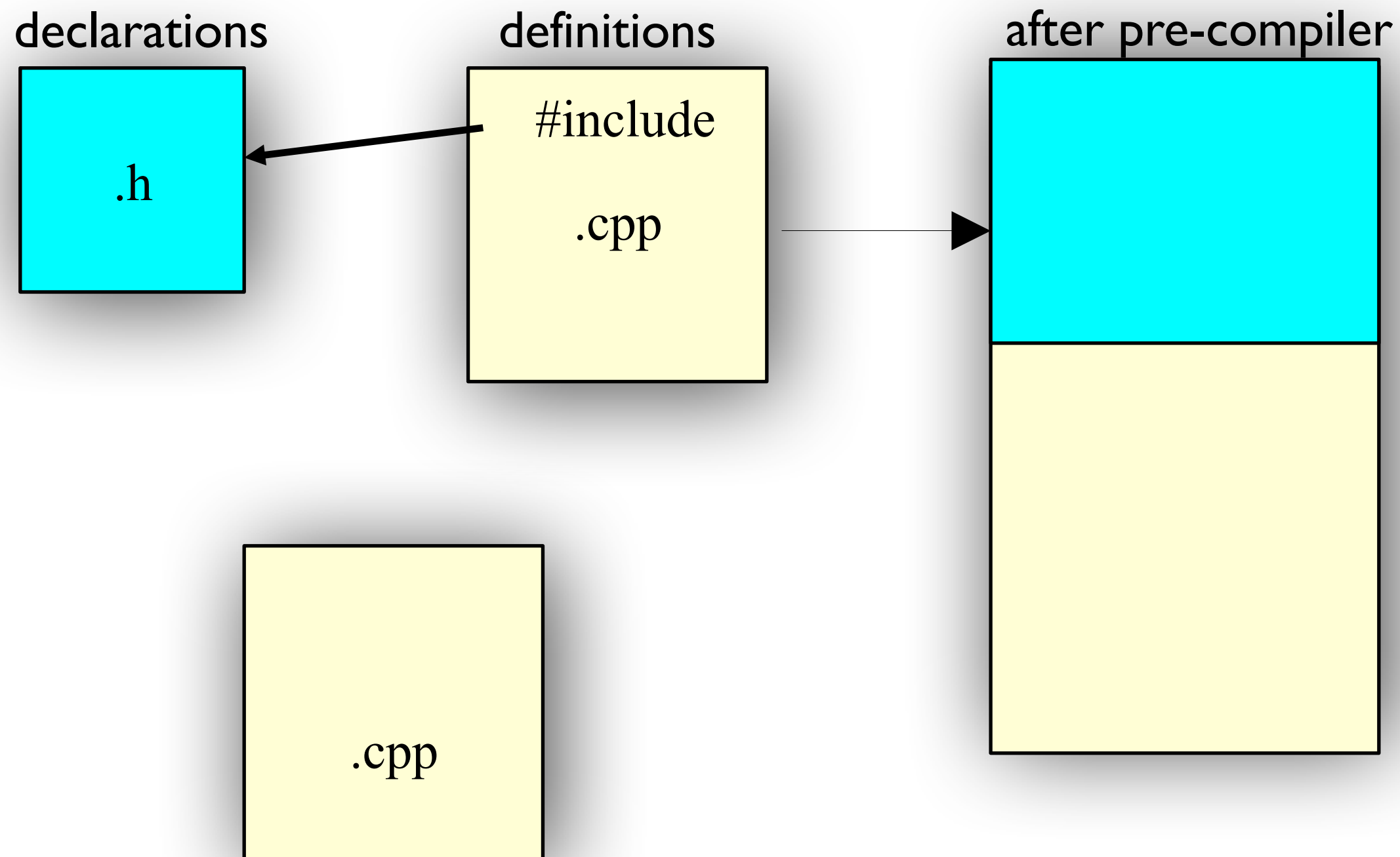
# Structure of C++ program



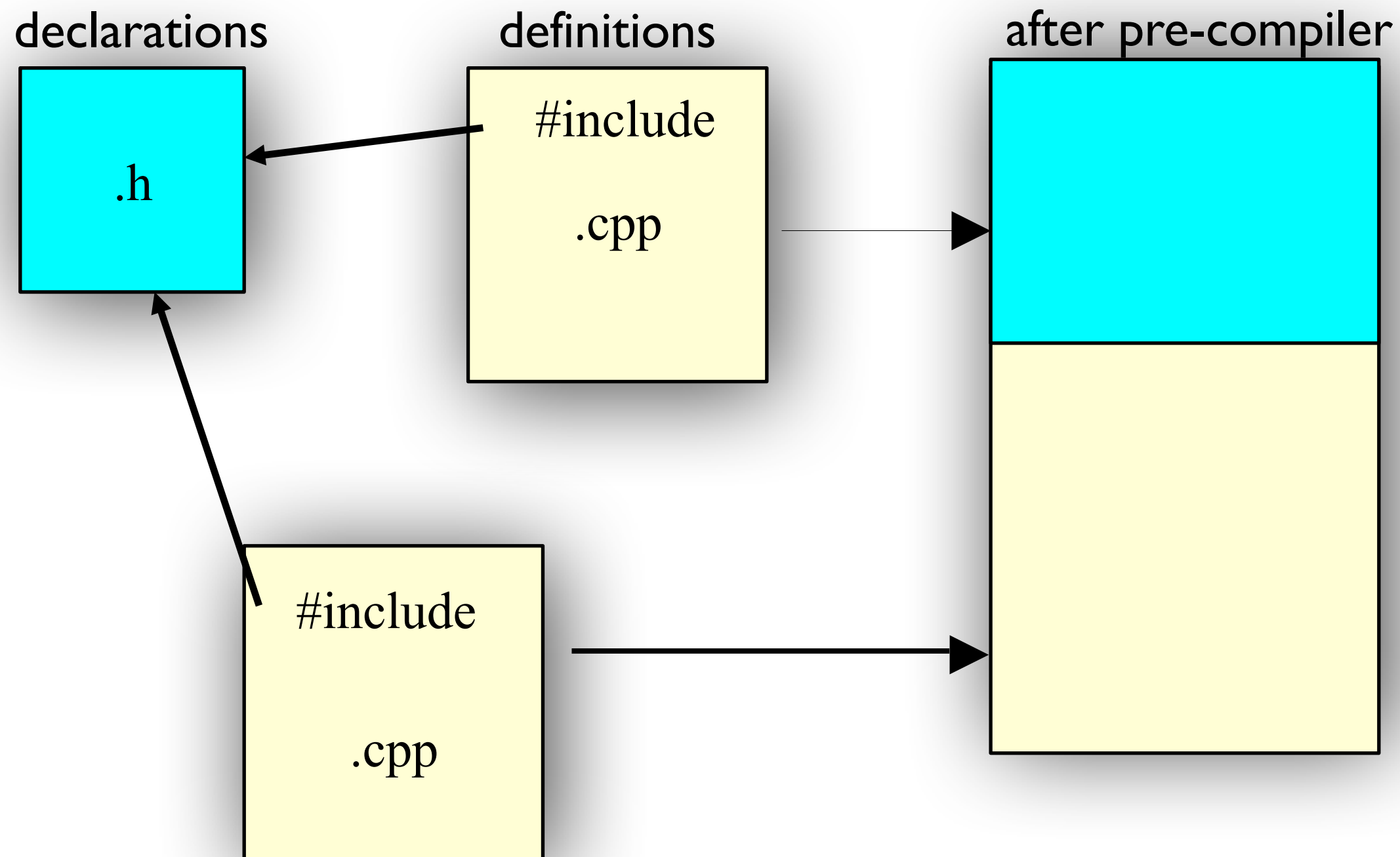
# Structure of C++ program



# Structure of C++ program



# Structure of C++ program



Other modules that use the functions

# Declarations vs. Definitions

- A .cpp file is a compile unit
- Only declarations are allowed to be in .h
  - extern variables
  - function prototypes
  - class/struct declaration

# #include



# #include

- `#include` is to insert the included file into the `.cpp` file at where the `#include` statement is.
- `#include "xx.h"`: first search in the current directory, then the directories declared somewhere
- `#include <xx.h>`: search in the specified directories

# Standard header file structure

```
#ifndef HEADER_FLAG  
#define HEADER_FLAG  
// Type declaration here...  
#endif // HEADER_FLAG
```

# Tips for header

1. One class declaration per header file
2. Same name with source file.
3. The contents of a header file is surrounded with  
`#ifndef    #define ...    #endif`

# The CMake utility