# Implementing objects and inheritance

**Object's memory map**

**Sharing of function**

**The 'this' pointer**

**Function invocations**

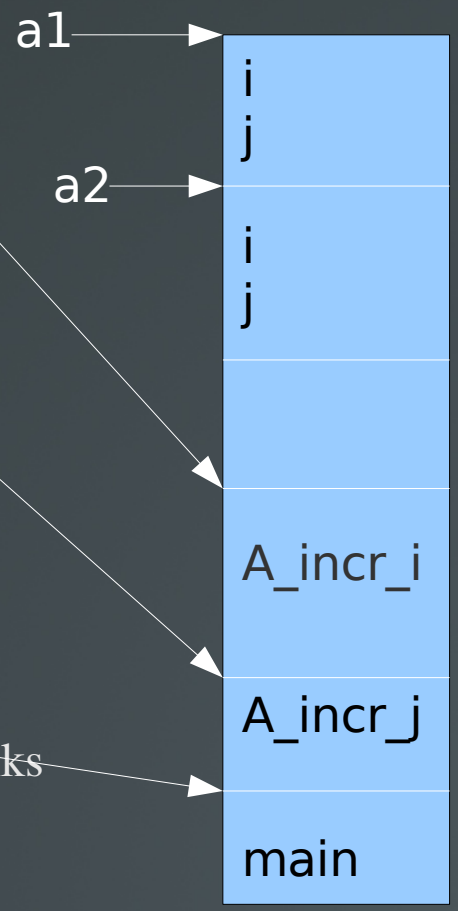
**Dynamic binding as per  the subsumption rules**

*PoPL course notes*

*Rushikesh K. Joshi*
*lectures on Aug 23,22,20*

# Standalone classes and their instances

```
A_incr_i (void *this, int x) {
        *((int *) (this+0)) +=x;
}
A_incr_j (void *this, int x) {
        *((int *) (this+sizeof int)) +=x;
}
main () {
        ..a1....a2 point to memory chunks
        A_incr_i (a1, 1);
        A_incr_i (a2,2);
        A_incr_j (a2,3);
}
```




```
class A {
        int i; int j;
        public incr_i ( int x) {
                i += x;
        }
        public incr_j (int x) {
                j+=x;
        }
};
main () {
        A *a1 = new A();
        A *a2 = new A();
        a1 -> incr_i(1);
        a2-> incr_i(2);
        a2->incr_j(3);
}
```

# The solution

- An instance's memory map contains all its state (the variables), and the class's function bodies are separated from the state chunks.

- shared function bodies

  To share a function body since a function may have to operate on multiple instances, we need to pass the memory reference of the object's state chunk.

  The 'this' pointer or 'self' reference

# The solution contd..

- Functions bodies are changed to reflect the indirect addressing of object's internal state through the 'this' pointer
- The invocations are changed to reflect the same by explicitly passing the receiver object's address as the first parameter
- The same this pointer can be made available to the programmer in source code
    - mainly used for returning itself

# Summary of the solution

We were able to share function bodies across many instances

&

The functions were able to find the locations of the object state variables

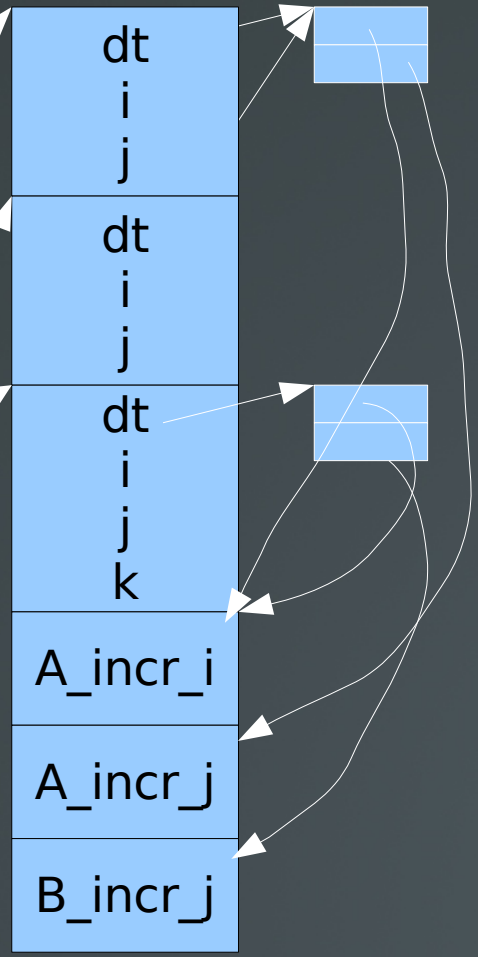
# Single Inheritance and instances

```
A_incr_i (void *this, int x) {
        *((int *) (this+sizeof dt)) +=x;
}
A_incr_j (void *this, int x) {
        *((int *) (this+sizeof dt+sizeof int)) +=x;
}
B _incr_j (void *this, int x) {
        *((int *) (this+sizeof dt+sizeof i) +=x;
        *((int ) (this+sizeof dt+2*sizeof int) +-1;
}
main () {
        ..a1....a2 point to memory chunks
        A_incr_i (a1, 1);
        A_incr_i (a2,2);
        A_incr_j (a2,3);
}
```

```
class A {
  protected int i; int j;
  public virtual incr_i ( int x) {i += x; }
  public virtual incr_j (int x) {j+=x;}
};
class B : public A{
  int k;
  public virtual incr_j( int x){j += x; k++;}
};
main () {
        A *a1 = new A();
        B *b1 = new B();
        A *a2 = b1;
        a1 -> incr_i(1);
        a1-> incr_j(2);
        a2 -> incr_i(3);
        a2-> incr_j(4);
        b1 -> incr_i(5);
        b1-> incr_j(6);
}
```

# Single Inheritance and instances

```
A_incr_i (void *this, int x) {
 *((int *) (this+sizeof dt)) +=x;
}
A_incr_j (void *this, int x) {
 *((int *) (this+sizeof dt+sizeof int)) +=x;
}
B _incr_j (void *this, int x) {
*((int *) (this+sizeof dt+sizeof i)) +=1;
(int ) (this+sizeod dt+2*sizeof int))+=1;
}
main () {
        A *a1 = new A();
        B *b1 = new B();
        A *a2 = b1; or new A()
        a1 -> incr_i(1);
        a1-> incr_j(2);
        a2 -> incr_i(3);
        a2-> incr_j(4);
        b1 -> incr_i(5);
        b1-> incr_j(6);
}
```




```
class A {
  protected int i; int j;
  public virtual incr_i ( int x) {i += x; }
  public virtual incr_j (int x) {j+=x;}
};
class B : public A {
  int k;
  public virtual incr_j( int x){j += x; k++;}
};
main () {
        A *a1 = new A();
        B *b1 = new B();
        A *a2 = b1; or new A()
        a1 -> incr_i(1);
        a1-> incr_j(2);
        a2 -> incr_i(3);
        a2-> incr_j(4);
        b1 -> incr_i(5);
        b1-> incr_j(6);
}
```

# The solution

- when the instances of subclasses are used as instances of superclasses, the respective functions should be called
- Which means we may not know at compile time the exact function bodies that will be called through a variable of a given type
- The solution is to make one dispatch table for every class and keep an address of this table in every instance.

# The solution contd..

- The dispatch table contains function pointers
- Invocations are now made through the dispatch table
- The this pointer scheme is the same as the one used earlier
- The table of a subclass preserves the order of entries in its superclass.

# Summary of the solution

We were able to locate the function from the instance even if the type of the variable was different

&

The functions were again able to find the right set of variables

# Additional problems of multiple inheritance

```
Class A {
    protected i,j;
    dynamically bound functions f,g,h;
}
class B {
    protected k,l;
    dynamically bound functions f,g,s;
}
class C inherits both A & B {
    protected m,n;
    dynamically bound functions f,g;
}
```

# What's the problem?

we may not be able to preserve the schema for the functions found in tables of the two superclasses within the table of the subclass.

solutions? (try in the lab..)