Compiling Object-Oriented Languages

Key features:

- inheritance (possibly multiple)
- subtyping & subtype polymorphism
- message passing, dynamic binding

Inheritance & subtype polymorphism

Represent subclass as extension of superclass's representation

```
class A {
  int i;
  char* p;
};
class B : A {
  float m;
  A* a;
};

B* b = new B;
...
A* a = b;       // subtype polymorphism
...
... a->i ...    // works
```


Multiple inheritance

```
class A {
  int i;
};
class B {
  int j;
};
class AB : A, B {
  int ij;
};

AB* ab = new AB;
A* a = ab;
... a->i ...    // works
B* b = ab;
... b->j ...    // breaks
```


A solution

[Stroustrup 87]

When casting to a non-primary superclass,
change pointer to point to the middle of the object

```
B* b = ab;      // adds sizeof(A) to ab
... b->j ...    // works
```

+ can access fields directly w/o overhead
+ supports MI

– some assignments, casts execute code
– complicates runtime system (e.g. GC, method lookup)

– breaks when casting through e.g. `void*`:

```
void* p = ab;
...
B* b = (B*) p;
... b->j ...    // breaks
```

Representing & calling virtual functions

```
class A {
  int i;
  virtual int foo() { return i; }
};
class B : A {
  int j;
  virtual int foo() { return j; }
  virtual void set_foo(int f) { j = f; }
};
...
A* a = ...;
... a->foo() ...    // what foo gets called?
```

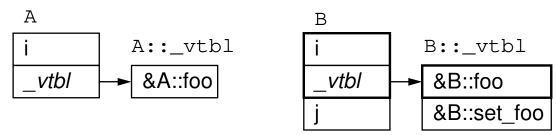
Could store pointer to virtual fn in each object

- + simple
- + reasonably fast to invoke virtual fn
- + flexible: each object could have its own methods
- – space-consuming

Virtual function tables

Observation:
each object in a class has the same virtual fn pointers

Idea:
factor out virtual fns into table shared by all instances of class



Call sequence:

```
... a->foo() ...
  ⇒
r1 := *(a + _vtbl_offset);
r2 := *(r1 + foo_index*4);
call *r2;
```

- + only 1 word of space overhead per object
- – extra indirection on call

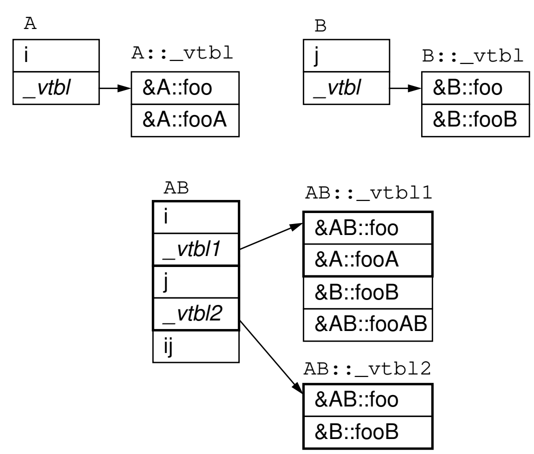
Multiple inheritance & virtual functions

Example source code:

```
class A {
  int i;
  virtual int foo()  { return fooA(); }
  virtual int fooA() { return i; }
};
class B {
  int j;
  virtual int foo()  { return fooB(); }
  virtual int fooB() { return j; }
};
class AB : A, B {
  int ij;
  virtual int foo()   { return fooAB(); }
  virtual int fooAB() { return ij; }
};
```


Object layout of example

Multiple vtbl pointers required in `AB` representation



A problem

```
AB* ab = new AB;
... ab->fooAB() ...// works
... ab->fooA()  ...// works
... ab->fooB()  ...// breaks
```

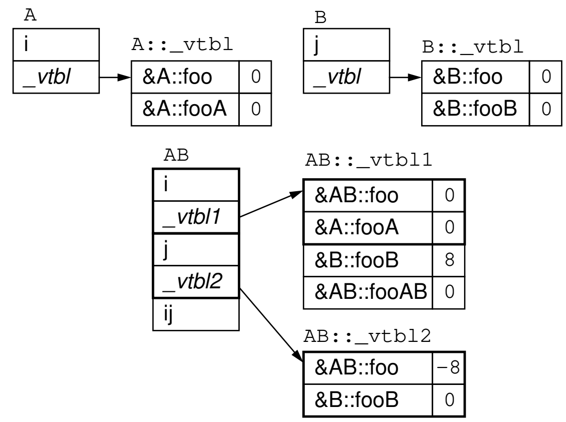
Need to perform implicit cast of receiver argument
as part of virtual function call

- – caller can't do it ⇒ doesn't know callee method
- – callee can't do it ⇒ doesn't know static type of actual
- + can be done in virtual tables
    - vtbl is specific to actual's static type in callee & callee method is known

One solution

Each entry of vtbl is a pair:

- pointer to callee function
- offset to be added to receiver pointer




```
... a->foo() ...
 ⇒
r1 := *(a + _vtbl_offset);
r2 := *(r1 + foo_index*8);
r3 := *(r1 + foo_index*8+4);
r4 := add a, r3;
call *r2;
```

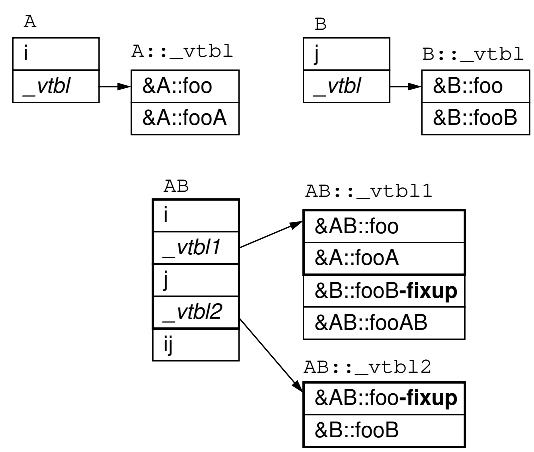

Evaluation

- + virtual functions can coexist with multiple inheritance
- – calling sequence has 2 more instructions
- – vtbls are twice as big
- – does not support implicit cast for other args, result
  ⇒ forces "nonvariance" for function overriding in C++

An alternative

Observation: most receiver offsets are 0

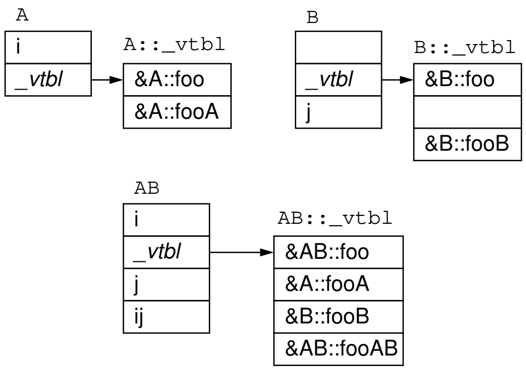
Idea: use single inheritance call sequence,
insert intermediary fns to implement non-0 offsets



- + fast call sequence, usually
- + pay for MI only when used

Another alternative

Insert padding in assigned field/function offsets to avoid conflicts among parents
[Dixon *et al.* 89], [Pugh & Weddell 90]



+ always fast access, invocation
+ no pointers to middle of objects
+ pay (at run-time) for MI only when used
– space cost
– complex link-time analysis (graph coloring!)