“So how do we lay out objects in heap anyway?”
Object layout

```c
class A {
    int a; int f(int);
};
class B : public A {
    int b; int g(int);
};
class X : public B {
    int c; int h(int);
};
...
C c;
c.g(42);
```

"So how do we include several parent objects?"

Object layout – virtual methods

```c
class A {
    int a; virtual int f(int);
    virtual int g(int);
    virtual int h(int);
};
class B : public A {
    int b; int g(int);
};
class C : public B {
    int c; int h(int);
};
...
C c;
c.g(42);
```

Multiple Base Classes

```c
class A {
    int a; int f(int);
};
class B {
    int b; int g(int);
};
class C : public A , public B {
    int c; int h(int);
};
...
C c;
c.g(42);
```

```c
% c = alloca %class.C
% 1 = getelementptr %c, i64 0, i32 0
% 2 = call i32 g(%class.B % 1, i32 42); g is statically known
```

```c
% c = alloca %class.C
% 1 = getelementptr %c, i64 0, i32 1
% 2 = call i32 g(%class.B % 1, i32 42); g is statically known
```
Multiple Base Classes

```java
class A {
    int a; int f(int);
};
class B {
    int b; int g(int);
};
class C : public A, public B {
    int c; int h(int);
};
...
C c;
c.g(42);
```

⚠️ `getelementptr` hides the \( \Delta B \) here!

Ambiguities

```java
class A { void f(int); }
class B { void f(int); }
class C : public A, public B {};
C* pc;
pc->f(42);
⚠️ Which method is called?
```

Solution I: Explicit qualification

```java
pc->A::f(42);
pc->B::f(42);
```

Solution II: Automatic resolution

Idea: The Compiler introduces a linear order on the nodes of the inheritance graph

Linearization

**Inheritance Relation** \( H \)  
Defined by ancestors.

**Multiplicity** \( M \)  
Defined by the order of multiple ancestors.

**Principles**

1. An inheritance mechanism (maps Object to sequence of ancestors) must follow the inheritance partial order \( H \).
2. The inheritance is a uniform mechanism, and its searches (→ total order) apply identical for all object properties (→fields/methods).
3. In any case the inheritance relation \( H \) excels the multiplicity \( M \).
4. When there is no contradiction between multiplicity \( M \) and inheritance \( H \), the inheritance search must follow the partial order \( H \cup M \).

Linearization algorithm candidates

Depth-First Search

```
A B C W
```

```
A B C
```

```
A B C W
```

```
A B C
```

```
A B C W
```

```
A B C
```

```
A B C W
```

```
A B C
```

```
A B C W
```

```
A B C
```

```
A B C W
```

```
A B C
```

```
A B C W
```

```
A B C
```
Linearization algorithm candidates

**Depth-First Search**
A B W C

⚠️ Principle 1 *inheritance* is violated.

**Breadth-First Search**
A B C W D

Linearization algorithm candidates

**Reverse Postorder Rightmost DFS**
A B F D C E G H W

✅ Linear extension of inheritance relation

**Reverse Postorder Rightmost DFS**
F E G D C B A

⚠️ But principle 4 *multiplicity* is violated!

Linearization algorithm candidates

**Reverse Postorder Rightmost DFS**
A B F D C E G H W

✅ Linear extension of inheritance relation

**Reverse Postorder Rightmost DFS**
A B F D C E G H W
**Linearization Algorithm**

Idea [Ducournau and Habib(1987)]

Successively perform Reverse Postorder Rightmost DFS and refine inheritance graph $G$ with *contradiction arcs*.

The reservoir set of potential *contradiction arcs* $CA$ is initially $M$, while the inheritance graph $G$ starts from $H$.

```
do
  1. search ← RPDFS$_G$
  2. CA ← \{contradiction arcs of upper search\} $\cap$ M
  3. G ← G $\cup$ CA;
  while (CA $\neq$ $\emptyset$) $\land$ (search violates $H \cup M$)
```

**Linearization algorithm candidates**

**Reverse Postorder Rightmost DFS**

- A B F D C E G H W
- ✔ Linear extension of inheritance relation

**Reverse Postorder Rightmost DFS**

- A B C D F G E
- ❌ But principle 4 *multiplicity* is violated!

---

**Linearization Algorithm**

Idea [Ducournau and Habib(1987)]

Successively perform Reverse Postorder Rightmost DFS and refine inheritance graph $G$ with *contradiction arcs*.

The reservoir set of potential *contradiction arcs* $CA$ is initially $M$, while the inheritance graph $G$ starts from $H$.

```
do
  1. search ← RPDFS$_G$
  2. CA ← \{contradiction arcs of upper search\} $\cap$ M
  3. G ← G $\cup$ CA;
  while (CA $\neq$ $\emptyset$) $\land$ (search violates $H \cup M$)
```

**Linearization vs. explicit qualification**

**Linearization**

- No switch/duplexer code necessary
- No explicit naming of qualifiers
- Unique super reference

**Qualification**

- More flexible, fine-grained
- Linearization choices may be awkward or unexpected

**Languages with automatic linearization exist**

- CLOS Common Lisp Object System
- Prerequisite for → Mixins
Linearization vs. explicit qualification

**Linearization**
- No switch/duplexer code necessary
- No explicit naming of qualifiers
- Unique `super` reference

**Qualification**
- More flexible, fine-grained
- Linearization choices may be awkward or unexpected

Languages with automatic linearization exist
- **CLOS** Common Lisp Object System
- Prerequisite for → Mixins