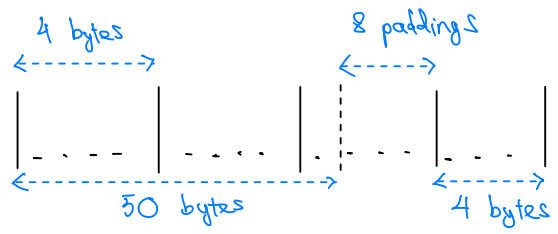
OBJECT-ORIENTED LANGUAGE

How Record store in memory

Consider this Record, let say our CPU reads 4 bytes at once

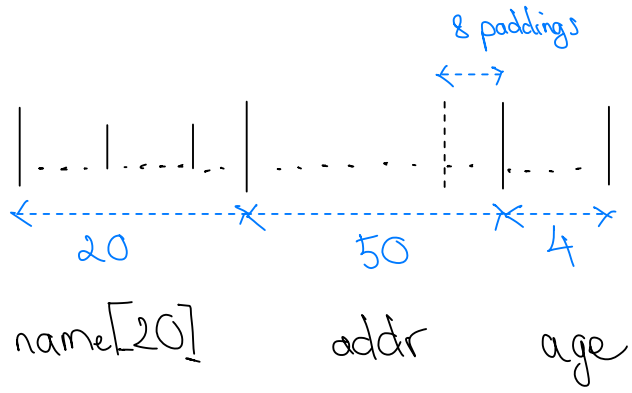
```
struct Address {
    char street[50];
    int zipcode;
};
```

store as →



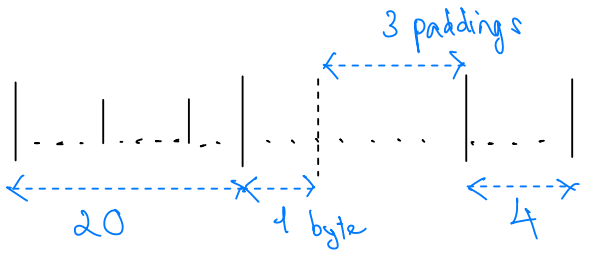

```
struct Person {
    char name[20];
    struct Address addr;
    int age;
};
```

store as →




```
struct PersonSlow {
    char name[20];
    struct Address *addr;
    int age;
};
```

store as →



Data alignment:

- Data alignment refers to how data is arranged in memory to match CPU's natural reading boundary.
- Rules of thumb:
  - Choosing starting address: Type want to start at addresses divisible by their size
  - Type Object is aligned if all sub-objects is aligned. (recursively)
  - Type Array is aligned if:
    - first element in Array follows Object rule.
    - subsequent elements is placed immidiately after previous one.

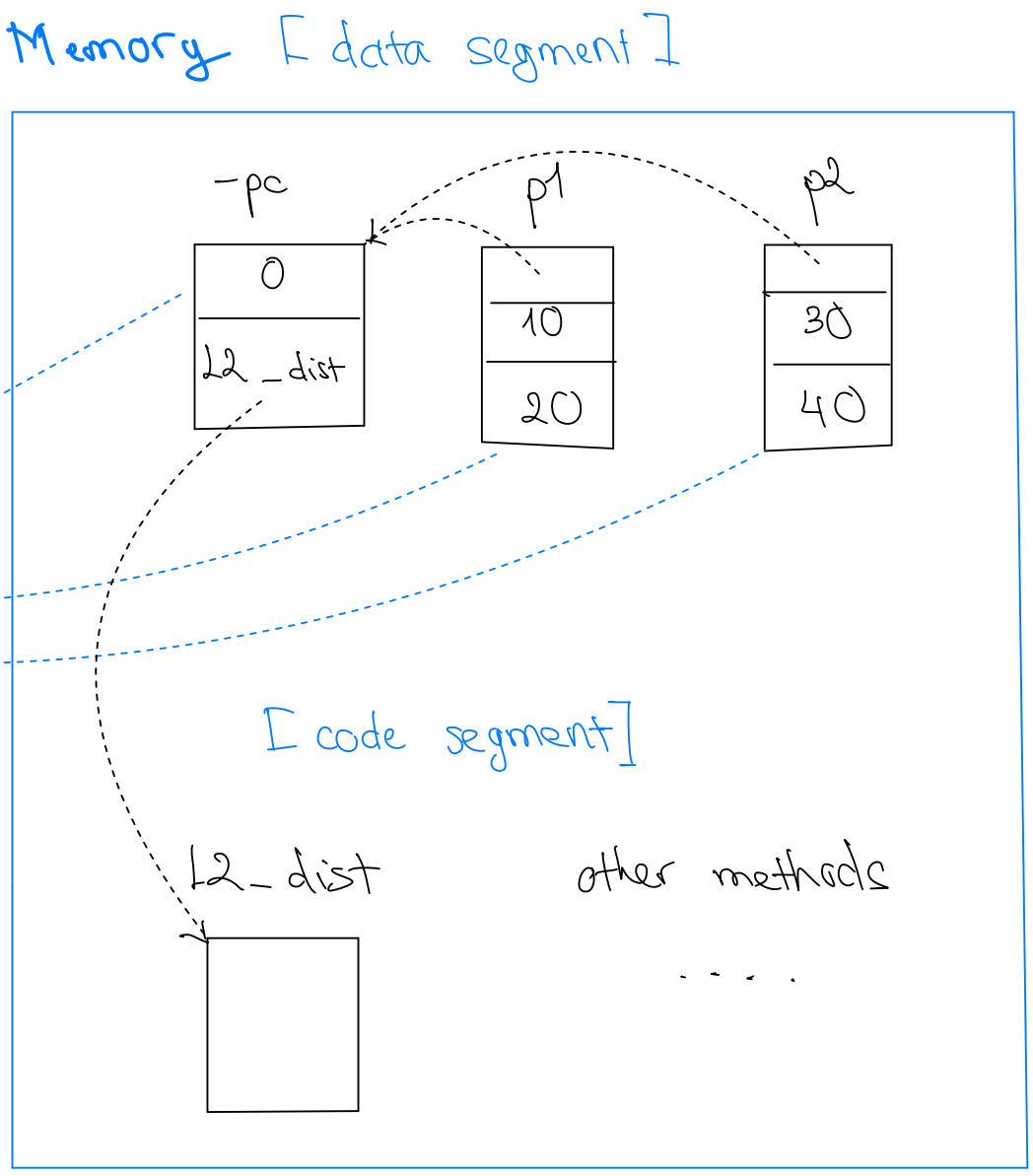
Building Classes / Objects from Records

```
extern double L2_dist(double x, double y);

typedef struct {
    int count;
    double (*dist)(double, double);
} _PointClass;

typedef struct {
    _PointClass *cp;
    int x, y;
} Point;

int main(void) {
    _PointClass _pc = {0, L2_dist};
    Point p1 = {_pc, 10, 20};
    Point p2 = {_pc, 30, 40};
    return 0;
}
```

method

class

object

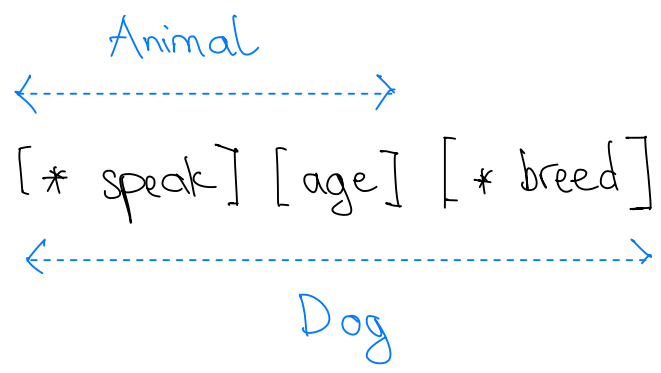


Inheritance:

Inheritance using structs:

```c
// "Parent"
struct Animal {
    void (*speak)(void* self);  // function pointer
    int age;
};

// "Child"
struct Dog {
    struct Animal base;  // Must be first!
    char* breed;         // Dog-specific data
};
```

Store as →



⇒ Conclusion: Its all about memory layout. In this example, anything comes first in Dog is inherited.

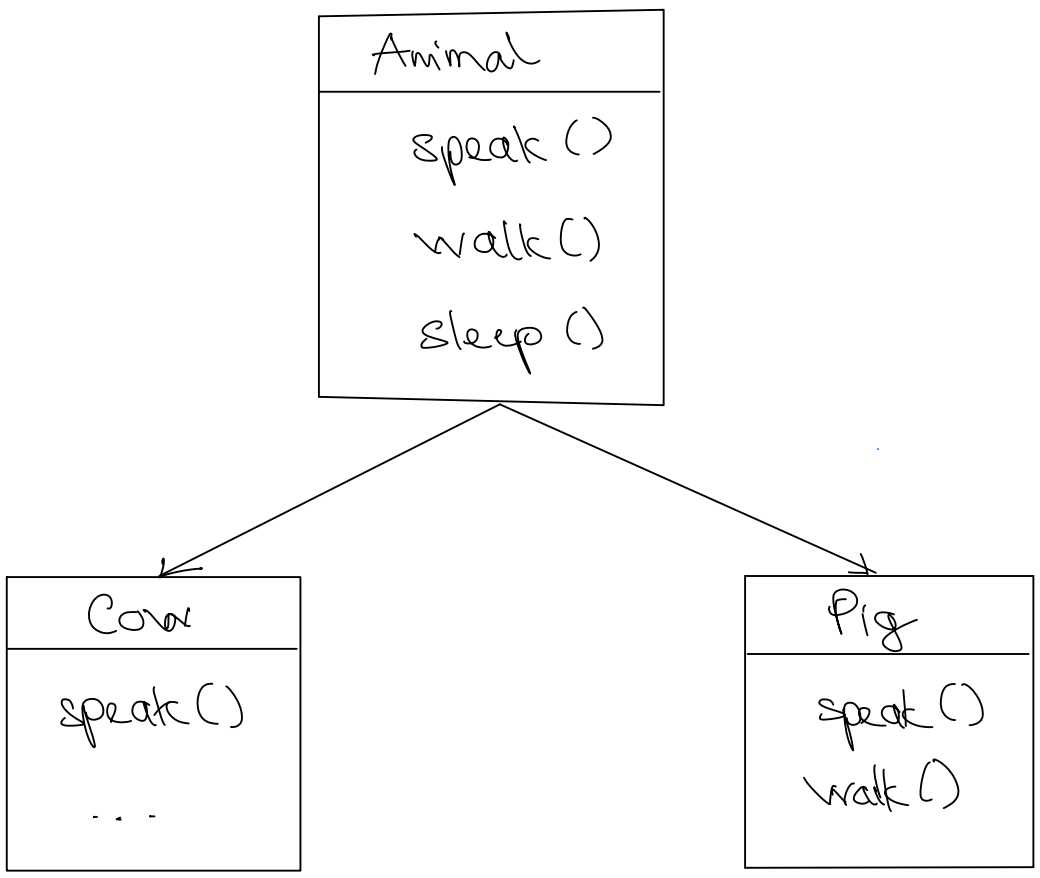
Now, with some syntaxtic sugar:

```c
// Becomes this in C++:
class Animal {
public:
    virtual void speak() { }  // Compiler creates vtable
    int age;
};

class Dog : public Animal {  // Compiler maintains same memory layout!
public:
    void speak() override { } // Goes into Dog's vtable
    char* breed;
};
```

do the same thing

Multiple Inheritance:

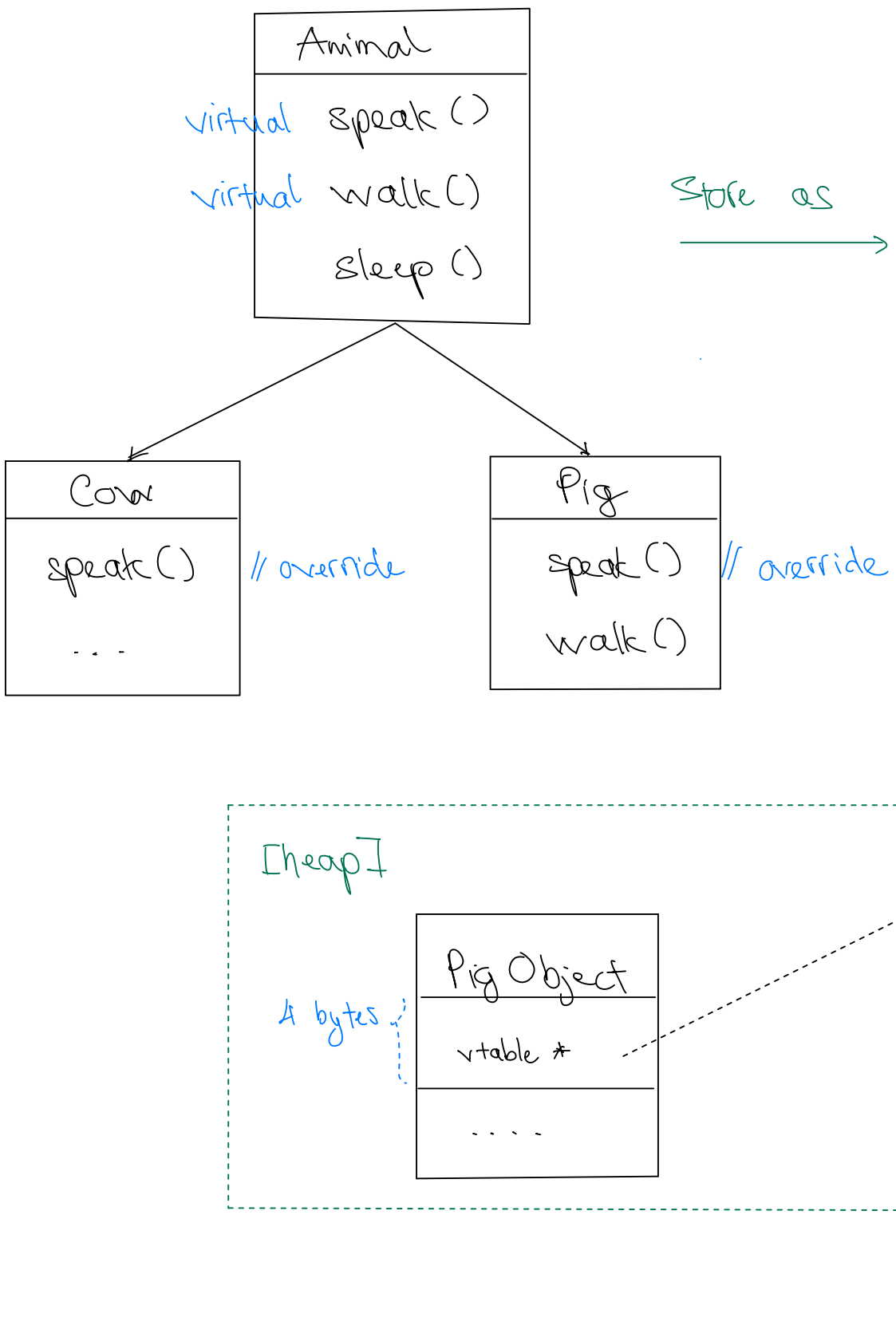
Problem: How does the code knows which "speak" implementation to run?




```
void main () {
    Animal * animal;
    if (something) {
        animal = new Cow;
    } else {
        animal = new Pig;
    }
    animal → speak();
}
```

?

Answer: virtual tables



Vtable rules:

- 1 vtable per type
- Only virtual functions are put in vtables
- All instances of the same type share the same vtable
- Only use vtables in the case involve polymorphism

Like this: `Animal* animal`

`animal = new Donkey`

not this: `Donkey d = new Donkey`

⇒ this case vtable is not needed