Object Orientation (Cont.)

CS143: lecture 19

Byron Zhong, July 31

Object-Oriented Programming

Dynamic Dispatch via Tagged Union

- A union is a type that can be one of the declared fields at a given time
- The fields overlap in memory
- C doesn't keep track of which field was set in a union and C doesn't prevent you from selecting the wrong union fields.
- When you select the wrong field, you choose a wrong interpretation of the bits and potentially read some uninitialized bits.
- A common way to keep track of the correct interpretation is to use *tagged union*.
  - structure of an enum and a union
  - Enum keeps track of the alternatives, and the union stores the data of one of them.

Object-Oriented Programming

Dynamic Dispatch via Tagged Union

- However, a tagged union is not extensible
- If we want to add another animal, every function needs to be changed.

```
enum animal_tag {
        CAT,
        DOG,
        ALLIGATOR,
};
```

```
const char *noise(struct animal *a)
{
        switch (a->tag) {
        ...
        case ALLIGATOR:
                ...
        ...
        }
}
```

```
void walk(struct animal *a)
{
        switch (a->tag) {
        ...
        case ALLIGATOR:
                ...
        ...
        }
}
```

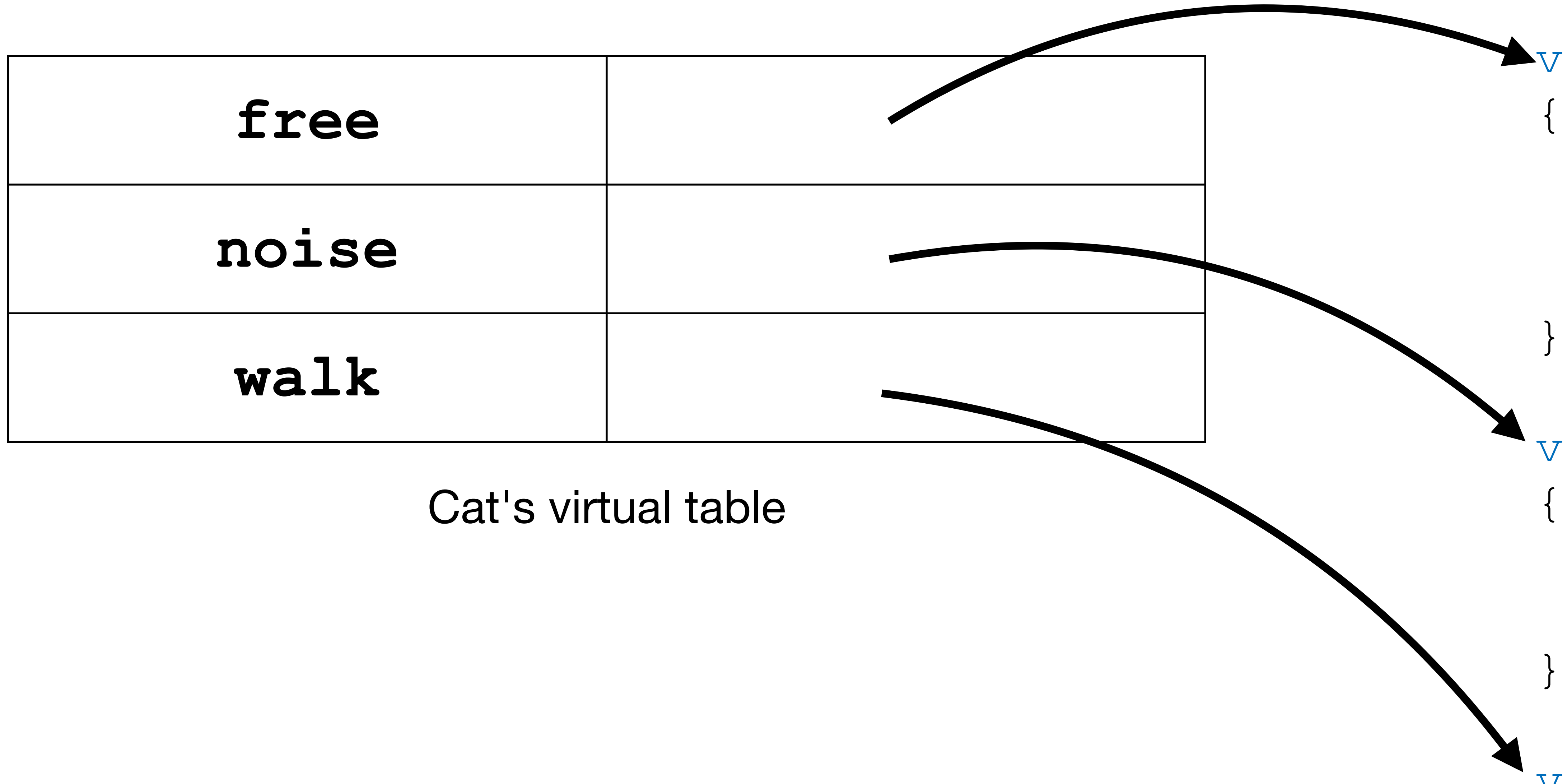
Object-Oriented Programming

Dynamic Dispatch via Virtual Table

- This is how most languages implement dynamic dispatch.
- Each *subclass* gets a *virtual table* (*vtable*), filled with pointers to its overridden functions.
- Each subclass *object* has a pointer to its class's virtual table, which is used to resolve function calls at runtime.

Object-Oriented Programming

Dynamic Dispatch via Virtual Table



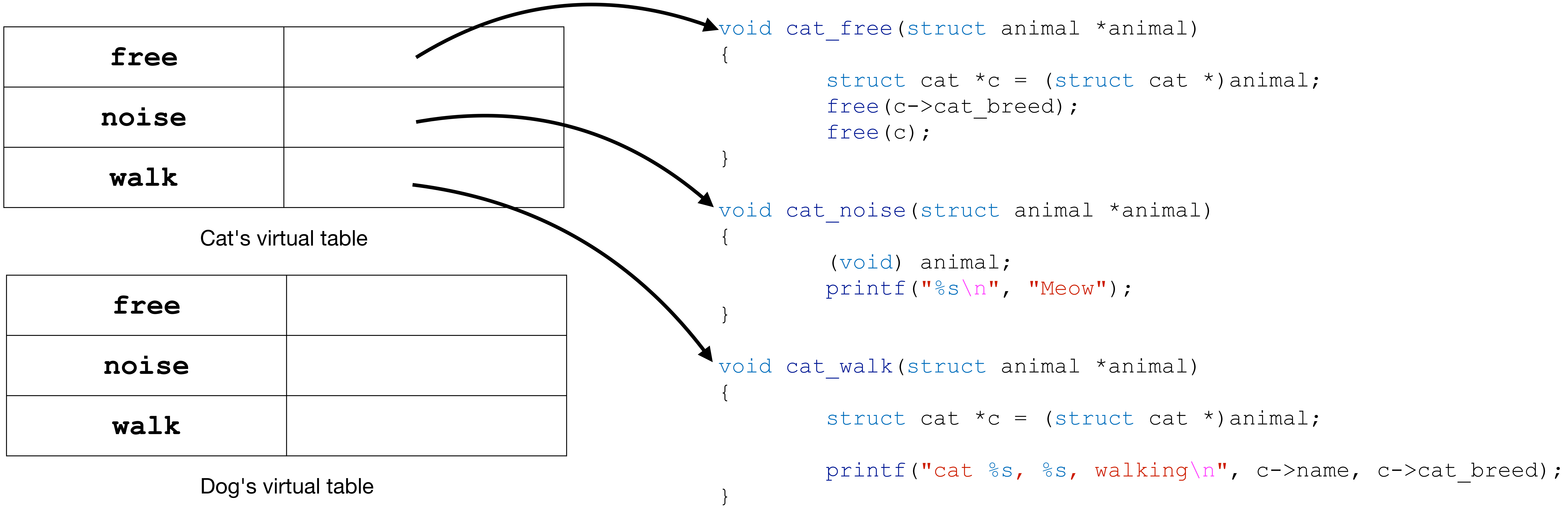
free	
noise	
walk	

Cat's virtual table

```
void cat_free(struct animal *animal)
{
        struct cat *c = (struct cat *)animal;
        free(c->cat_breed);
        free(c);
}

void cat_noise(struct animal *animal)
{
        (void) animal;
        printf("%s\n", "Meow");
}

void cat_walk(struct animal *animal)
{
        struct cat *c = (struct cat *)animal;

        printf("cat %s, %s, walking\n", c->name, c->cat_breed);
}
```

Object-Oriented Programming

Dynamic Dispatch via Virtual Table



free	
noise	
walk	

Cat's virtual table

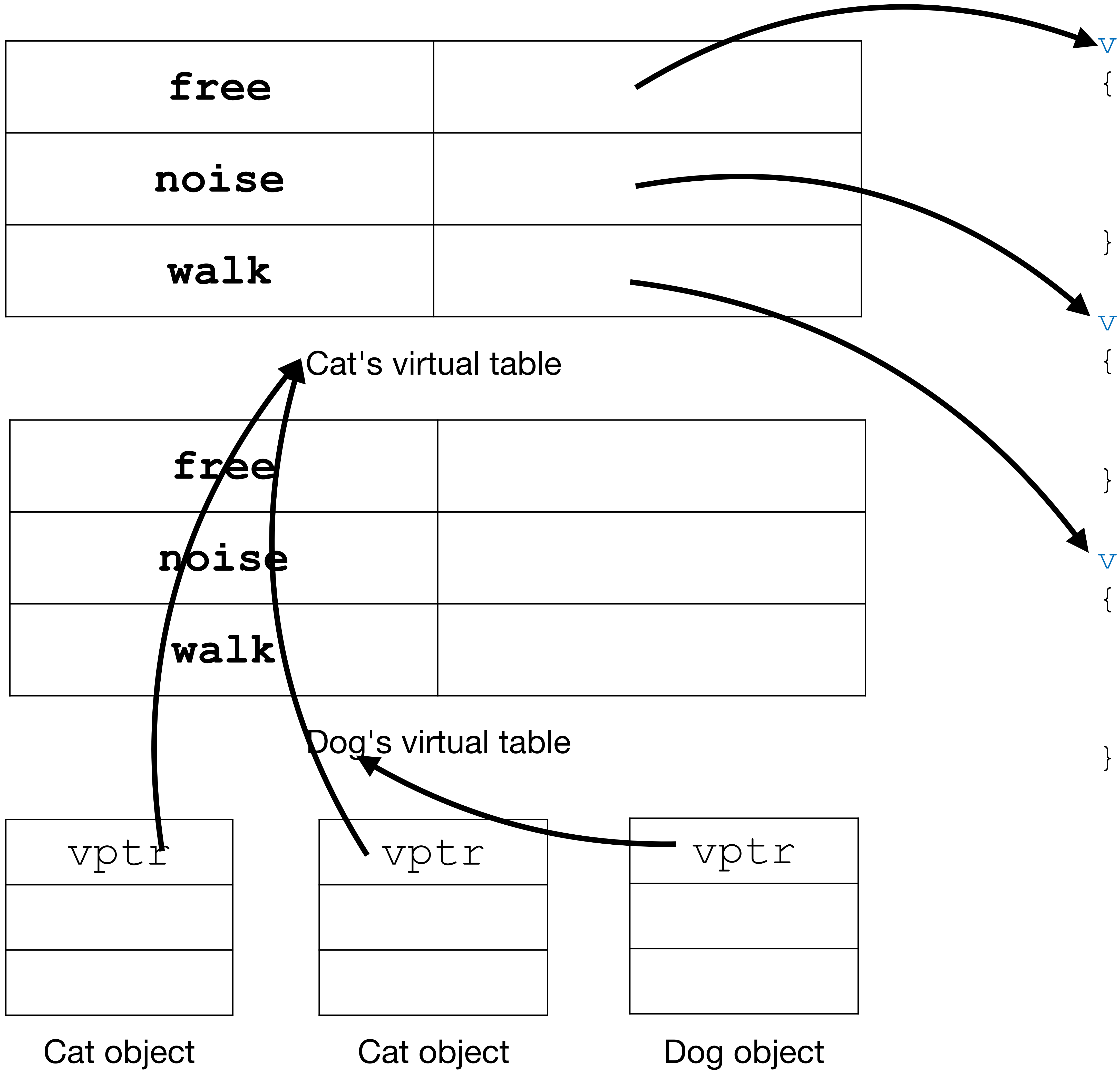
free	
noise	
walk	

Dog's virtual table

```
void cat_free(struct animal *animal)
{
        struct cat *c = (struct cat *)animal;
        free(c->cat_breed);
        free(c);
}

void cat_noise(struct animal *animal)
{
        (void) animal;
        printf("%s\n", "Meow");
}

void cat_walk(struct animal *animal)
{
        struct cat *c = (struct cat *)animal;

        printf("cat %s, %s, walking\n", c->name, c->cat_breed);
}
```

Object-Oriented Programming

Dynamic Dispatch via Virtual Table



free	
noise	
walk	

Cat's virtual table

free	
noise	
walk	

Dog's virtual table

Cat object

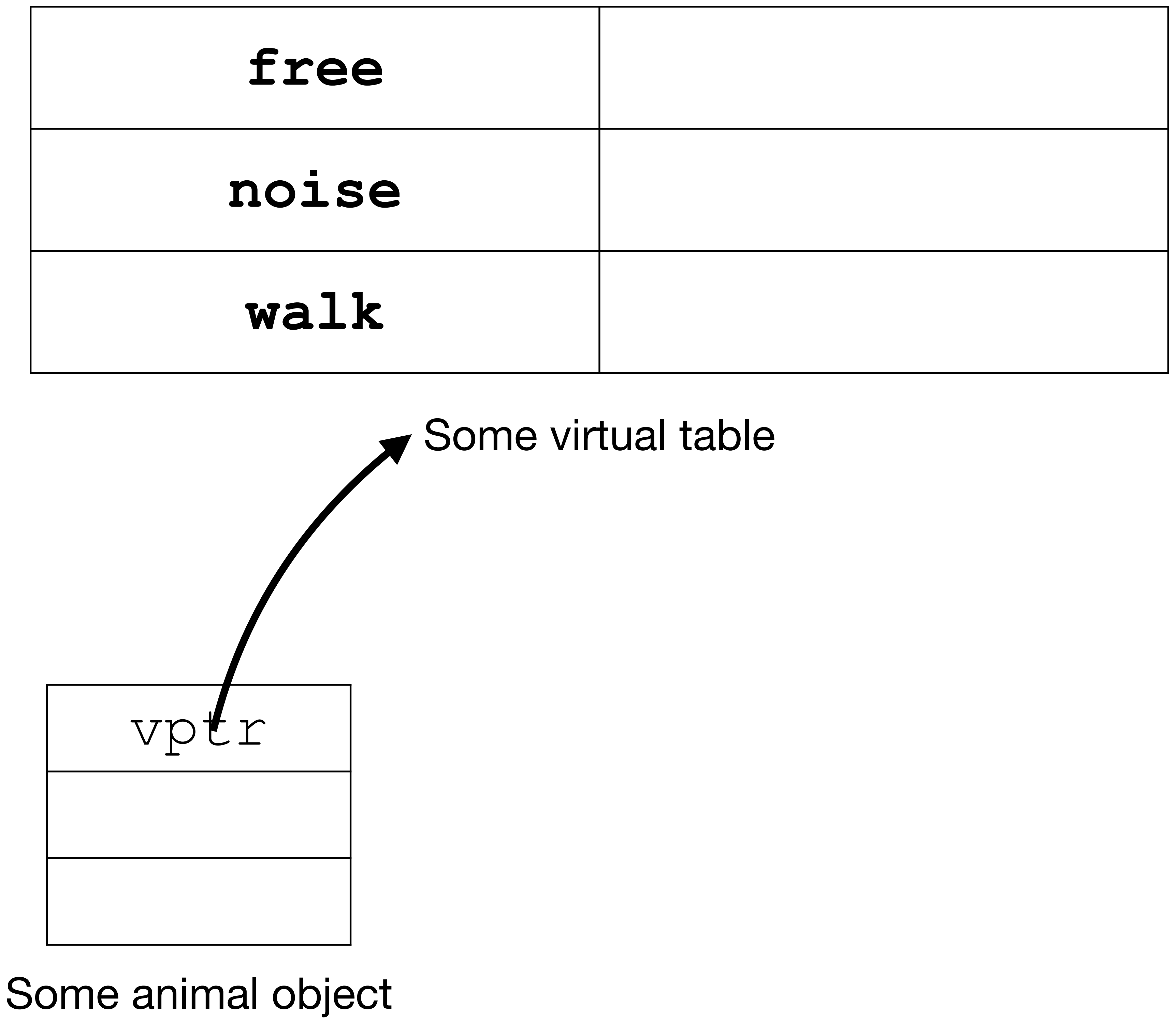
Cat object

Dog object

```
void cat_free(struct animal *animal)
{
        struct cat *c = (struct cat *)animal;
        free(c->cat_breed);
        free(c);
}

void cat_noise(struct animal *animal)
{
        (void) animal;
        printf("%s\n", "Meow");
}

void cat_walk(struct animal *animal)
{
        struct cat *c = (struct cat *)animal;

        printf("cat %s, %s, walking\n", c->name, c->cat_breed);
}
```

Object-Oriented Programming

Dynamic Dispatch via Virtual Table



`free`	
`noise`	
`walk`	

Some virtual table

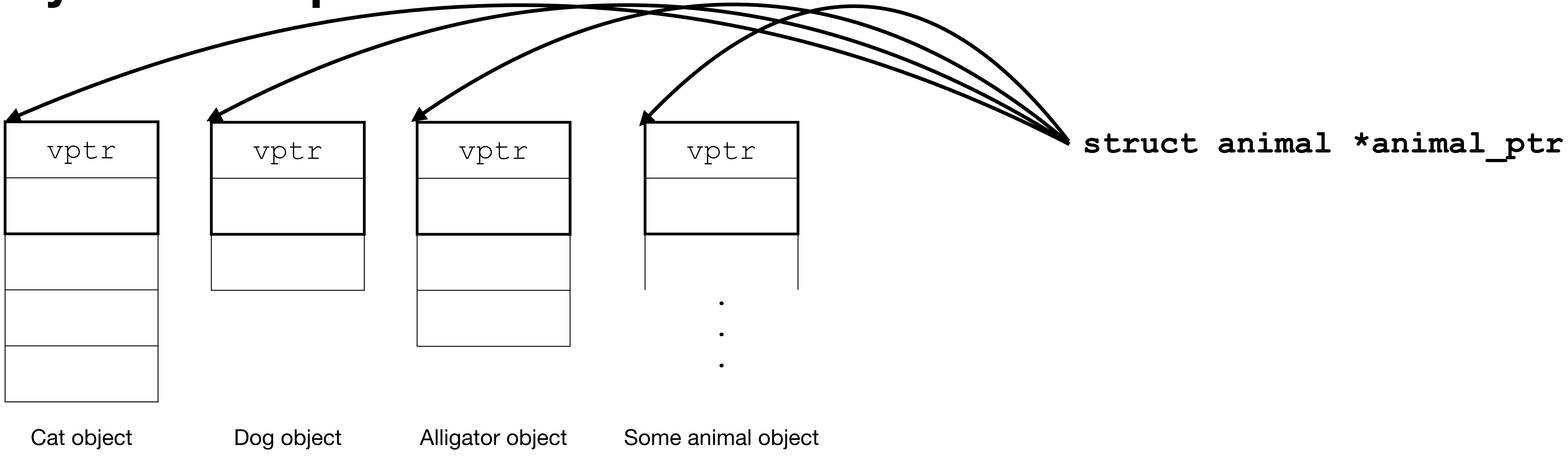
`vptr`

Some animal object

- If the animal is a cat, we know that the vptr points to Cat's virtual table
- If the animal is a dog, we know that the vptr points to Dog's virtual table
- `animal->vptr->noise()` calls the correct noise function via the virtual table

Object-Oriented Programming

Dynamic Dispatch via Virtual Table



`vptr` `vptr` `vptr` `vptr`

`struct animal *animal_ptr`

Cat object

Dog object

Alligator object

Some animal object

Object-Oriented Programming

Dynamic Dispatch via Virtual Table

```
#include <vtable-demo>
```

Bits (Again)

Integer Promotion and Demotion

- Changing the number of bits that represents a number
- `char <-> short <-> int <-> long`
- `uint8_t <-> uint16_t <-> uint32_t <-> uint64_t`
- You convert the types (length) of a number without changing the number it represents (provided that the number is within the range)
- What happens to the bits?

Bits (Again)

Unsigned Promotion

`uint8_t`	1010 1010	170

Bits (Again)

Unsigned Promotion

`uint8_t`	`1010 1010`	170
`uint16_t`	`0000 0000 1010 1010`	170

Bits (Again)

Unsigned Promotion

`uint8_t`	`1010 1010`	170
`uint16_t`	`0000 0000 1010 1010`	170
`uint32_t`	`0000 0000 0000 0000 0000 0000 1010 1010`	170

Bits (Again)

Signed Promotion

uint8_t	1010 1010	170
uint16_t	0000 0000 1010 1010	170
uint32_t	0000 0000 0000 0000 0000 0000 1010 1010	170
int8_t	1010 1010	-86

Bits (Again)

Signed Promotion

uint8_t	1010 1010	170
uint16_t	0000 0000 1010 1010	170
uint32_t	0000 0000 0000 0000 0000 0000 1010 1010	170
int8_t	1010 1010	-86
int16_t		-86

Bits (Again)

Signed Promotion

`uint8_t`	`1010 1010`	170
`uint16_t`	`0000 0000 1010 1010`	170
`uint32_t`	`0000 0000 0000 0000 0000 0000 1010 1010`	170

`int8_t`	`1010 1010`	-86
`int16_t`	`0000 0000 1010 1010`	170

Bits (Again)

Signed Promotion

`uint8_t`	1010 1010	170
`uint16_t`	0000 0000 1010 1010	170
`uint32_t`	0000 0000 0000 0000 0000 0000 1010 1010	170

`int8_t`	1010 1010	-86
`int16_t`	1111 1111 1010 1010	-86

Bits (Again)

Signed Promotion

1	0	1	0
-2^3	2^2	2^1	2^0

Bits (Again)

Signed Promotion

1	1	0	1	0
-2^4	2^3	2^2	2^1	2^0

Bits (Again)

Signed Promotion

1 1 0 1 0

$-2 * 2^3$ 2^3 2^2 2^1 2^0

Bits (Again)

Signed Promotion

$$\underbrace{1\quad 1}_{-2^3}\quad \underset{2^2}{0}\quad \underset{2^1}{1}\quad \underset{2^0}{0}$$

Bits (Again)

Signed Promotion

$$\underbrace{1 \quad 1 \quad 1}_{-2^3} \quad \underset{2^2}{0} \quad \underset{2^1}{1} \quad \underset{2^0}{0}$$

Bits (Again)

Signed Promotion

$$\underbrace{1\quad 1\quad 1\quad 1}_{-2^3}\quad \underset{2^2}{0}\quad \underset{2^1}{1}\quad \underset{2^0}{0}$$

Bits (Again)

Signed Promotion

$$\underbrace{1\quad 1\quad 1\quad 1\quad 1}_{-2^3}\quad \underset{2^2}{0}\quad \underset{2^1}{1}\quad \underset{2^0}{0}$$

Bits (Again)

Signed Promotion

`uint8_t`	`1010 1010`	170
`uint16_t`	`0000 0000 1010 1010`	170
`uint32_t`	`0000 0000 0000 0000 0000 0000 1010 1010`	170

`int8_t`	`1010 1010`	-86
`int16_t`	`1111 1111 1010 1010`	-86
`int32_t`	`1111 1111 1111 1111 1111 1111 1010 1010`	-86

Bits (Again)

Signed Promotion

`uint8_t`	1010 1010	170
`uint16_t`	0000 0000 1010 1010	170
`uint32_t`	0000 0000 0000 0000 0000 0000 1010 1010	170

`int8_t`	1010 1010	-86
`int16_t`	1111 1111 1010 1010	-86
`int32_t`	1111 1111 1111 1111 1111 1111 1010 1010	-86

`int8_t`	0010 1010	42
`int16_t`	0000 0000 0010 1010	42
`int32_t`	0000 0000 0000 0000 0000 0000 0010 1010	42

Bits (Again)

Promotion

- To convert an n-bit unsigned number to an $(n+1)$-bit unsigned number:
  - Add more 0's
- To convert an n-bit signed number to an $(n+1)$-bit signed number:
  - If the number is negative, add more 1's
  - Otherwise, add more 0's

Bits (Again)

Demotion

`uint8_t`	`1010 1010`	`170`
`uint16_t`	`0000 0000 1010 1010`	`170`
`uint32_t`	`0000 0000 0000 0000 0000 0000 1010 1010`	`170`

`int8_t`	`1010 1010`	`-86`
`int16_t`	`1111 1111 1010 1010`	`-86`
`int32_t`	`1111 1111 1111 1111 1111 1111 1010 1010`	`-86`

`int8_t`	`0010 1010`	`42`
`int16_t`	`0000 0000 0010 1010`	`42`
`int32_t`	`0000 0000 0000 0000 0000 0000 0010 1010`	`42`

Bits (Again)

Promotion

- To convert an $(n + 1)$-bit number to an n-bit number:
  - Just take the last n bits!

Final Exam

- Thursday, August 3, 6:00pm - 7:20pm in class
- You will have 80 minutes -- show up a few minutes earlier if you can
- Cheat sheet: 1 letter-size, double-sided, *hand-written* note
- Topics:
  - Heavily biased towards the second half of the quarter
  - But *material from the first half could appear*
- Q&A on Tuesday -- bring questions

Final Exam

Topics

- Numbers: Binary, Decimal, Hexadecimal, 2's complement, endianness
- Bitwise operators: Operators, bit-packing, masks, ...
- C stuff: `struct`, `enum`, `union`, `&`, `*`, function pointers
- Hash table: Chaining and probing
- Threads: concept, `pthread_create`, `pthread_join`

Final Exam

Not topics

- Assembly language
- Virtual memory
- Virtual table
- Sorting
- Time complexity