UNM SCHOOL *of* ENGINEERING

Department of Computer Science

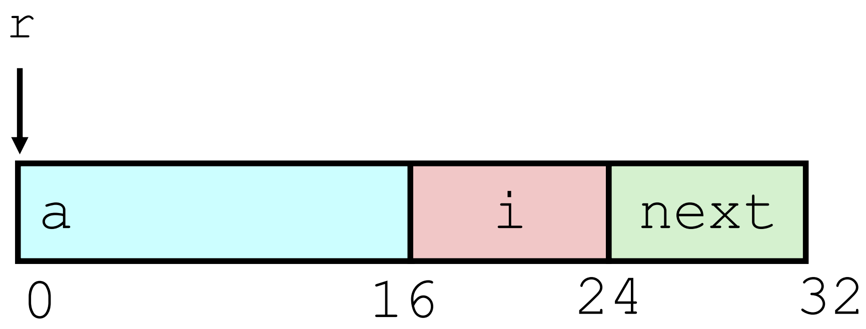
Lecture #13

Complex Data and Control Structures (2): structs and their alignment Section 3.9

Prof. Soraya Abad-Mota, PhD

Structure Representation

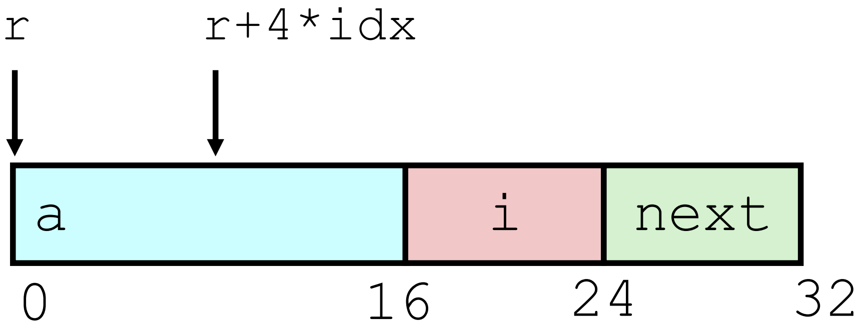
```
struct rec {
    int a[4];
    size_t i;
    struct rec *next;
};
```



- Structure represented as block of memory
  - **Big enough to hold all of the fields**
- Fields ordered according to declaration
  - **Even if another ordering could yield a more compact representation**
- Compiler determines overall size + positions of fields
  - **Machine-level program has no understanding of the structures in the source code**

Generating Pointer to Structure Member

```
struct rec {
    int a[4];
    size_t i;
    struct rec *next;
};
```



- Generating Pointer to Array Element
  - Offset of each structure member determined at compile time
  - Compute as **`r + 4*idx`**

```
int *get_ap
 (struct rec *r, size_t idx)
{
  return &r->a[idx];
}
```

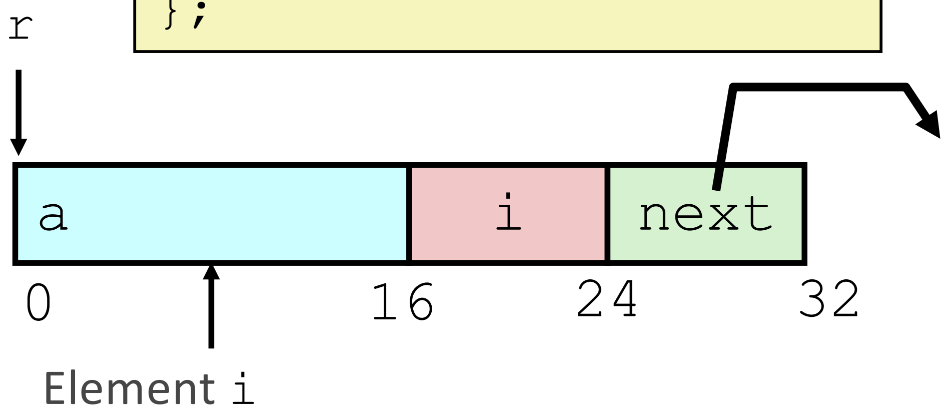
```
  # r in %rdi, idx in %rsi
  leaq  (%rdi,%rsi,4), %rax
  ret
```

Following Linked List

- C Code

```
struct rec {
    int a[4];
    int i;
    struct rec *next;
};
```

```
void set_val
  (struct rec *r, int val)
{
  while (r) {
    int i = r->i;
    r->a[i] = val;
    r = r->next;
  }
}
```



Register	Value
%rdi	r
%rsi	val

```
.L11:                              # loop:
  movslq  16(%rdi), %rax           #   i = M[r+16]
  movl    %esi, (%rdi,%rax,4)      #   M[r+4*i] = val
  movq    24(%rdi), %rdi           #   r = M[r+24]
  testq   %rdi, %rdi               #   Test r
  jne     .L11                     #   if !=0 goto loop
```

Practice

- Problem 3.41 p. 268

```
struct prob {
     int  *p;
     struct {
          int x;
          int y;
      }  s;
      struct prob  *next;
};
```

Problem 3.41 (cont.)

A. Offsets (in bytes) of the following fields?

p:

s.x:

s.y:

next:

B. How many total bytes does the structure require?

Problem 3.41 (cont.)

C. Assembly code for void sp_init(struct prob *sp)

```
movl   12(%rdi), %eax
movl   %eax,  8(%rdi)
leaq    8(%rdi), %rax
movq  %rax, (%rdi)
movq  %rdi,  16(%rdi)
ret
```

Fill in the C code for sp_init

Skeleton code for sp_init

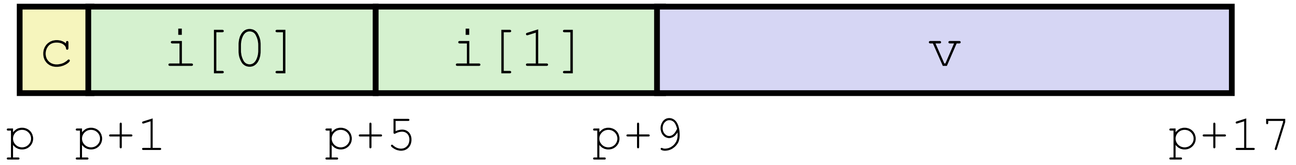
```
void sp_init(struct prob *sp){
   sp->s.x =
   sp->p  =
   sp->next =
```

Practice more with

- Problem 3.42
- You may do it on your own.
- If we have time today, we can go over it at the end, once we have finished structure alignment and unions, but please make sure you do it.

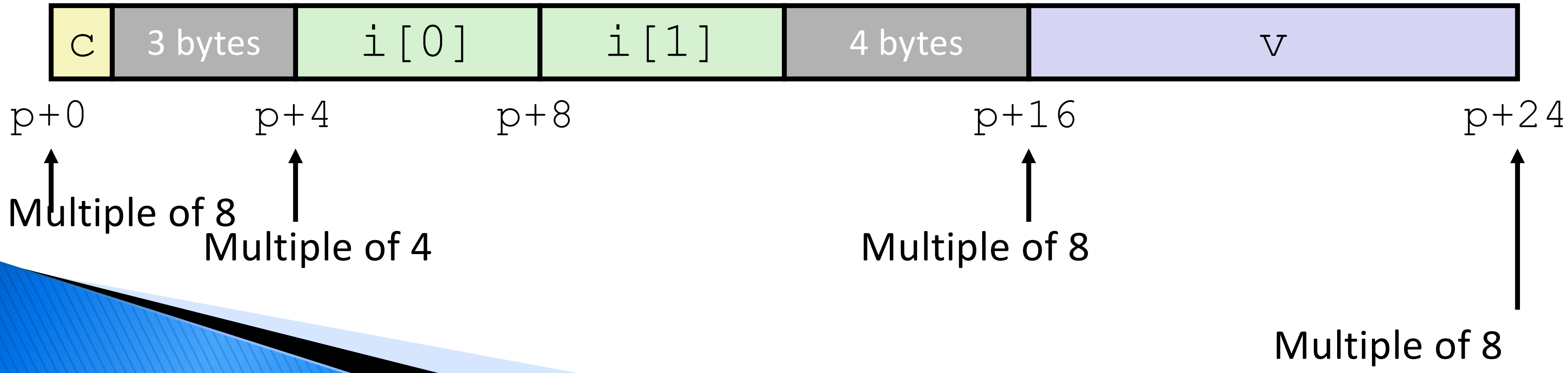
Structures & Alignment

- Unaligned Data



```
struct S1 {
  char c;
  int i[2];
  double v;
} *p;
```

- Aligned Data
  - Primitive data type requires K bytes
  - (its) Address (inside the structure) must be multiple of K



Alignment Principles

- Aligned Data
  - Primitive data type require K bytes ➔ Address must be multiple of K
  - Required on some machines; advised on x86-64
- Motivation for Aligning Data
  - Memory accessed by (aligned) chunks of 4 or 8 bytes (syst. dependent)
    - Inefficient to load or store datum that spans quad word boundaries
    - Virtual memory trickier when datum spans 2 pages
- Compiler
  - Inserts gaps in structure to ensure correct *alignment of fields*

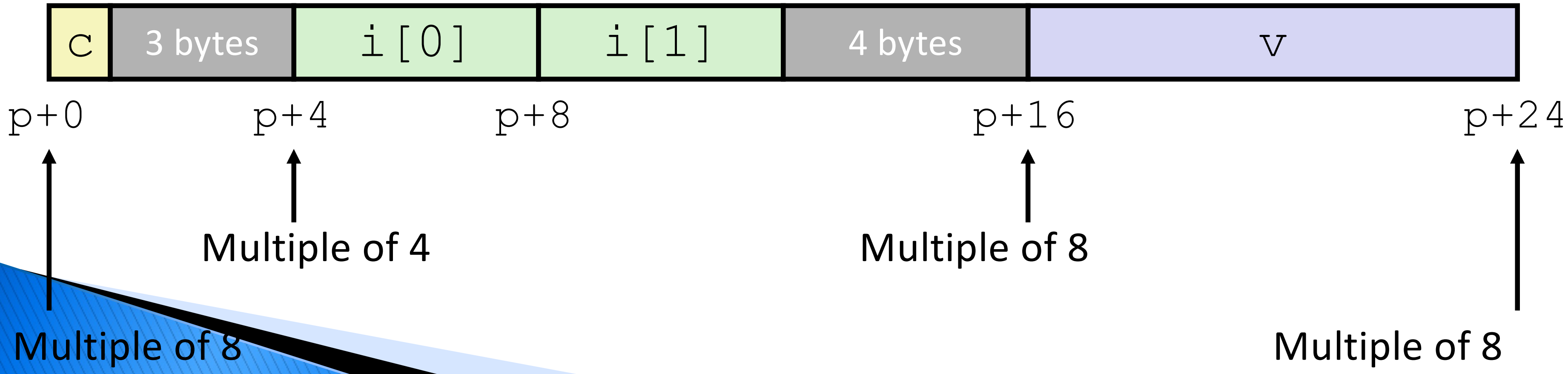
Specific Cases of Alignment (x86-64)

- K
- 1 byte: **`char`**, …
  - no restrictions on address
- 2 bytes: **`short`**, …
  - lowest 1 bit of address must be 0_2
- 4 bytes: **`int`**, **`float`**, …
  - lowest 2 bits of address must be 00_2
- 8 bytes: **`double`**, `long`, **`char *`**, …
  - lowest 3 bits of address must be 000_2
- 16 bytes: **`long double`** (GCC on Linux)
  - lowest 4 bits of address must be 0000_2

Satisfying Alignment with Structures

- Within structure:
  - Must satisfy each field's alignment requirement
- Overall structure placement
  - The structure has alignment requirement K_{struct}
    - K_{struct} = Largest alignment of any field
  - Initial address & structure length must be multiples of K_{struct}
- Example:
  - K_{struct} = 8, due to **`double`** element

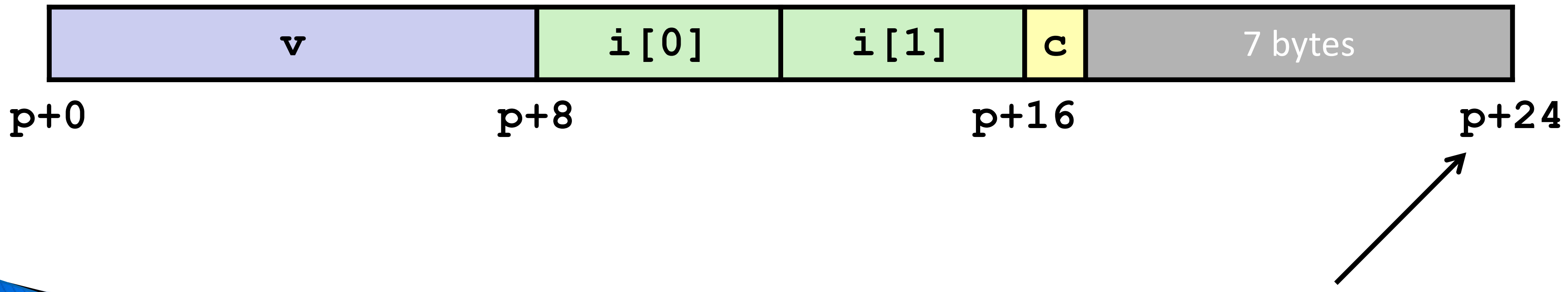
```
struct S1 {
  char c;
  int i[2];
  double v;
} *p;
```



Meeting Overall Alignment Requirement

- For largest alignment requirement K
- Overall structure must be multiple of K_{struct}
- More compact but requires user providing diff. field order

```
struct S2 {
  double v;
  int i[2];
  char c;
} *p;
```



Multiple of K=8

Practice Problem 3.44

- A. struct P1 {int I; char c; int j; char d;};
  - offset of each field?
  - total size of the structure?
  - alignment requirement, K= ?

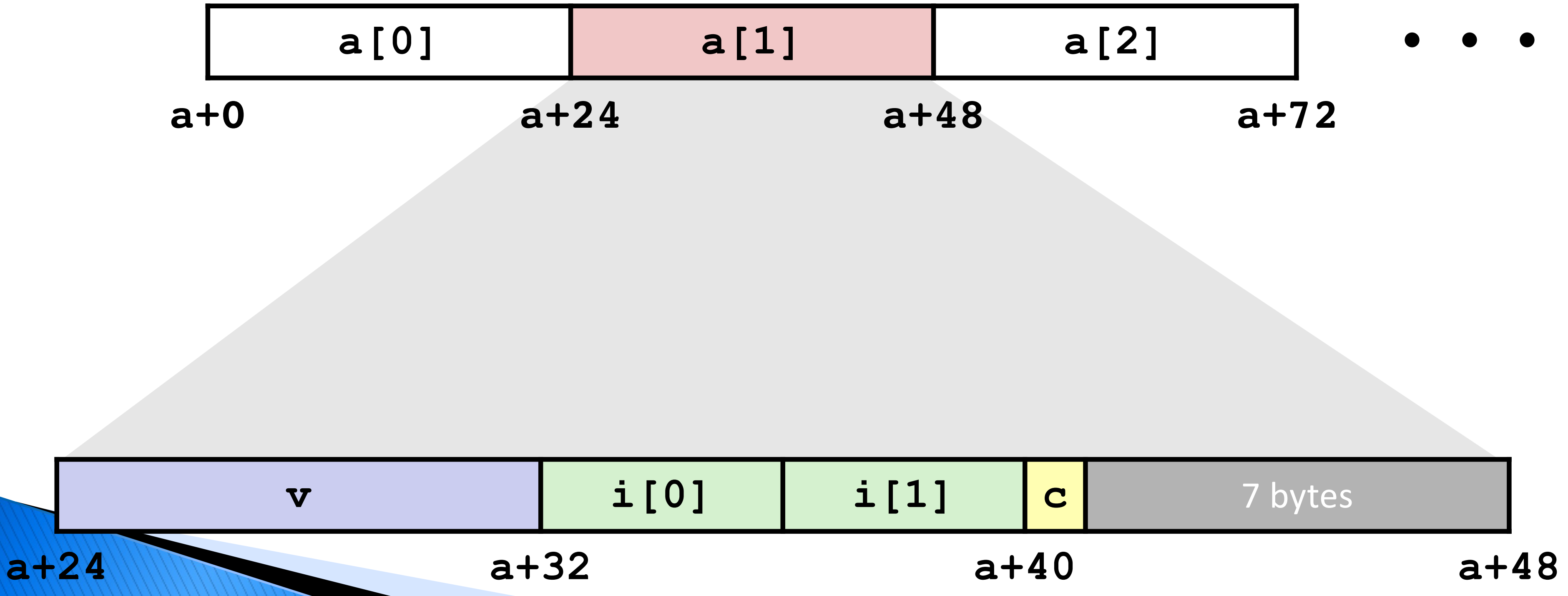
Practice Problem 3.44 (cont.)

- B. `struct P2 {int i; char c; char d;long j;};`
  - offset of each field?
  - total size of the structure?
  - alignment requirement, K= ?
- Continue with parts C, D and E of this problem

Arrays of Structures

- Overall structure length multiple of K
- Must satisfy alignment requirement for every element (aligned to 24 bytes; where is the padding?)

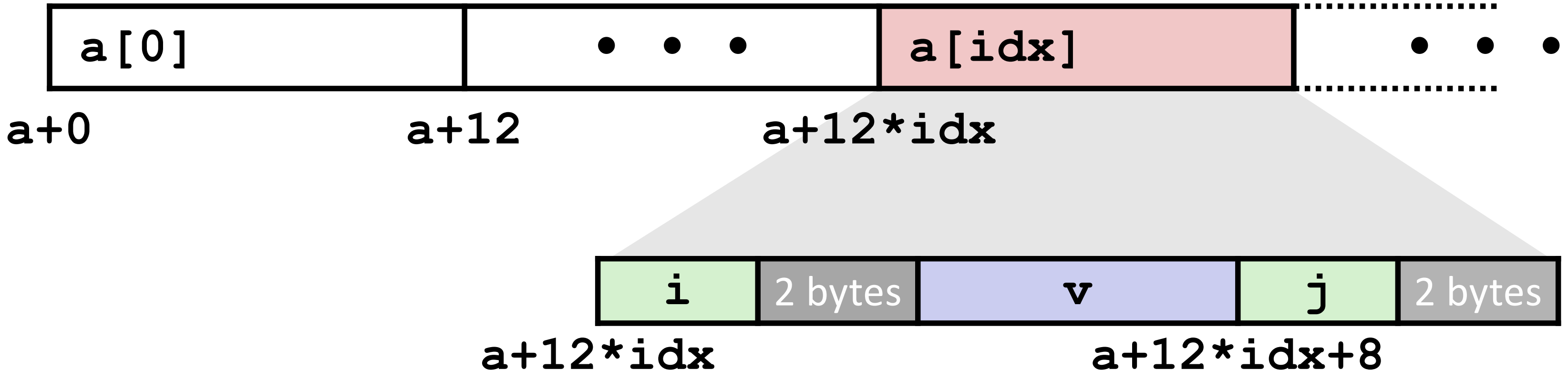
```
struct S2 {
  double v;
  int i[2];
  char c;
} a[10];
```



Accessing Array Elements

```
struct S3 {
  short i;
  float v;
  short j;
} a[10];
```

- Compute array offset 12*idx
  - **`sizeof(S3)`**, including alignment spacers
- Element **`j`** is at offset 8 within structure
- Assembler gives offset **`a+8`**
  - Resolved during linking



```
short get_j(int idx)
{
  return a[idx].j;
}
```

```
 # %rdi = idx
leaq (%rdi,%rdi,2),%rax # 3*idx
movzwl a+8(,%rax,4),%eax
```

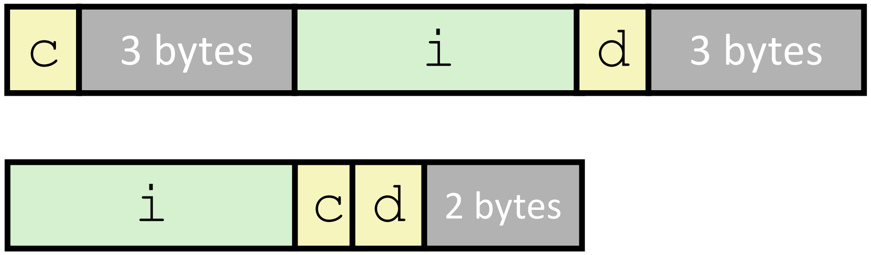
Saving Space

- Put large data types first

```
struct S4 {
  char c;
  int i;
  char d;
} *p;
```

```
struct S5 {
  int i;
  char c;
  char d;
} *p;
```

- Effect (K=4) (Multiples of 4)



But this must be done by the programmer, the compiler will not change the order of the fields in the structure, it only adds padding.

End of section 3.9.1 and 3.9.3

Now section 3.9.2 on Unions

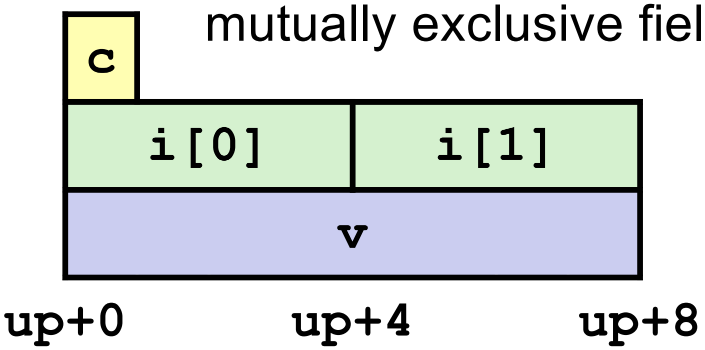
Union Allocation

- Allocate according to largest element
- Can only use one field at a time

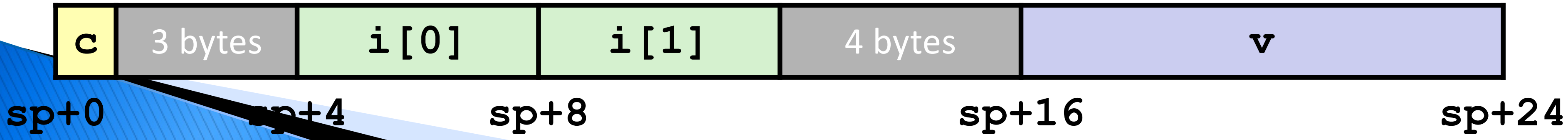
```
union U1 {
  char c;
  int i[2];
  double v;
} *up;
```

```
struct S1 {
  char c;
  int i[2];
  double v;
} *sp;
```

All fields reference the same block (for mutually exclusive fields). Overlay decl. Circumvents type system.

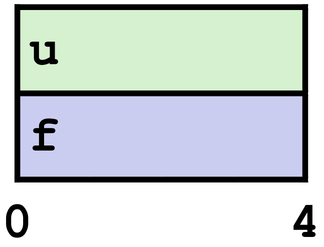


Unlike struct in which diff. fields reference diff. blocks



Using Union to Access Bit Patterns

```
typedef union {
  float f;
  unsigned u;
} bit_float_t;
```



Same location

```
float bit2float(unsigned u)
{
  bit_float_t arg;
  arg.u = u;
  return arg.f;
}
```

```
unsigned float2bit(float f)
{
  bit_float_t arg;
  arg.f = f;
  return arg.u;
}
```

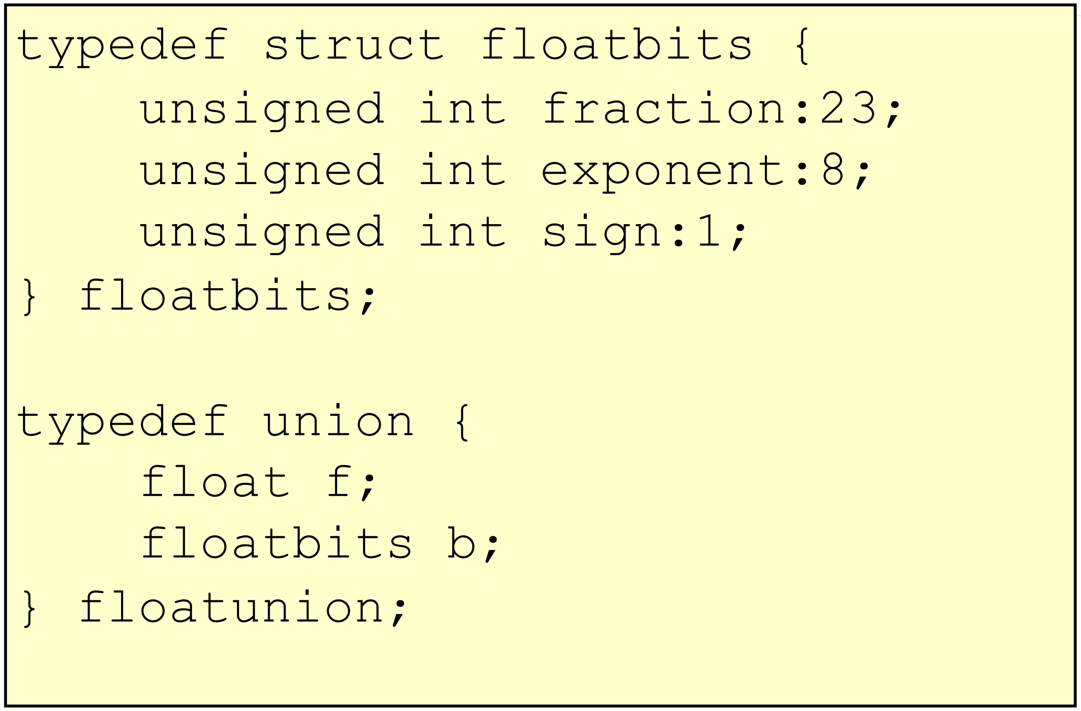
Same as **(float) u**?

Same as **(unsigned) f**?

Two separate locations

Unions, Structures, and Bitfields

- Can even break up structures and unions into smaller pieces:



```
typedef struct floatbits {
    unsigned int fraction:23;
    unsigned int exponent:8;
    unsigned int sign:1;
} floatbits;

typedef union {
    float f;
    floatbits b;
} floatunion;
```

Stop here