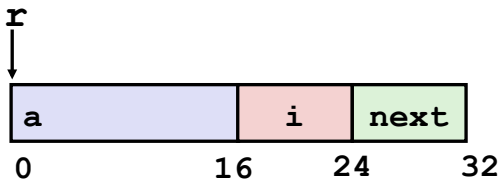
C Structs

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




```
struct rec x;
struct rec y;

x.i = 5;
x.a[1] = 2;
x.next = &y;

y = x; // copy full struct
```

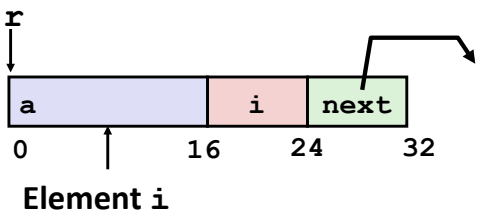
```
struct rec* z;
z = &y; // just a pointer copy

// form 1
(*z).i = 7; // NOT z.i = 7;

// form 2 (preferred)
z->i = 7;
```


typedef

```
// give type T another name: U
typedef T U;


// example: defines a type "struct rec"
// and typedefs it with another name "rec"
typedef struct rec {
  ...
} rec;


rec x; // now can omit "struct"
x.i = 5;


rec* p = (rec*) malloc(sizeof(rec));
p->i = 3;
```

Linked List Example

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



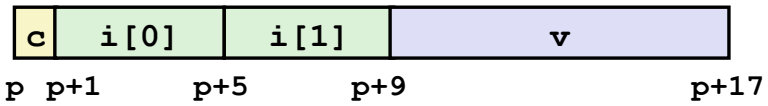

```
void set_val
  (struct rec* r, int val)
{
  while (r) {
    int i = r->i;
    r->a[i] = val;
    r = r->next;
  }
}
```

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
```

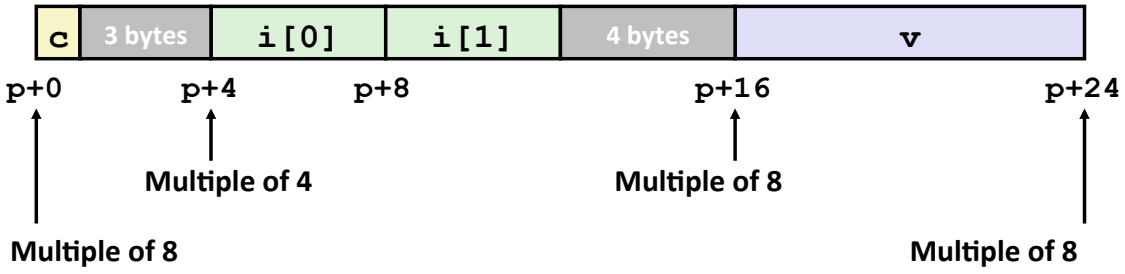

Data Alignment

Unaligned Data



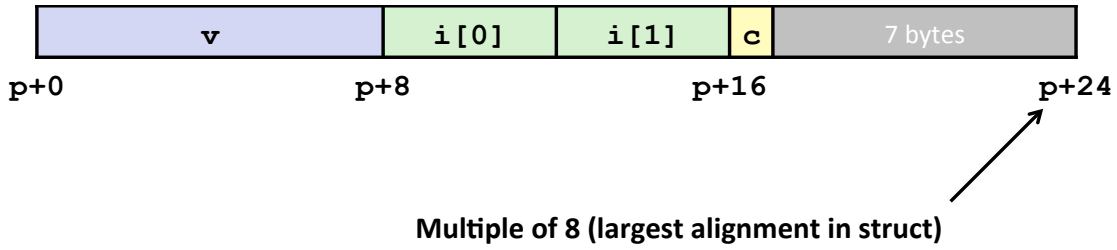

```
struct S1 {
  char c;
  int i[2];
  double v;
} *p;
```

Aligned Data



Struct Data Alignment

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

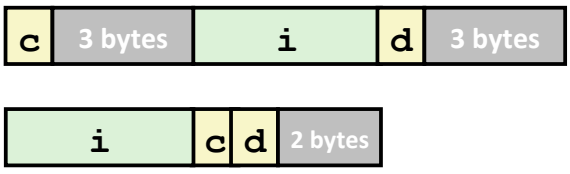


Saving Space

Put large data types first

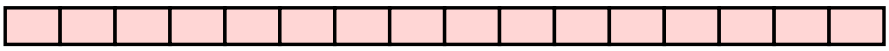
```
struct S4 {
  char c;
  int i;
  char d;
} *p;
```

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

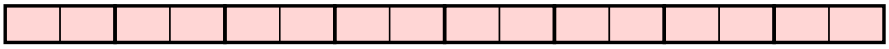


Floating Point: YMM/XMM Registers

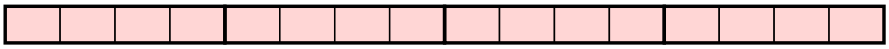
- 16 single-byte integers



- 8 16-bit integers



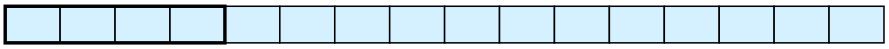
- 4 32-bit integers



- 4 single-precision floats

- 2 double-precision floats

- 1 single-precision float



- 1 double-precision float