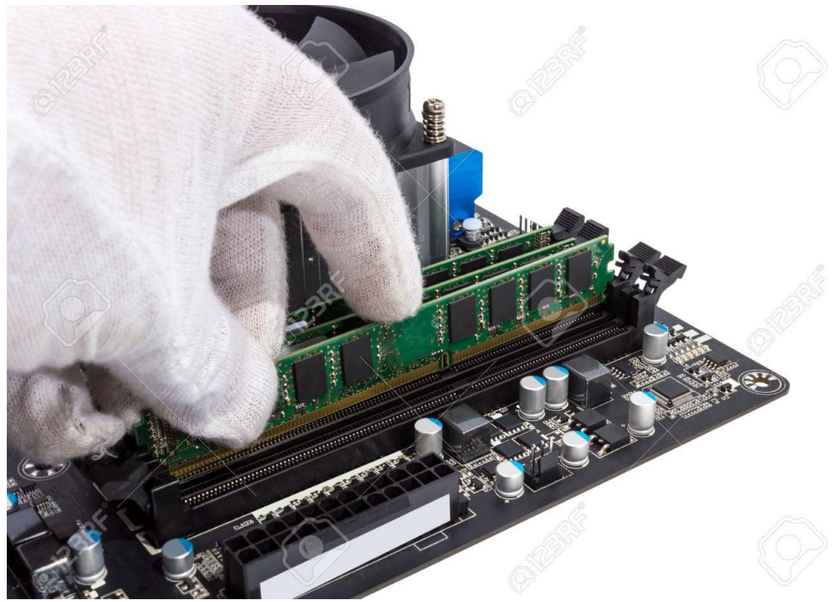
Memory and HelloWorld

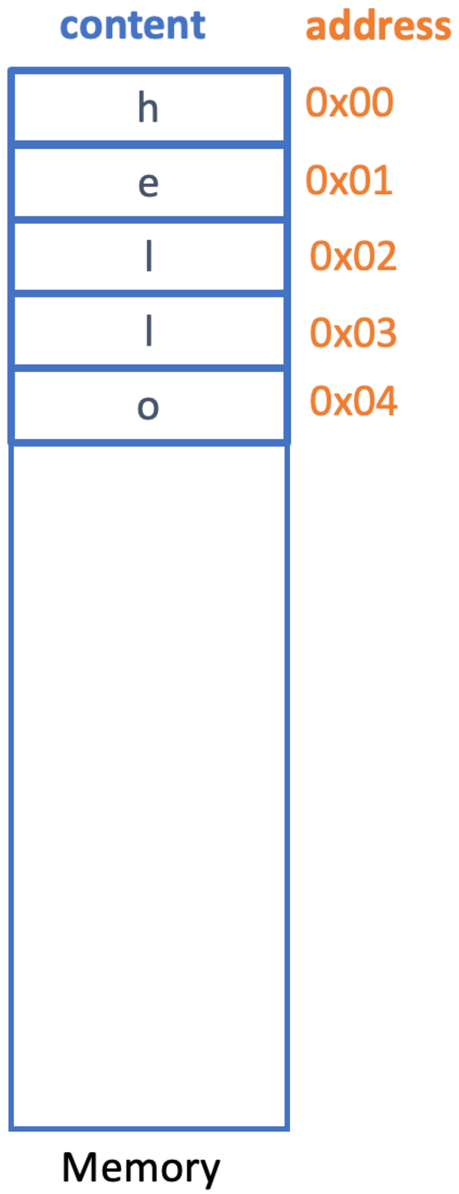
Yu-Ju Huang
Slides adapted from Yunhao Zhang

Memory

- Memory is an array of bytes
- An index into this array is called an ”address”
- **Content** and **Address**



https://www.123rf.com/



Memory address space

- content and address
  - e.g., 32bit address space can represent up to 2^32 bytes

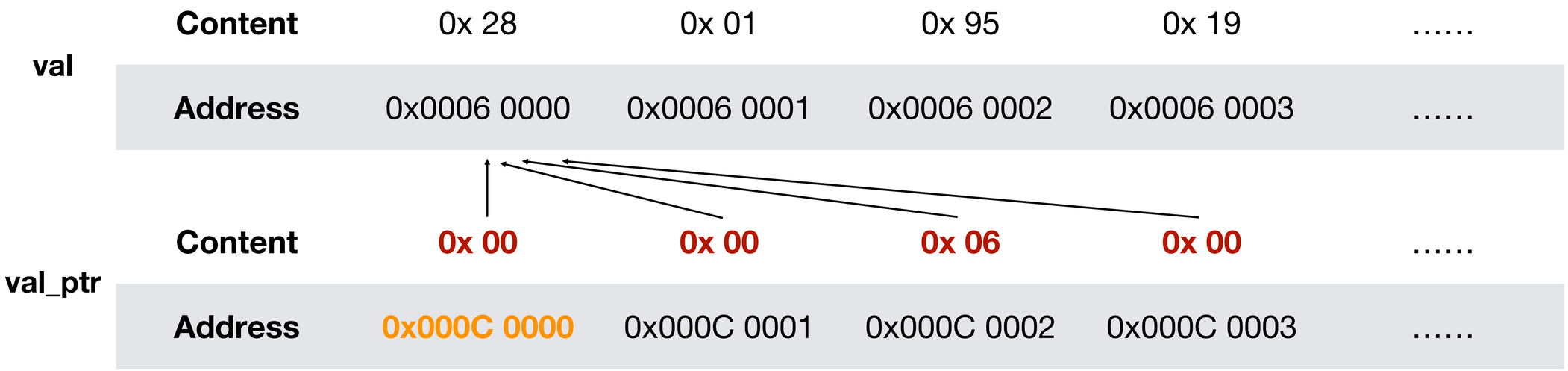
Content	1st byte	2nd byte	3rd byte	……	2^32th byte
Address	0x0000 0000	0x0000 0001	0x0000 0002	……	0xFFFF FFFF

```
int val = 0x19950128; // a global variable in C

// compiler decides where to put val in memory
// say the compiler decides to put val at 0x0006_0000
```

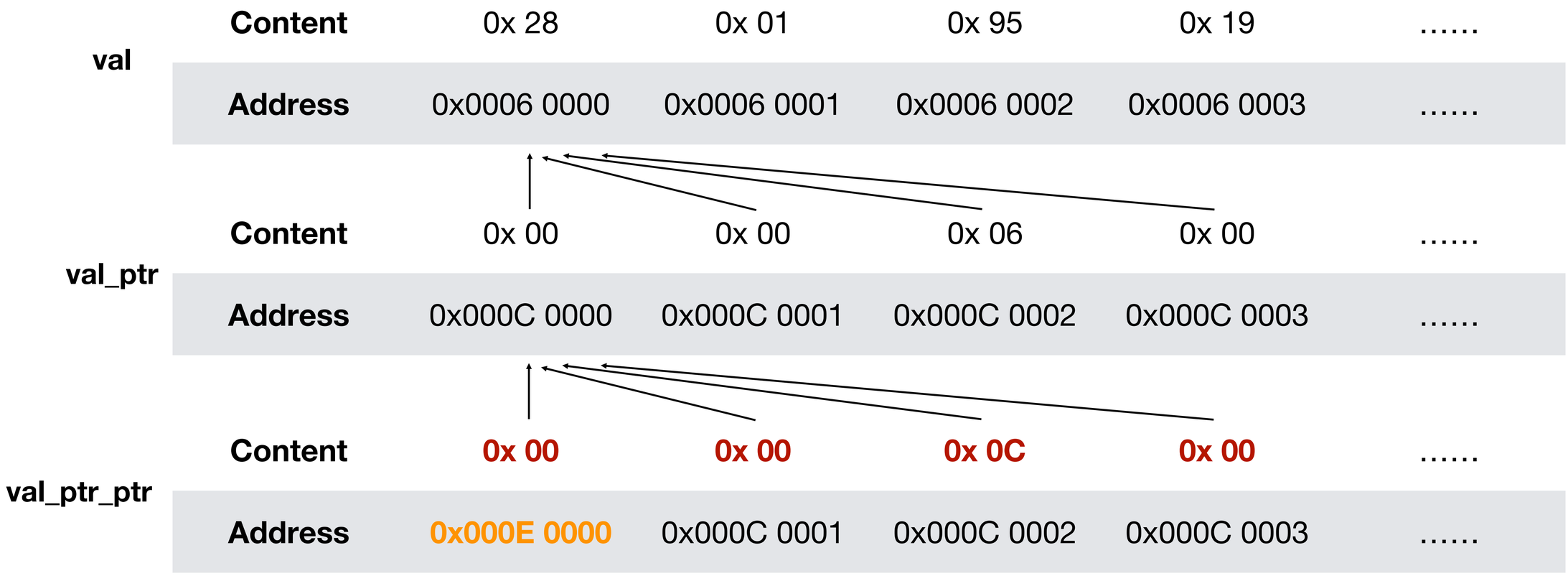
val

Content	0x 28	0x 01	0x 95	0x 19	.....
Address	**0x0006 0000**	0x0006 0001	0x0006 0002	0x0006 0003	.....

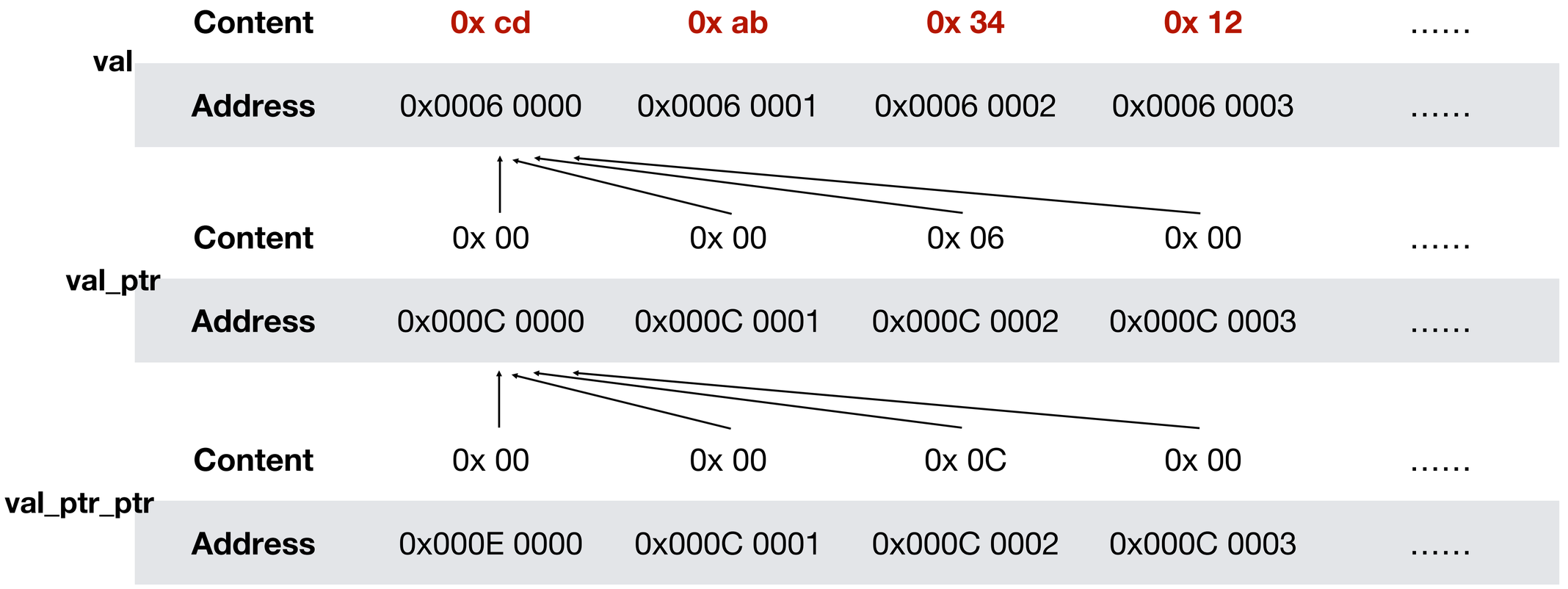
```
int val = 0x19950128;

int* val_ptr;       // another global variable
                    // say val_ptr is put at 0x000C_0000
val_ptr = &val;     //  &val=0x0006_0000
```



```
int val = 0x19950128;
int* val_ptr = &val;

int** val_ptr_ptr = &val_ptr; // a third variable
// say val_ptr_ptr is put at 0xE0000
```

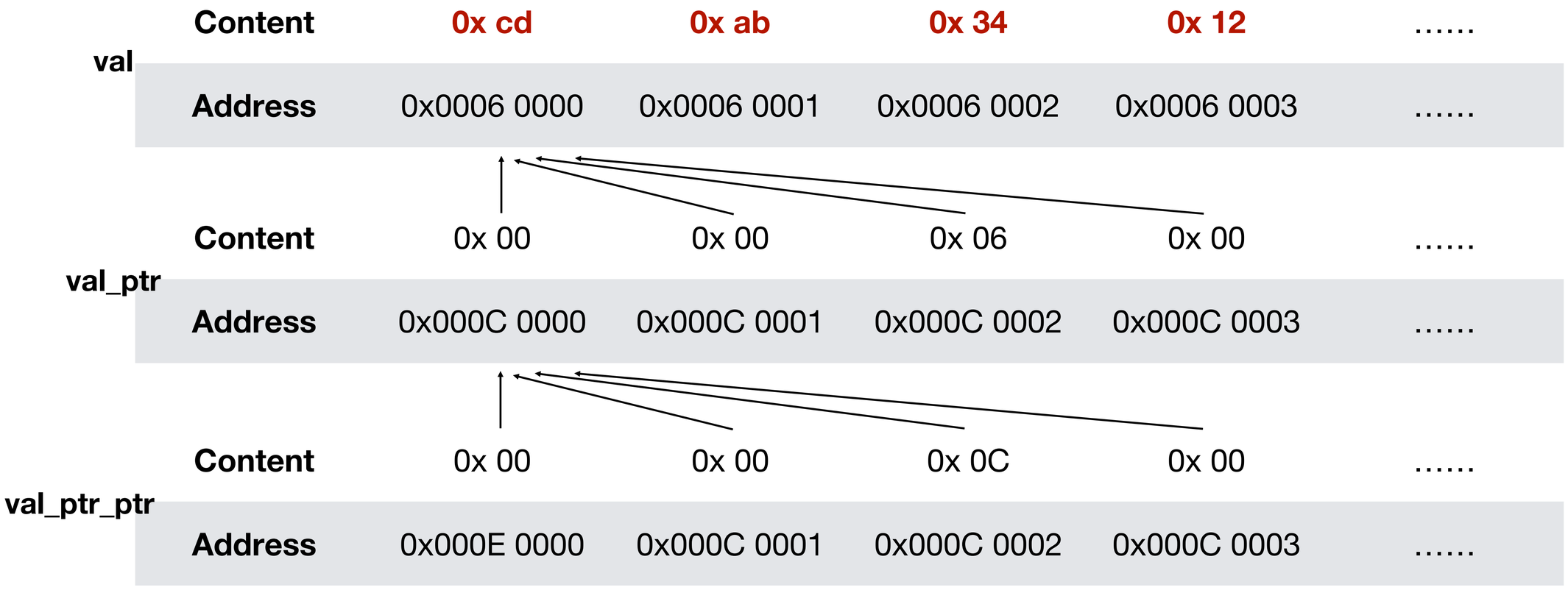


```
int val = 0x19950128;
int* val_ptr = &val;
int** val_ptr_ptr = &val_ptr;

void func_a() { val = 0x1234abcd; }
```



Write these 4 bytes

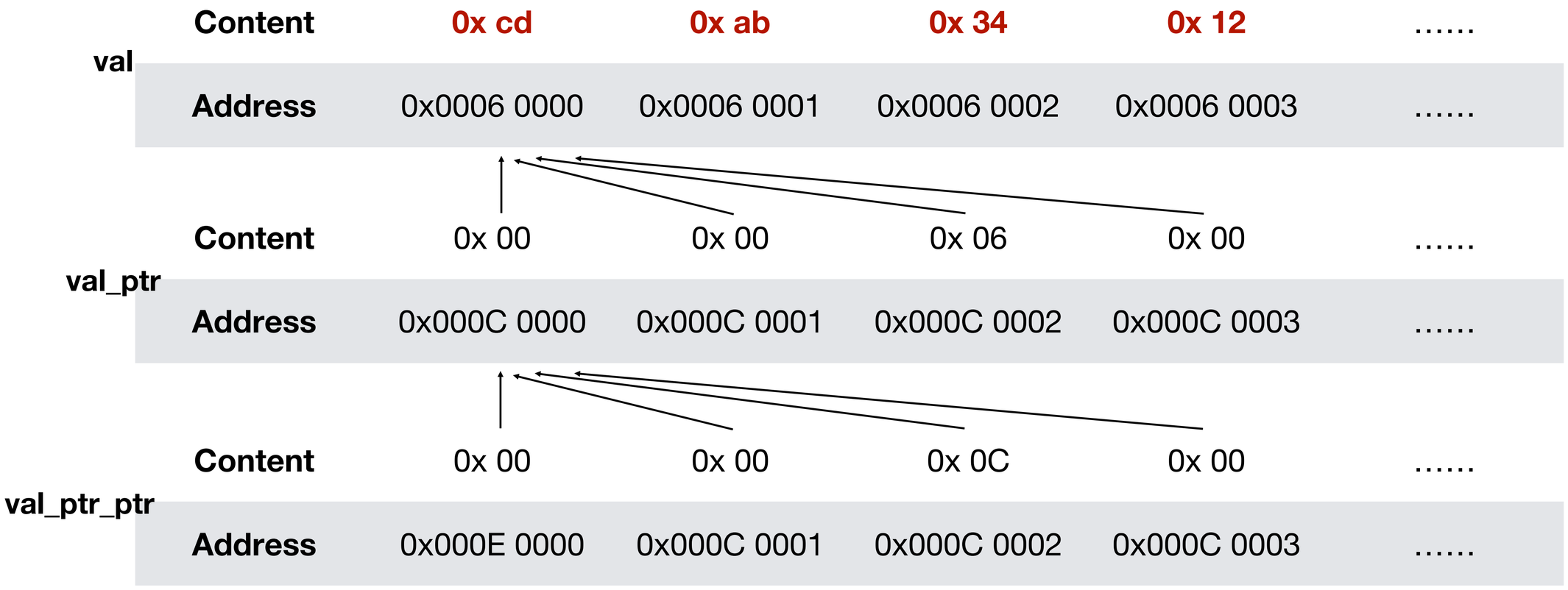
```
int val = 0x19950128;
int* val_ptr = &val;
int** val_ptr_ptr = &val_ptr;

void func_a() { *val_ptr = 0x1234abcd; }
```



Then, write these 4 bytes

First, read these 4 bytes

```
int val = 0x19950128;
int* val_ptr = &val;
int** val_ptr_ptr = &val_ptr;

void func_a() { **val_ptr_ptr = 0x1234abcd; }
```



Lastly, write these 4 bytes

Then, read these 4 bytes

First, read these 4 bytes

More types for variables

Type	sizeof(Type)	Type	sizeof(Type) (32bit CPU)	sizeof(Type) (64bit CPU)
char	1	**char***	4	8
int	4	**int***	4	8
long long	8	**long long***	4	8
float	4	**float***	4	8
void	cannot define variable of void type	**void***	4	8

Type casting

```
int val = 0x19955343;
int* val_ptr = &val;
char* char_ptr = (char*) val_ptr
char c1 = *char_ptr;        // 0x43-> ‘C’
char c2 = *(char_ptr+1);    // 0x53-> ‘S’
```

Content	0x 43	0x 53	0x 95	0x 19	.....
Address	0x0006 0000	0x0006 0001	0x0006 0002	0x0006 0003	.....
Content	0x 00	0x 00	0x 06	0x 00	.....
Address	0x000C 0000	0x000C 0001	0x000C 0002	0x000C 0003	.....

Then, read
1 byte here

First, read
these 4 bytes

Content	0x 43	.....	.....	.....	.....
Address	0x000F 0000	.....	.....	.....	.....

Lastly, write
1 byte here

Why void pointer?

```
char item0 = ‘a’;
int item1 = 0x19950128;
float item2 = 3.14;

queue_t q = queue_new();

// queue is generic
queue_enqueue(q, &item0);
queue_enqueue(q, &item1);
queue_enqueue(q, &item2);
```

```
// It’s up to the user of the
// queue to decide whether
// the item is char, int or
// other types
```

```
char item0 = ‘a’;
int item1 = 0x19950128;
float item2 = 3.14;

queue_t q = queue_new();

// queue is generic
queue_enqueue(q, &item0);
queue_enqueue(q, &item1);
queue_enqueue(q, &item2);
```

```
char* item3;
int* item4;
float* item5;

queue_dequeue(q, &item3);
queue_dequeue(q, &item4);
queue_dequeue(q, &item5);

// now
// item3 == &item0
// item4 == &item1
// item5 == &item2
```

Why pointer of pointer?

```
char* item3;
int* item4;
float* item5;

queue_dequeue(q, &item3);
queue_dequeue(q, &item4);
queue_dequeue(q, &item5);

// item3 == &item0
// item4 == &item1
// item5 == &item2
```

```
// queue_dequeue needs to
// modify item3, so it takes
// the address of item3, namely
// &item3, as parameter

// and &item3 is a pointer to
// a pointer with type void**
```

```
// Global variable
char item0 = ‘a’;

void test() {
  char* item3;
  queue_enqueue(q, &item0);
  queue_dequeue(q, &item3);
  // item3 == &item0
}
```

```
int queue_dequeue(queue_t q,
void** pitem){

pitem = … // modifies the local
          // variable pitem

*pitem = … // modifies the item3
           // variable in test()

**pitem = … // modifies ???

// only one above is needed

}
```

malloc/free

- Allocate memory on-demand
- Allocate memory when the size is determined during runtime
- If *X* allocates memory, *X* frees it.
  - Queue implementation allocates memory, Queue needs to free it.
  - Queue test allocates memory, test needs to free it.

Let’s put it all together

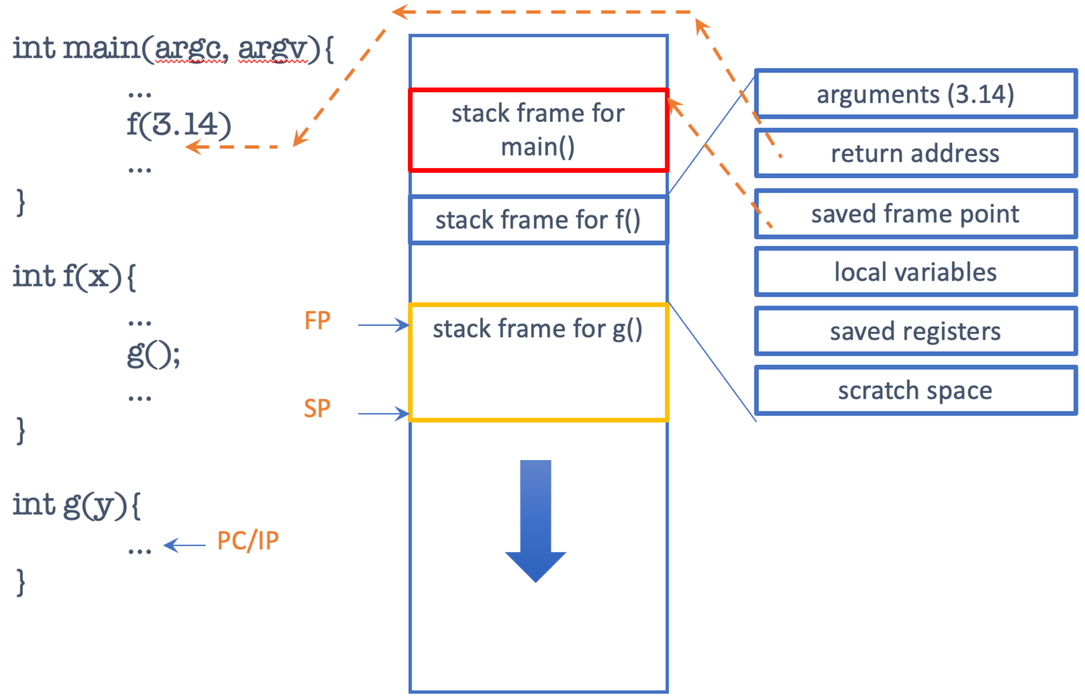
HelloWorld

```c
int main() {
    char* msg = "Hello, World!\n\r";
    terminal_write(msg, 15);

    /* Uncomment this line of code
     * when implementing formatted output
     */
    /*
    printf("%d is a number\n", 1234);
    printf("%s is a string\n", "abcd");
    printf("%s-%d is awesome!\n", "egos", 2000);

    printf("%c is character $\n", '$');
    printf("%c is character 0\n", (char)48);
    printf("%x is integer 1234 in hexadecimal\n", 1234);
    printf("%u is the maximum of unsigned int\n", (unsigned int)0xFFFFFFFF);
    printf("%p is the hexadecimal address of the hello-world string\n", msg);
    printf("%lu is the maximum of unsigned long long\n", 0xFFFFFFFFFFFFFFFFUL);
    */

    return 0;
}
```

Prolog/Epilog

```c
int main() {
    char* msg = "Hello, World!\n\r";
    terminal_write(msg, 15);

    return 0;
}
```

```
80000078 <main>:
main():
80000078:   fe010113           addi    sp,sp,-32
8000007c:   00112e23           sw  ra,28(sp)
80000080:   00812c23           sw  s0,24(sp)
80000084:   02010413           addi    s0,sp,32
80000088:   800007b7           lui a5,0x80000
8000008c:   0b878793           addi    a5,a5,184 # 800000b8 <main+0x40>
80000090:   fef42623           sw  a5,-20(s0)
80000094:   00f00593           li  a1,15
80000098:   fec42503           lw  a0,-20(s0)
8000009c:   f75ff0ef           jal 80000010 <terminal_write>
800000a0:   00000793           li  a5,0
800000a4:   00078513           mv  a0,a5
800000a8:   01c12083           lw  ra,28(sp)
800000ac:   01812403           lw  s0,24(sp)
800000b0:   02010113           addi    sp,sp,32
800000b4:   00008067           ret
```



Loading constant string

```c
int main() {
    char* msg = "Hello, World!\n\r";
    terminal_write(msg, 15);

    return 0;
}
```

```
80000078 <main>:
main():
80000078:   fe010113           addi    sp,sp,-32
8000007c:   00112e23           sw  ra,28(sp)
80000080:   00812c23           sw  s0,24(sp)
80000084:   02010413           addi    s0,sp,32
80000088:   800007b7           lui a5,0x80000
8000008c:   0b878793           addi    a5,a5,184 # 800000b8 <main+0x40>
80000090:   fef42623           sw  a5,-20(s0)
80000094:   00f00593           li  a1,15
80000098:   fec42503           lw  a0,-20(s0)
8000009c:   f75ff0ef           jal 80000010 <terminal_write>
800000a0:   00000793           li  a5,0
800000a4:   00078513           mv  a0,a5
800000a8:   01c12083           lw  ra,28(sp)
800000ac:   01812403           lw  s0,24(sp)
800000b0:   02010113           addi    sp,sp,32
800000b4:   00008067           ret
```

Function call

```c
int main() {
    char* msg = "Hello, World!\n\r";
    terminal_write(msg, 15);

    return 0;
}
```

```
80000078 <main>:
main():
80000078:   fe010113           addi    sp,sp,-32
8000007c:   00112e23           sw  ra,28(sp)
80000080:   00812c23           sw  s0,24(sp)
80000084:   02010413           addi    s0,sp,32
80000088:   800007b7           lui a5,0x80000
8000008c:   0b878793           addi    a5,a5,184 # 800000b8 <main+0x40>
80000090:   fef42623           sw  a5,-20(s0)
80000094:   00f00593           li  a1,15
80000098:   fec42503           lw  a0,-20(s0)
8000009c:   f75ff0ef           jal 80000010 <terminal_write>
800000a0:   00000793           li  a5,0
800000a4:   00078513           mv  a0,a5
800000a8:   01c12083           lw  ra,28(sp)
800000ac:   01812403           lw  s0,24(sp)
800000b0:   02010113           addi    sp,sp,32
800000b4:   00008067           ret
```

terminal_write

```
1  void terminal_write(const char *str, int len) {
2      for (int i = 0; i < len; i++) {
3          *(char*)(0x10000000UL) = str[i];
4      }
5  }
```

Function return value

```c
int main() {
    char* msg = "Hello, World!\n\r";
    terminal_write(msg, 15);

    return 0;
}
```

```
80000078 <main>:
main():
80000078:   fe010113           addi    sp,sp,-32
8000007c:   00112e23           sw  ra,28(sp)
80000080:   00812c23           sw  s0,24(sp)
80000084:   02010413           addi    s0,sp,32
80000088:   800007b7           lui a5,0x80000
8000008c:   0b878793           addi    a5,a5,184 # 800000b8 <main+0x40>
80000090:   fef42623           sw  a5,-20(s0)
80000094:   00f00593           li  a1,15
80000098:   fec42503           lw  a0,-20(s0)
8000009c:   f75ff0ef           jal 80000010 <terminal_write>
800000a0:   00000793           li  a5,0
800000a4:   00078513           mv  a0,a5
800000a8:   01c12083           lw  ra,28(sp)
800000ac:   01812403           lw  s0,24(sp)
800000b0:   02010113           addi    sp,sp,32
800000b4:   00008067           ret
```

Today

- P0 (Queue) due next week
- P1 (HelloWorld) release today, due in two weeks
  - Find your teammate!
- Next week: user-level thread library