

变量和指针变量

信息科学技术学院
wangzhao@pku.edu.cn



高级语言 中的 变量

什么是 变量？



一种 存放数据 的 容器



高级语言 中的 变量

变量是如何实现的？



一个变量 对应于
存储器中的若干连续字节



高级语言 中的 变量

- 每个变量 都有一个 字符串形式的名字

| 地址 | 存储单元 | |
|-----------|-----------------|------------------|
| 0 (0000) | 1 0 0 1 0 1 0 1 | → 1Byte 4Byte |
| 1 (0001) | 1 1 0 0 1 0 1 0 | |
| 2 (0010) | 0 1 0 0 1 0 1 1 | |
| 3 (0011) | 0 1 1 0 0 1 0 0 | |
| 4 (0100) | 1 0 1 0 1 0 1 1 | 8Byte |
| | 1 1 1 0 0 1 0 1 | |
| | 1 1 0 1 0 1 1 0 | |
| ... | 1 0 0 0 1 0 0 1 | |
| | 1 1 0 1 0 1 1 0 | |
| | 1 0 0 1 0 1 1 1 | |
| | 0 0 0 0 1 0 0 0 | |
| | 1 0 1 0 1 1 1 1 | |
| | 0 0 0 1 1 1 0 1 | |
| | 0 0 1 1 1 0 0 0 | |
| 14 (1110) | 1 0 0 0 0 0 1 0 | |
| 15 (1111) | 0 1 0 1 1 1 0 1 | |

//变量

char a; //字符

short x; //短整数

int len; //整数

long max; //长整数

float number; //单精度
浮点数

a的地址: 0000 (1)

x的地址: 0001 (2)

len的地址: 0100 (4)

max的地址: 1000 (8)

number的地址: 1100 (12)



高级语言 中的 变量

变量的 优点?



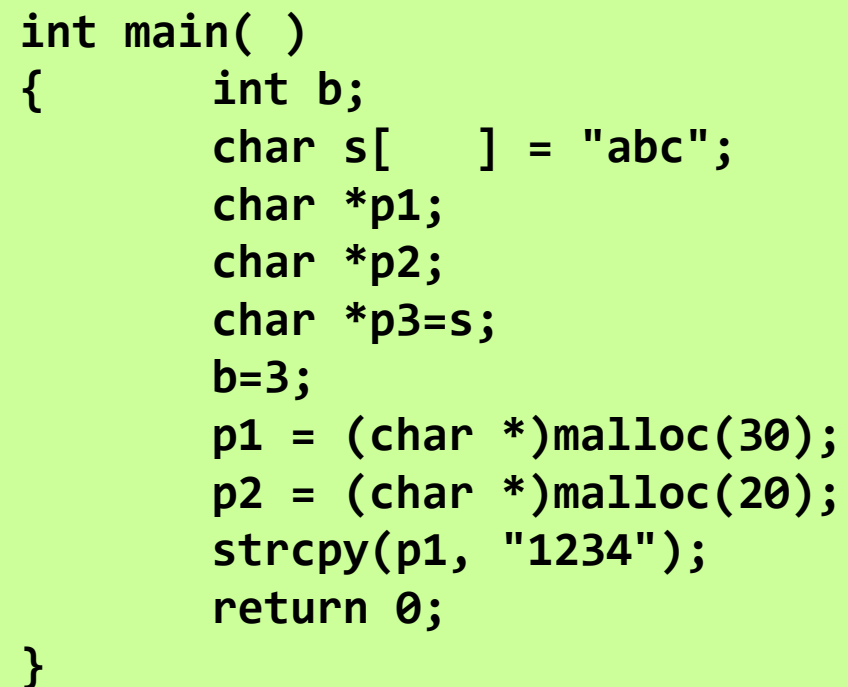
1. 编程人员不需要关心变量的具体存放位置
2. 变量的名字可以帮助人们直观的理解变量所代表的物理含义
3. 在程序中，通过变量的名字就可以实现对变量中数据的读和写





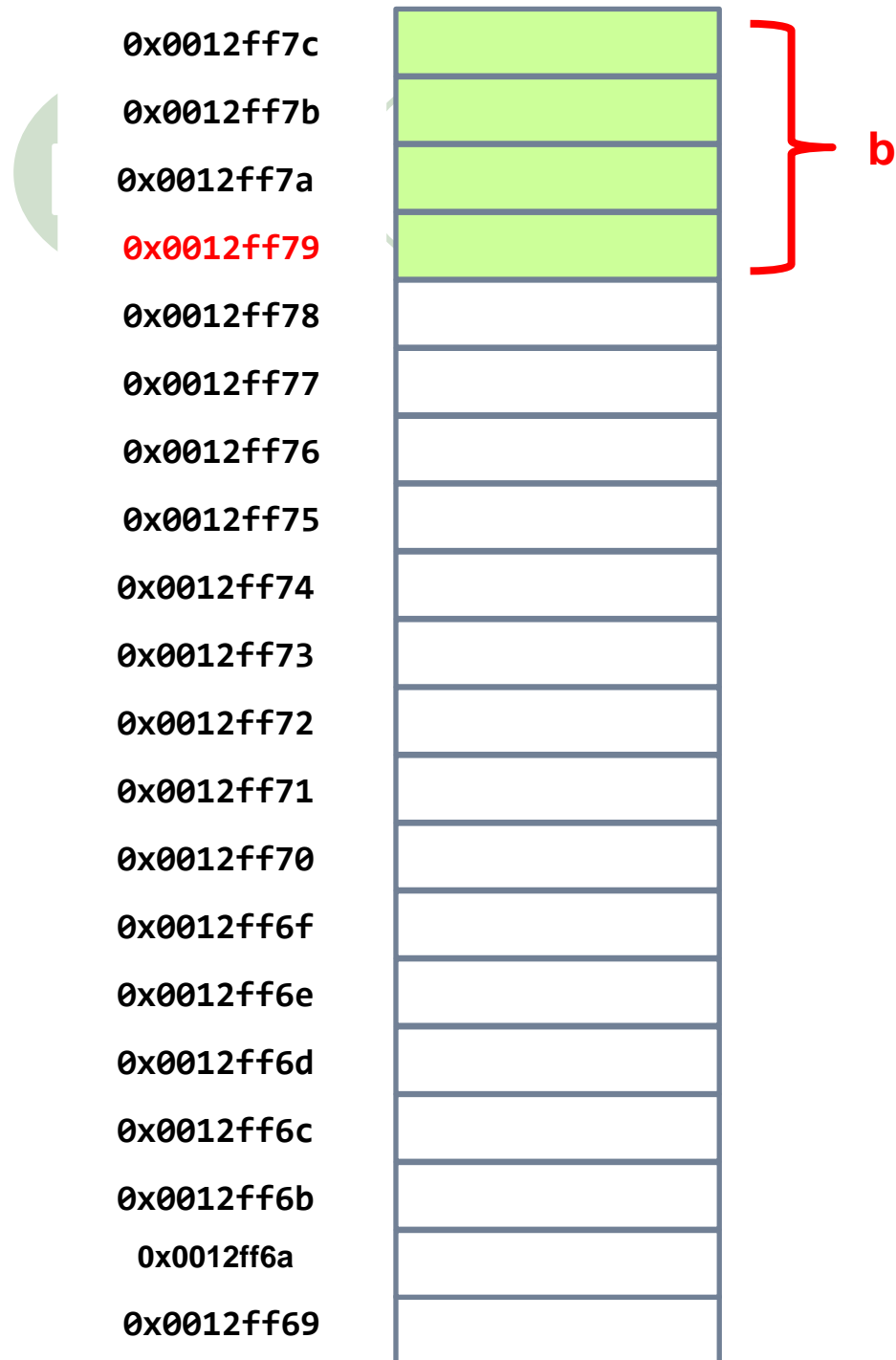
A diagram of a memory stack. On the left, a list of 20 memory addresses is shown, starting from 0x0012ff7c at the top and decreasing by 1 to 0x0012ff69 at the bottom. To the right of these addresses is a vertical column of 20 empty rectangular boxes, each corresponding to an address. A light green semi-circular shape is positioned to the left of the top four addresses (0x0012ff7c to 0x0012ff77).

| | |
|------------|--|
| 0x0012ff7c | |
| 0x0012ff7b | |
| 0x0012ff7a | |
| 0x0012ff79 | |
| 0x0012ff78 | |
| 0x0012ff77 | |
| 0x0012ff76 | |
| 0x0012ff75 | |
| 0x0012ff74 | |
| 0x0012ff73 | |
| 0x0012ff72 | |
| 0x0012ff71 | |
| 0x0012ff70 | |
| 0x0012ff6f | |
| 0x0012ff6e | |
| 0x0012ff6d | |
| 0x0012ff6c | |
| 0x0012ff6b | |
| 0x0012ff6a | |
| 0x0012ff69 | |

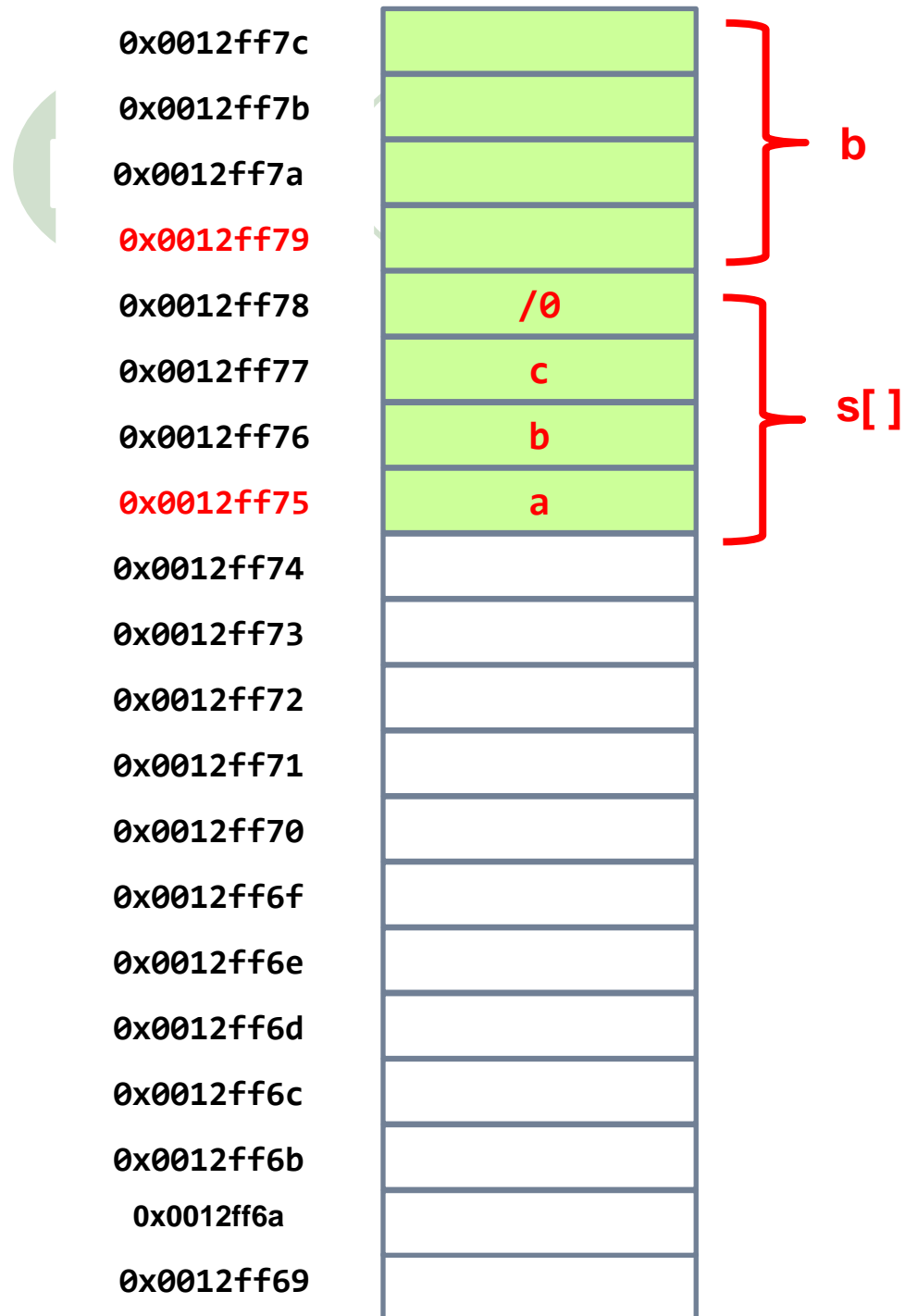


A C code snippet is displayed on a light green rectangular background. The code defines a main function that declares an integer b, a character array s containing "abc", and three character pointers p1, p2, and p3. p1 and p2 are allocated memory using malloc (30 and 20 bytes respectively), and p3 is assigned the address of s. The string "1234" is copied into p1 using strcpy. The function returns 0. In the background, three light green circles are visible.

```
int main( )
{
    int b;
    char s[ ] = "abc";
    char *p1;
    char *p2;
    char *p3=s;
    b=3;
    p1 = (char *)malloc(30);
    p2 = (char *)malloc(20);
    strcpy(p1, "1234");
    return 0;
}
```

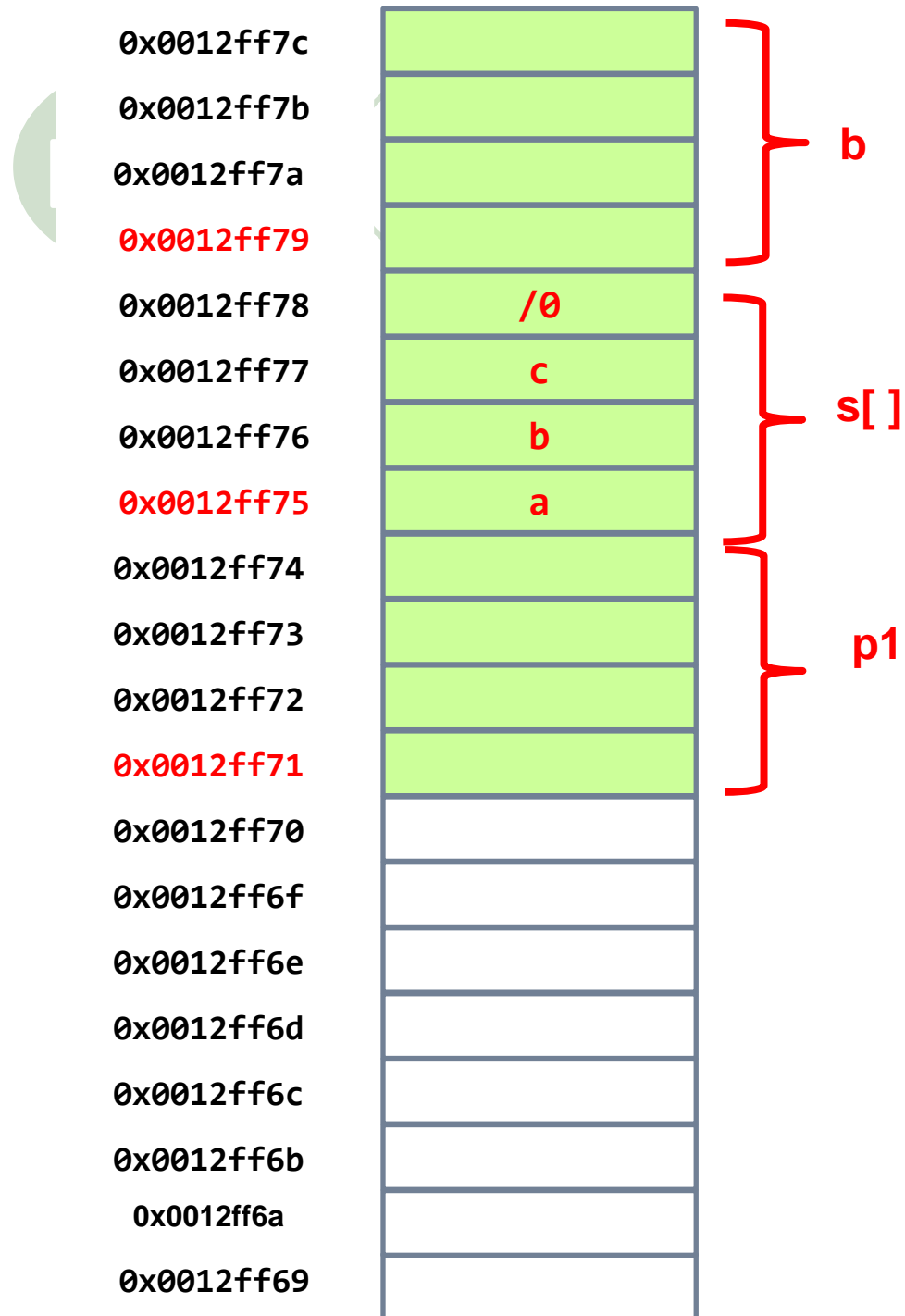


```
int main( )  
{  
    int b;  
    char s[ ] = "abc";  
    char *p1;  
    char *p2;  
    char *p3=s;  
    b=3;  
    p1 = (char *)malloc(30);  
    p2 = (char *)malloc(20);  
    strcpy(p1, "1234");  
    return 0;  
}
```



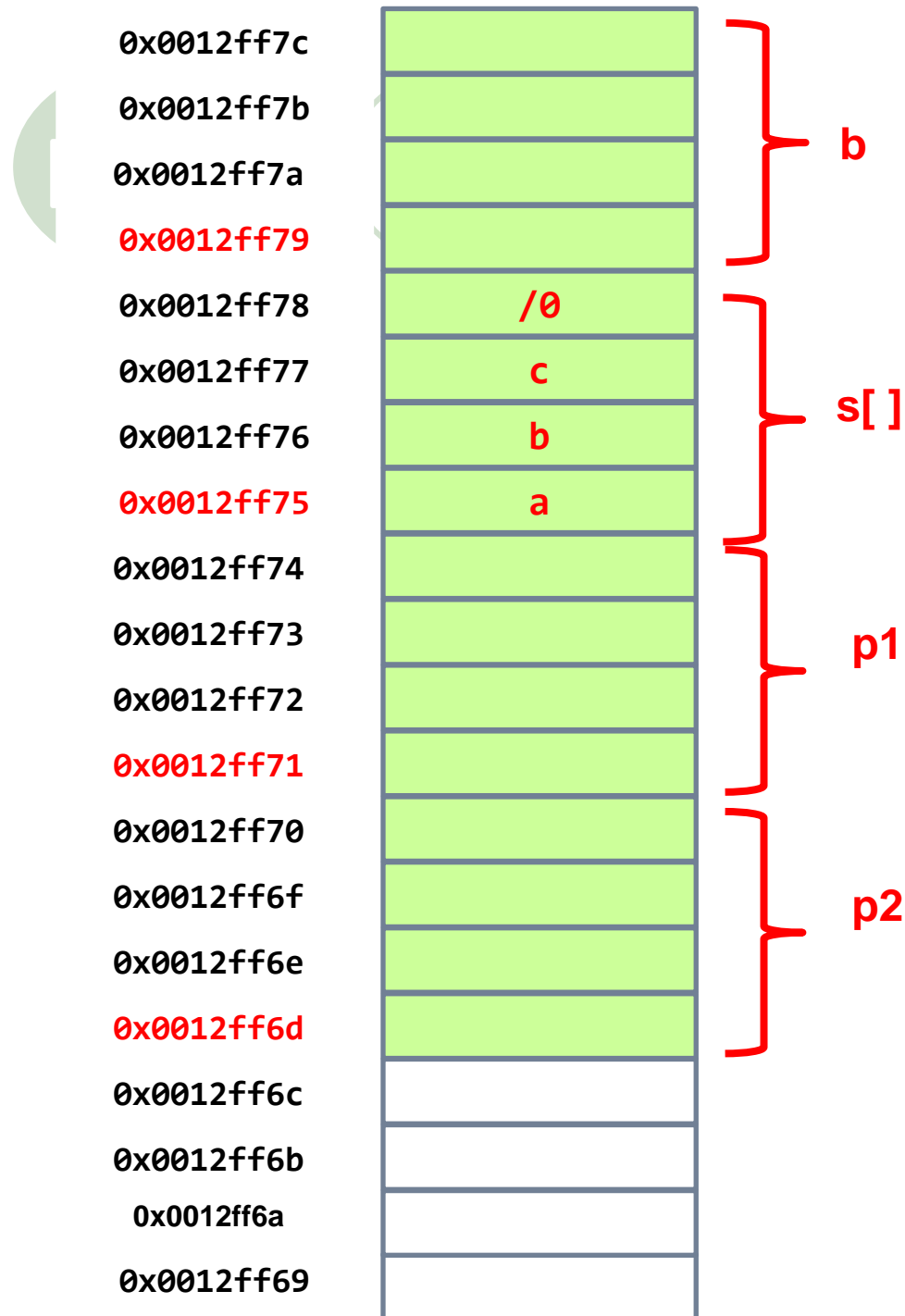
```
int main( )
{
    int b;
    char s[ ] = "abc";
    char *p1;
    char *p2;
    char *p3=s;
    b=3;
    p1 = (char *)malloc(30);
    p2 = (char *)malloc(20);
    strcpy(p1, "1234");
    return 0;
}
```





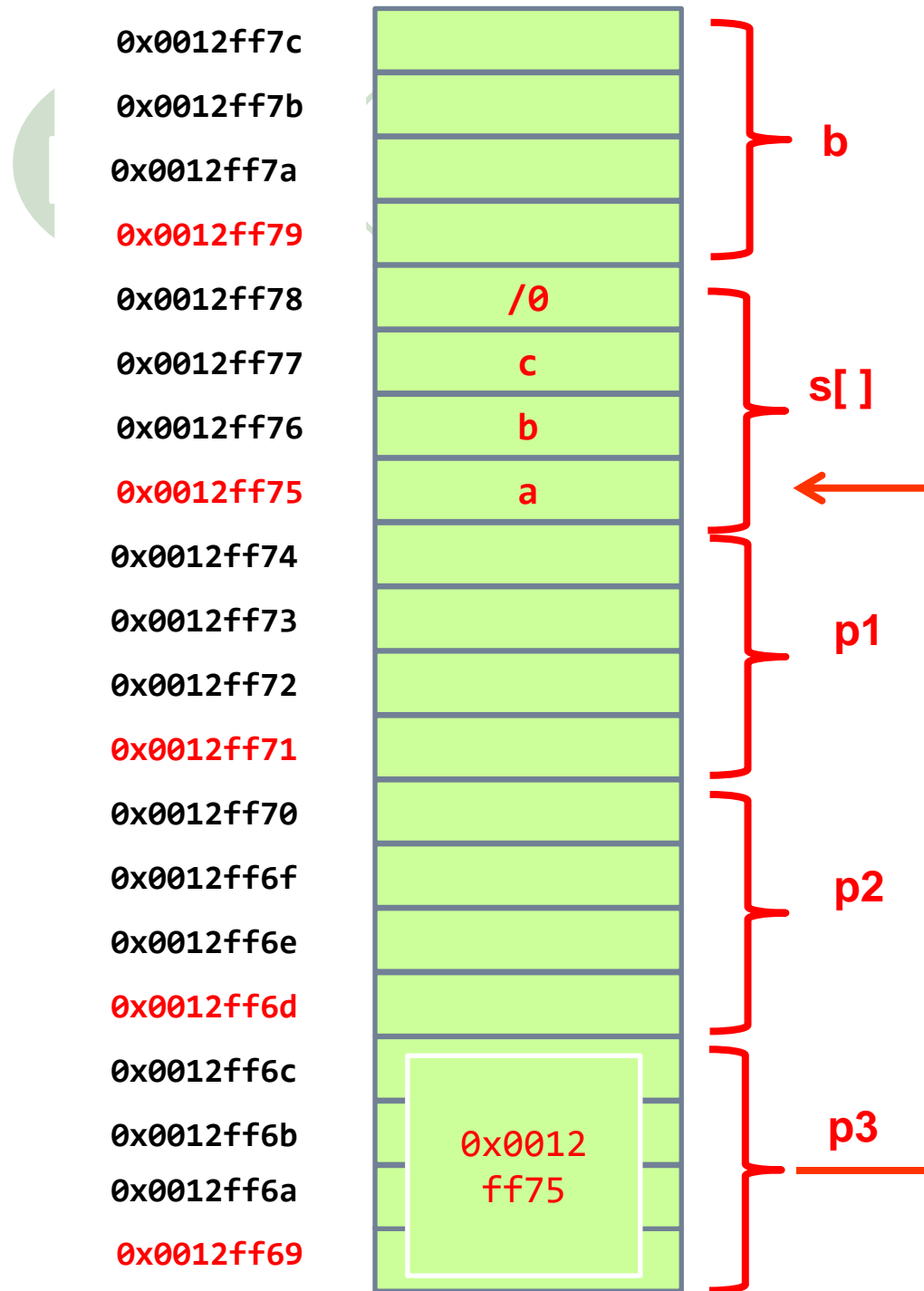
```
int main( )
{
    int b;
    char s[ ] = "abc";
    char *p1;
    char *p2;
    char *p3=s;
    b=3;
    p1 = (char *)malloc(30);
    p2 = (char *)malloc(20);
    strcpy(p1, "1234");
    return 0;
}
```



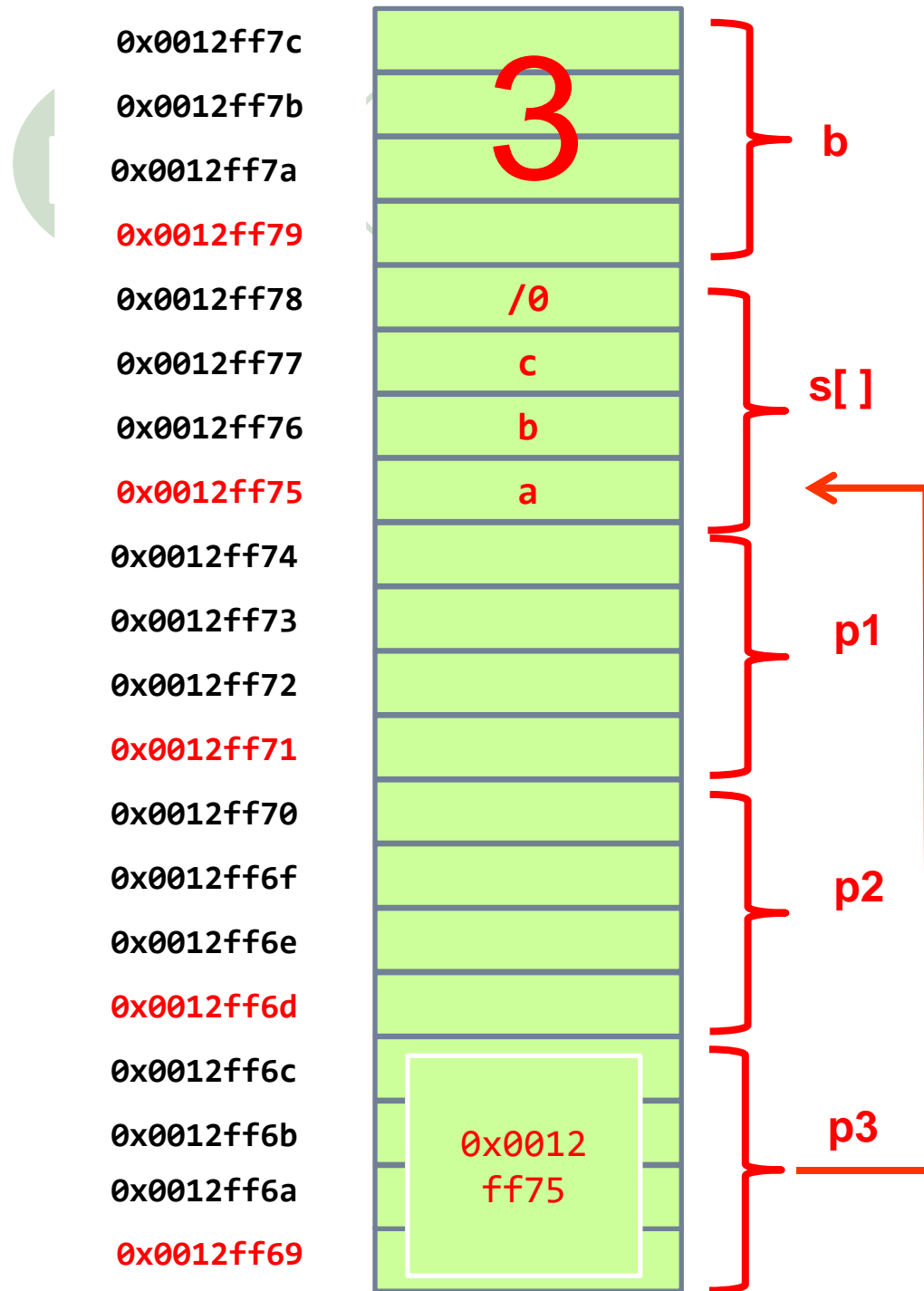


```
int main( )
{
    int b;
    char s[    ] = "abc";
    char *p1;
    char *p2;
    char *p3=s;
    b=3;
    p1 = (char *)malloc(30);
    p2 = (char *)malloc(20);
    strcpy(p1, "1234");
    return 0;
}
```

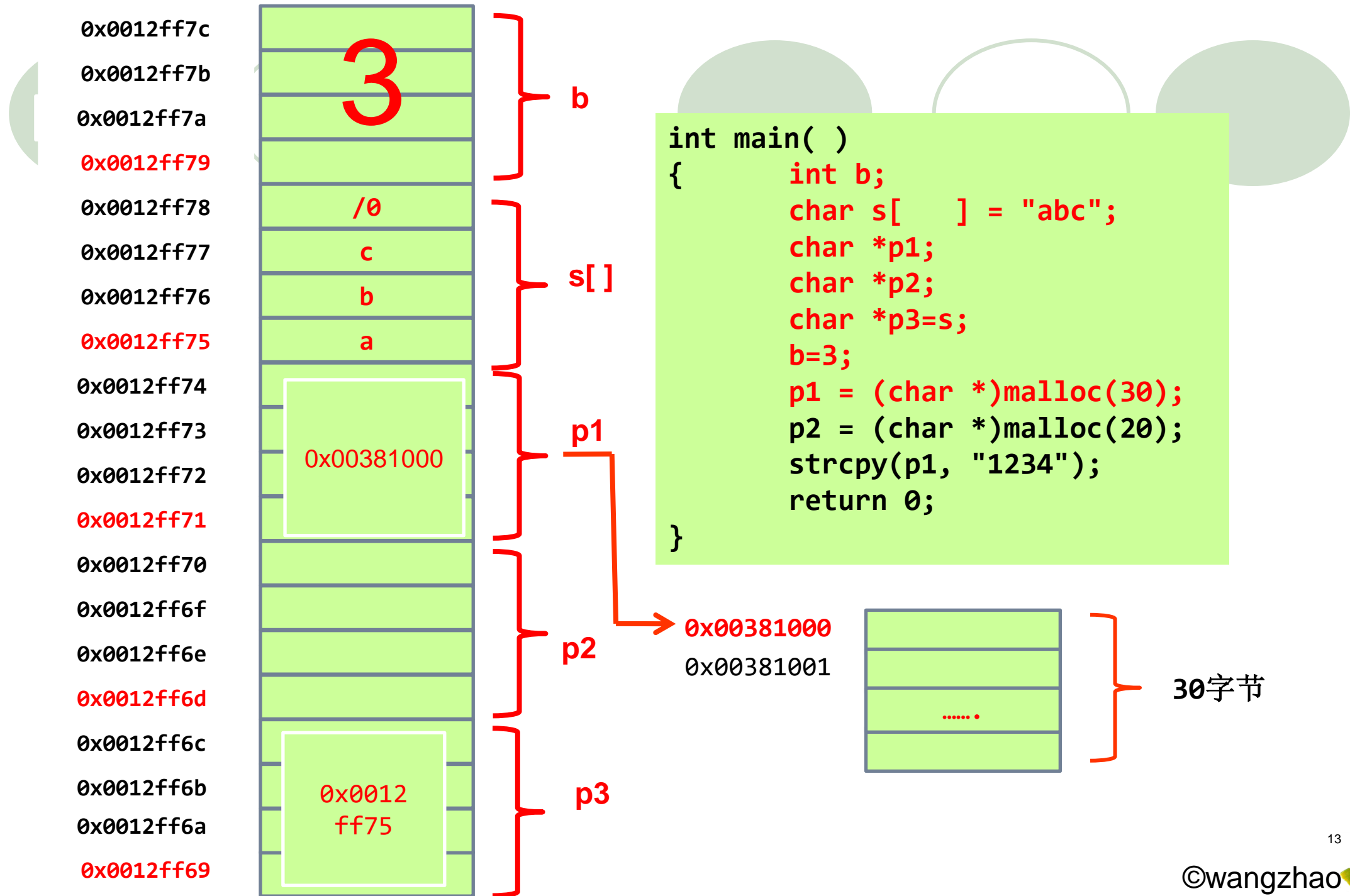


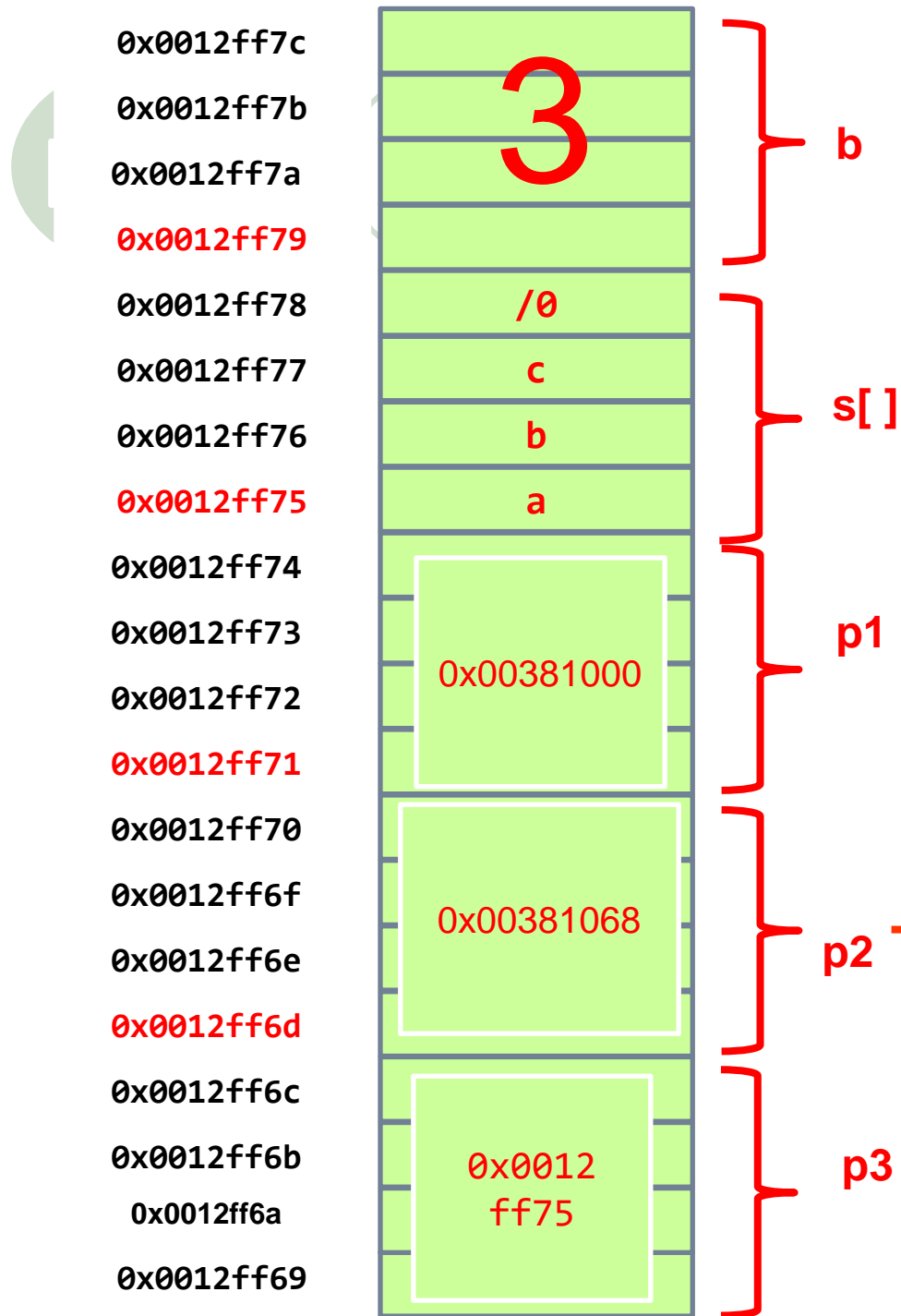


```
int main( )
{
    int b;
    char s[  ] = "abc";
    char *p1;
    char *p2;
    char *p3=s;
    b=3;
    p1 = (char *)malloc(30);
    p2 = (char *)malloc(20);
    strcpy(p1, "1234");
    return 0;
}
```



```
int main( )
{
    int b;
    char s[    ] = "abc";
    char *p1;
    char *p2;
    char *p3=s;
    b=3;
    p1 = (char *)malloc(30);
    p2 = (char *)malloc(20);
    strcpy(p1, "1234");
    return 0;
}
```

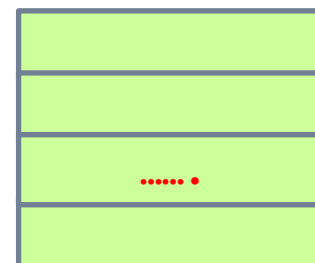




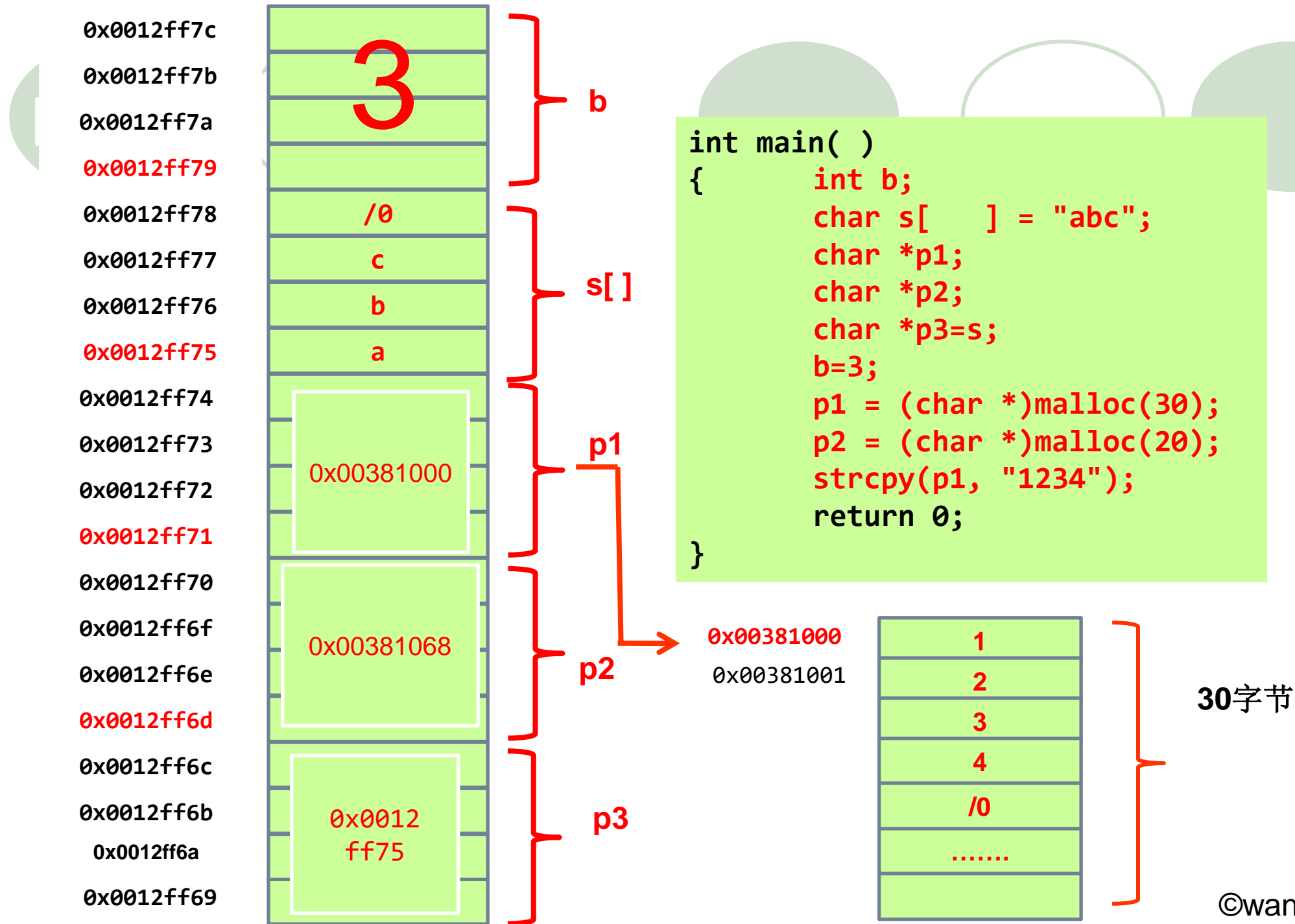
```
int main( )
{
    int b;
    char s[    ] = "abc";
    char *p1;
    char *p2;
    char *p3=s;
    b=3;
    p1 = (char *)malloc(30);
    p2 = (char *)malloc(20);
    strcpy(p1, "1234");
    return 0;
}
```



0x00381068
0x00381069



20字节



动态分配内存的函数

- **malloc** 函数

- 其函数原型为 **void * malloc(unsigned int size);**
- 其作用是在内存的动态存储区域中分配一个长度为**size**的连续空间.如果未成功, 返回空指针**NULL**.

- 函数

- 其函数原型为 **void free(void *p);**
- 其作用是释放由**p**指向的内存区.

ANSI 新标准增加了**void**指针类型 (**void ***) , 即可以定义一个指针变量, 但不指定它是指向哪一种数据类型。



指针变量可以有空值

- 即该指针变量不指向任何变量
- `char * p;`
- `p=NULL;`
 - `# define NULL 0`
 - `p=NULL;`
 - `/* 表示p不指向任何有用单元。*/`
 - 任何指针都可以与NULL作比较
 - `if(p==NULL)`

```
char *sa;  
sa=(char *)malloc(sizeof(char));  
if(sa==NULL)  
{  
    printf("No space for sa");  
    return 0;  
} /*检验内存是否分配成功*/
```



动态分配

VS

静态分配

```
大学的一天{  
    数算B-2班 {  
        教室1=malloc(148人教室);  
        在教室1 (理教201) 上课;  
        free(教室1)  
    }  
  
    化学概论{  
        教室2=malloc(148人教室)  
        在教室2 (理教201) 上课;  
        free(教室2)  
    }  
    ...  
    ...  
}
```

```
中学的一天{  
    高三 6班[60人] ; // 三教307  
    语文{  
        上课;  
    }  
  
    数学{  
        上课;  
    }  
  
    英语{  
        上课;  
    }  
    ...  
    ...  
}
```

