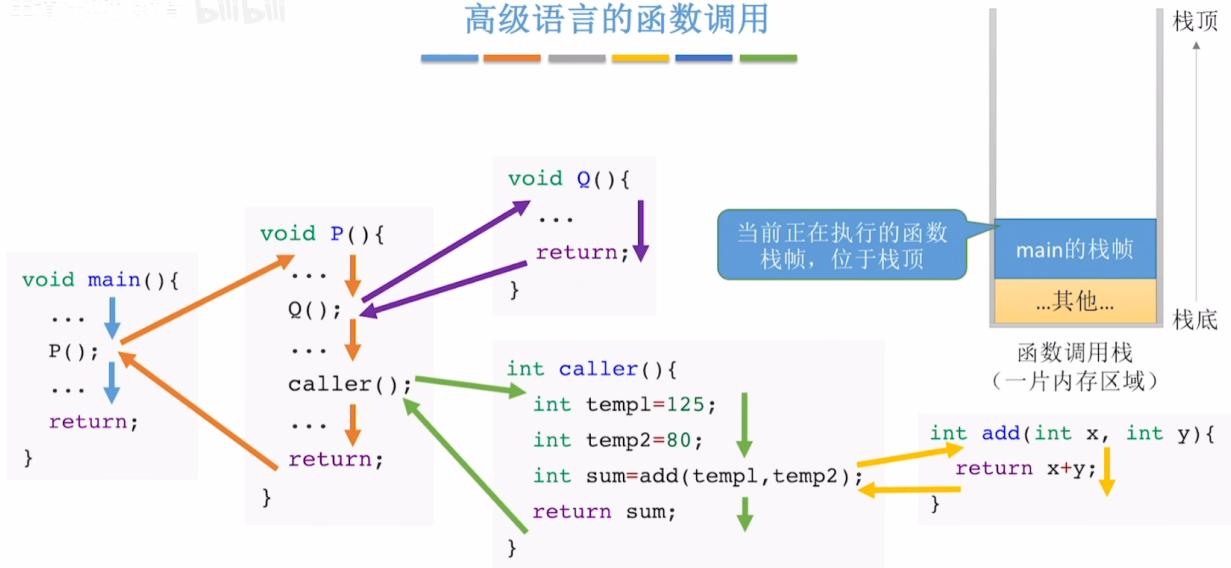


# 过程调用的机器级表示

## 高级语言的函数调用

来源：https://www.bilibili.com

### 高级语言的函数调用



## x86 汇编语言的函数调用



函数调用指令: call <函数名>  
函数返回指令: ret

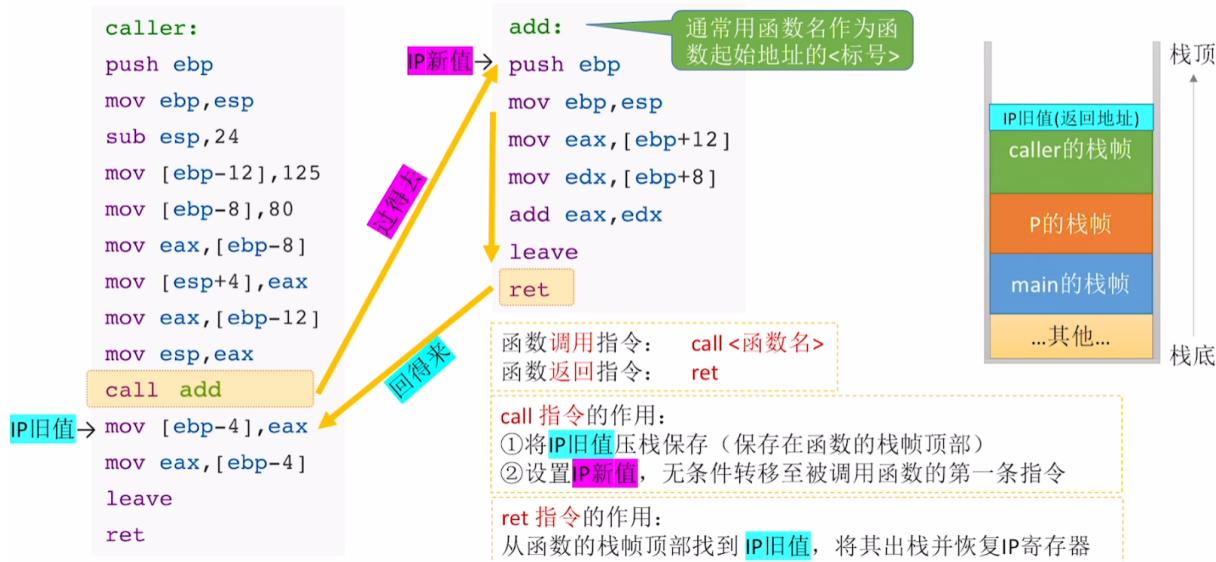
### x86汇编语言的函数调用



## call、ret 指令

## call、ret指令

注: x86处理器中  
程序计数器 PC (Program Counter)  
通常被称为 IP (Instruction Pointer)



## 小朋友，你是否有很多问号？

```

int caller(){
    int temp1=125;
    int temp2=80;
    int sum=add(temp1,temp2);
    return sum;
}
  
```

int add(int x, int y){
 return x+y;
}



如何传递调用参数、返回值？

为什么倒过来了？

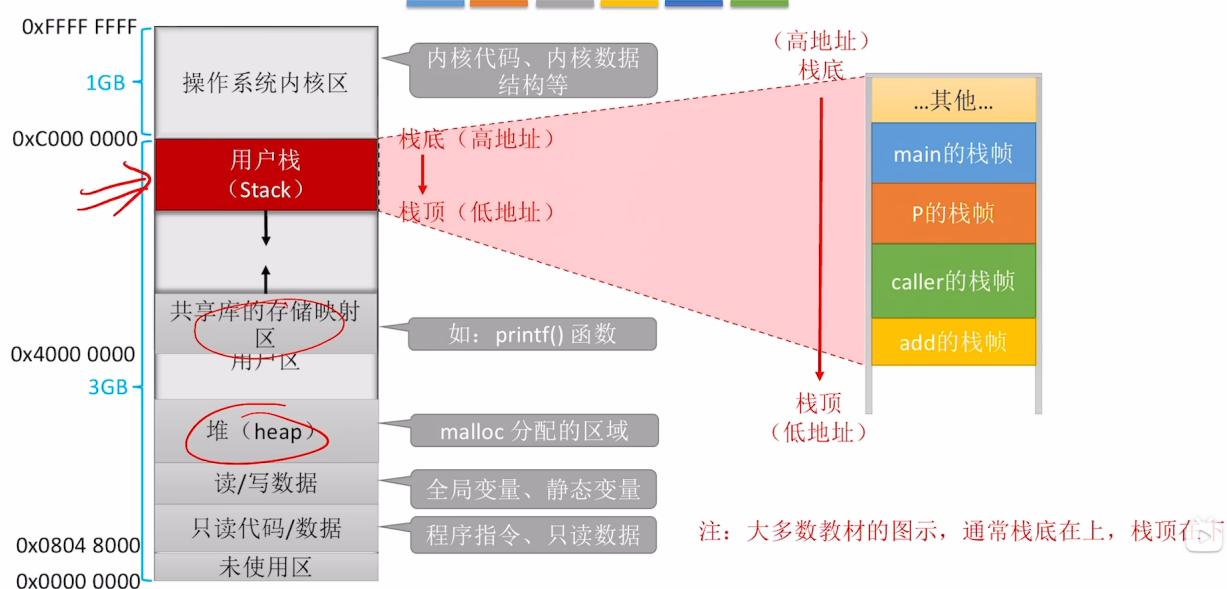
如何访问栈帧里的数据？

栈帧内可能包含哪些内容？

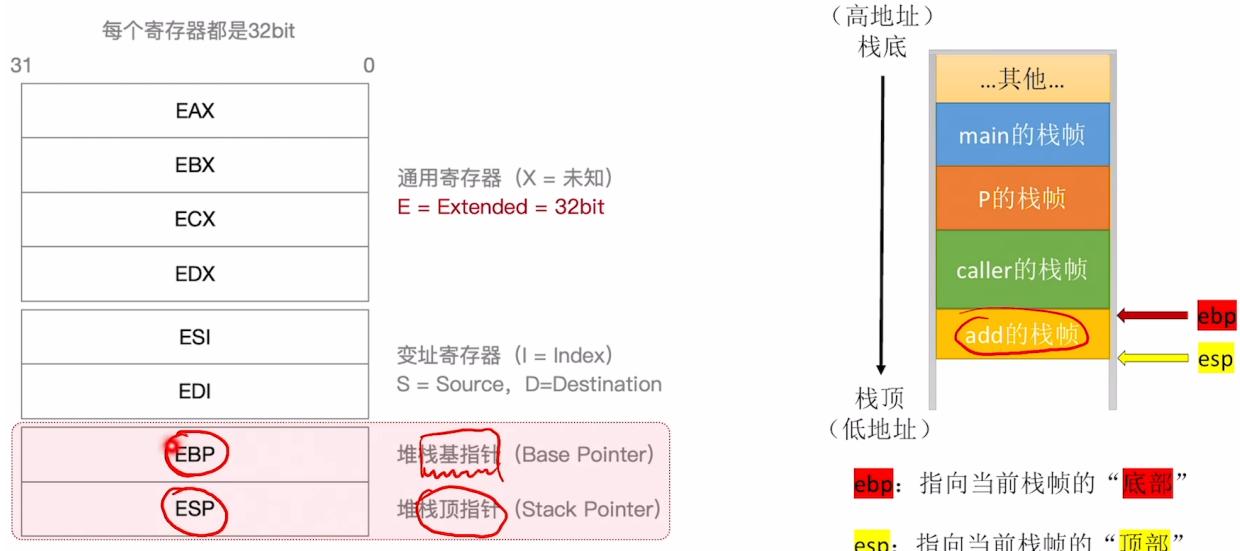


## 函数调用栈在内存中的位置

## 函数调用栈在内存中的位置



## 标记栈帧范围：EBP、ESP寄存器

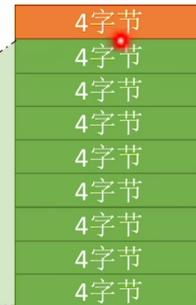


## 标记栈帧范围：EBP、ESP寄存器

(高地址)  
栈底



栈顶  
(低地址)



对栈帧内数据的访问，都是基于 ebp、esp 进行的



标记一处地点



知道这是谁的地盘么

内存地址

**ebp**: 指向当前栈帧的“底部”

注: x86 系统中，默认以4字节为栈的操作单位

**esp**: 指向当前栈帧的“顶部”

## 访问栈帧数据：push、pop 指令

### 访问栈帧数据：push、pop 指令

eax寄存器: 666

**push、pop** 指令实现入栈、出栈操作，x86 默认以4字节为单位。指令格式如下：

Push 🐾 // 先让 esp 减 4，再将 🐾 压入  
Pop 🐾 // 栈顶元素出栈写入 🐾，再让 esp 加 4

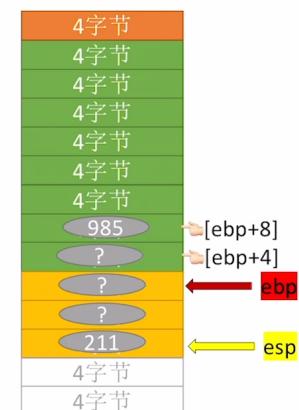
注1: 🐾 可以是立即数、寄存器、主存地址  
注2: 🐾 可以是寄存器、主存地址

例：

push eax	# 将寄存器 eax 的值压栈
push 985	# 将立即数 985 压栈
push [ebp+8]	# 将主存地址 [ebp+8] 里的数据压栈
<b>pop eax</b>	# 栈顶元素出栈，写入寄存器 eax
<b>pop [ebp+8]</b>	# 栈顶元素出栈，写入主存地址 [ebp+8]

(高地址)  
栈底

栈顶  
(低地址)



**ebp**: 指向当前栈帧的“底部”

**esp**: 指向当前栈帧的“顶部”

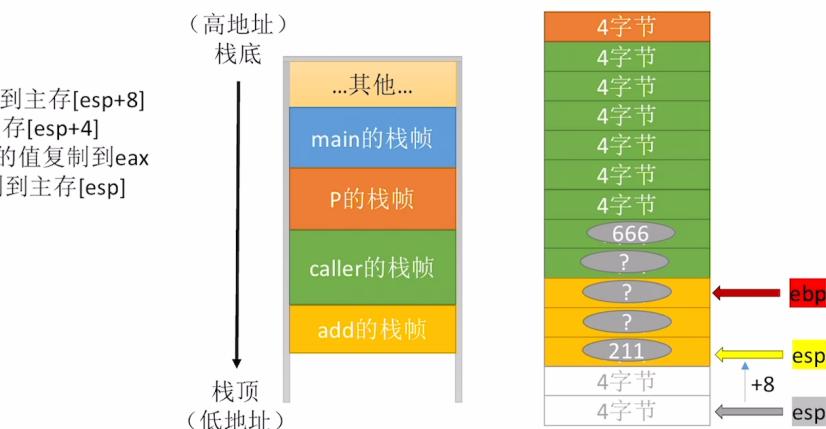
## 访问栈帧数据：mov 指令

例:

```

sub esp, 12      #栈顶指针-12
mov [esp+8], eax #将eax的值复制到主存[esp+8]
mov [esp+4], 958 #将985复制到主存[esp+4]
mov eax, [ebp+8] #将主存[ebp+8]的值复制到eax
mov [esp],eax    #将eax的值复制到主存[esp]
add esp, 8       #栈顶指针+8

```



- 可以用 **mov 指令**，结合 **esp**、**ebp** 指针访问栈帧数据
- 可以用减法/加法指令，即 **sub/add** 修改栈顶指针 **esp** 的值

**ebp**: 指向当前栈帧的“底部”

**esp**: 指向当前栈帧的“顶部”

## 总结: 如何访问栈帧?

### 总结: 如何访问栈帧?

#### 方法一:

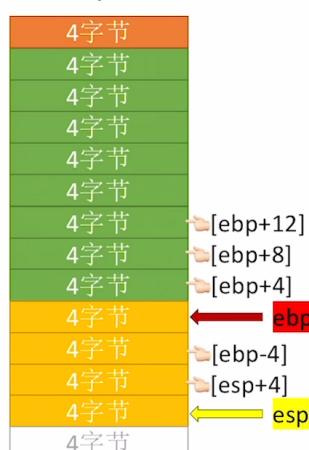
```

Push 🐾 // 先让esp减4, 再将 🐾压入
Pop 🐾  // 栈顶元素出栈写入 🐾, 再让 esp加4

```

注1: 🐾 可以是立即数、寄存器、主存地址

注2: 🐾 可以是寄存器、主存地址



#### 方法二:

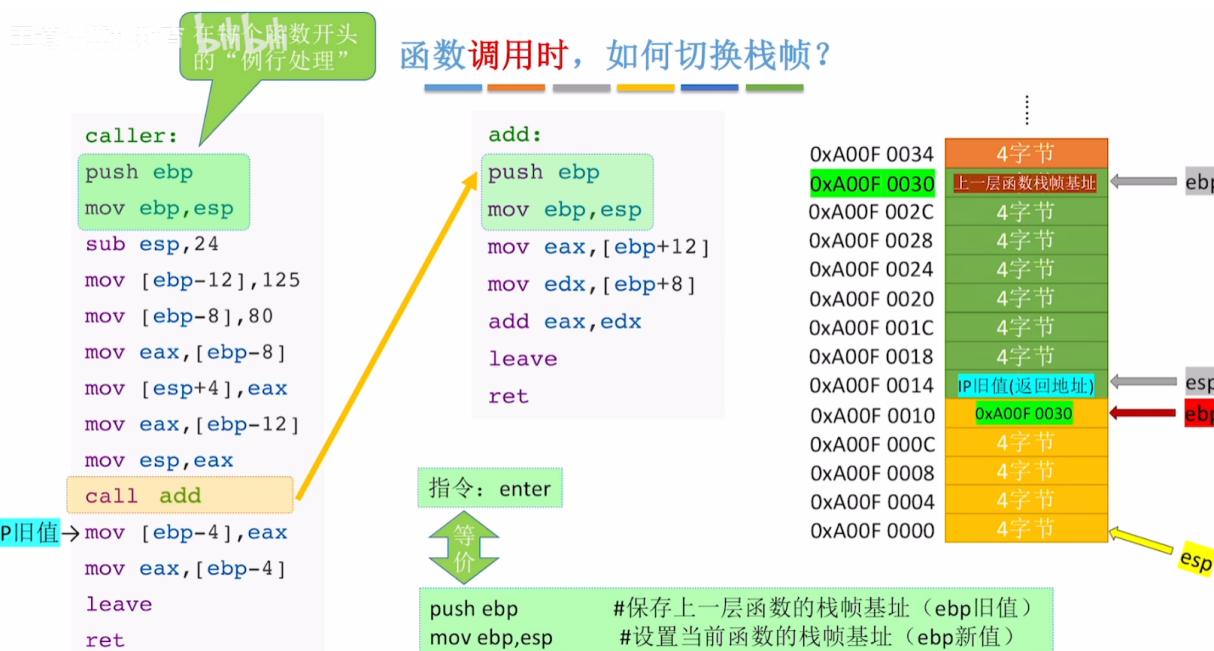
```
mov 指令, 结合 esp、ebp 指针访问栈帧数据
```

注: 可以用减法/加法指令，即 **sub/add** 修改栈顶指针 **esp** 的值

**ebp**: 指向当前栈帧的“底部”

**esp**: 指向当前栈帧的“顶部”

## 函数调用时，如何切换栈帧?

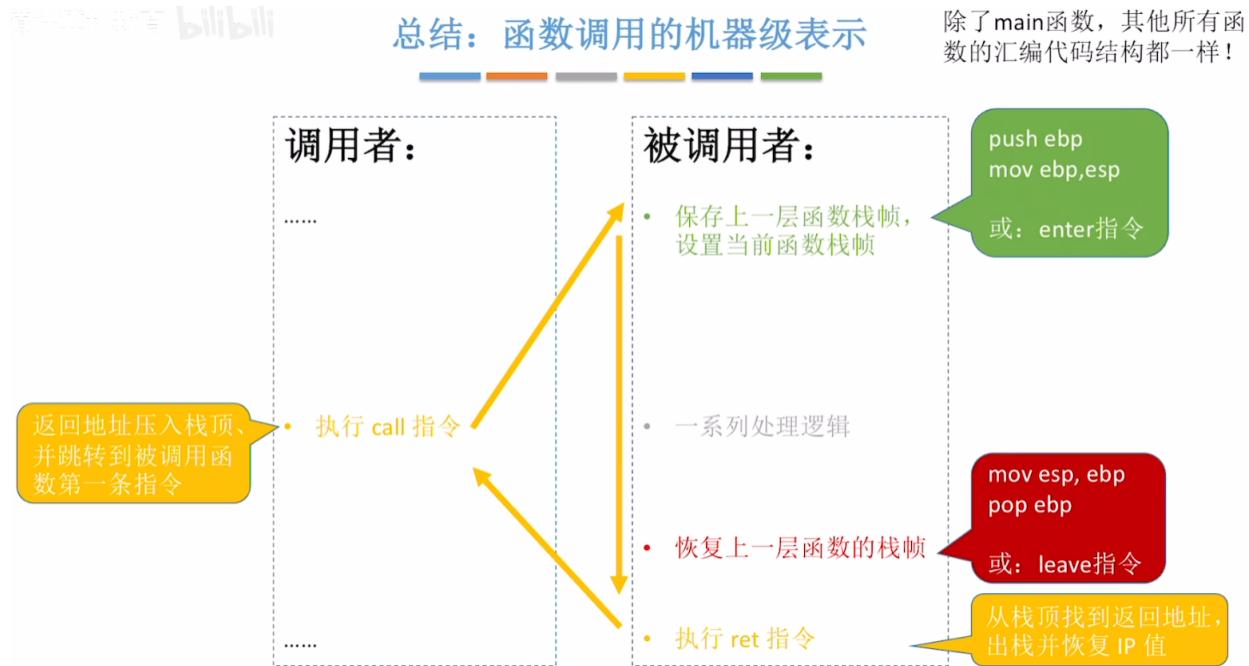


## 函数返回时，如何切换栈帧？

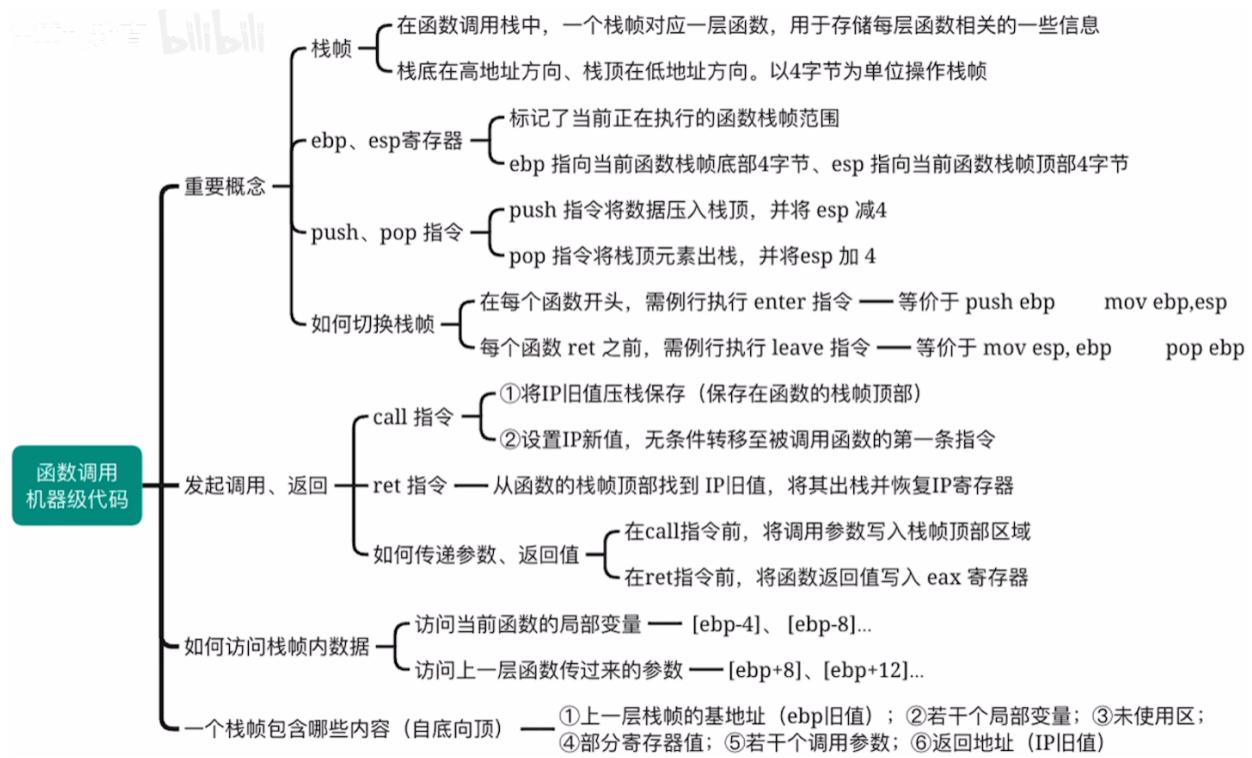




## 总结：函数调用的机器级表示



## 一个栈帧内可能包含哪些内容？



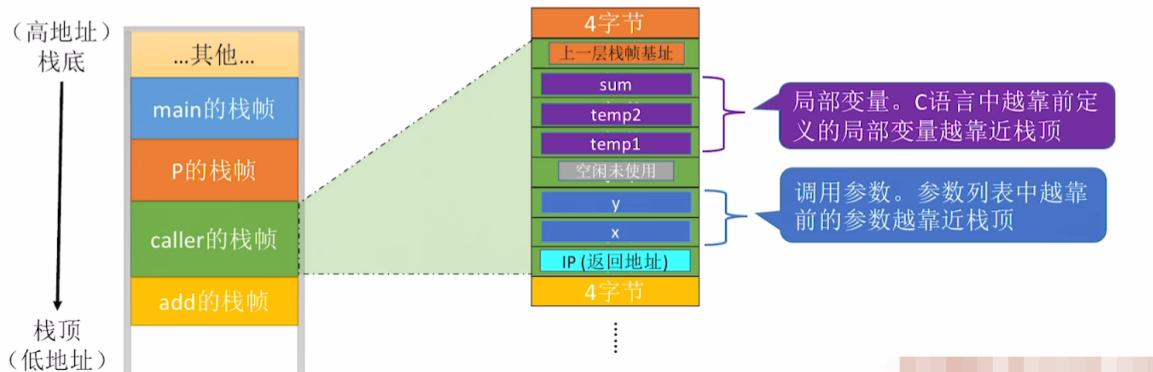
拿好教育  
和你一起  
学习  
开源、自由、非常流行

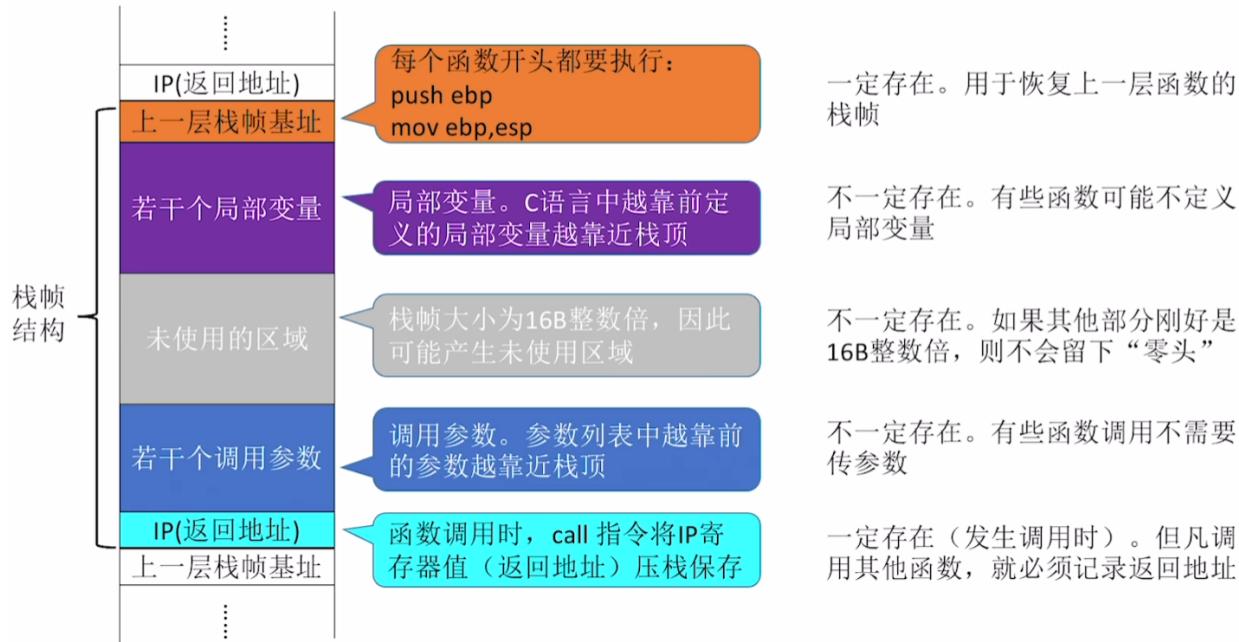
## 一个栈帧内可能包含哪些内容？

- gcc 编译器将每个栈帧大小设置为 16B 的整数倍（当前函数的栈帧除外），因此栈帧内可能出现空闲未使用的区域。
- 通常将局部变量集中存储在栈帧底部区域
- 通常将调用参数集中存储在栈帧顶部区域
- 栈帧最底部一定是上一层栈帧基址（ebp旧值）
- 栈帧最顶部一定是返回地址（当前函数的栈帧除外）

```
int caller(){
    int temp1=125;
    int temp2=80;
    int sum=add(temp1,temp2);
    return sum;
}

int add(int x, int y){
    return x+y;
}
```





## 汇编代码实战



## 总结：函数调用的机器级表示

