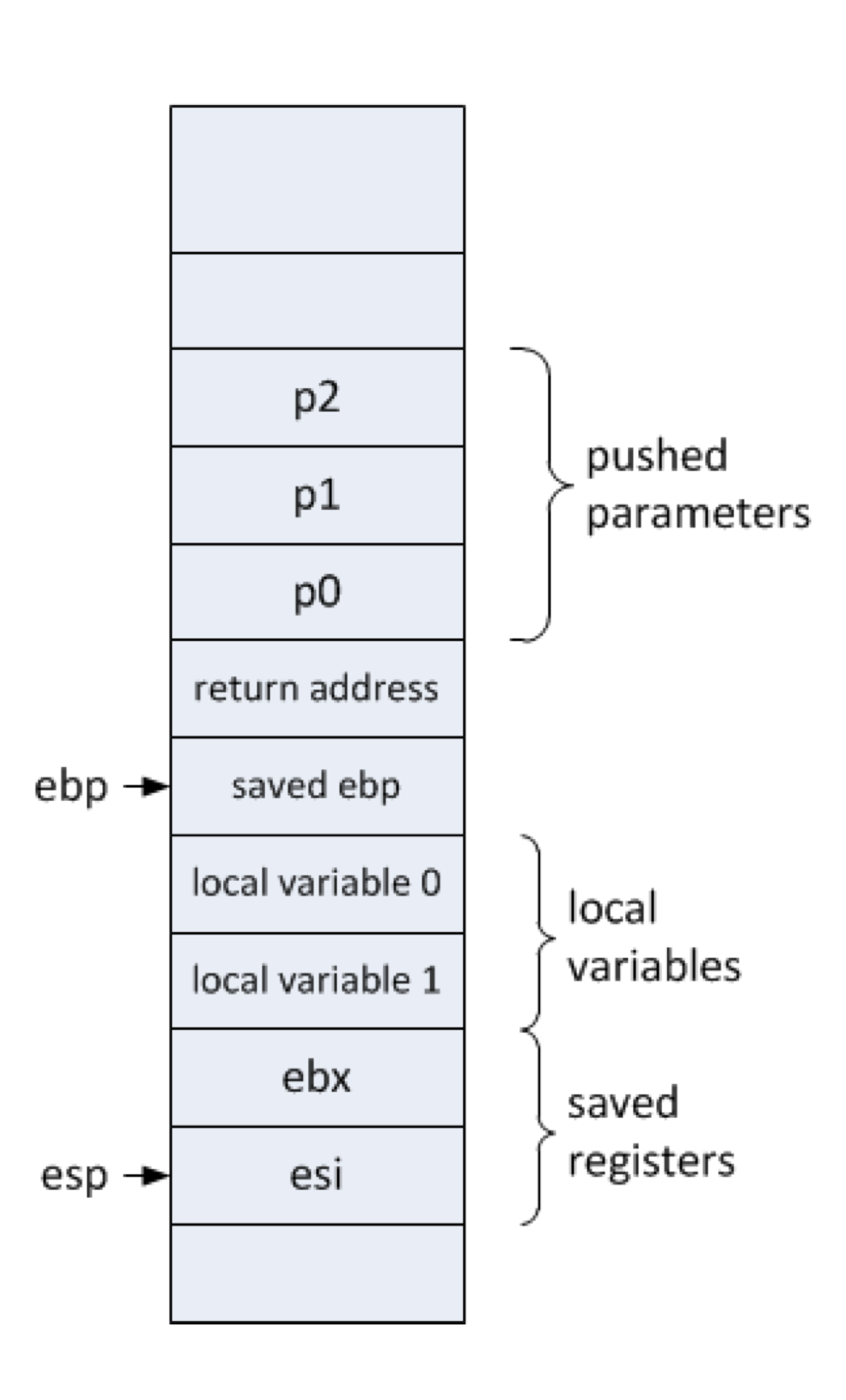
//Have no clue if these are right, pls correct as needed :)

# Q.1

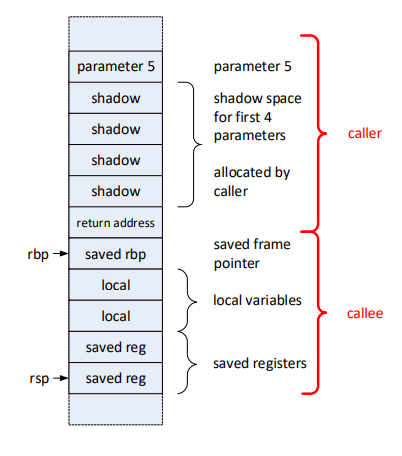
### IA32

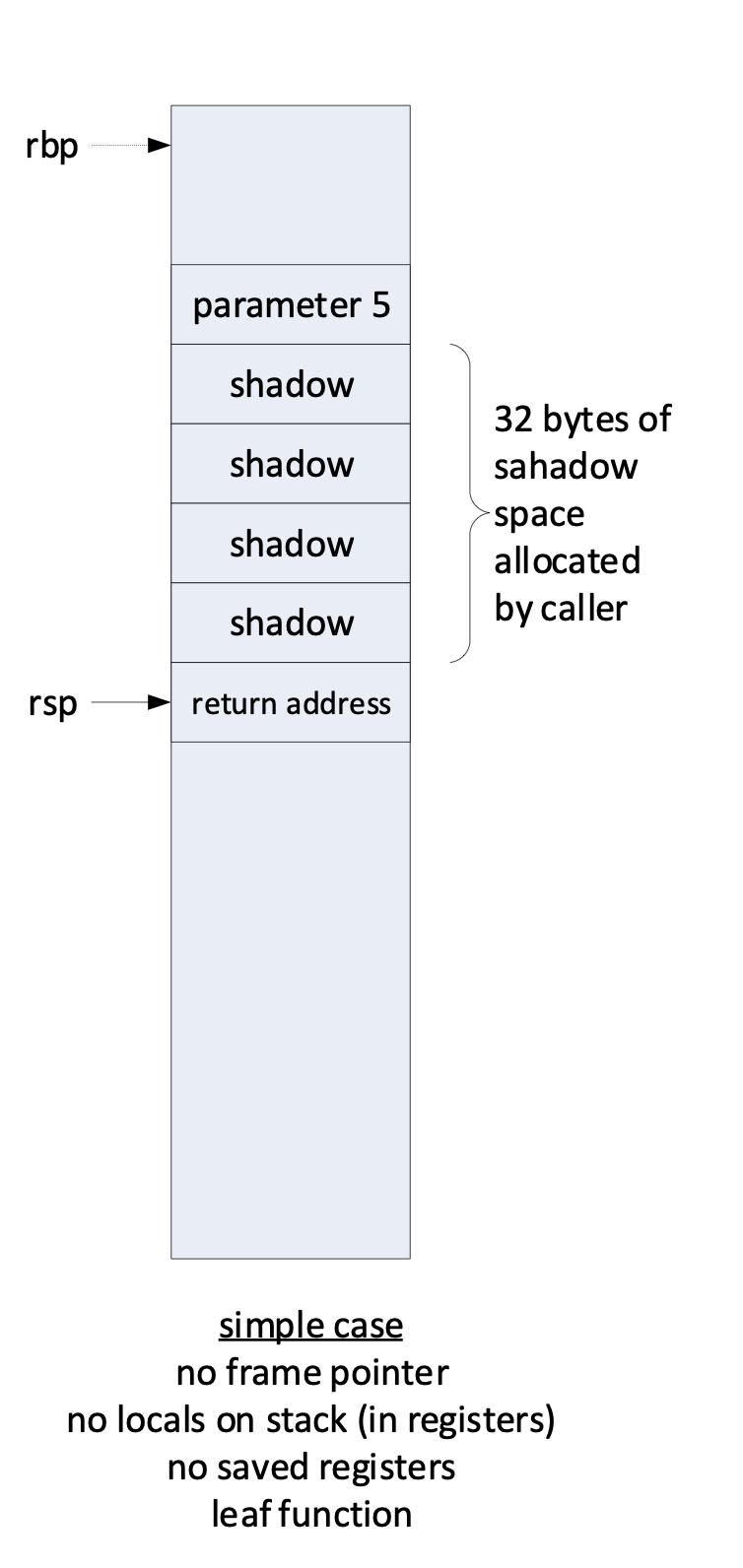
* Has fewer registers than a typical RISC processor
* Register eax is the accumulator. Ebp and esp are the frame pointer and stack pointer respectively
* Registers are all 32 bits. Possible to access the first 8 bits using register name al, the 8th to 15th buts usingA ah and the first 16 bits using ax
* Function result returned in register eax.
* Eax, ecx and edx are volatile, they are not preserved across function calls
* Parameters to a function are pushed on the stack from right to left
* Parameters and local variables are accessed relative to ebp, the frame pointer
* Parameters are accessed at positive offsets from ebp
* Local variables are accessed at negative offsets



### x64

* 8 additional registers to IA32, r8 to r15
* When calling a function, the first four parameters are passed in rcx, rdx, r8 and r9
* Additional parameters are passed on the stack from right to left
* Rax, rcx, rdx, r8, r9, r10 and r11 are volatile
* So many registers means that local variables can be stored in registers
* No need for a frame pointer and no need to save/restore registers
* It is essential to allocate 32 bytes of shadow space on the stack, before a function is called. It can be used by the compiler to leave a copy of the register values (first four parameters passed) on the stack for later inspection in the debugger.
* The shadow space must be deallocated afterwards





### Main advantage of x64 procedure calling convention compared with the IA32 procedure calling convention

* Parameters passed through registers
* x64 uses a simpler stack frame and more registers are available
* Not necessary to allocate individual variables because **shadow** **space** is allocated instead.
* Larger Number of GPRs means regs can be used for local vars, no need to use frame pointers to access them or no need to save / restore vars at the end.

### 

### Convert code to IA32

max: push ebp ; save ebp

mov ebp, esp ; ebp -> new stack frame

sub esp, 4 ; space for local var v

mov [esp-4], [esp+8] ; int v = a

cmp [esp+12], [esp-4] ; if (b > v)

jl max1 ; {

mov [esp-4], [esp+12] ; v =b

max1: cmp [esp+16], [esp-4] ; if (c > v)

jl endif ; {

mov [esp-4], [esp+16] ; v =c

endif: mov eax, [esp-4] ; eax = v

mov esp, ebp ;

pop ebp ;

ret 0 ;

p: push ebp ;

mov ebp, esp ;

push ebx ;

push [ebp+12] ; j

push [ebp+8] ; i

push g ; g

call max ;

push [ebp+20] ; l

push [ebp+16] ; k

push eax ; max(g, i , j)

call max ;

pop ebx ;

mov esp, ebp ;

pop ebp ;

ret 0 ;

### Convert code to x64

.data

public g

g QWORD 4

.code

max: mov r10, rdi ; v = a

cmp rsi, r10 ; if (b > v)

jl max1 ; {

mov r10, rsi ; v = b

max1: cmp rdx, r10 ; if (c > v)

jl max2 ; {

mov r10, rdx ; v = c

max2: mov rax, r10 ;

ret ;

public p:

sub rsp, 32

mov r12, rcx

mov r13, rdx

mov r14, r8

mov r15, r9

mov rcx, g

mov rdx, r12

mov r8, r13

call max

mov rcx, rax

mov rdx, r14

mov r8, r15

call min

add rsp, 32

ret 0