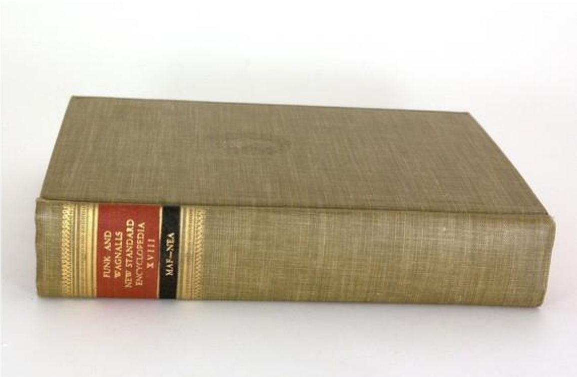
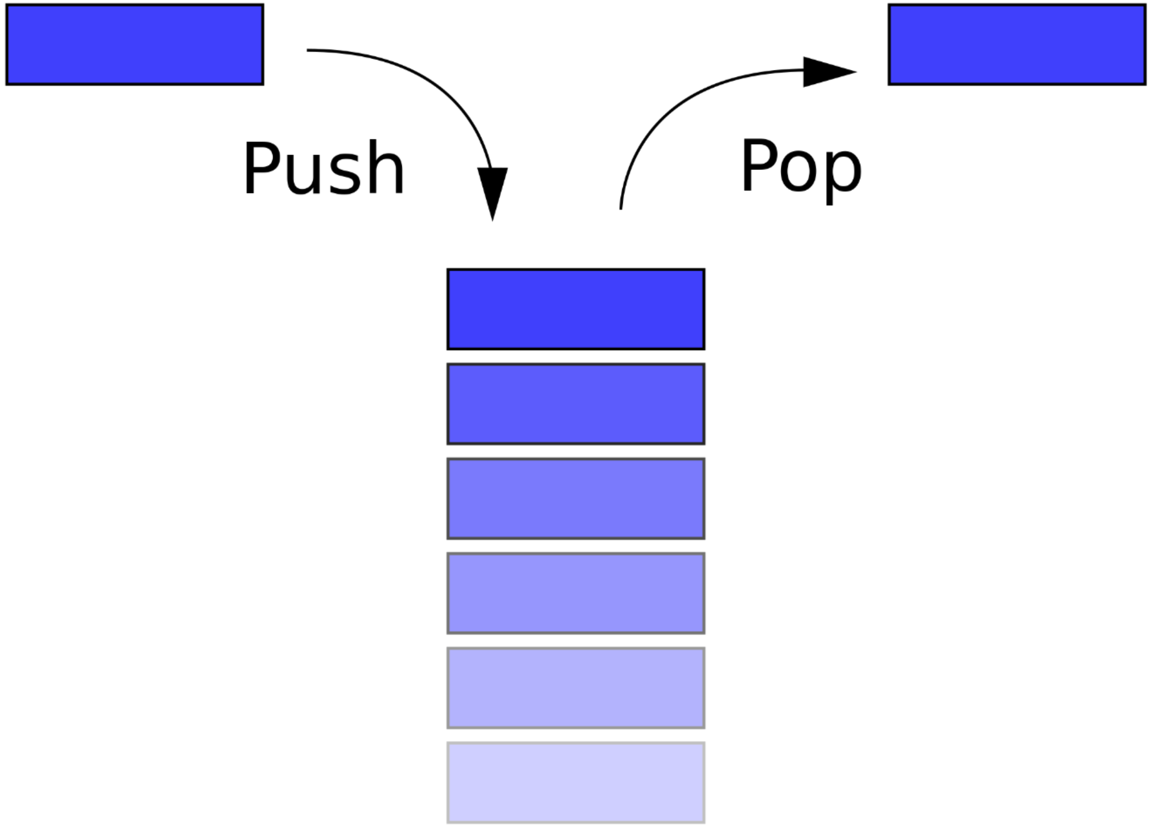
X86 Stack

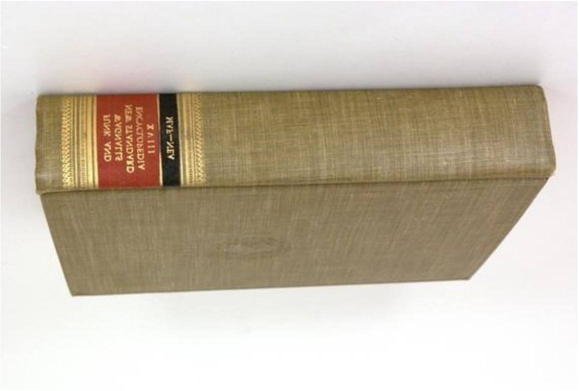
Computer Systems Section 3.7

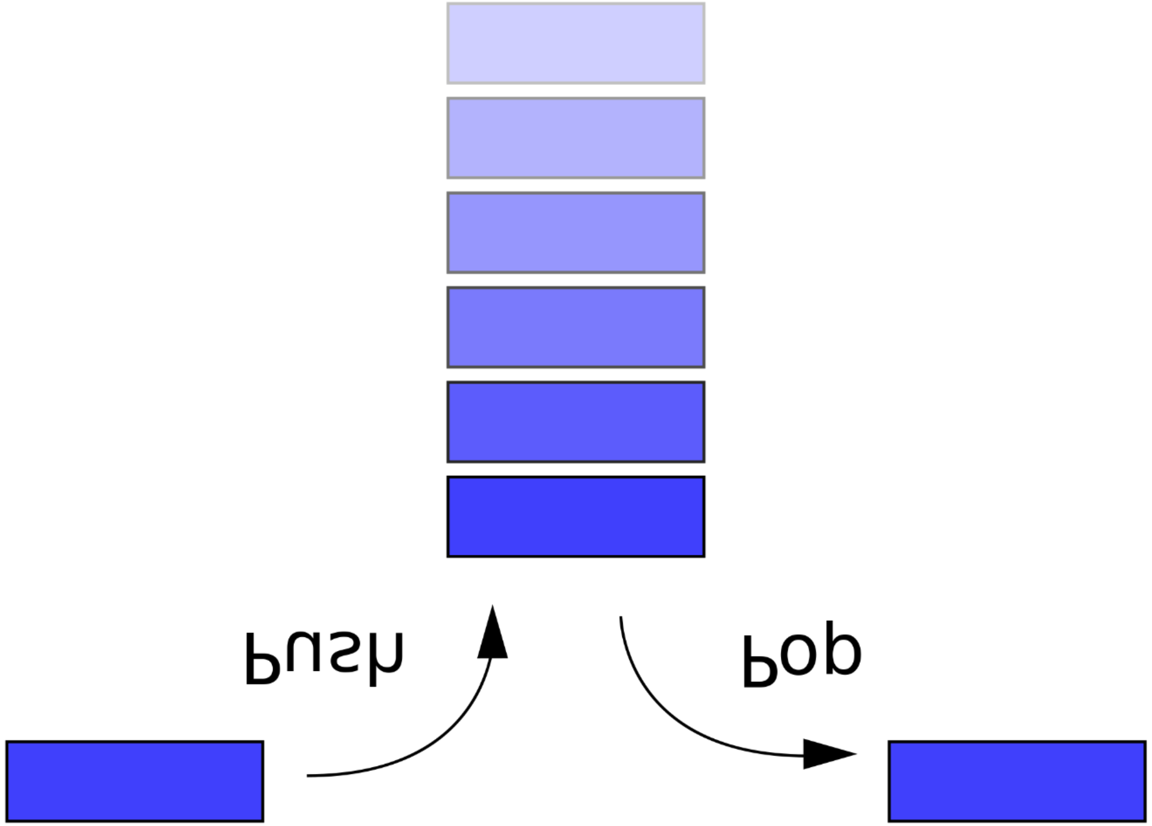
The Stack (as we learned in CS-120)



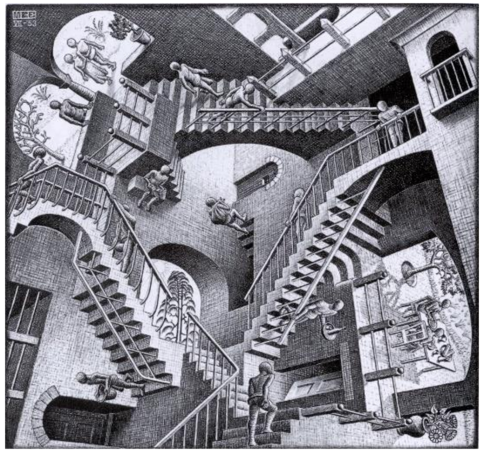


The x86 stack





Terminology Warning!

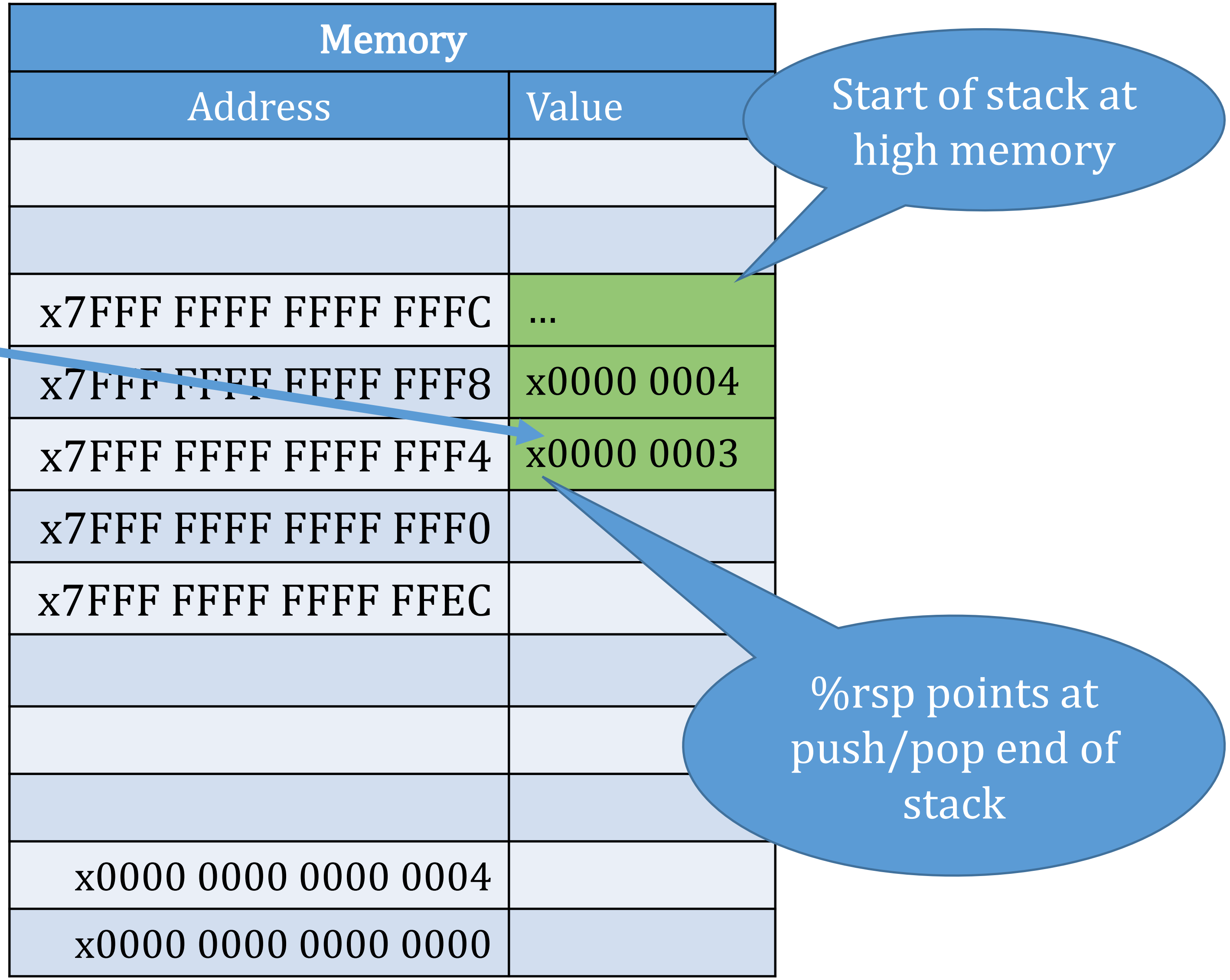


- The textbook uses the convention:
  push and pop occurs at the "top" of the stack
- In x86 the "top" of the stack is at the "bottom" of memory
- I prefer calling the "top" of the stack the top of memory
  - push and pop therefore occurs at the "bottom" of the stack

- To avoid confusion, I will try to say "high address" and "low address" rather than "top" and "bottom"

x86 Stack

Reg	Value
rsp	x7FFF FFFF FFFF FFF4
rax	x0000 0000 0000 000E



Memory	
Address	Value
x7FFF FFFF FFFF FFFC	…
x7FFF FFFF FFFF FFF8	x0000 0004
x7FFF FFFF FFFF FFF4	x0000 0003
x7FFF FFFF FFFF FFF0	
x7FFF FFFF FFFF FFEC	
x0000 0000 0000 0004	
x0000 0000 0000 0000	

Start of stack at high memory

%rsp points at push/pop end of stack

- Memory above %rsp is in use
- Memory below %rsp is available

x86 Stack

Reg	Value
rsp	x7FFF FFFF FFFF FFF4
rax	x???? ???? 0000 000E

push %eax

Memory	
Address	Value
xFFFF FFFC	
xFFFF FFF8	x0000 0004
xFFFF FFF4	x0000 0003
xFFFF FFF0	
xFFFF FFEC	
x0000 0004	
x0000 0000	

x86 Stack

Reg	Value
rsp	x7FFF FFFF FFFF FFF0
rax	x???? ???? 0000 000E

Memory	
Address	Value
xFFFF FFFC	
xFFFF FFF8	x0000 0004
xFFFF FFF4	x0000 0003
xFFFF FFF0	x0000 000E
xFFFF FFEC	
x0000 0004	
x0000 0000	

push %eax

```
subq  $4,%rsp
movl %eax,(%rsp)
```

x86 Stack

Reg	Value
rsp	x7FFF FFFF FFFF FFF4
rax	x???? ???? 0000 000E
rbx	x???? ???? 0000 000E

Memory	
Address	Value
xFFFF FFFC	
xFFFF FFF8	x0000 0004
xFFFF FFF4	x0000 0003
xFFFF FFF0	x0000 000E
xFFFF FFEC	
x0000 0004	
x0000 0000	

pop %ebx

```
movl (%rsp),%ebx
addq $4,%rsp
```

x86 Stack

Reg	Value
rsp	x7FFF FFFF FFFF FFEC
rax	x???? ???? 0000 000E

Memory	
Address	Value
xFFFF FFFC	
xFFFF FFF8	x0000 0004
xFFFF FFF4	x0000 0003
xFFFF FFF0	x0000 0000
xFFFF FFEC	x0000 000E
x0000 0004	
x0000 0000	

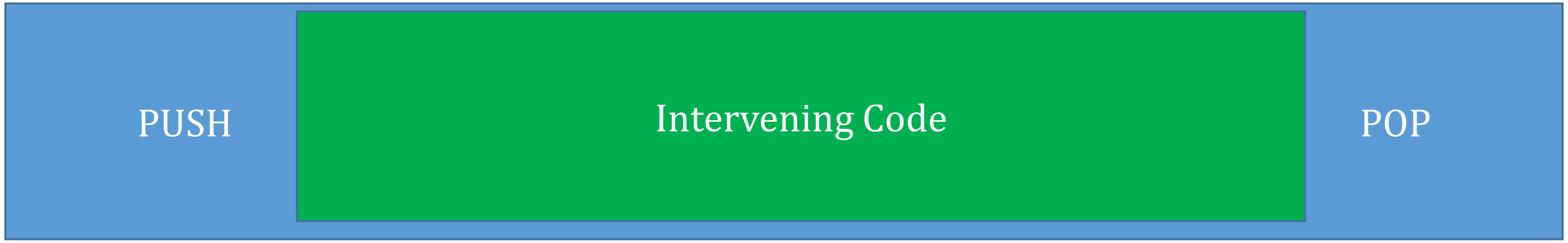
pushq %eax

subq $8,%rsp
movq %eax,(%rsp)

Stack Etiquette

- Rule 1: Push first
- Rule 2: Everything I push, I will also pop

- If intervening code follows etiquette, stack will work
- If I follow etiquette, I can be intervening code



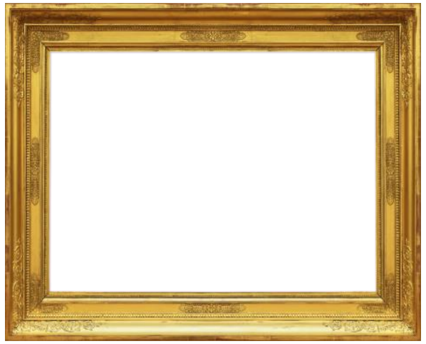
PUSH

Intervening Code

POP

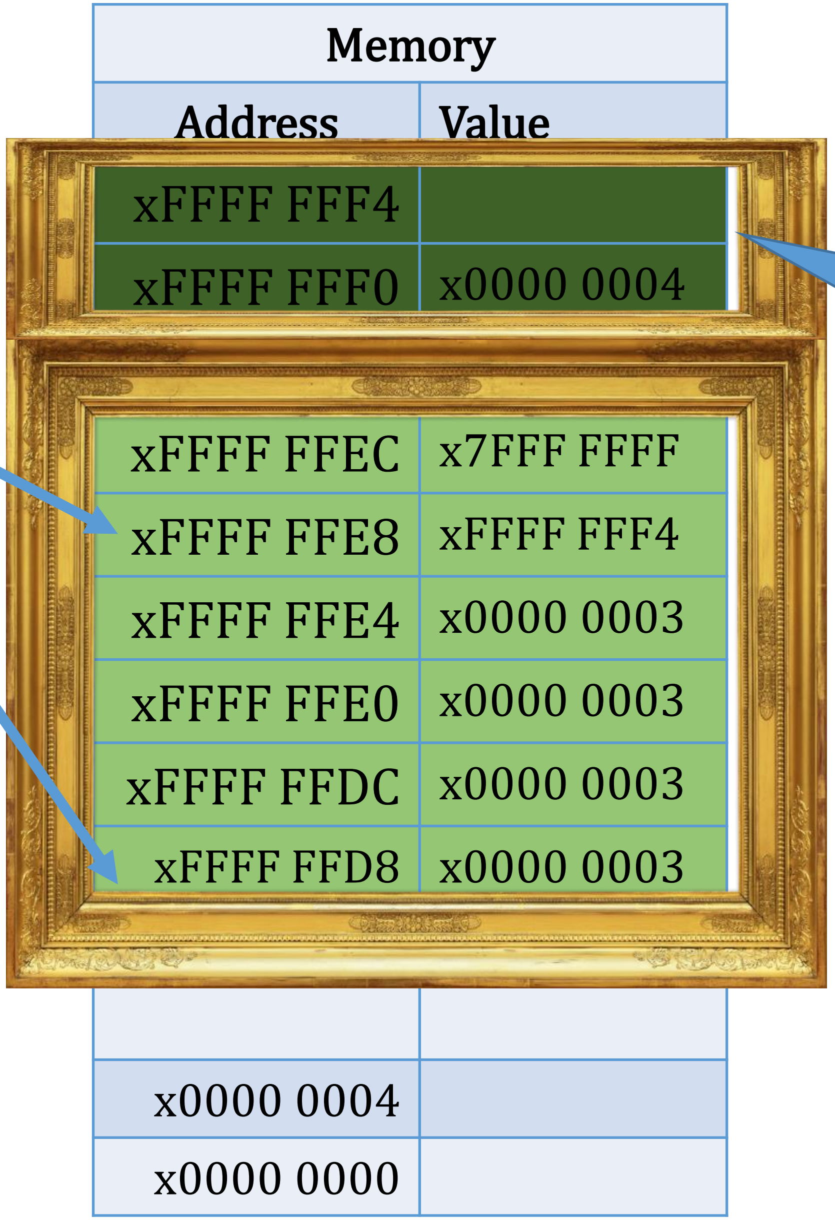
Use the stack for Function Invocation

- When a function is invoked, it’s preamble pushes invocation specific information on the stack
- When a function returns, the function specific information is popped off the stack, and the stack is restored to caller’s state
- The information associated with a function invocation is called an invocation record, or “stack frame”



Stack Frame

Reg	Value
rbp	x7FFF FFFF FFFF FFE8
rsp	x7FFF FFFF FFFF FFD8
rax	x???? ???? 0000 000E



Memory	
Address	Value
xFFFF FFF4	
xFFFF FFF0	x0000 0004
xFFFF FFEC	x7FFF FFFF
xFFFF FFE8	xFFFF FFF4
xFFFF FFE4	x0000 0003
xFFFF FFE0	x0000 0003
xFFFF FFDC	x0000 0003
xFFFF FFD8	x0000 0003
x0000 0004	
x0000 0000	

Previous Frame directly above current frame

Current Frame: Eight byte words between addresses in %rbp and %rsp

Example Call Stack

```
1. int addem(int x, int y);
2. int main() {
3.      int a=addem(3,4);
4.      a=addem(a,4);
5.      return 0;
6. }
7. int addem(int x, int y) { return x+y;}
```

Inv	Fn	args	vars	Ret
OS	main		a=	

What’s In a Stack Frame?

- Information to restore caller’s stack frame
- Space for Local Variable Values
- Space for saved state
- Space for parameter copies
- Return address (when calling functions)

How Big is a Stack Frame?

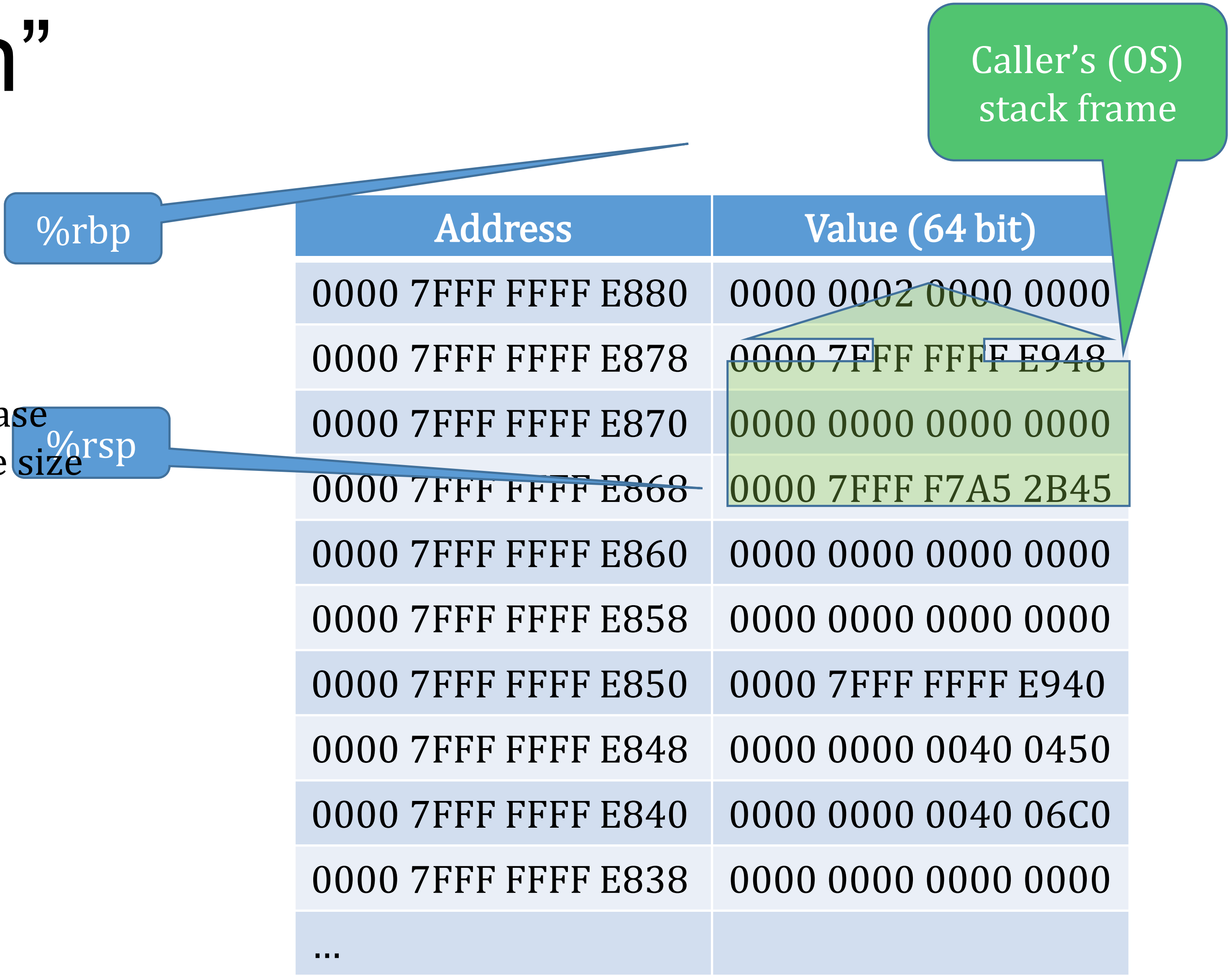
Info	Size
Caller's frame info	8 bytes
Local Variables	? (different for each function)
Copies of Parameter Values	? (different for each function)
Saved State	? (different for each function)
Return Address	8 bytes (if needed)
Total	**8+???**

When I am called…

- My caller’s stack frame is still active
- I need to save information about my callers frame
- I need to create my own stack frame

At entry to “main”

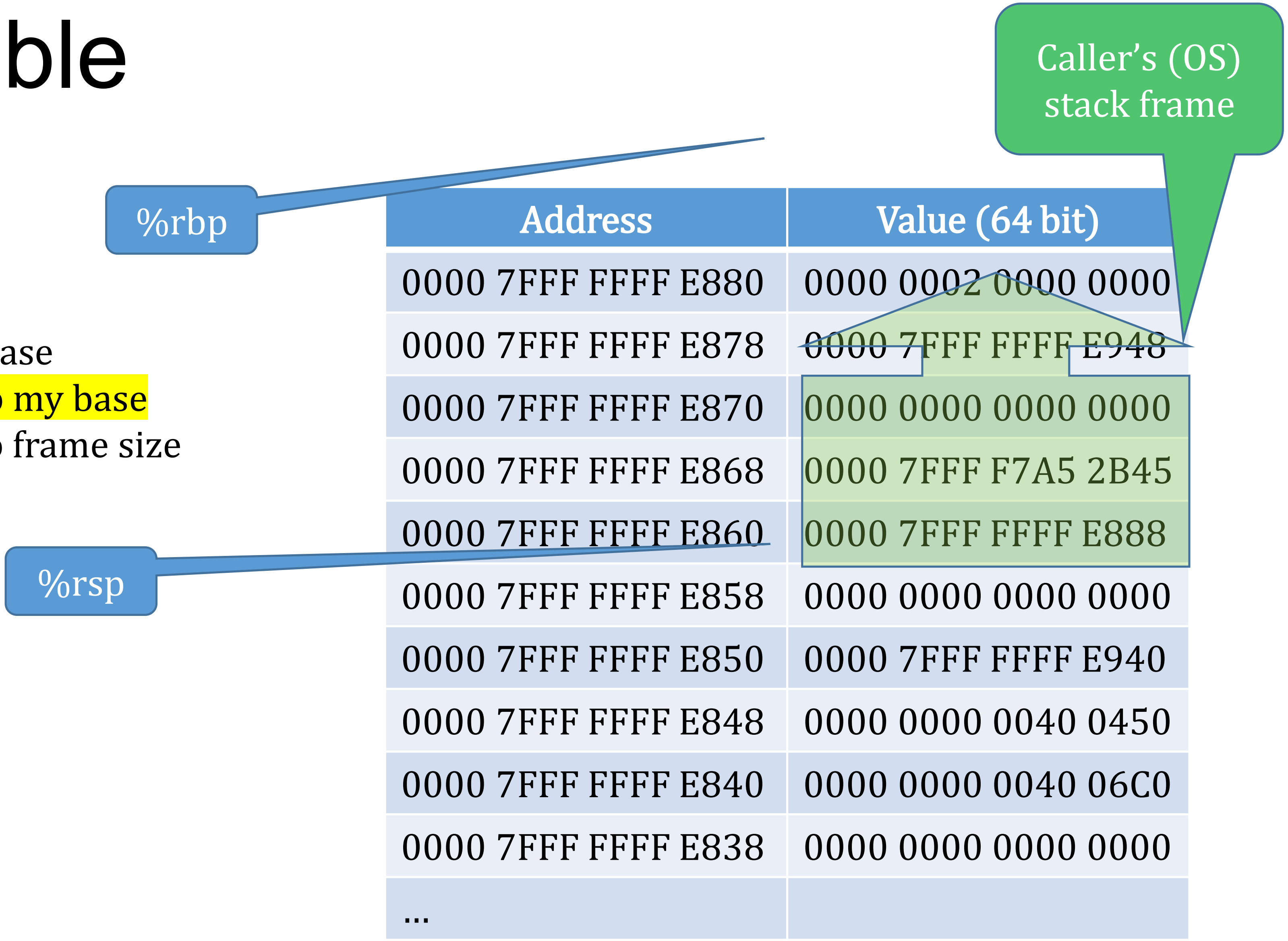
```
pushq     %rbp           ; Save caller’s base
movq      %rsp, %rbp  ; Reset %rbp to my base
subq      $32, %rsp      ; Reset %rsp to frame size
```



Address	Value (64 bit)
0000 7FFF FFFF E880	0000 0002 0000 0000
0000 7FFF FFFF E878	0000 7FFF FFFF E948
0000 7FFF FFFF E870	0000 0000 0000 0000
0000 7FFF FFFF E868	0000 7FFF F7A5 2B45
0000 7FFF FFFF E860	0000 0000 0000 0000
0000 7FFF FFFF E858	0000 0000 0000 0000
0000 7FFF FFFF E850	0000 7FFF FFFF E940
0000 7FFF FFFF E848	0000 0000 0040 0450
0000 7FFF FFFF E840	0000 0000 0040 06C0
0000 7FFF FFFF E838	0000 0000 0000 0000
…	

Main’s Preamble

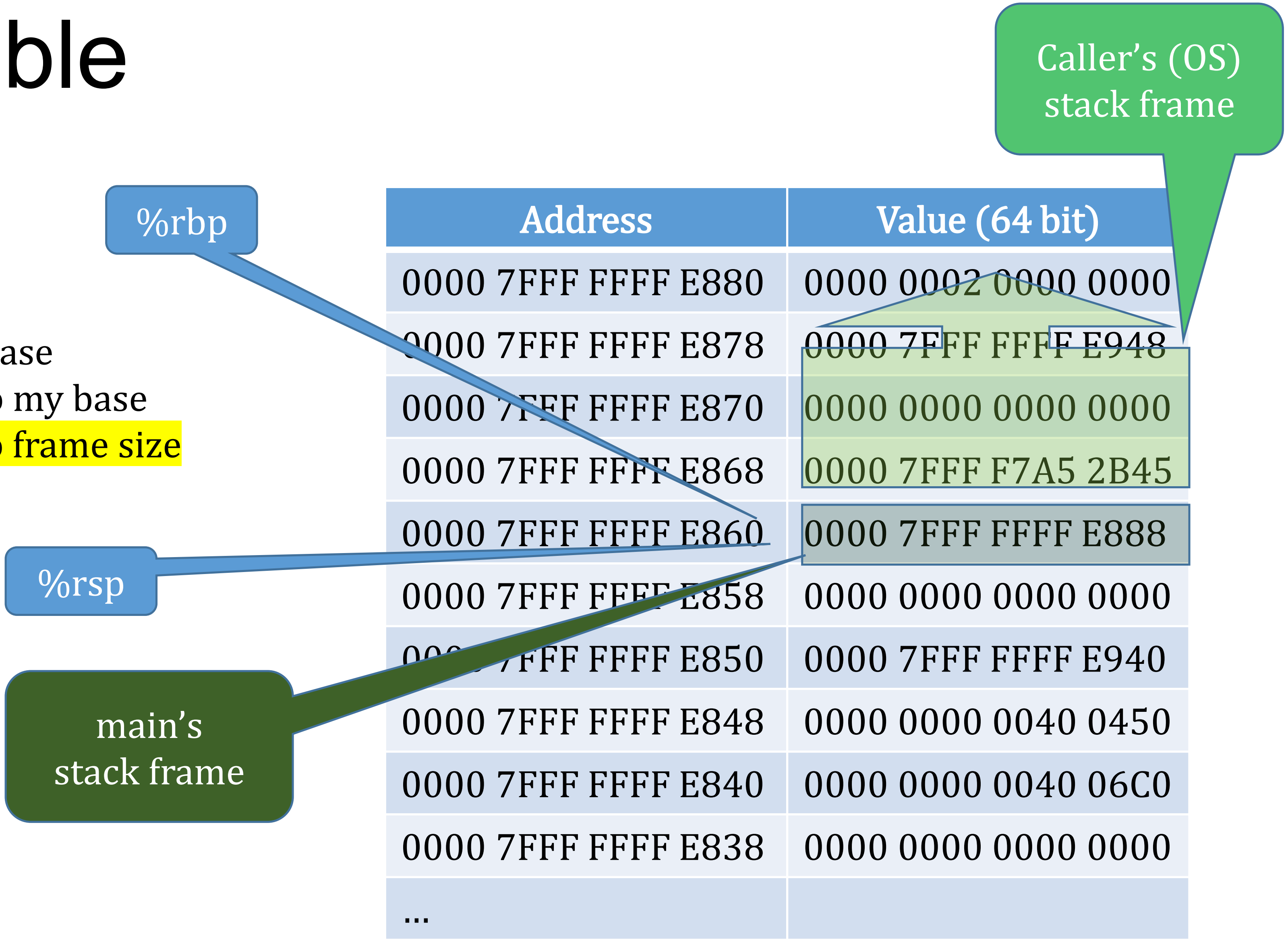
```
pushq     %rbp          ; Save caller’s base
movq      %rsp, %rbp    ; Reset %rbp to my base
subq      $32, %rsp     ; Reset %rsp to frame size
```



Address	Value (64 bit)
0000 7FFF FFFF E880	0000 0002 0000 0000
0000 7FFF FFFF E878	0000 7FFF FFFF E948
0000 7FFF FFFF E870	0000 0000 0000 0000
0000 7FFF FFFF E868	0000 7FFF F7A5 2B45
0000 7FFF FFFF E860	0000 7FFF FFFF E888
0000 7FFF FFFF E858	0000 0000 0000 0000
0000 7FFF FFFF E850	0000 7FFF FFFF E940
0000 7FFF FFFF E848	0000 0000 0040 0450
0000 7FFF FFFF E840	0000 0000 0040 06C0
0000 7FFF FFFF E838	0000 0000 0000 0000
…	

Main’s Preamble

```
pushq     %rbp           ; Save caller’s base
movq      %rsp, %rbp  ; Reset %rbp to my base
subq      $32, %rsp      ; Reset %rsp to frame size
```



Caller’s (OS) stack frame

%rbp

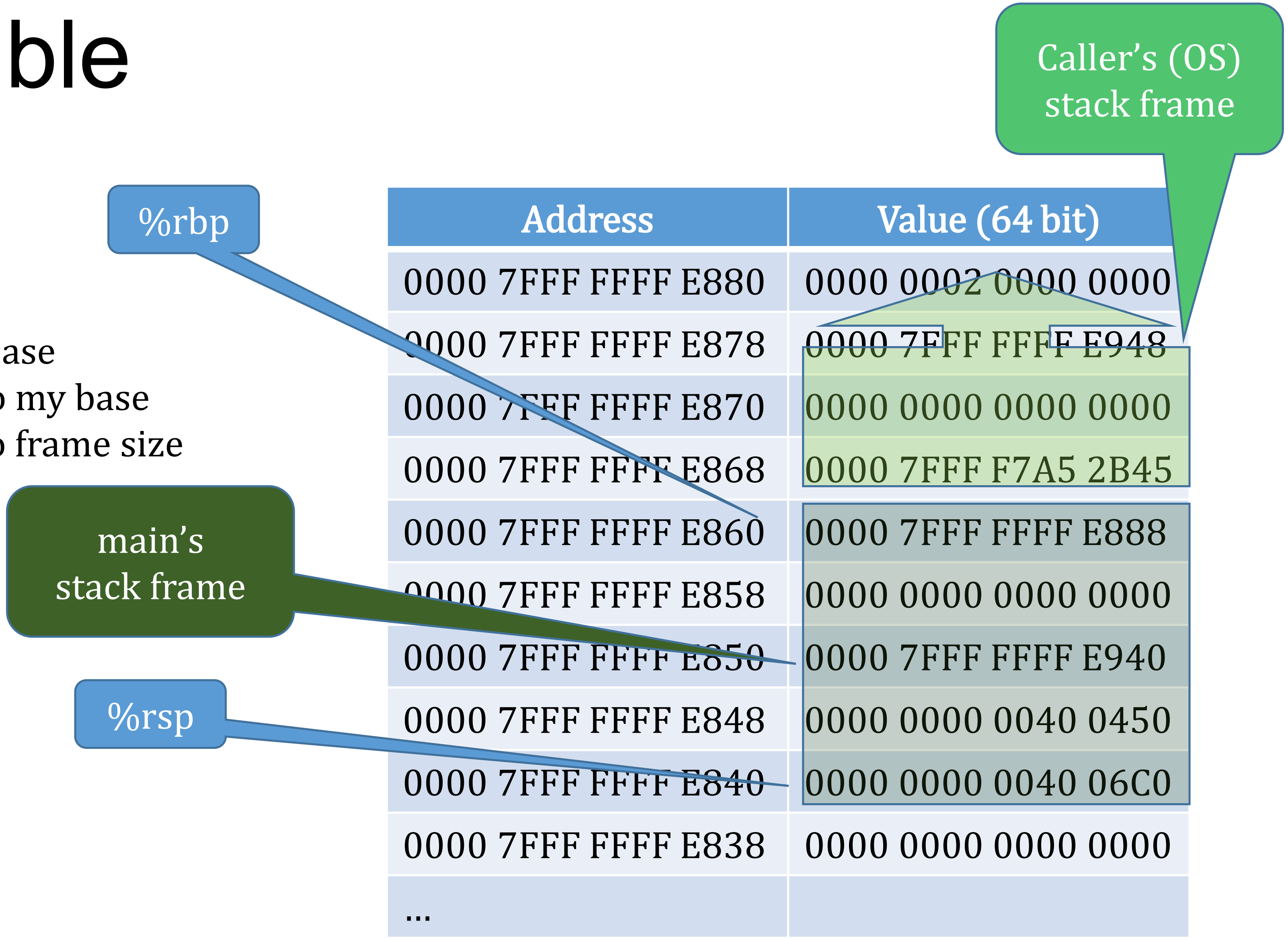
%rsp

main’s stack frame

Address	Value (64 bit)
0000 7FFF FFFF E880	0000 0002 0000 0000
0000 7FFF FFFF E878	0000 7FFF FFFF E948
0000 7FFF FFFF E870	0000 0000 0000 0000
0000 7FFF FFFF E868	0000 7FFF F7A5 2B45
0000 7FFF FFFF E860	0000 7FFF FFFF E888
0000 7FFF FFFF E858	0000 0000 0000 0000
0000 7FFF FFFF E850	0000 7FFF FFFF E940
0000 7FFF FFFF E848	0000 0000 0040 0450
0000 7FFF FFFF E840	0000 0000 0040 06C0
0000 7FFF FFFF E838	0000 0000 0000 0000
…	

Main's Preamble

```
pushq     %rbp           ; Save caller's base
movq      %rsp, %rbp   ; Reset %rbp to my base
subq      $32, %rsp      ; Reset %rsp to frame size
; main x86 instructions
```



Address	Value (64 bit)
0000 7FFF FFFF E880	0000 0002 0000 0000
0000 7FFF FFFF E878	0000 7FFF FFFF E948
0000 7FFF FFFF E870	0000 0000 0000 0000
0000 7FFF FFFF E868	0000 7FFF F7A5 2B45
0000 7FFF FFFF E860	0000 7FFF FFFF E888
0000 7FFF FFFF E858	0000 0000 0000 0000
0000 7FFF FFFF E850	0000 7FFF FFFF E940
0000 7FFF FFFF E848	0000 0000 0040 0450
0000 7FFF FFFF E840	0000 0000 0040 06C0
0000 7FFF FFFF E838	0000 0000 0000 0000
…	

Example Call Stack

```
1. int addem(int x, int y);
2. int main() {
3.     int a=addem(3,4);
4.      a=addem(a,4);
5.      return 0;
6. }
7. int addem(int x, int y) { return x+y;}
```

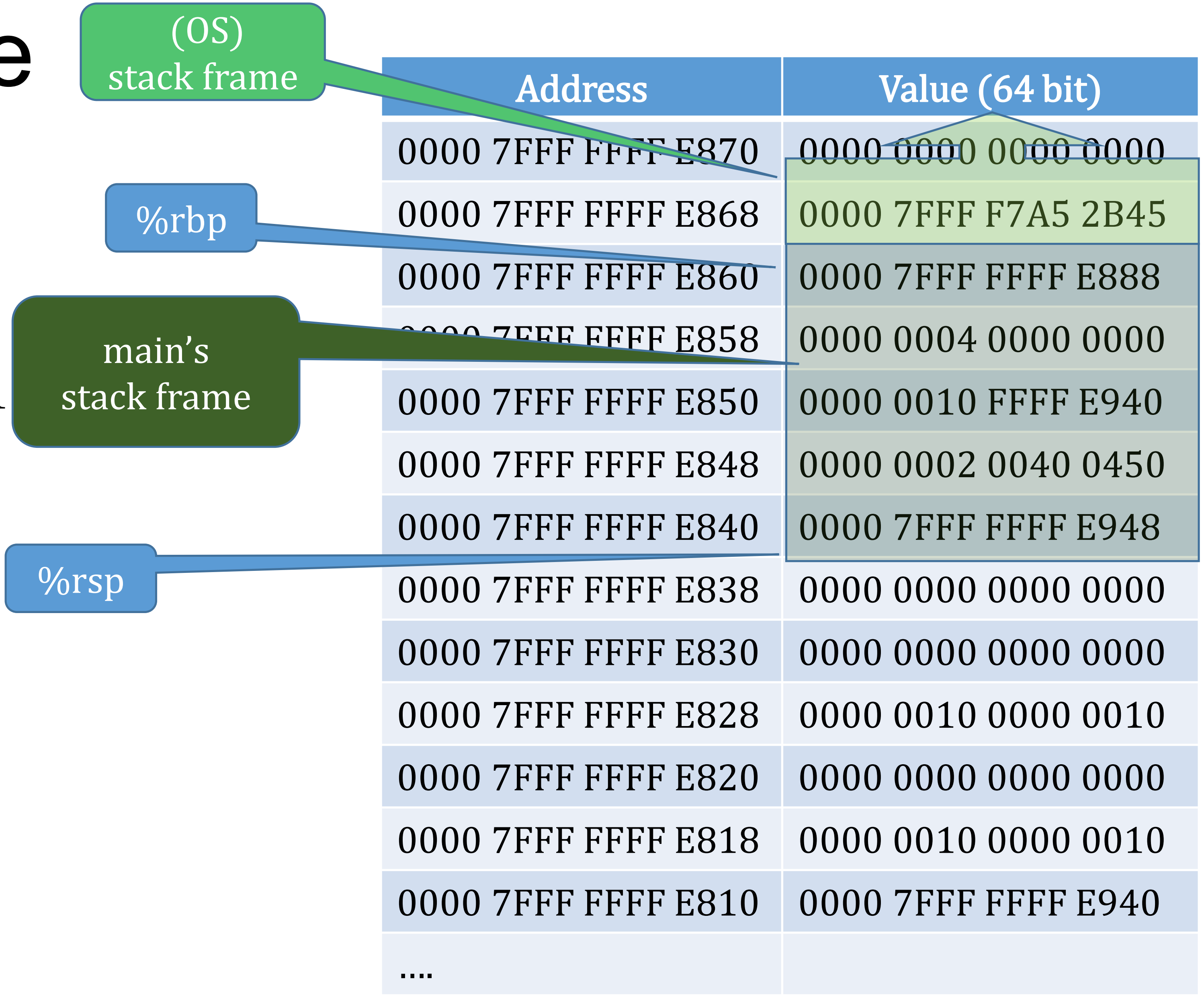
Inv	Fn	args	vars	Ret
3.10	addem	x=3,y=4		

Inv	Fn	args	vars	Ret
OS	main		a=	

In main’s code

pushq %rip
jmp add

```
...
4005D4 callq      addem  ; at 400621
4005D9 mov        %eax,-08x(%rbp)
...
```



(OS) stack frame

%rbp

main’s stack frame

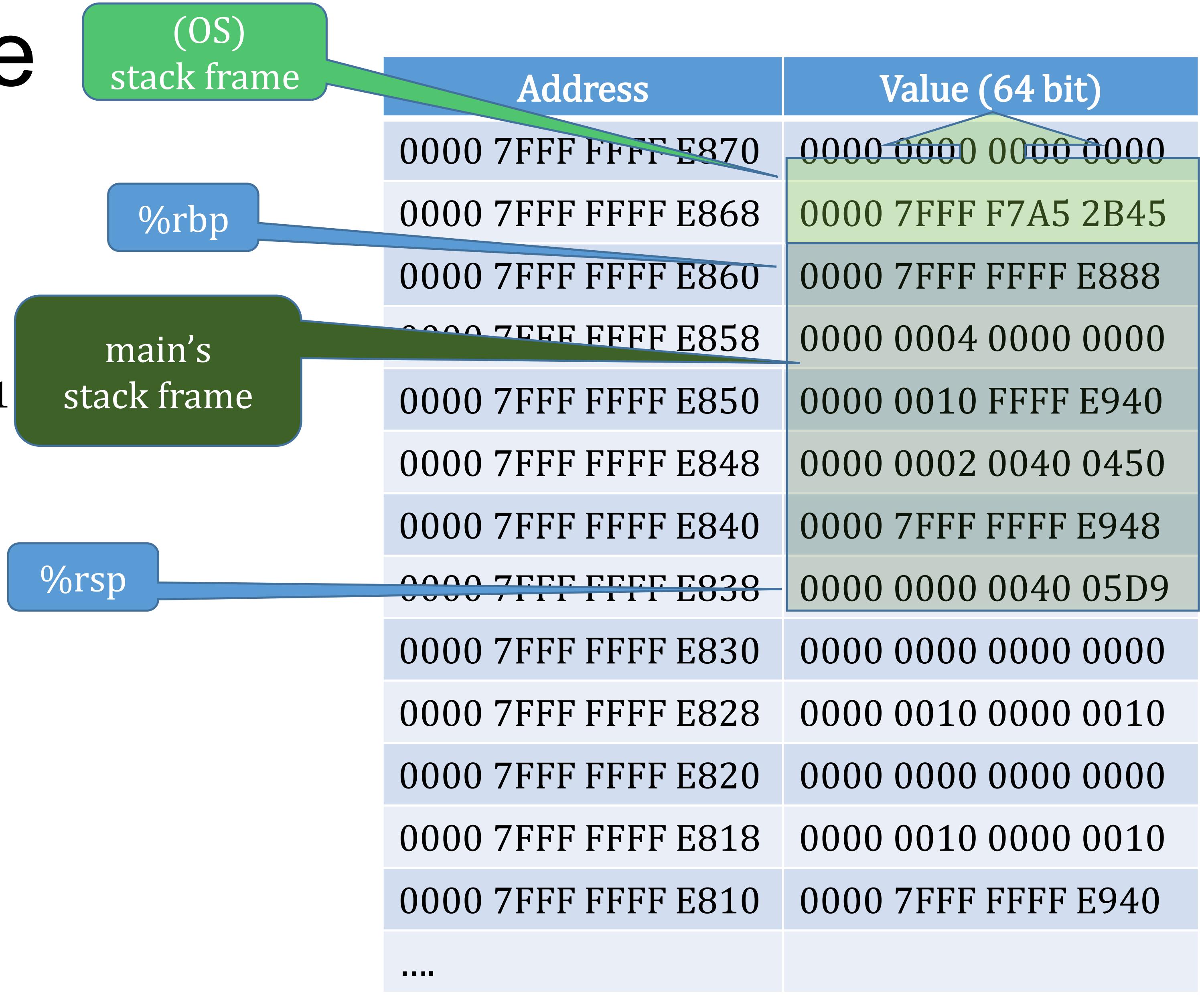
%rsp

Address	Value (64 bit)
0000 7FFF FFFF E870	0000 0000 0000 0000
0000 7FFF FFFF E868	0000 7FFF F7A5 2B45
0000 7FFF FFFF E860	0000 7FFF FFFF E888
0000 7FFF FFFF E858	0000 0004 0000 0000
0000 7FFF FFFF E850	0000 0010 FFFF E940
0000 7FFF FFFF E848	0000 0002 0040 0450
0000 7FFF FFFF E840	0000 7FFF FFFF E948
0000 7FFF FFFF E838	0000 0000 0000 0000
0000 7FFF FFFF E830	0000 0000 0000 0000
0000 7FFF FFFF E828	0000 0010 0000 0010
0000 7FFF FFFF E820	0000 0000 0000 0000
0000 7FFF FFFF E818	0000 0010 0000 0010
0000 7FFF FFFF E810	0000 7FFF FFFF E940
….	

In main's code

pushq %rip
jmp addem

```
...
4005D4 callq      addem  ; at 400621
4005D9 mov        %eax,-08x(%rbp)
...
```



(OS) stack frame

%rbp

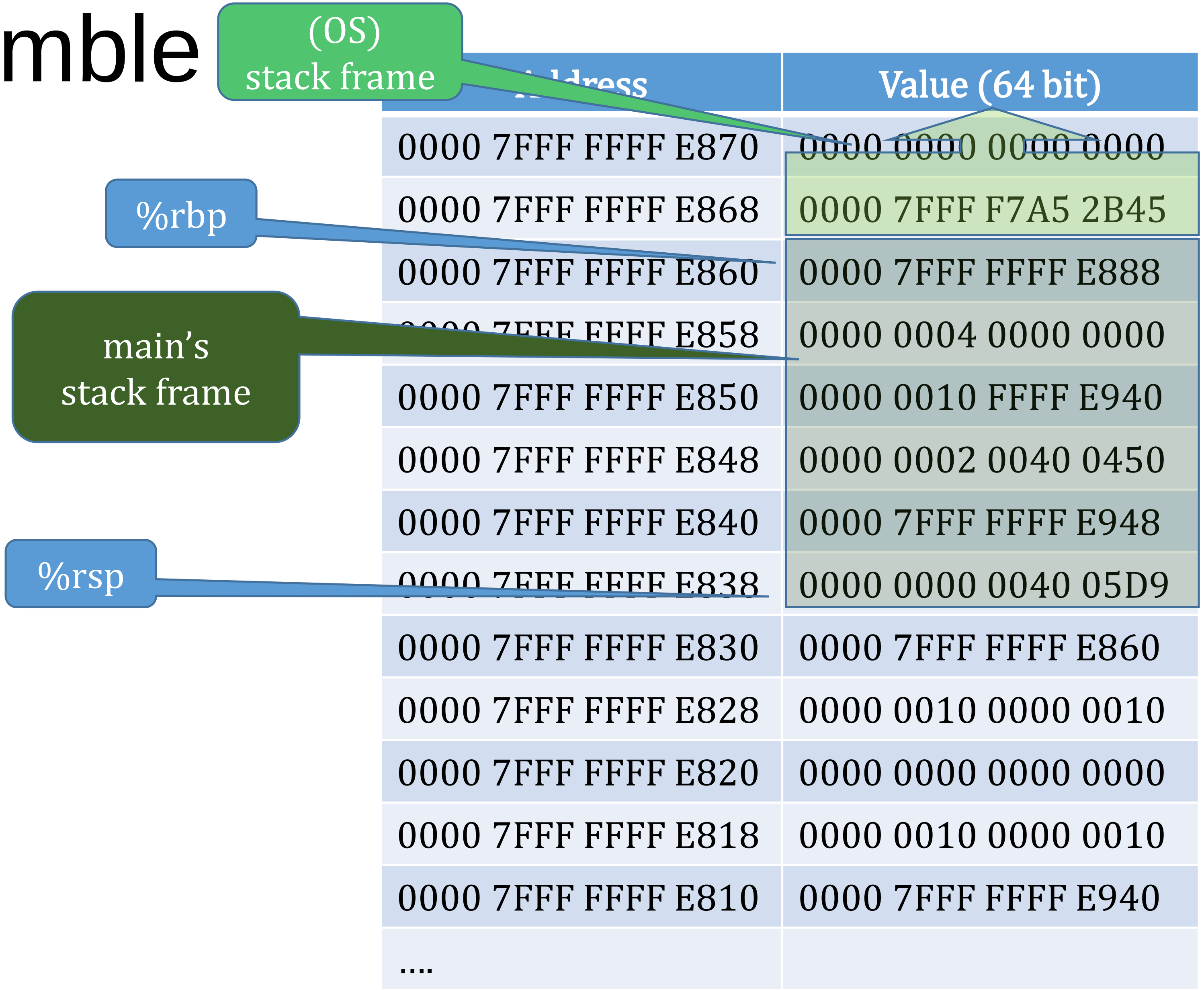
main's stack frame

%rsp

Address	Value (64 bit)
0000 7FFF FFFF E870	0000 0000 0000 0000
0000 7FFF FFFF E868	0000 7FFF F7A5 2B45
0000 7FFF FFFF E860	0000 7FFF FFFF E888
0000 7FFF FFFF E858	0000 0004 0000 0000
0000 7FFF FFFF E850	0000 0010 FFFF E940
0000 7FFF FFFF E848	0000 0002 0040 0450
0000 7FFF FFFF E840	0000 7FFF FFFF E948
0000 7FFF FFFF E838	0000 0000 0040 05D9
0000 7FFF FFFF E830	0000 0000 0000 0000
0000 7FFF FFFF E828	0000 0010 0000 0010
0000 7FFF FFFF E820	0000 0000 0000 0000
0000 7FFF FFFF E818	0000 0010 0000 0010
0000 7FFF FFFF E810	0000 7FFF FFFF E940
….	

addem’s preamble

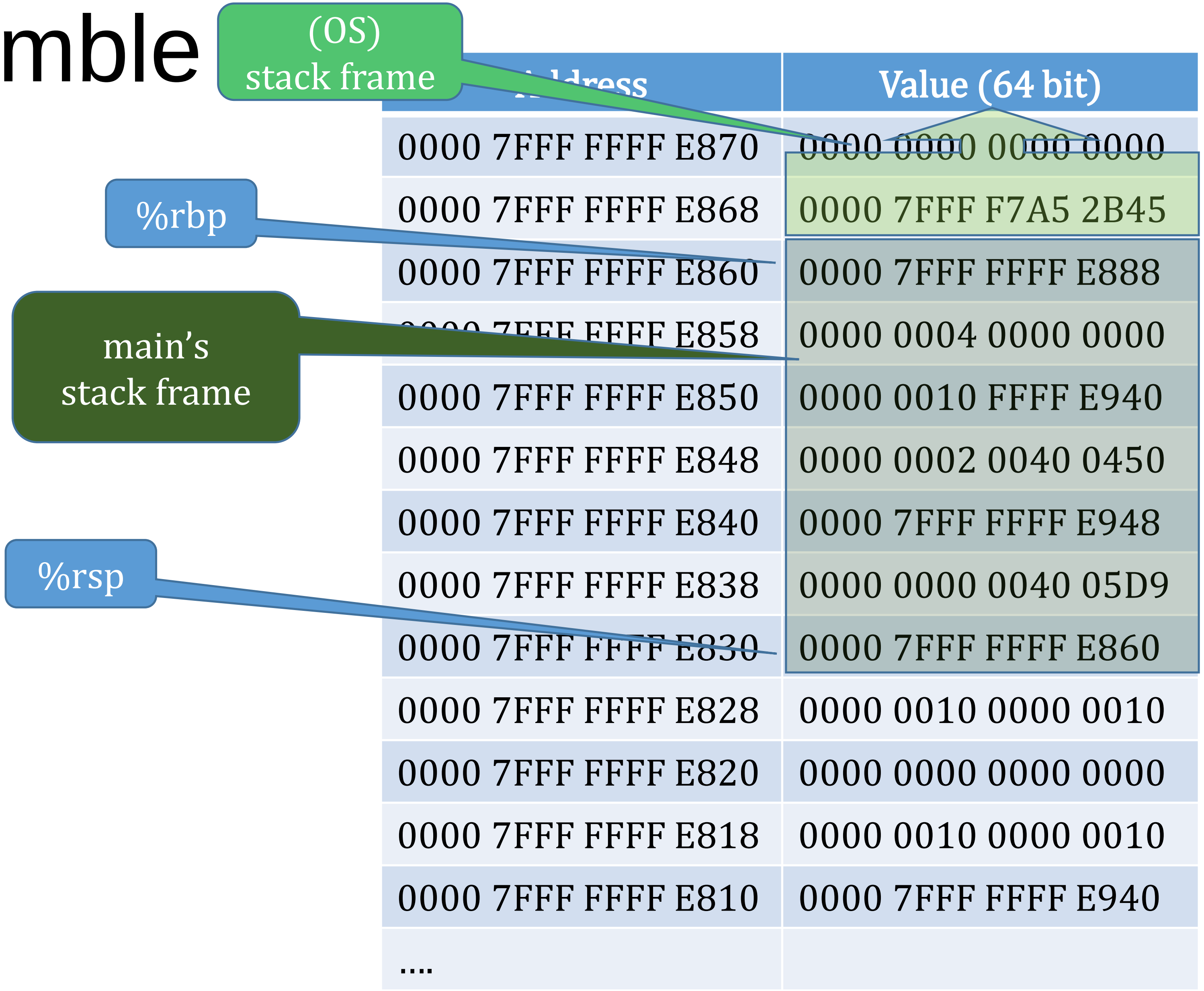
```
pushq     %rbp
movq      %rsp, %rbp
…
```



Address	Value (64 bit)
0000 7FFF FFFF E870	0000 0000 0000 0000
0000 7FFF FFFF E868	0000 7FFF F7A5 2B45
0000 7FFF FFFF E860	0000 7FFF FFFF E888
0000 7FFF FFFF E858	0000 0004 0000 0000
0000 7FFF FFFF E850	0000 0010 FFFF E940
0000 7FFF FFFF E848	0000 0002 0040 0450
0000 7FFF FFFF E840	0000 7FFF FFFF E948
0000 7FFF FFFF E838	0000 0000 0040 05D9
0000 7FFF FFFF E830	0000 7FFF FFFF E860
0000 7FFF FFFF E828	0000 0010 0000 0010
0000 7FFF FFFF E820	0000 0000 0000 0000
0000 7FFF FFFF E818	0000 0010 0000 0010
0000 7FFF FFFF E810	0000 7FFF FFFF E940
….	

addem’s preamble

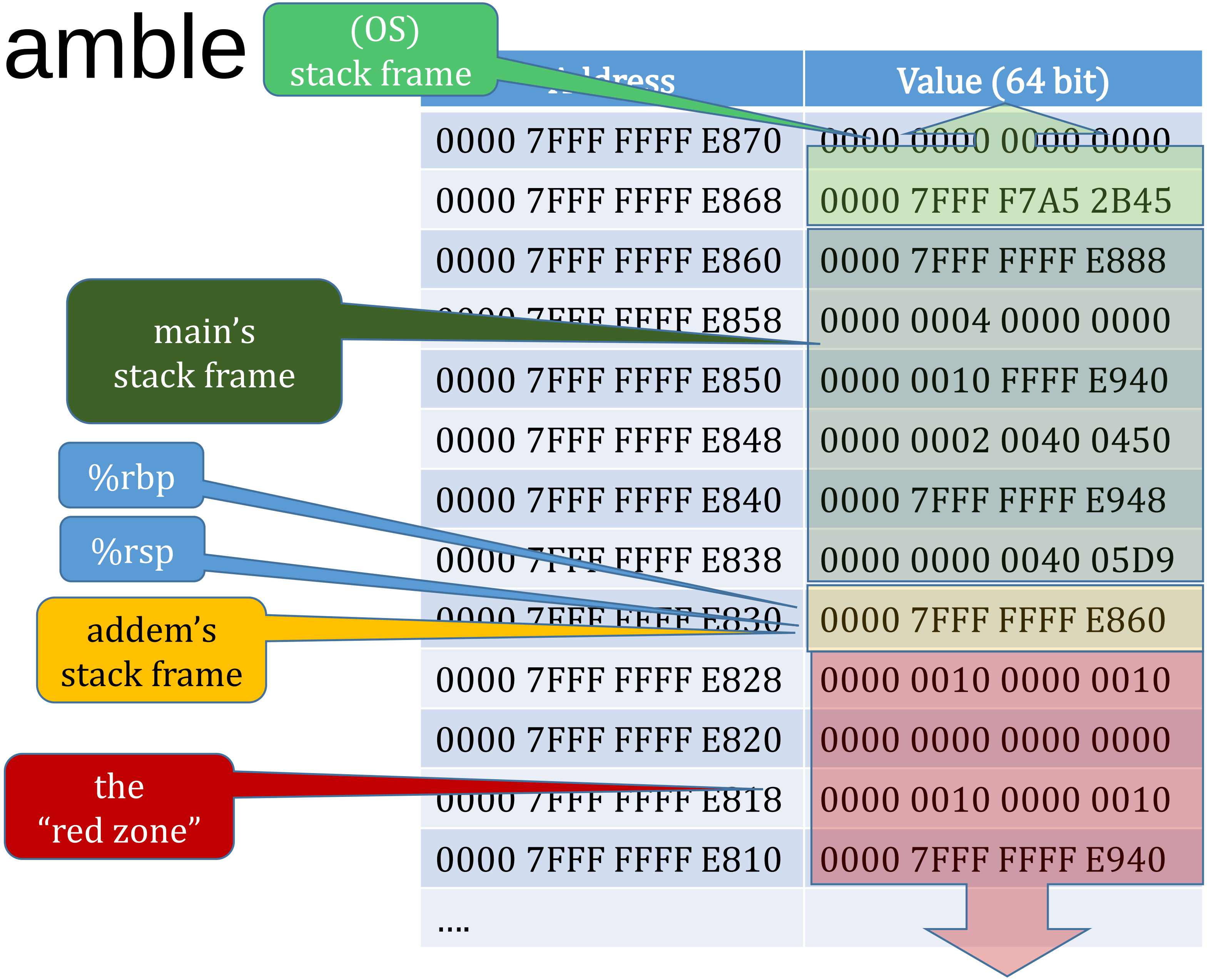
```
pushq     %rbp
movq      %rsp, %rbp
…
```



Address	Value (64 bit)
0000 7FFF FFFF E870	0000 0000 0000 0000
0000 7FFF FFFF E868	0000 7FFF F7A5 2B45
0000 7FFF FFFF E860	0000 7FFF FFFF E888
0000 7FFF FFFF E858	0000 0004 0000 0000
0000 7FFF FFFF E850	0000 0010 FFFF E940
0000 7FFF FFFF E848	0000 0002 0040 0450
0000 7FFF FFFF E840	0000 7FFF FFFF E948
0000 7FFF FFFF E838	0000 0000 0040 05D9
0000 7FFF FFFF E830	0000 7FFF FFFF E860
0000 7FFF FFFF E828	0000 0010 0000 0010
0000 7FFF FFFF E820	0000 0000 0000 0000
0000 7FFF FFFF E818	0000 0010 0000 0010
0000 7FFF FFFF E810	0000 7FFF FFFF E940
….	

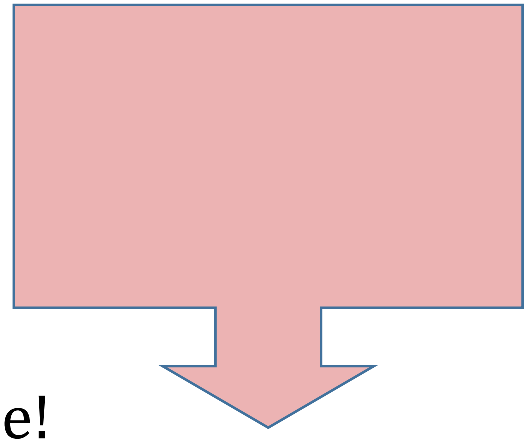
addem's preamble

```
pushq     %rbp
movq      %rsp, %rbp
...
```



Address	Value (64 bit)
0000 7FFF FFFF E870	0000 0000 0000 0000
0000 7FFF FFFF E868	0000 7FFF F7A5 2B45
0000 7FFF FFFF E860	0000 7FFF FFFF E888
0000 7FFF FFFF E858	0000 0004 0000 0000
0000 7FFF FFFF E850	0000 0010 FFFF E940
0000 7FFF FFFF E848	0000 0002 0040 0450
0000 7FFF FFFF E840	0000 7FFF FFFF E948
0000 7FFF FFFF E838	0000 0000 0040 05D9
0000 7FFF FFFF E830	0000 7FFF FFFF E860
0000 7FFF FFFF E828	0000 0010 0000 0010
0000 7FFF FFFF E820	0000 0000 0000 0000
0000 7FFF FFFF E818	0000 0010 0000 0010
0000 7FFF FFFF E810	0000 7FFF FFFF E940
….	

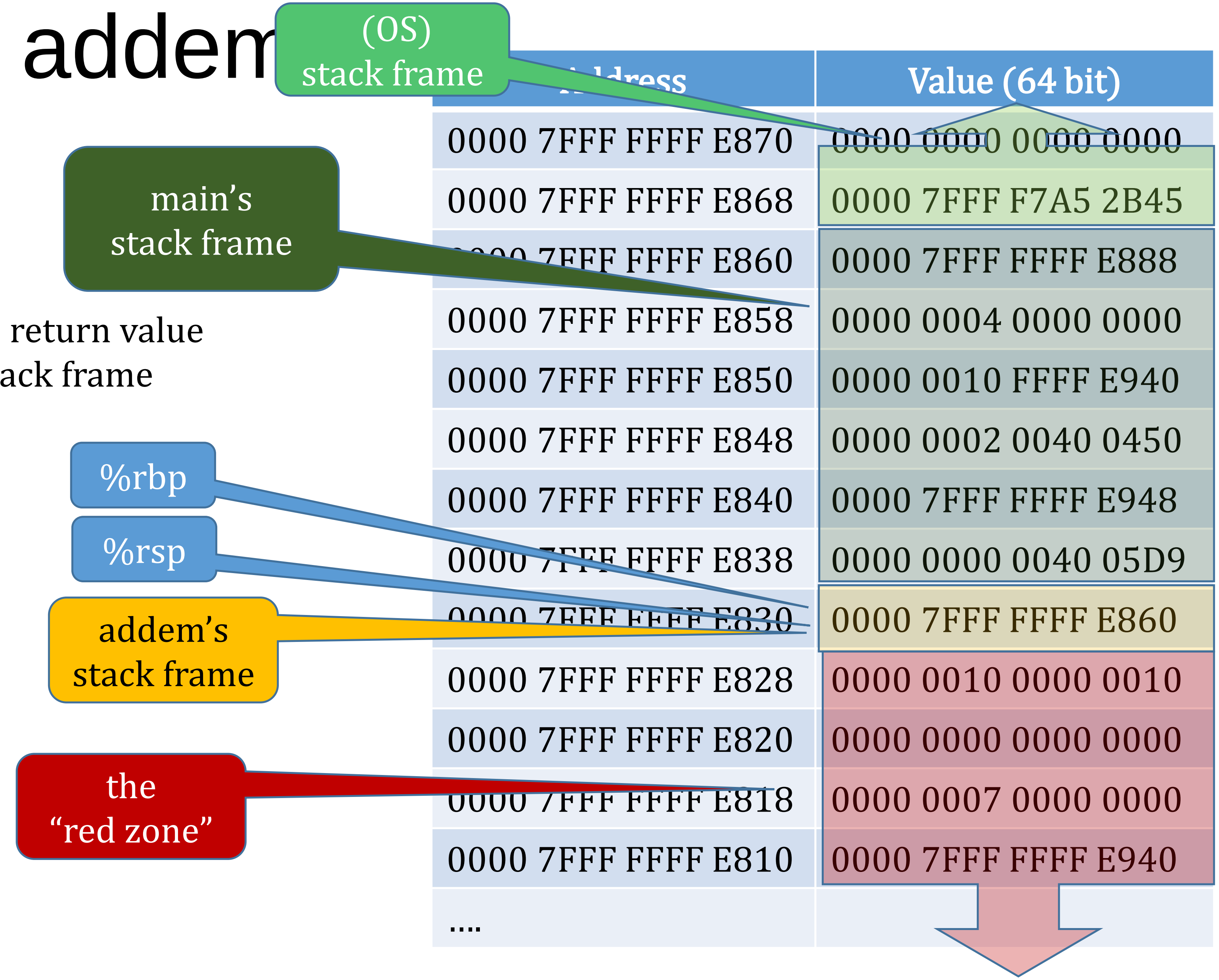
The “red zone”



- 128 bytes (16 addresses) below %rsp (below stack)
- Operating system will not modify the red zone
  - Any asynchronous interrupts may not modify the red zone!
- If this function does not invoke other functions (leaf function) it may use the red zone WITHOUT modifying %rsp
  - If lower level functions are called, the red zone would get modified
- gcc uses the red zone for local variables, parameters, and saved registers for leaf functions
- Saves instructions to modify %rsp in entry and return

Return from addem

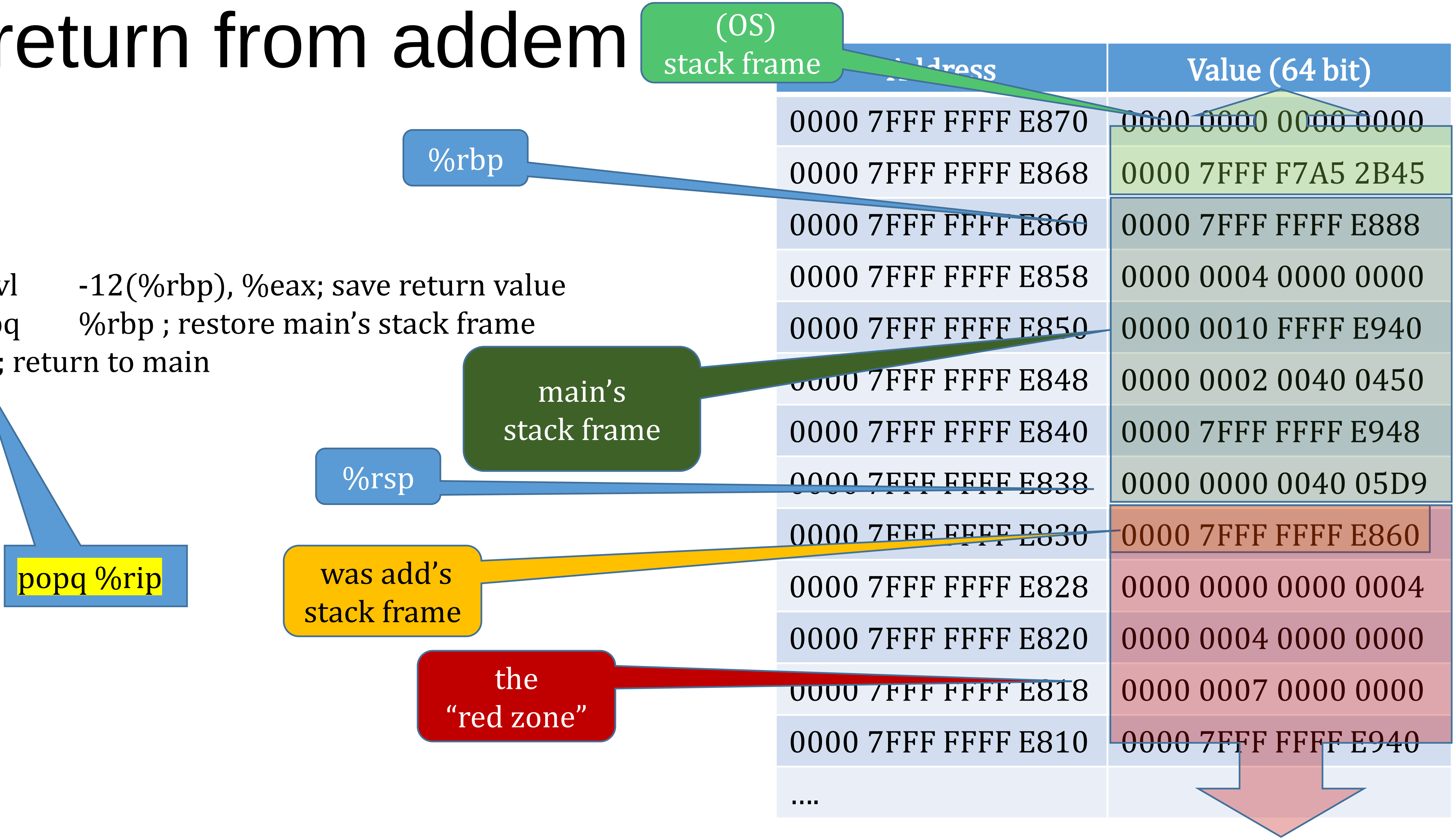
```
movl    +12(%rbp), %eax; save return value
popq    %rbp ; restore main's stack frame
ret
```



Address	Value (64 bit)
0000 7FFF FFFF E870	0000 0000 0000 0000
0000 7FFF FFFF E868	0000 7FFF F7A5 2B45
0000 7FFF FFFF E860	0000 7FFF FFFF E888
0000 7FFF FFFF E858	0000 0004 0000 0000
0000 7FFF FFFF E850	0000 0010 FFFF E940
0000 7FFF FFFF E848	0000 0002 0040 0450
0000 7FFF FFFF E840	0000 7FFF FFFF E948
0000 7FFF FFFF E838	0000 0000 0040 05D9
0000 7FFF FFFF E830	0000 7FFF FFFF E860
0000 7FFF FFFF E828	0000 0010 0000 0010
0000 7FFF FFFF E820	0000 0000 0000 0000
0000 7FFF FFFF E818	0000 0007 0000 0000
0000 7FFF FFFF E810	0000 7FFF FFFF E940
….	

return from addem

```
movl      -12(%rbp), %eax; save return value
popq      %rbp ; restore main's stack frame
ret ; return to main
```



Address	Value (64 bit)
0000 7FFF FFFF E870	0000 0000 0000 0000
0000 7FFF FFFF E868	0000 7FFF F7A5 2B45
0000 7FFF FFFF E860	0000 7FFF FFFF E888
0000 7FFF FFFF E858	0000 0004 0000 0000
0000 7FFF FFFF E850	0000 0010 FFFF E940
0000 7FFF FFFF E848	0000 0002 0040 0450
0000 7FFF FFFF E840	0000 7FFF FFFF E948
0000 7FFF FFFF E838	0000 0000 0040 05D9
0000 7FFF FFFF E830	0000 7FFF FFFF E860
0000 7FFF FFFF E828	0000 0000 0000 0004
0000 7FFF FFFF E820	0000 0004 0000 0000
0000 7FFF FFFF E818	0000 0007 0000 0000
0000 7FFF FFFF E810	0000 7FFF FFFF E940
….	

return from addem

```
...
4005D4 callq      addem  ; at 400621
4005D9 mov        %eax,-08x(%rbp)
...
```

(OS) stack frame

%rbp

main's stack frame

%rsp

Address	Value (64 bit)
0000 7FFF FFFF E870	0000 0000 0000 0000
0000 7FFF FFFF E868	0000 7FFF F7A5 2B45
0000 7FFF FFFF E860	0000 7FFF FFFF E888
0000 7FFF FFFF E858	0000 0007 0000 0000
0000 7FFF FFFF E850	0000 0010 FFFF E940
0000 7FFF FFFF E848	0000 0002 0040 0450
0000 7FFF FFFF E840	0000 7FFF FFFF E948
0000 7FFF FFFF E838	0000 0000 0040 05D9
0000 7FFF FFFF E830	0000 7FFF FFFF E860
0000 7FFF FFFF E828	0000 0000 0000 0004
0000 7FFF FFFF E820	0000 0004 0000 0000
0000 7FFF FFFF E818	0000 0007 0000 0000
0000 7FFF FFFF E810	0000 7FFF FFFF E940
....	

Example Call Stack

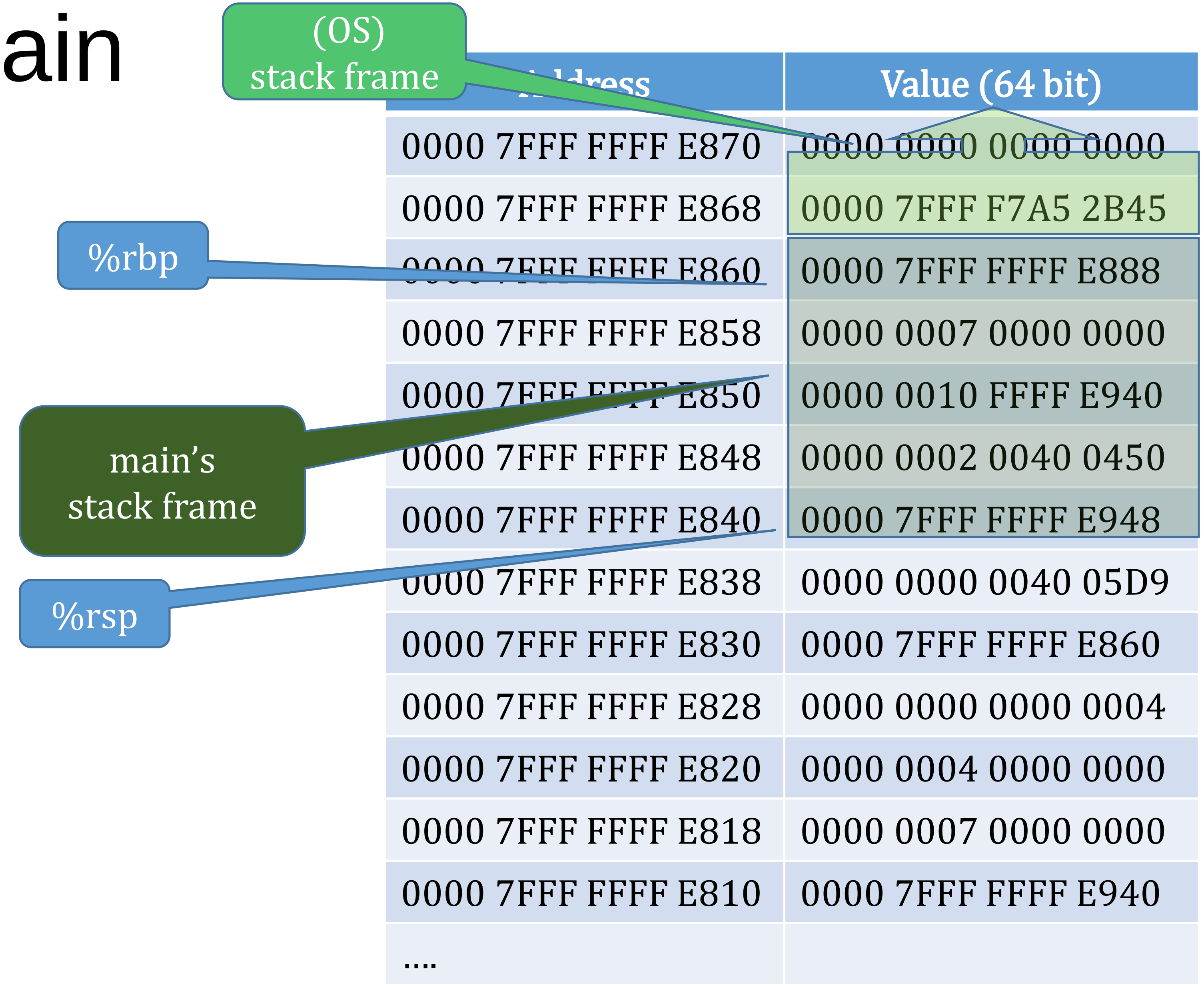
```
1. int addem(int x, int y);
2. int main() {
3.     int a=addem(3,4);
4.      a=addem(a,4);
5.      return 0;
6. }
7. int addem(int x, int y) { return x+y;}
```

Inv	Fn	args	vars	Ret
OS	main		a=7	

return from main

```
leave ; restore OS stack frame
ret
```

```
movq      %rbp,%rsp
popq      %rbp
```

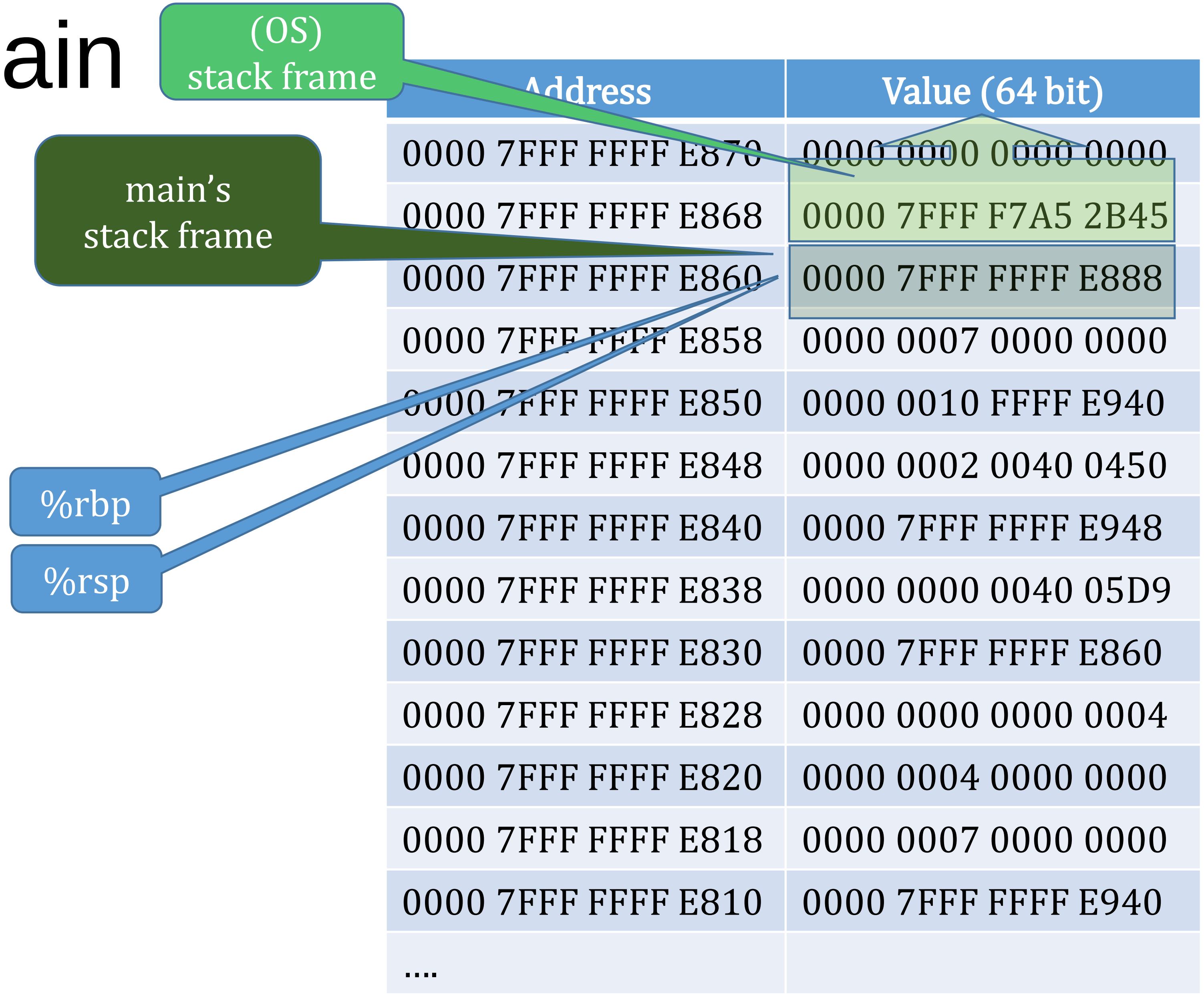


Address	Value (64 bit)
0000 7FFF FFFF E870	0000 0000 0000 0000
0000 7FFF FFFF E868	0000 7FFF F7A5 2B45
0000 7FFF FFFF E860	0000 7FFF FFFF E888
0000 7FFF FFFF E858	0000 0007 0000 0000
0000 7FFF FFFF E850	0000 0010 FFFF E940
0000 7FFF FFFF E848	0000 0002 0040 0450
0000 7FFF FFFF E840	0000 7FFF FFFF E948
0000 7FFF FFFF E838	0000 0000 0040 05D9
0000 7FFF FFFF E830	0000 7FFF FFFF E860
0000 7FFF FFFF E828	0000 0000 0000 0004
0000 7FFF FFFF E820	0000 0004 0000 0000
0000 7FFF FFFF E818	0000 0007 0000 0000
0000 7FFF FFFF E810	0000 7FFF FFFF E940
….	

return from main

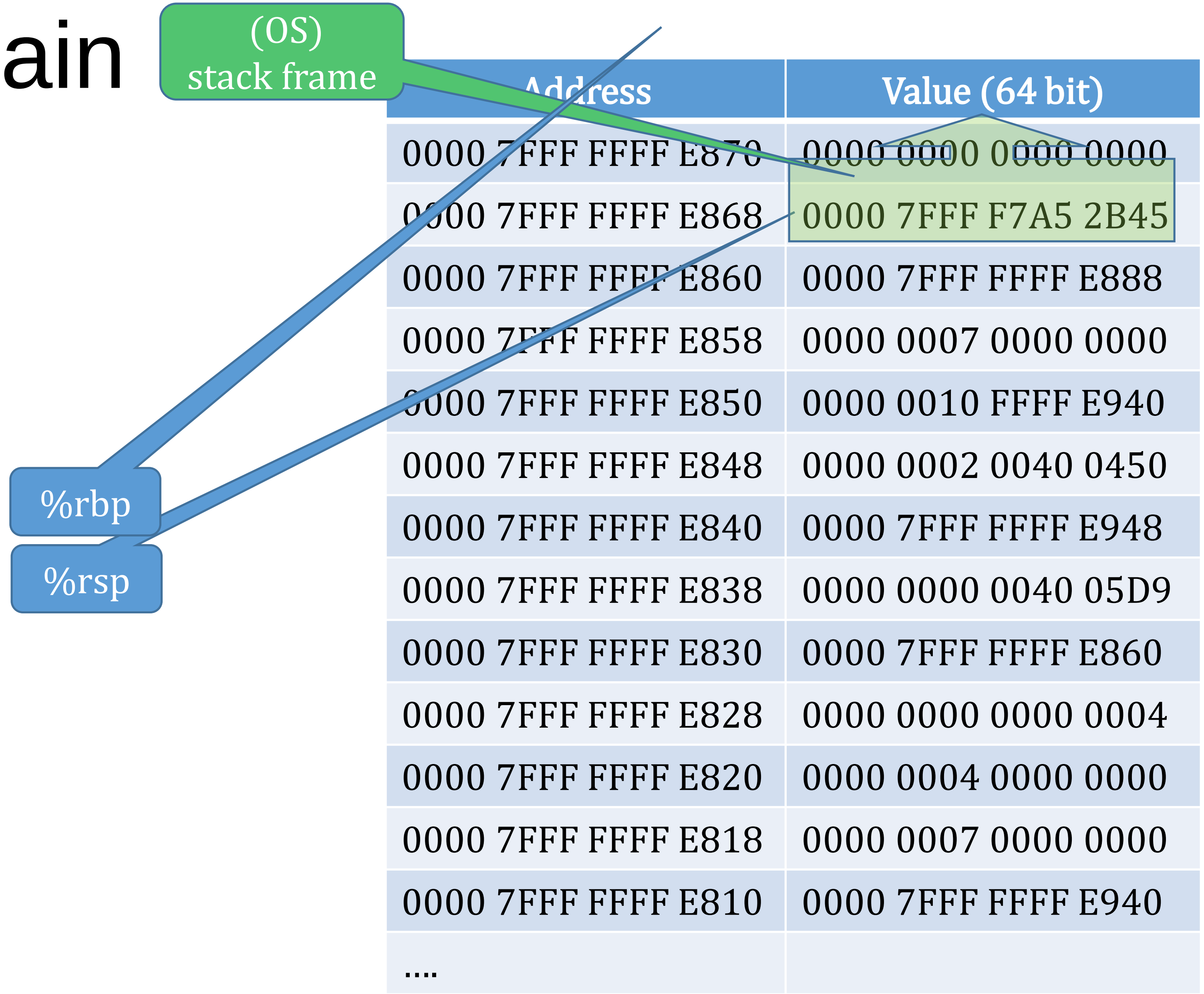
```
leave ; restore OS stack frame
ret
```

```
movq      %rbp,%rsp
popq      %rbp
```



Address	Value (64 bit)
0000 7FFF FFFF E870	0000 0000 0000 0000
0000 7FFF FFFF E868	0000 7FFF F7A5 2B45
0000 7FFF FFFF E860	0000 7FFF FFFF E888
0000 7FFF FFFF E858	0000 0007 0000 0000
0000 7FFF FFFF E850	0000 0010 FFFF E940
0000 7FFF FFFF E848	0000 0002 0040 0450
0000 7FFF FFFF E840	0000 7FFF FFFF E948
0000 7FFF FFFF E838	0000 0000 0040 05D9
0000 7FFF FFFF E830	0000 7FFF FFFF E860
0000 7FFF FFFF E828	0000 0000 0000 0004
0000 7FFF FFFF E820	0000 0004 0000 0000
0000 7FFF FFFF E818	0000 0007 0000 0000
0000 7FFF FFFF E810	0000 7FFF FFFF E940
….	

return from main



(OS)
stack frame

%rbp

%rsp

Address	Value (64 bit)
0000 7FFF FFFF E870	0000 0000 0000 0000
0000 7FFF FFFF E868	0000 7FFF F7A5 2B45
0000 7FFF FFFF E860	0000 7FFF FFFF E888
0000 7FFF FFFF E858	0000 0007 0000 0000
0000 7FFF FFFF E850	0000 0010 FFFF E940
0000 7FFF FFFF E848	0000 0002 0040 0450
0000 7FFF FFFF E840	0000 7FFF FFFF E948
0000 7FFF FFFF E838	0000 0000 0040 05D9
0000 7FFF FFFF E830	0000 7FFF FFFF E860
0000 7FFF FFFF E828	0000 0000 0000 0004
0000 7FFF FFFF E820	0000 0004 0000 0000
0000 7FFF FFFF E818	0000 0007 0000 0000
0000 7FFF FFFF E810	0000 7FFF FFFF E940
….	