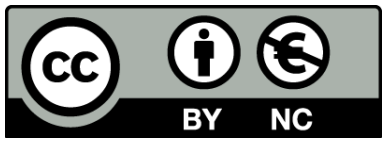
Architecture et technologie des ordinateurs

- La pile -

Sébastien Fourey
Hugo Descoubes

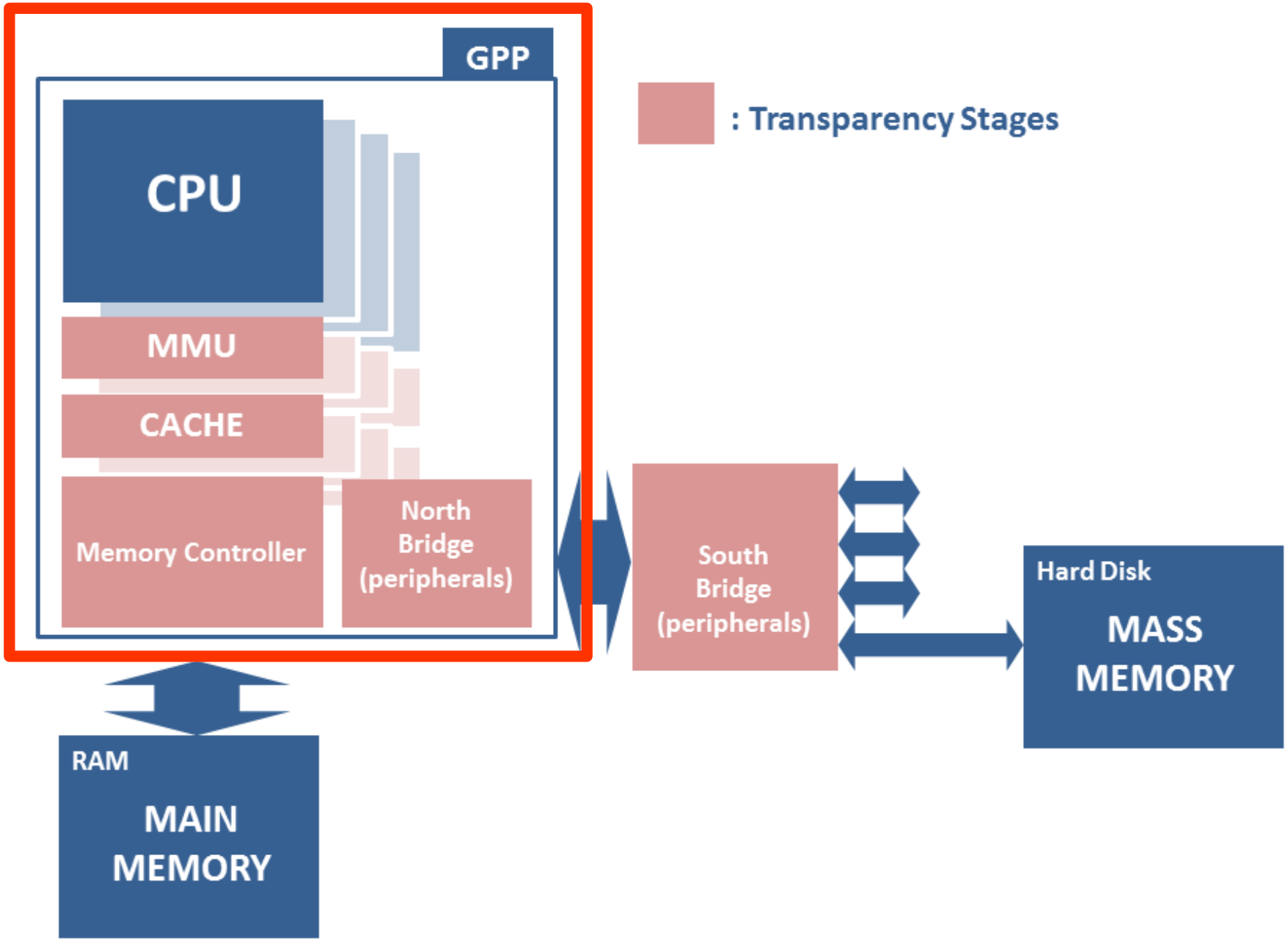
ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS
CENTRE DE RECHERCHE

L'École des INGÉNIEURS Scientifiques

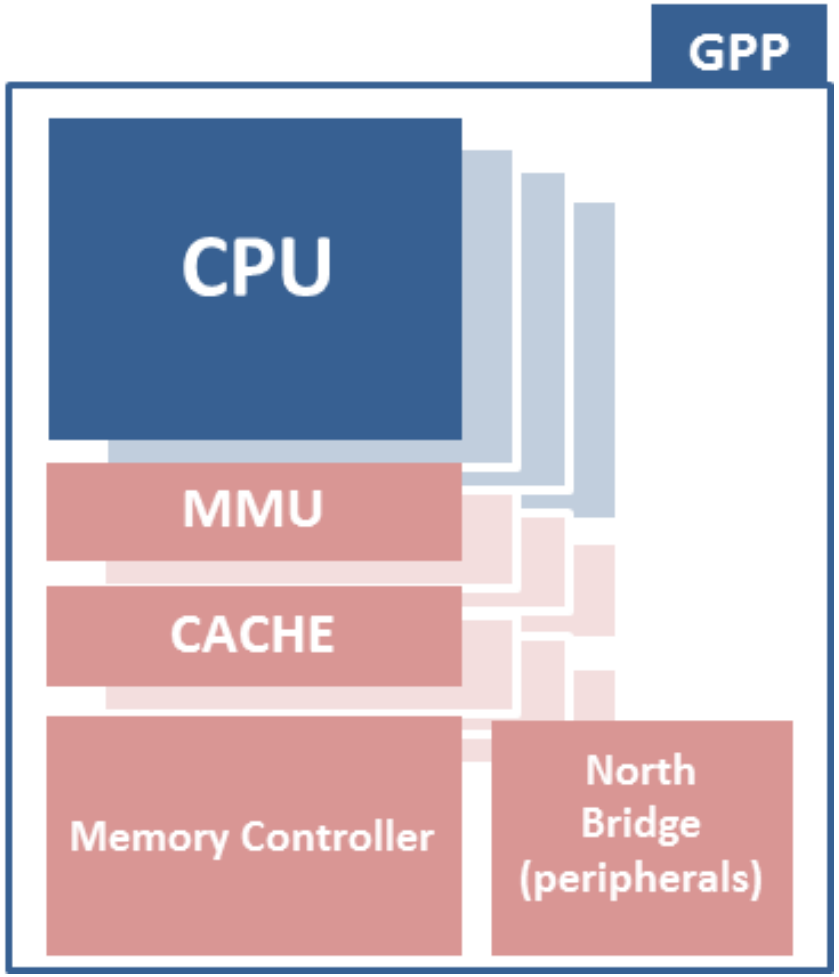


Décembre 2020

Rappel


GPP
CPU
MMU
CACHE
Memory Controller
North Bridge (peripherals)
: Transparency Stages
South Bridge (peripherals)
Hard Disk
MASS MEMORY
RAM
MAIN MEMORY

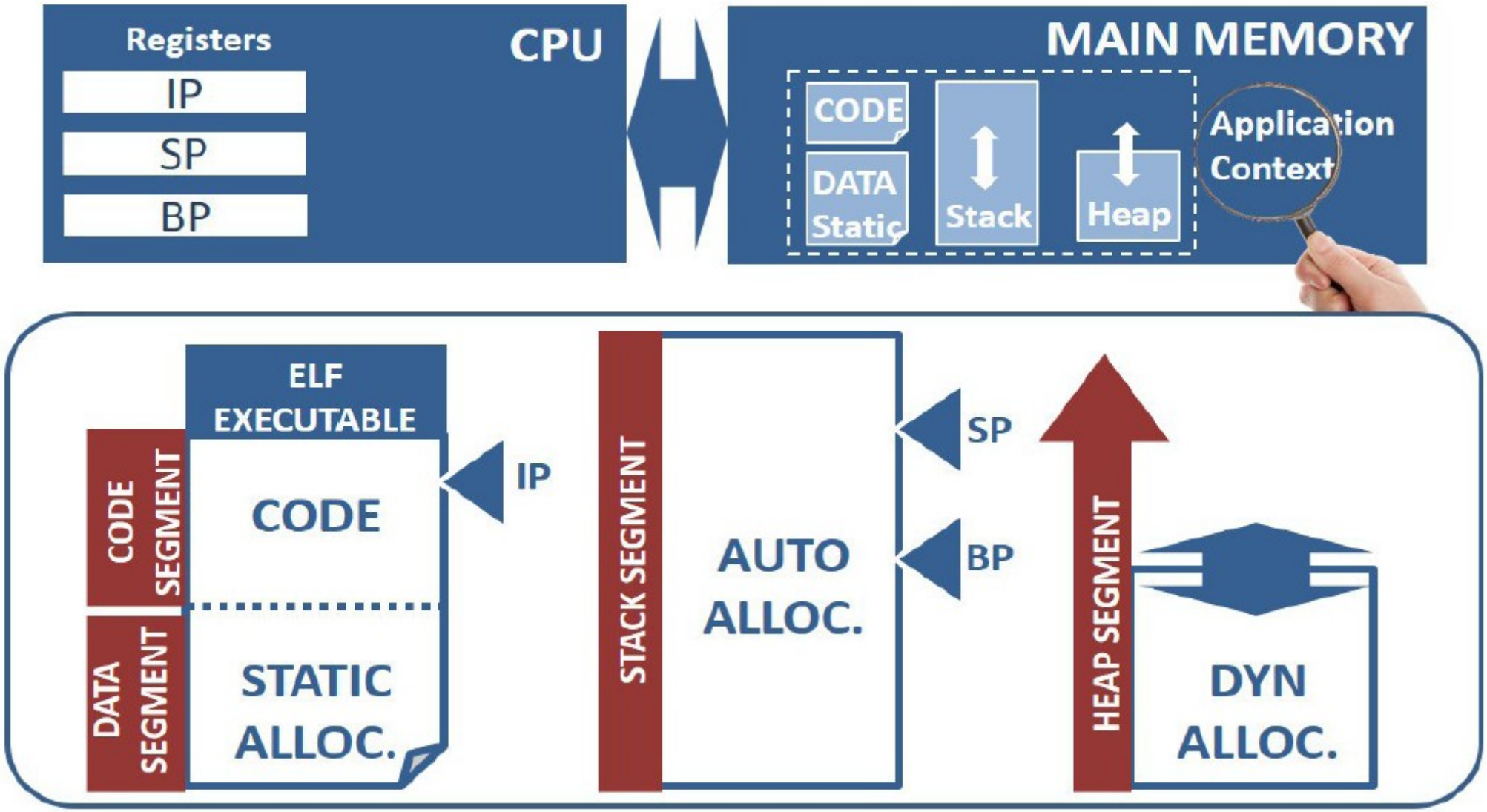
GPP & CPU



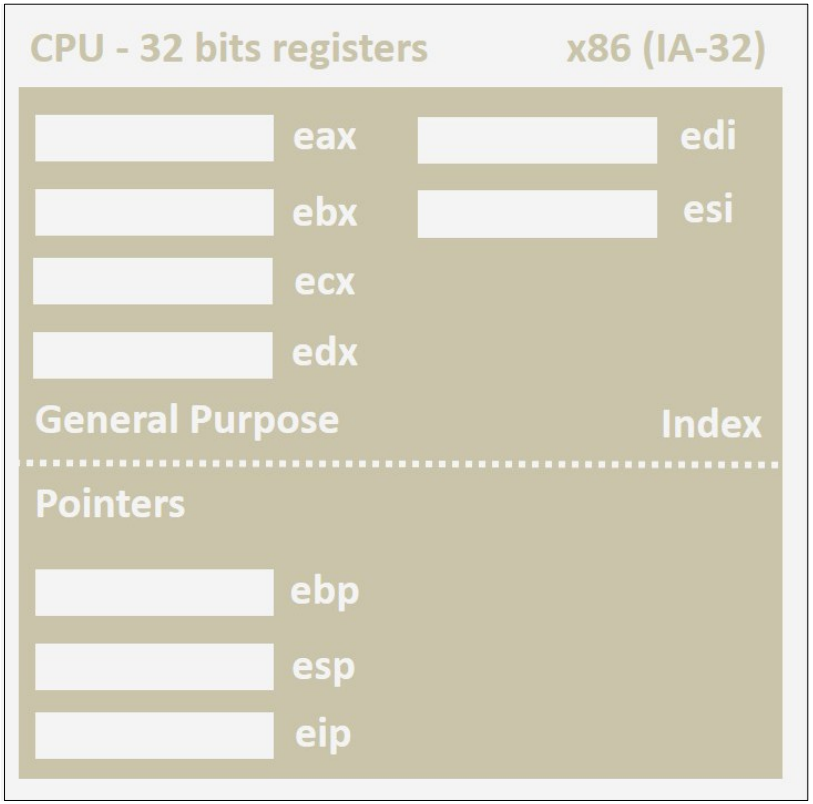
GPP : General Purpose Processor

CPU : Central Processing Unit

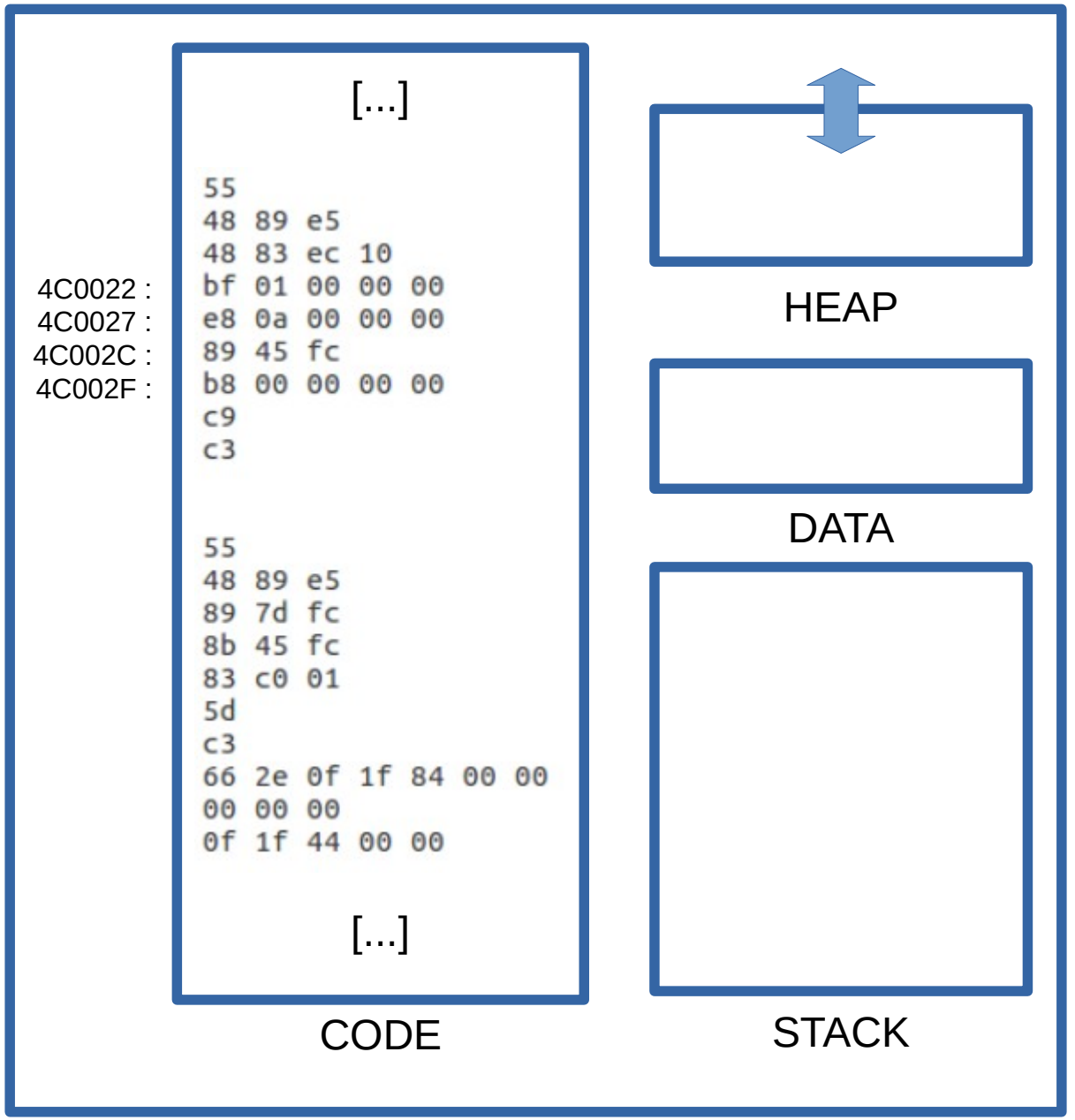
CPU & Mémoire


Registers
IP
SP
BP
CPU
MAIN MEMORY
CODE
DATA
Static
Stack
Heap
Application
Context
ELF
EXECUTABLE
CODE
SEGMENT
CODE
IP
DATA
SEGMENT
STATIC
ALLOC.
STACK SEGMENT
AUTO
ALLOC.
SP
BP
HEAP SEGMENT
DYN
ALLOC.

Programme, mémoire et pile

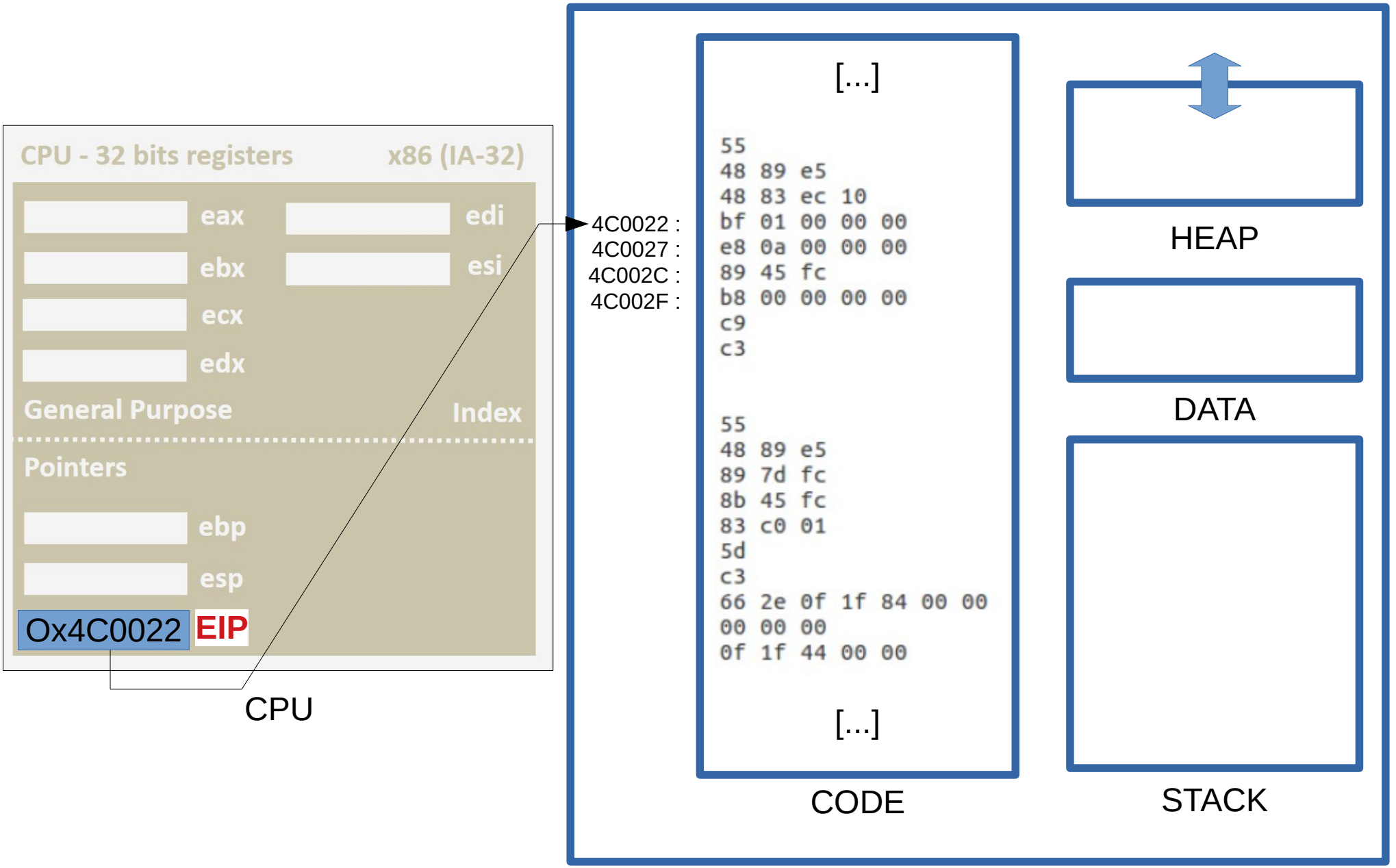


CPU

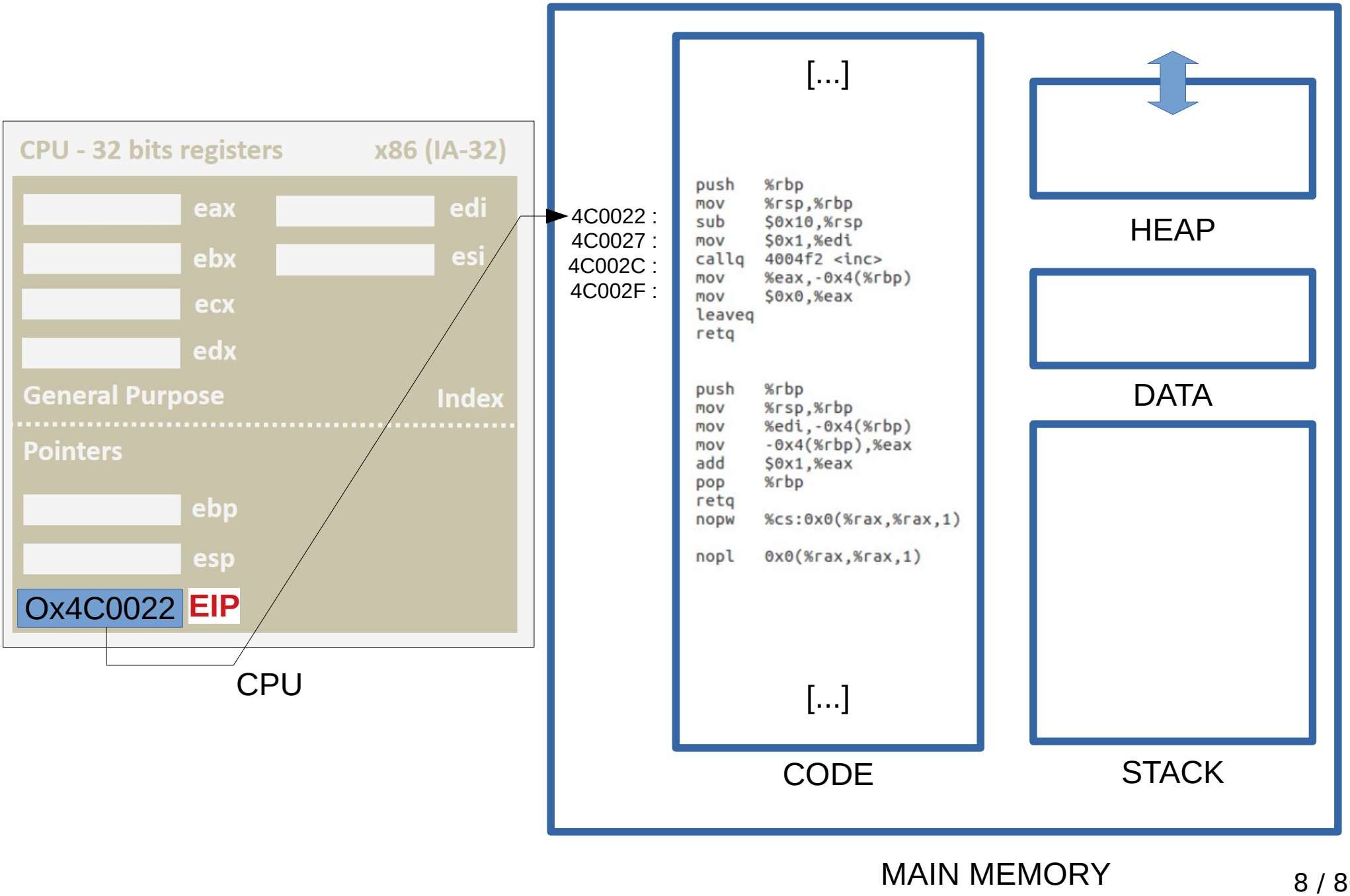


MAIN MEMORY

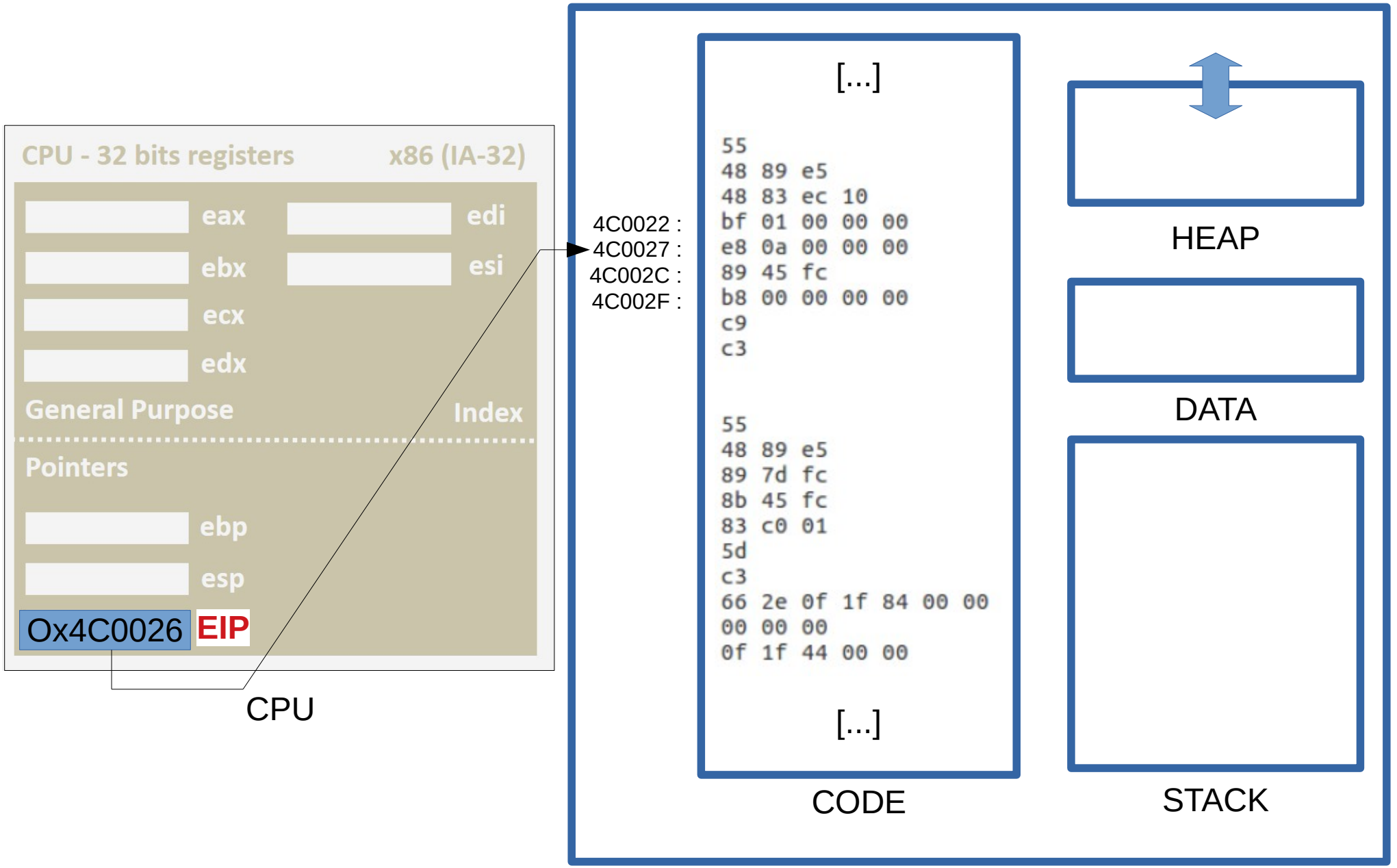
Programme, mémoire et pile



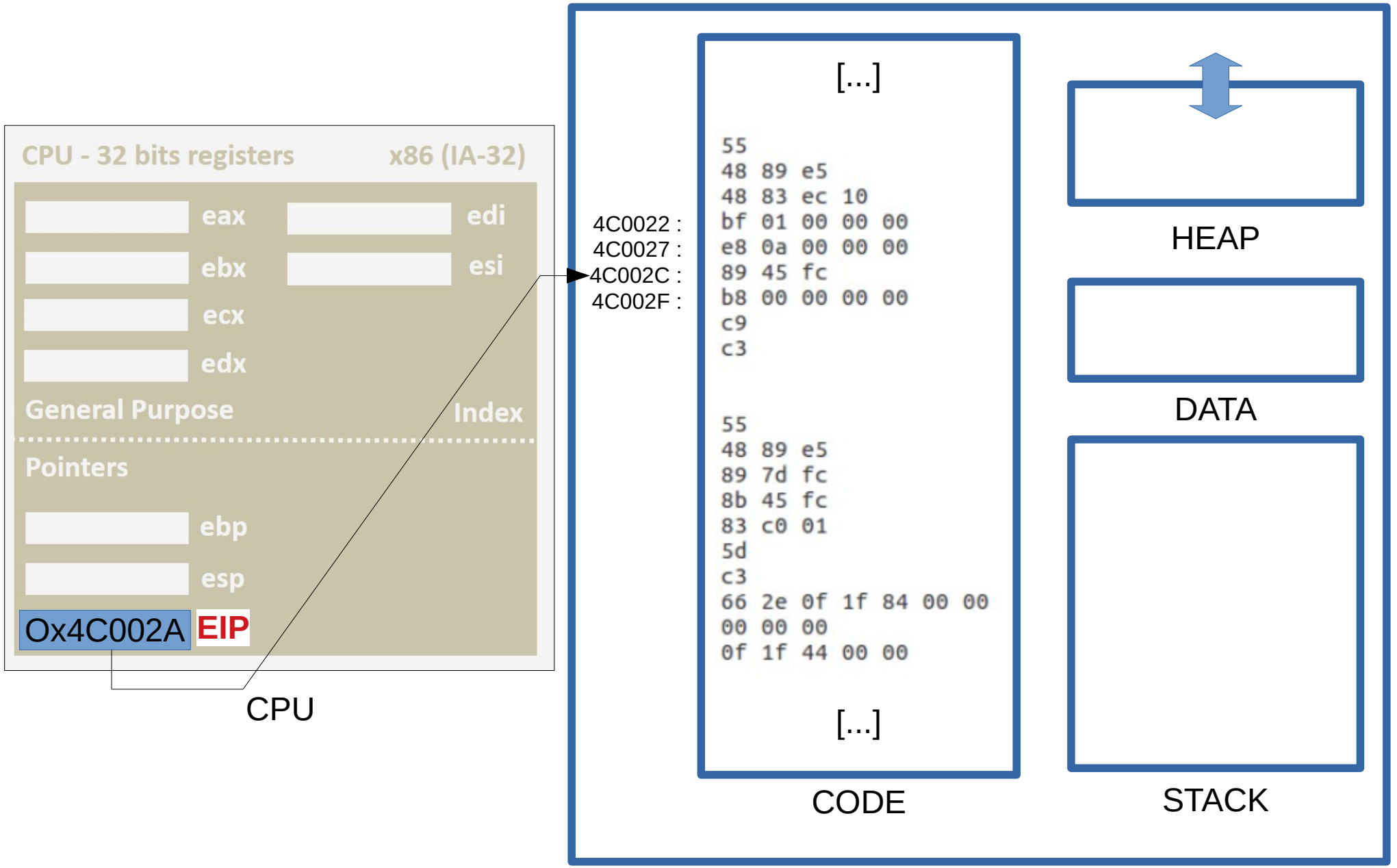
Programme, mémoire et pile



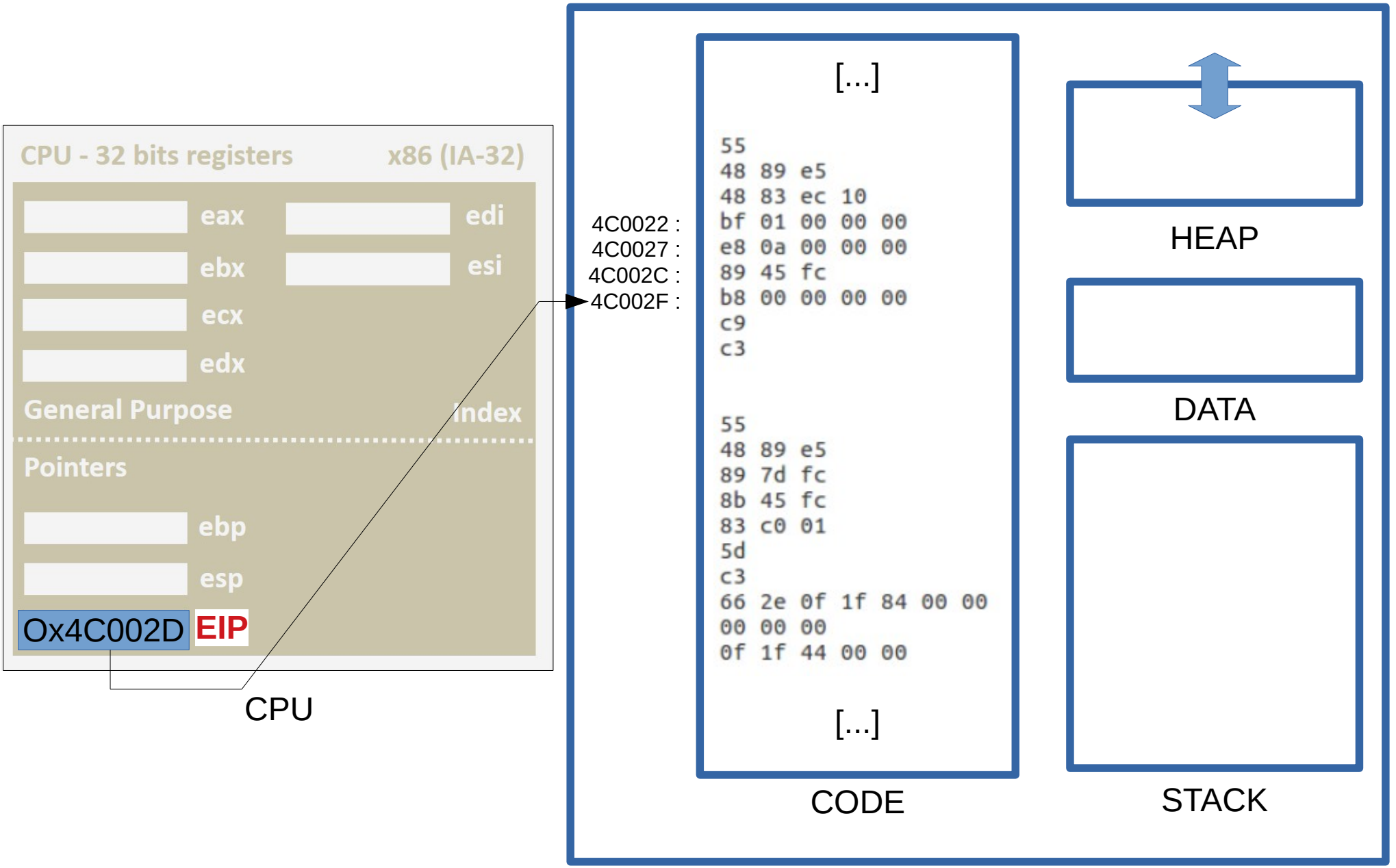
Programme, mémoire et pile



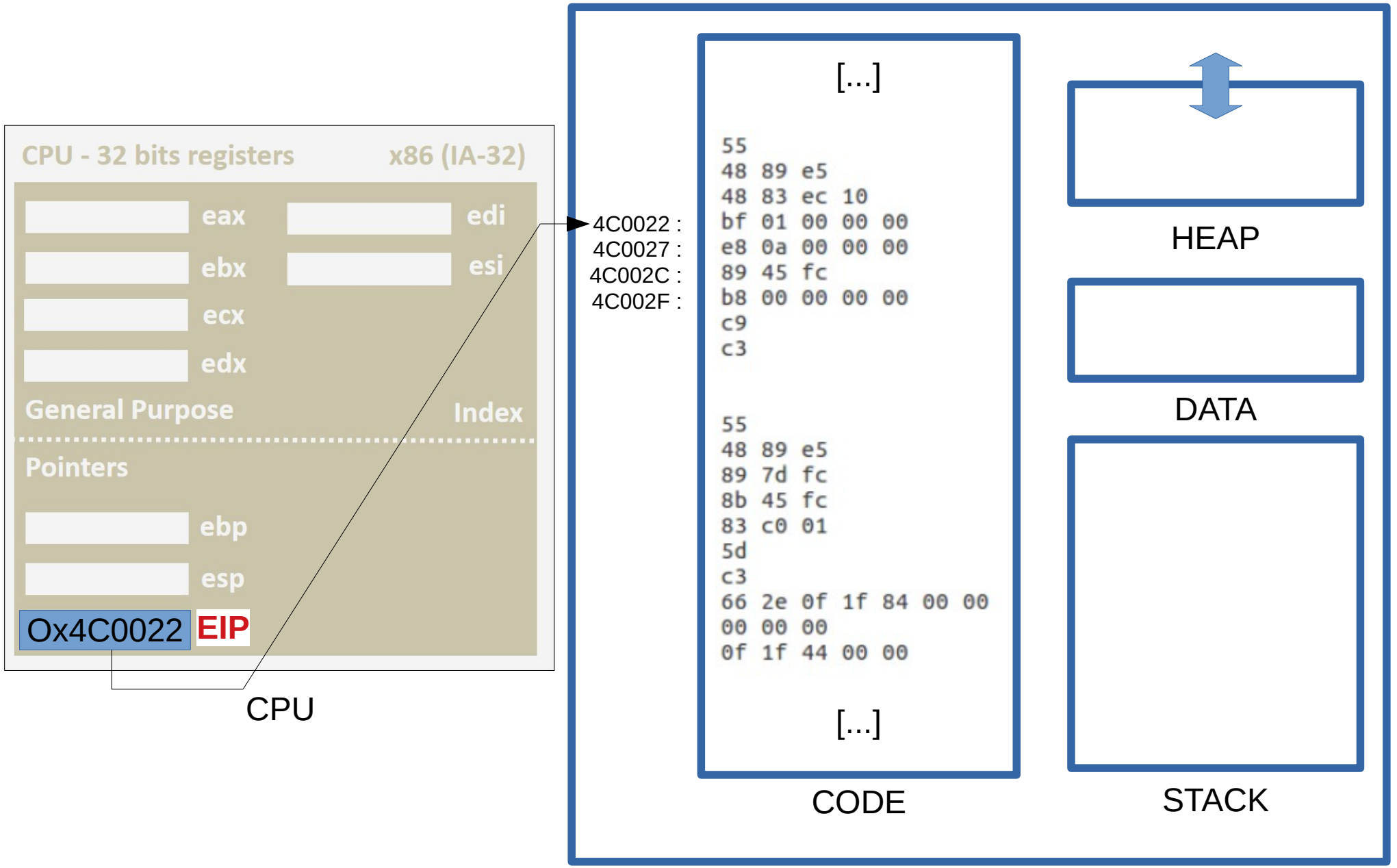
Programme, mémoire et pile



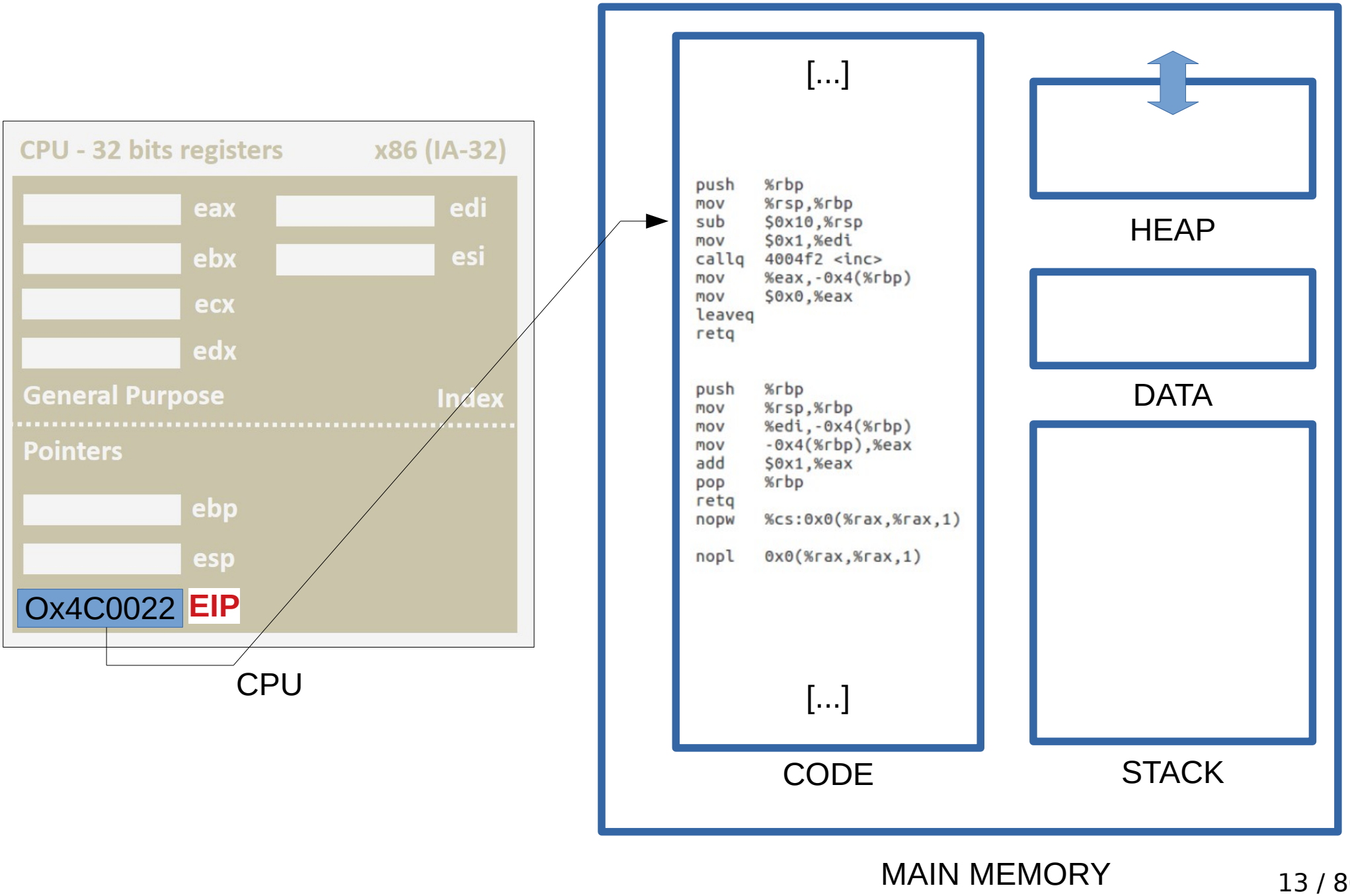
Programme, mémoire et pile



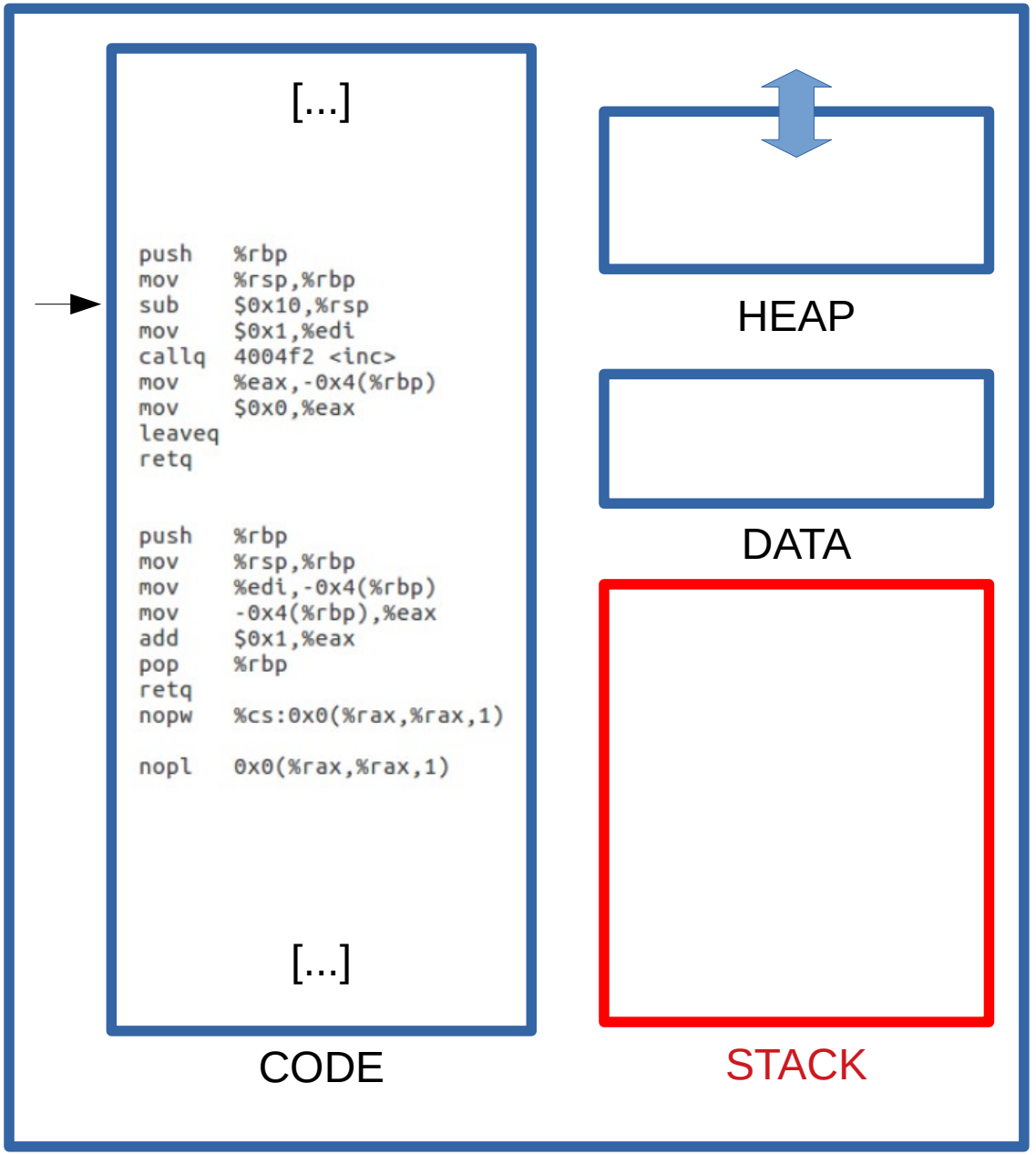
Programme, mémoire et pile

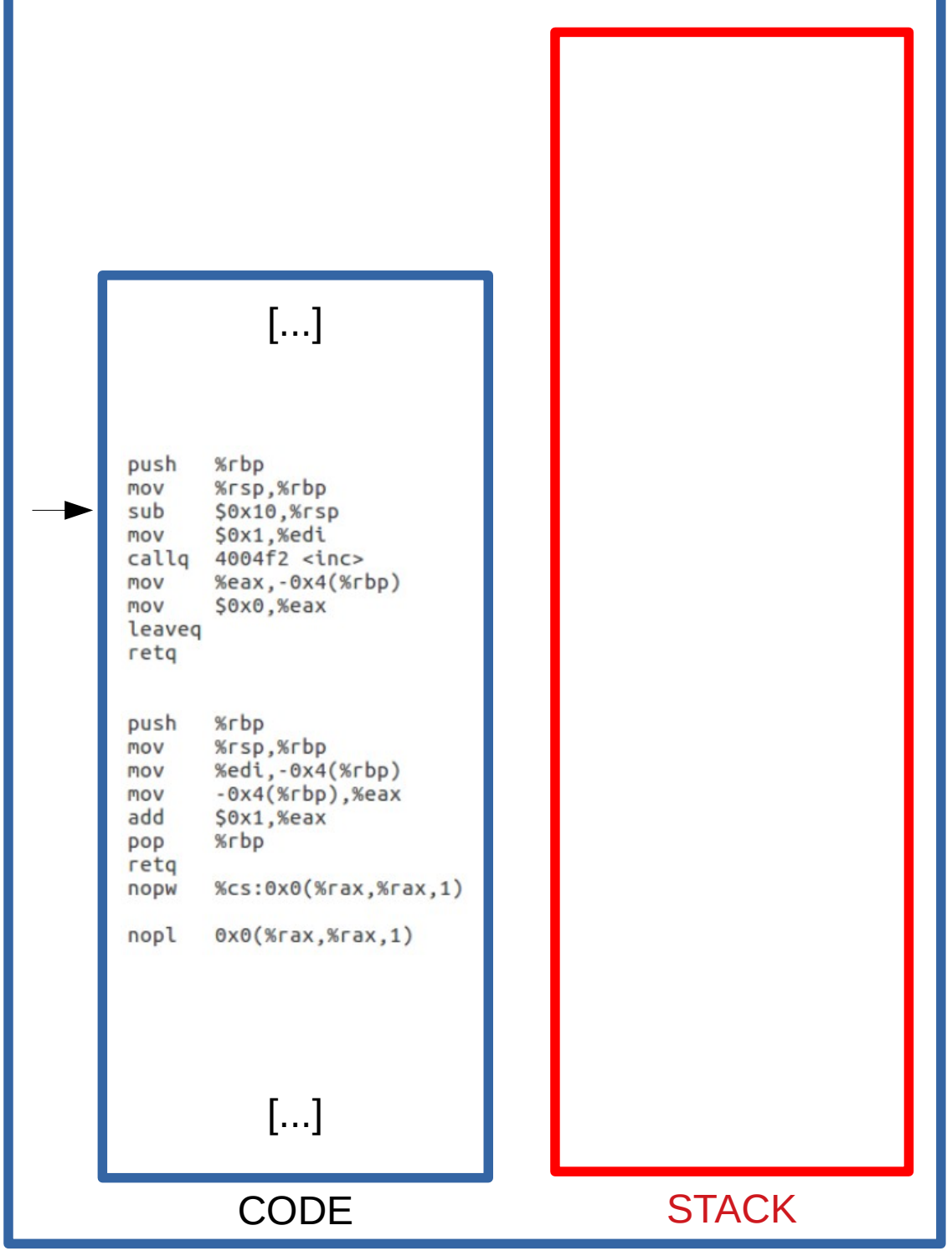


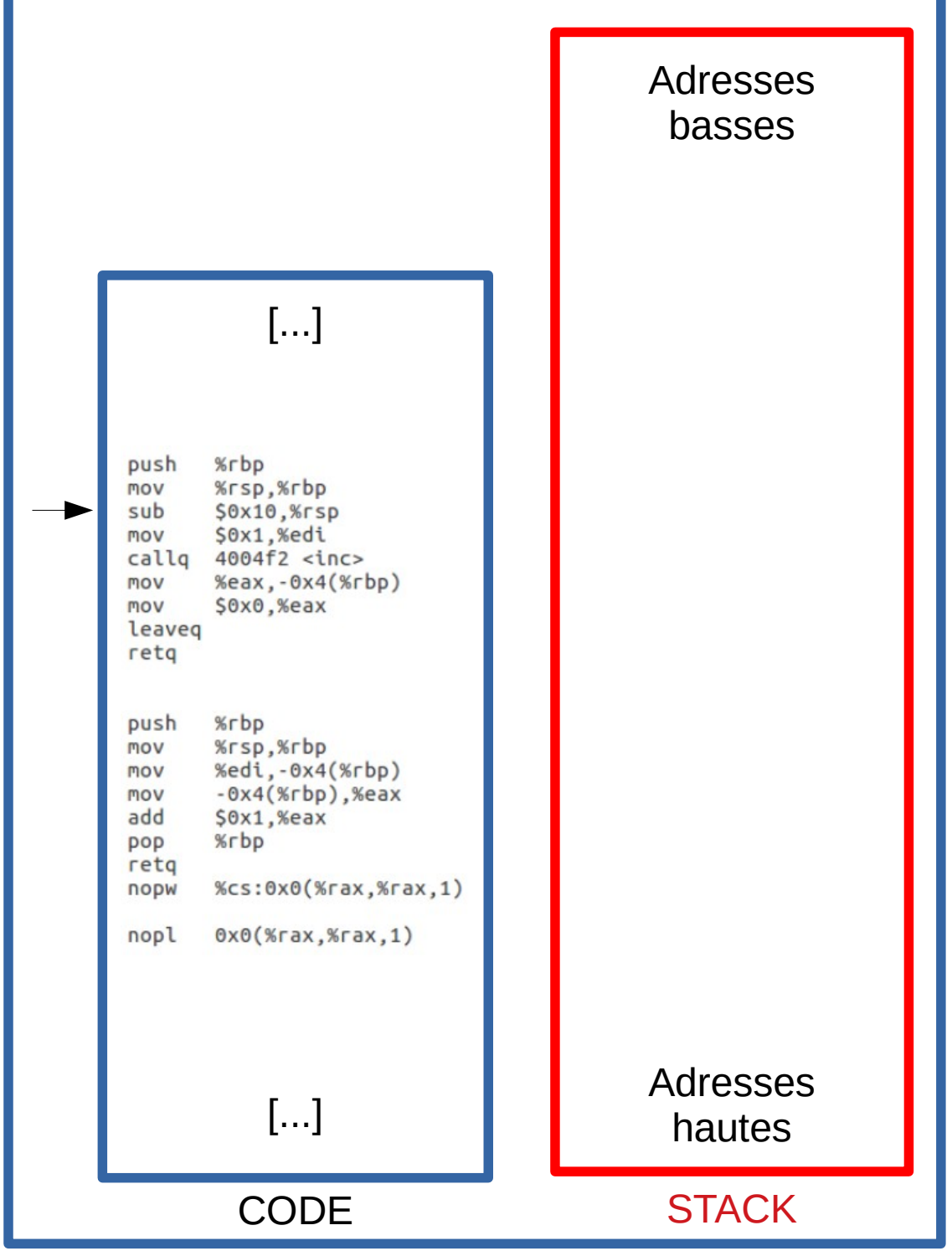
Programme, mémoire et pile



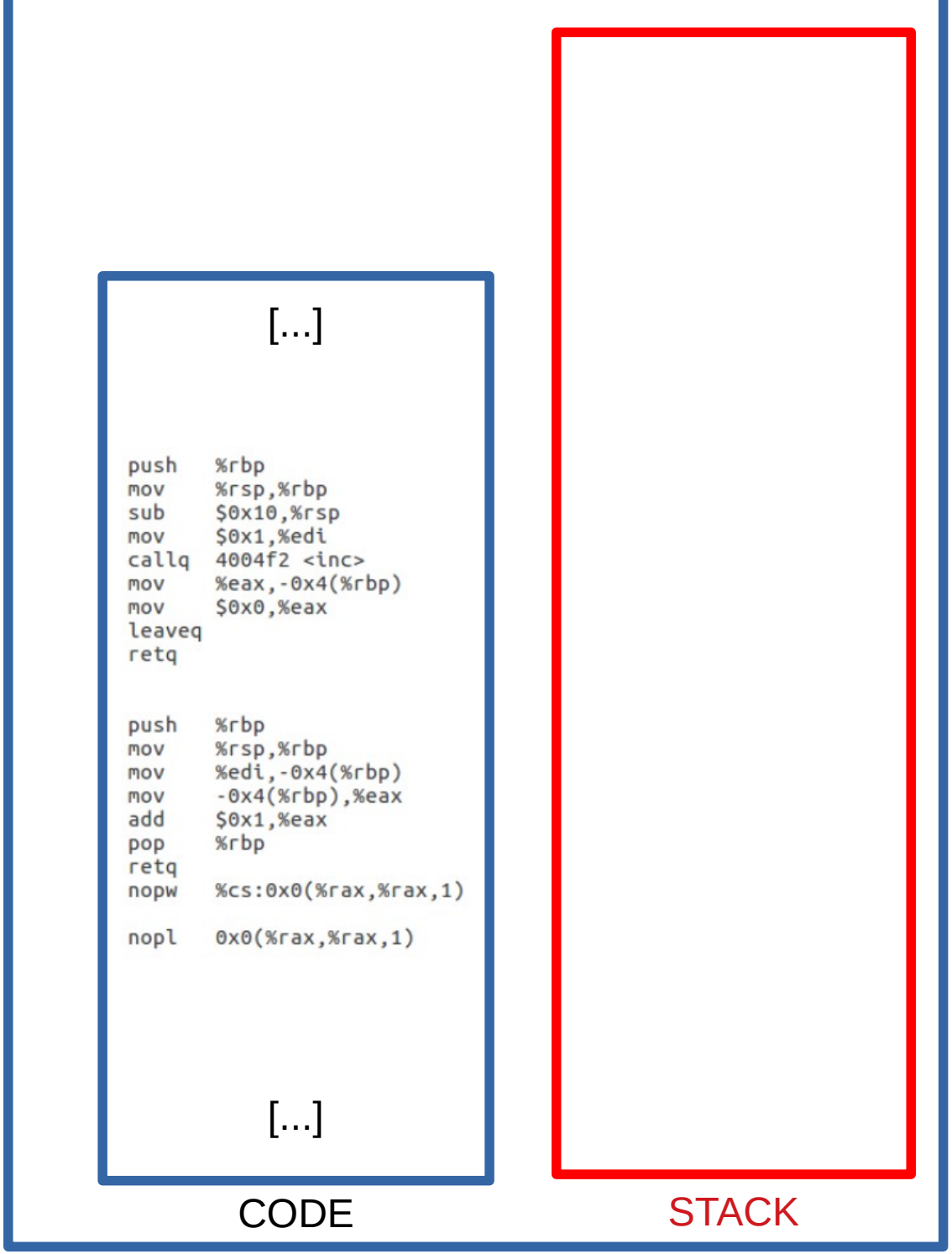
La Pile (STACK)



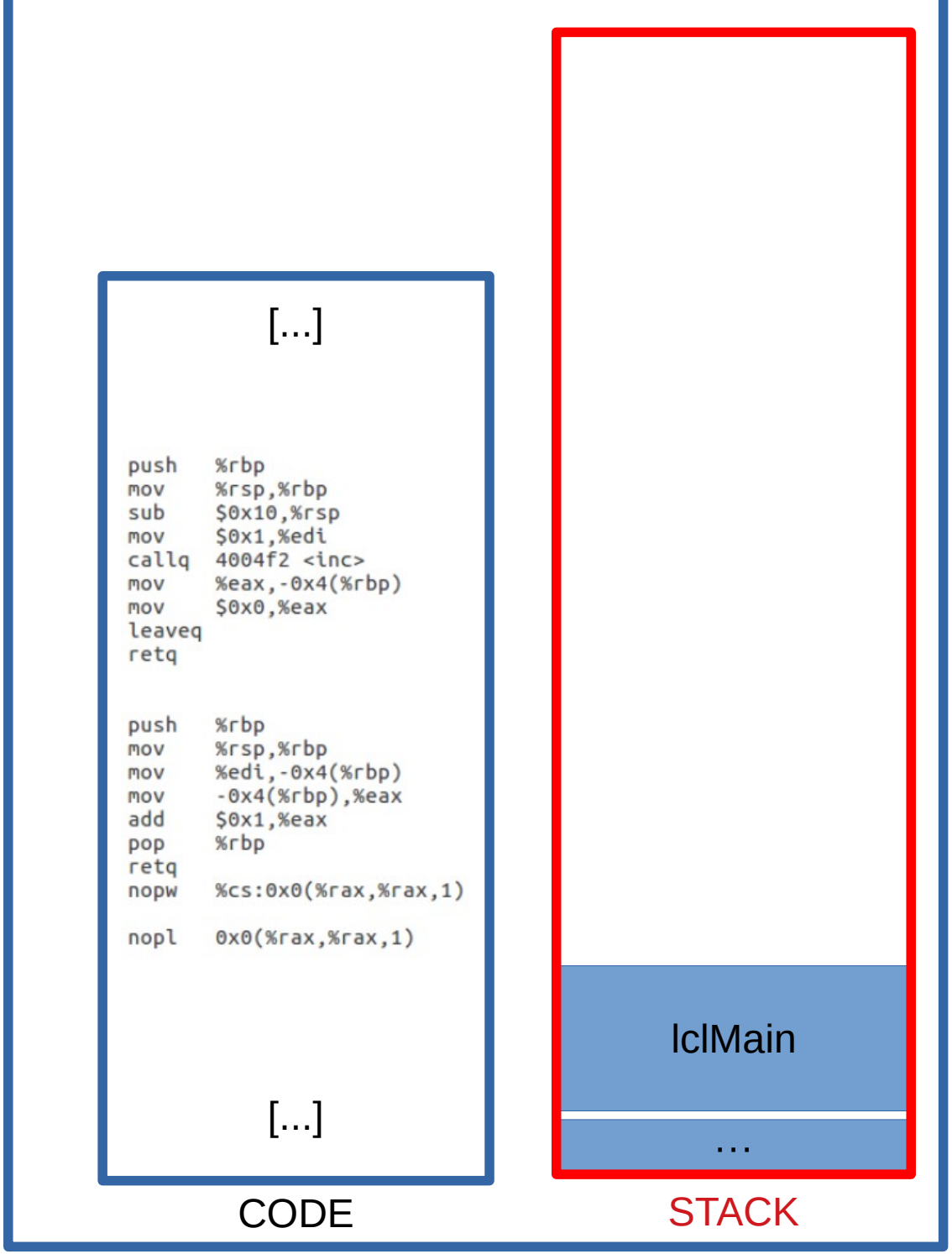

[...]
push %rbp
mov %rsp,%rbp
sub $0x10,%rsp
mov $0x1,%edi
callq 4004f2 <inc>
mov %eax,-0x4(%rbp)
mov $0x0,%eax
leaveq
retq
push %rbp
mov %rsp,%rbp
mov %edi,-0x4(%rbp)
mov -0x4(%rbp),%eax
add $0x1,%eax
pop %rbp
retq
nopw %cs:0x0(%rax,%rax,1)
nopl 0x0(%rax,%rax,1)
[...]
CODE
STACK
MAIN MEMORY


Adresses
basses
[...]
push %rbp
mov %rsp,%rbp
sub $0x10,%rsp
mov $0x1,%edi
callq 4004f2 <inc>
mov %eax,-0x4(%rbp)
mov $0x0,%eax
leaveq
retq
push %rbp
mov %rsp,%rbp
mov %edi,-0x4(%rbp)
mov -0x4(%rbp),%eax
add $0x1,%eax
pop %rbp
retq
nopw %cs:0x0(%rax,%rax,1)
nopl 0x0(%rax,%rax,1)
[...]
Adresses
hautes
CODE
STACK
MAIN MEMORY

```c
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}

int main() {
  int lclMain = 5;
  bar();
  foo();
}
```

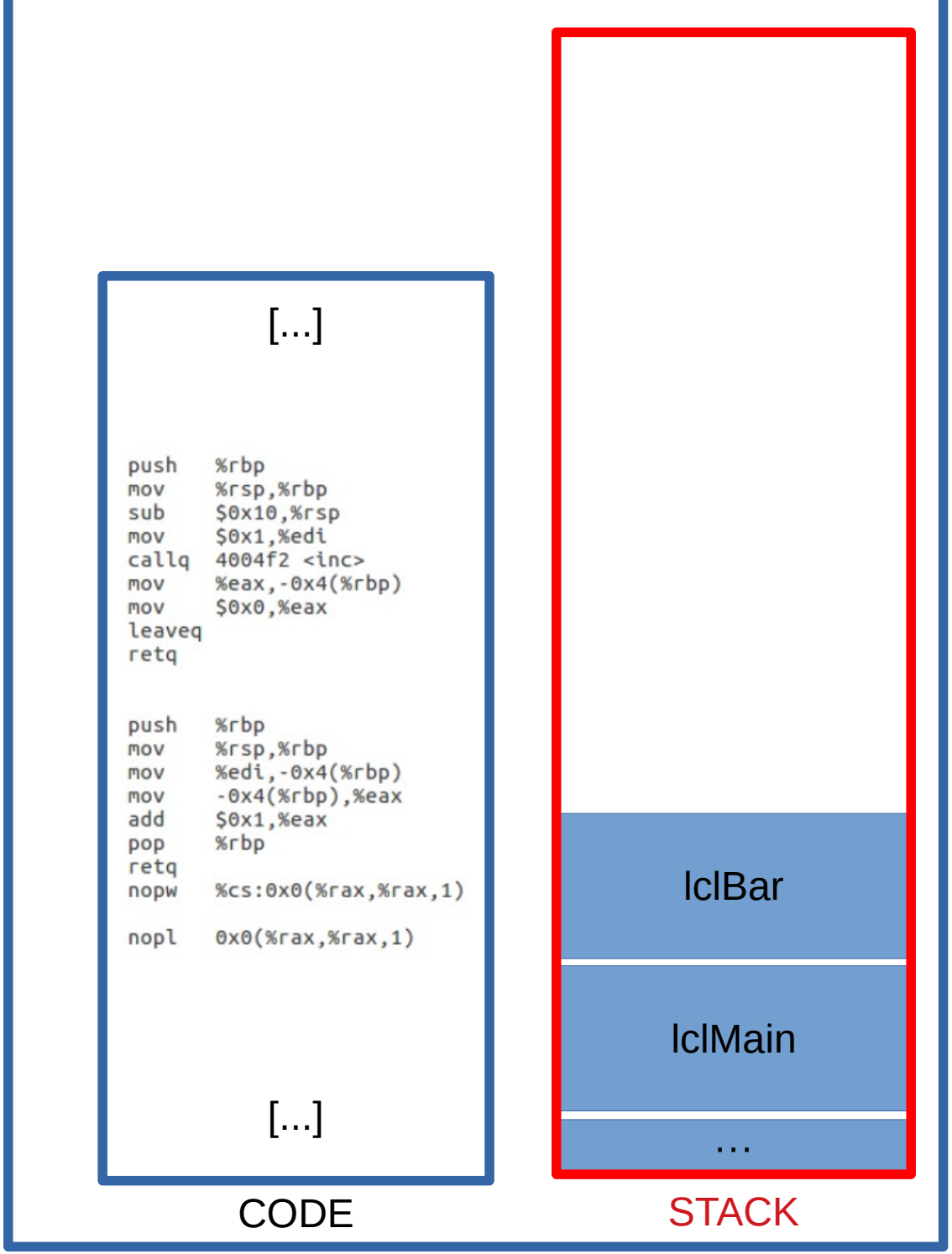


```c
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}

int main() {
  int lclMain = 5;
  bar();
  foo();
}
```



Contexte main()

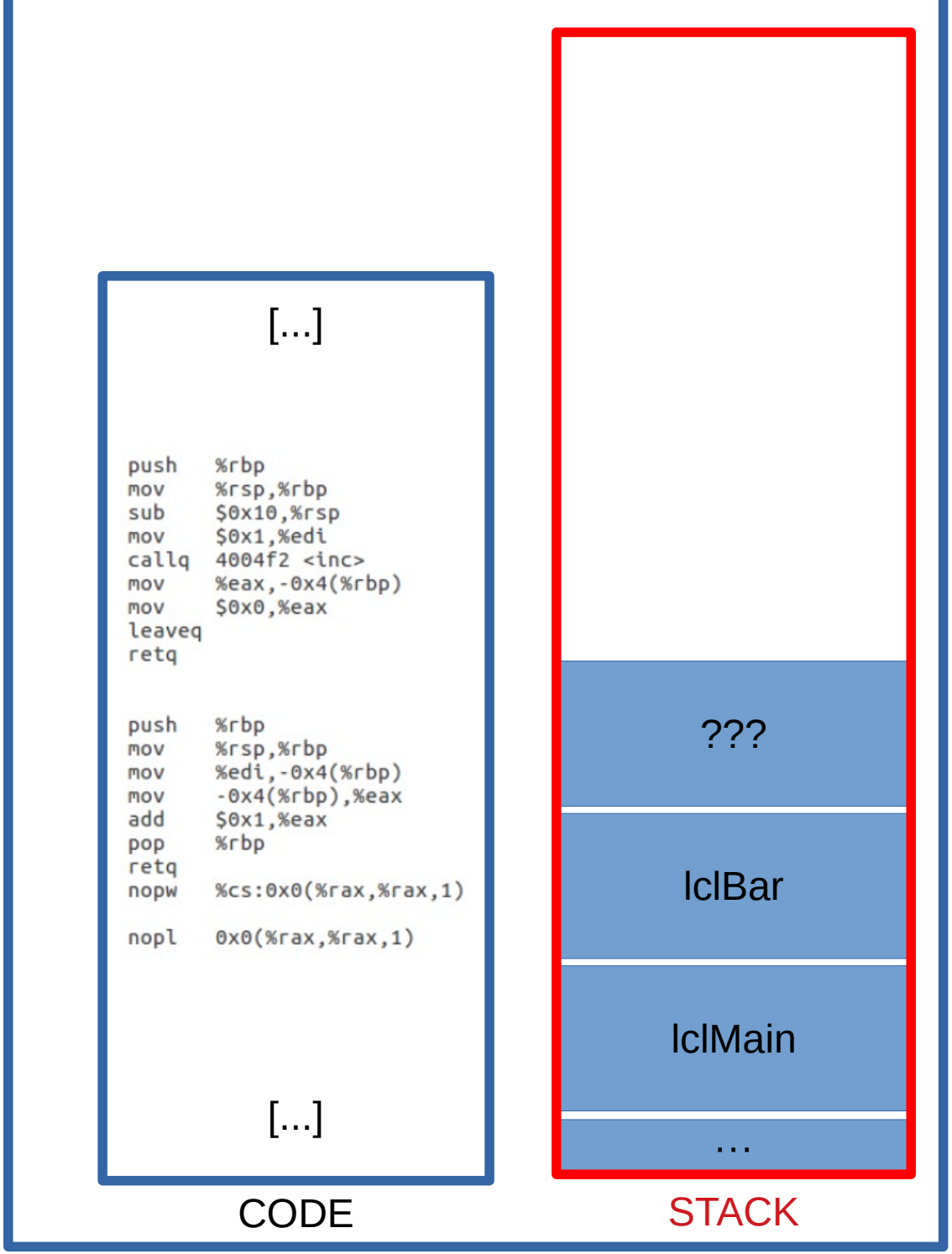
```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}

int main() {
  int lclMain = 5;
  bar();
  foo();
}
```



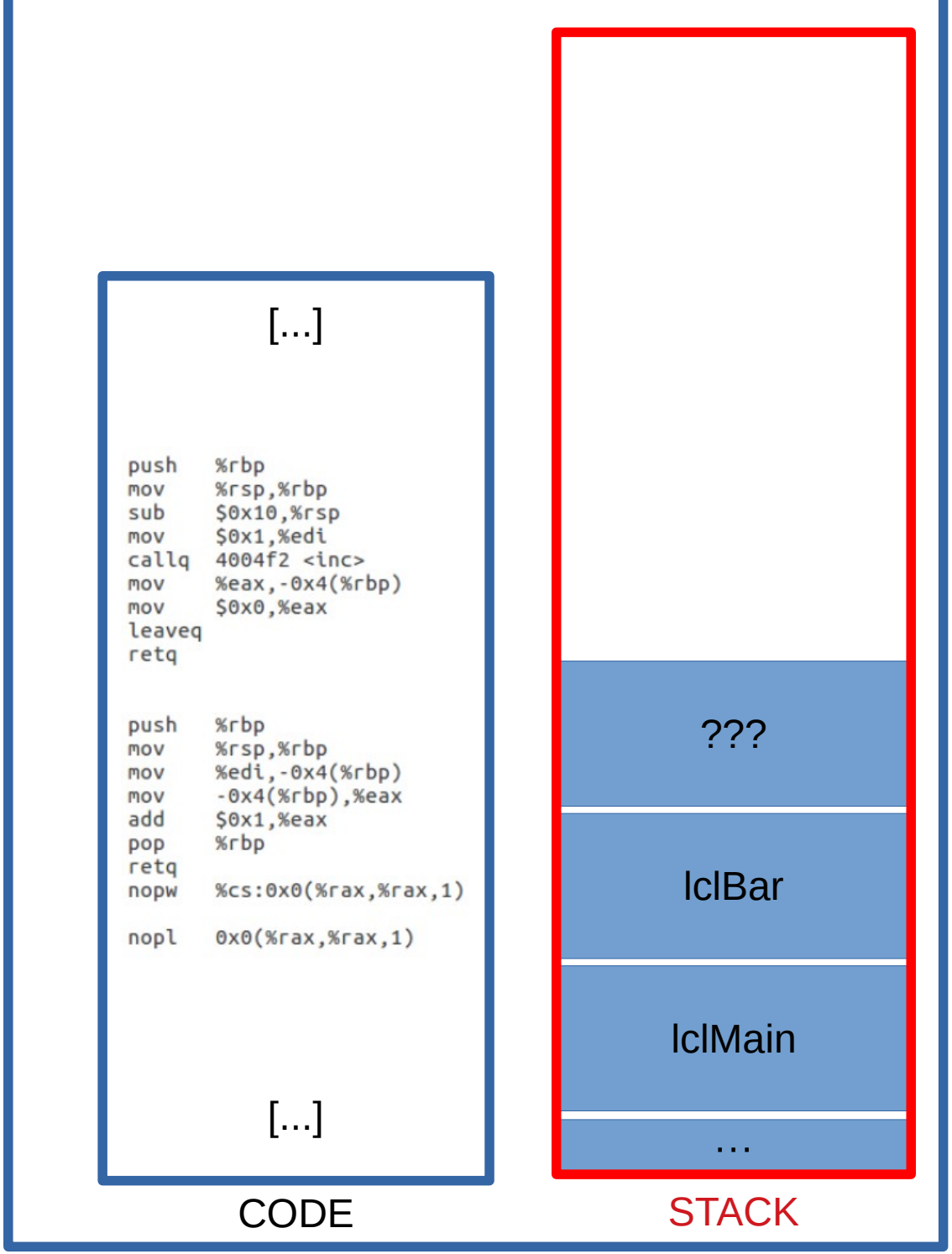
Contexte bar()

Contexte main()

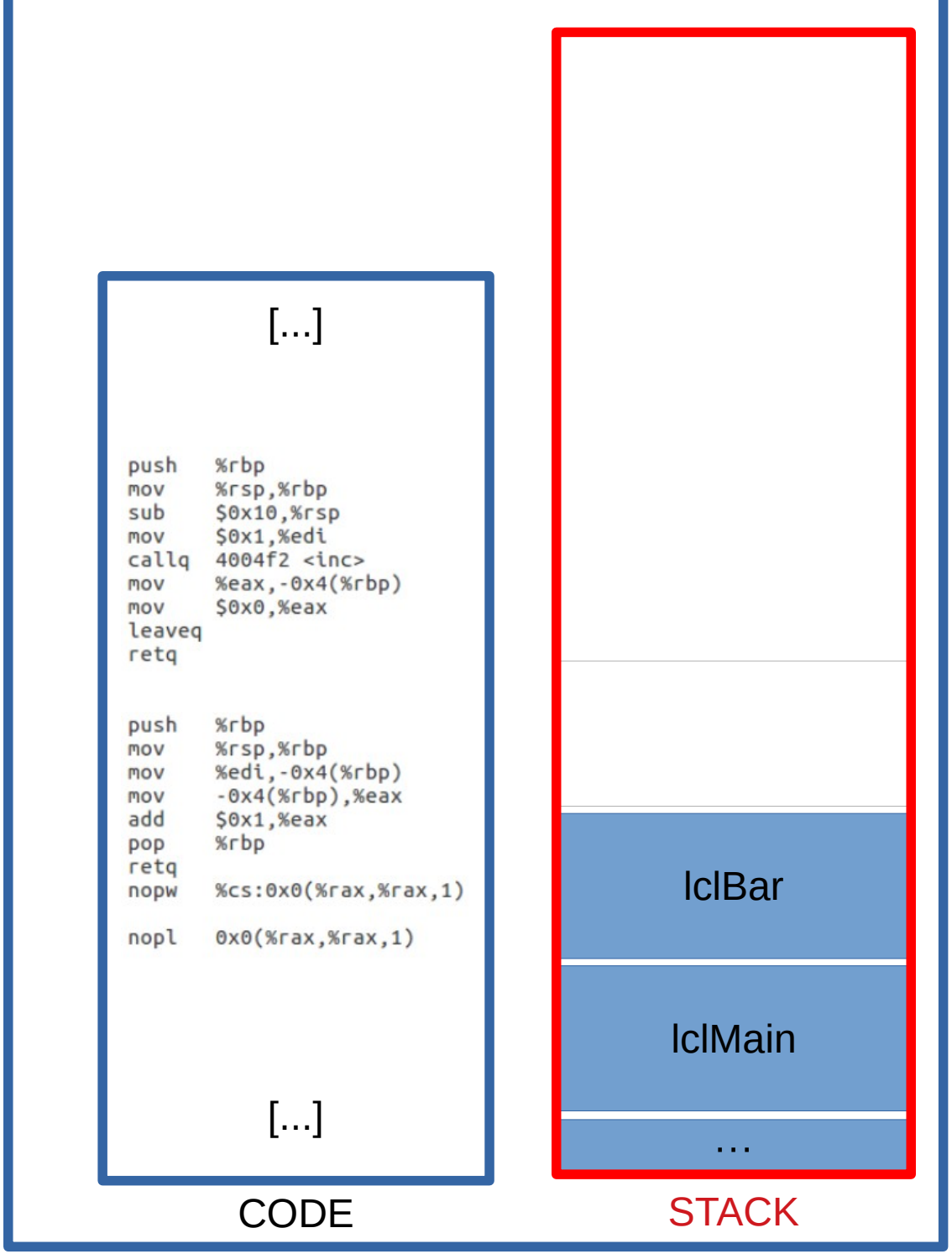
```
int printf()
  {...}

void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}

int main() {
  int lclMain = 5;
  bar();
  foo();
}
```



```
int printf()
{...}

void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}

int main() {
  int lclMain = 5;
  bar();
  foo();
}
```



```
int printf()
{...}

void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}

int main() {
  int lclMain = 5;
  bar();
  foo();
}
```

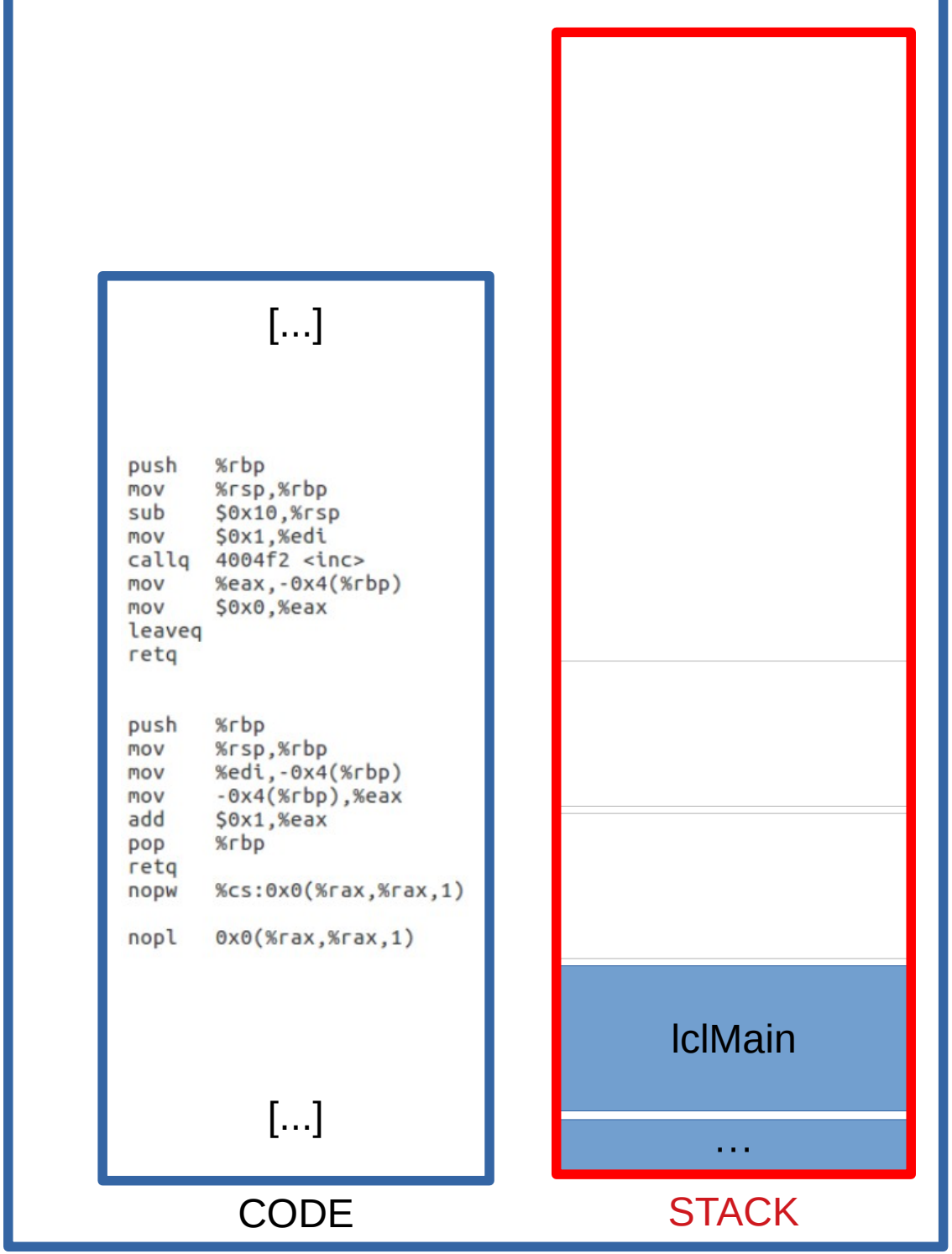


Contexte printf()

Contexte bar()

Contexte main()

```
int printf()
{...}

void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}

int main() {
  int lclMain = 5;
  bar();
  foo();
}
```

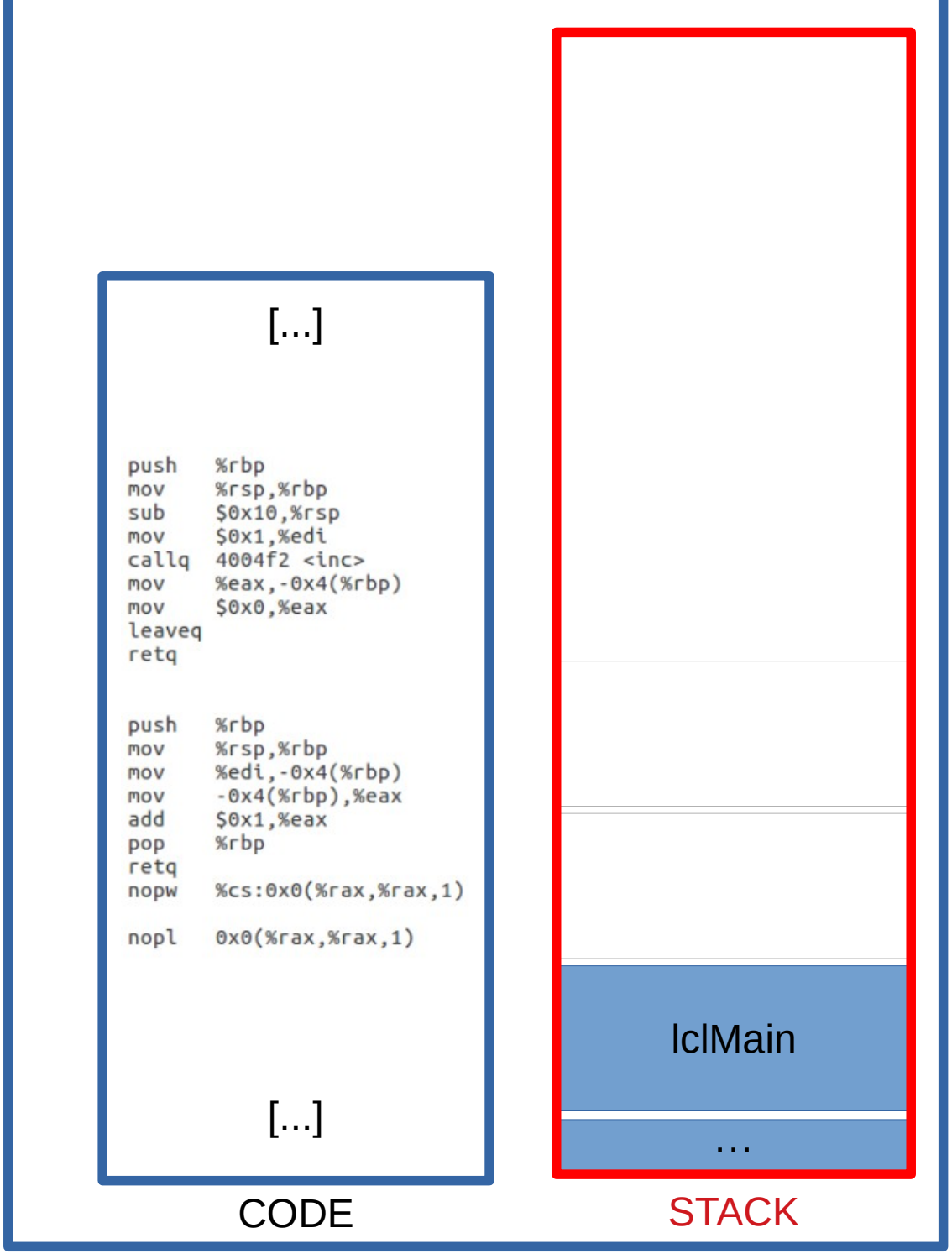


Contexte printf()

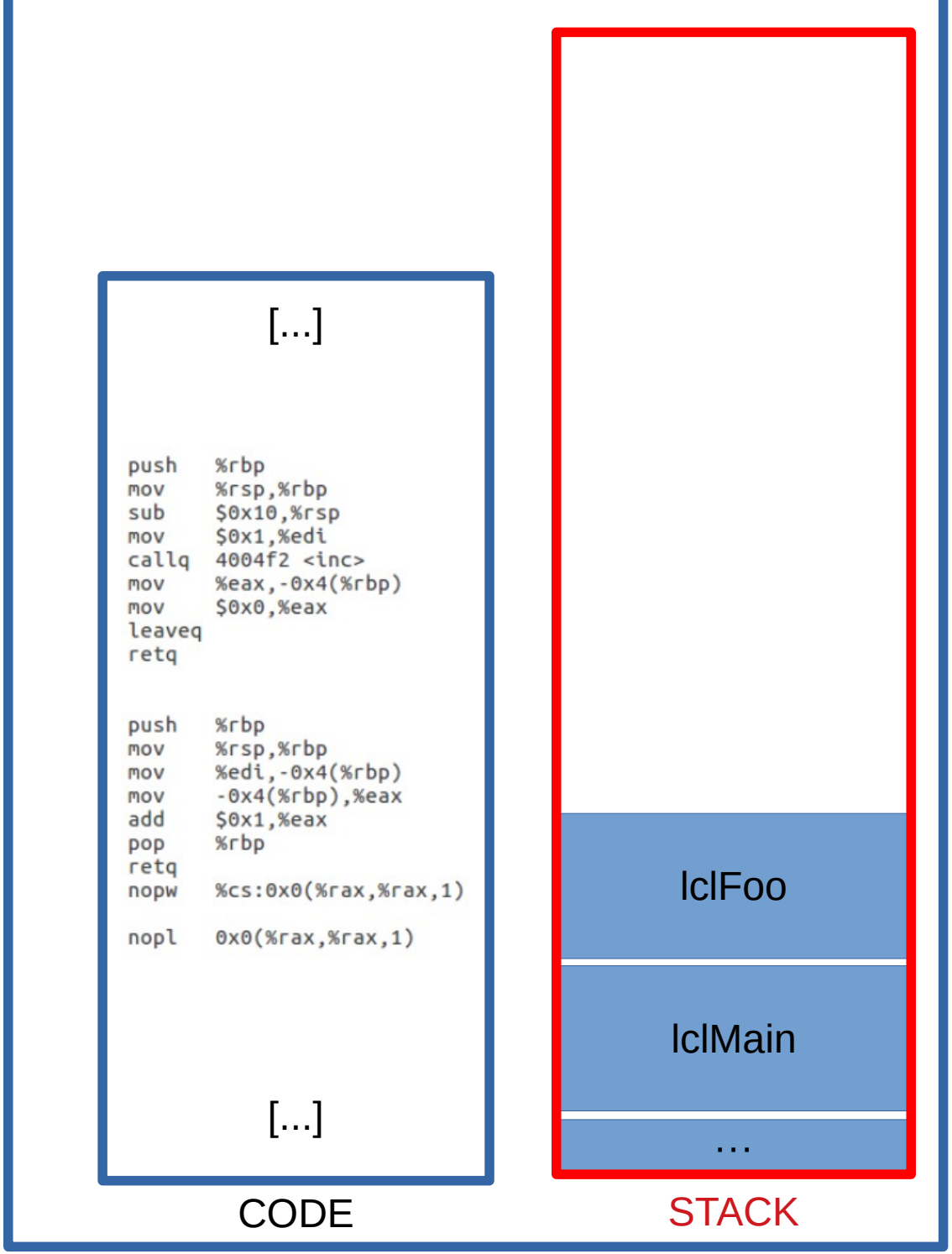
Contexte bar()

Contexte main()

```
   int printf()
   {...}

void foo() {
   int lclFoo = 10;
   bar();
}

void bar() {
   int lclBar = 20;
   printf("OK\n");
}

int main() {
   int lclMain = 5;
   bar();
   foo();
}
```



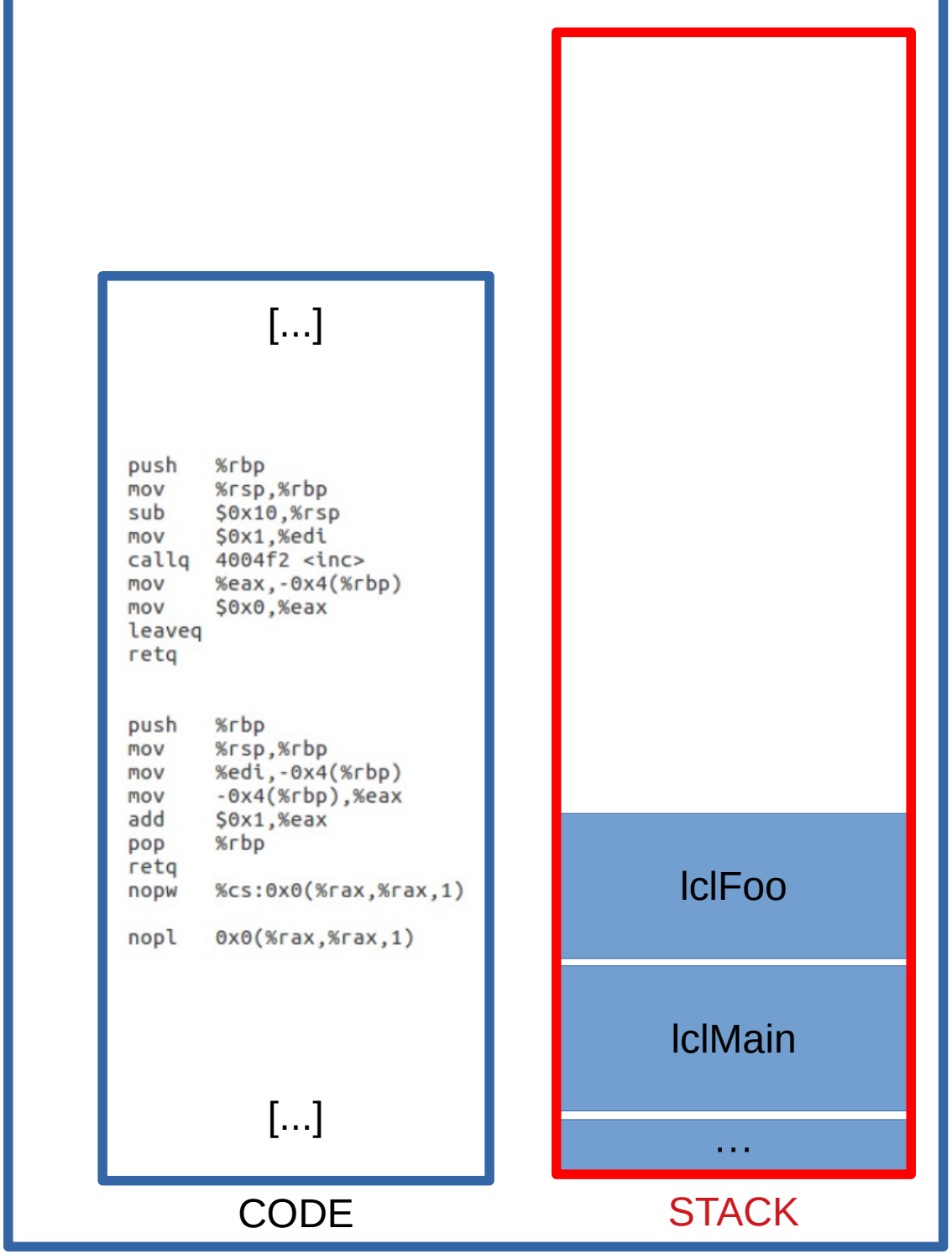
```
   int printf()
   {...}

void foo() {
   int lclFoo = 10;
   bar();
}

void bar() {
   int lclBar = 20;
   printf("OK\n");
}

int main() {
   int lclMain = 5;
   bar();
   foo();
}
```



Contexte foo()

Contexte main()

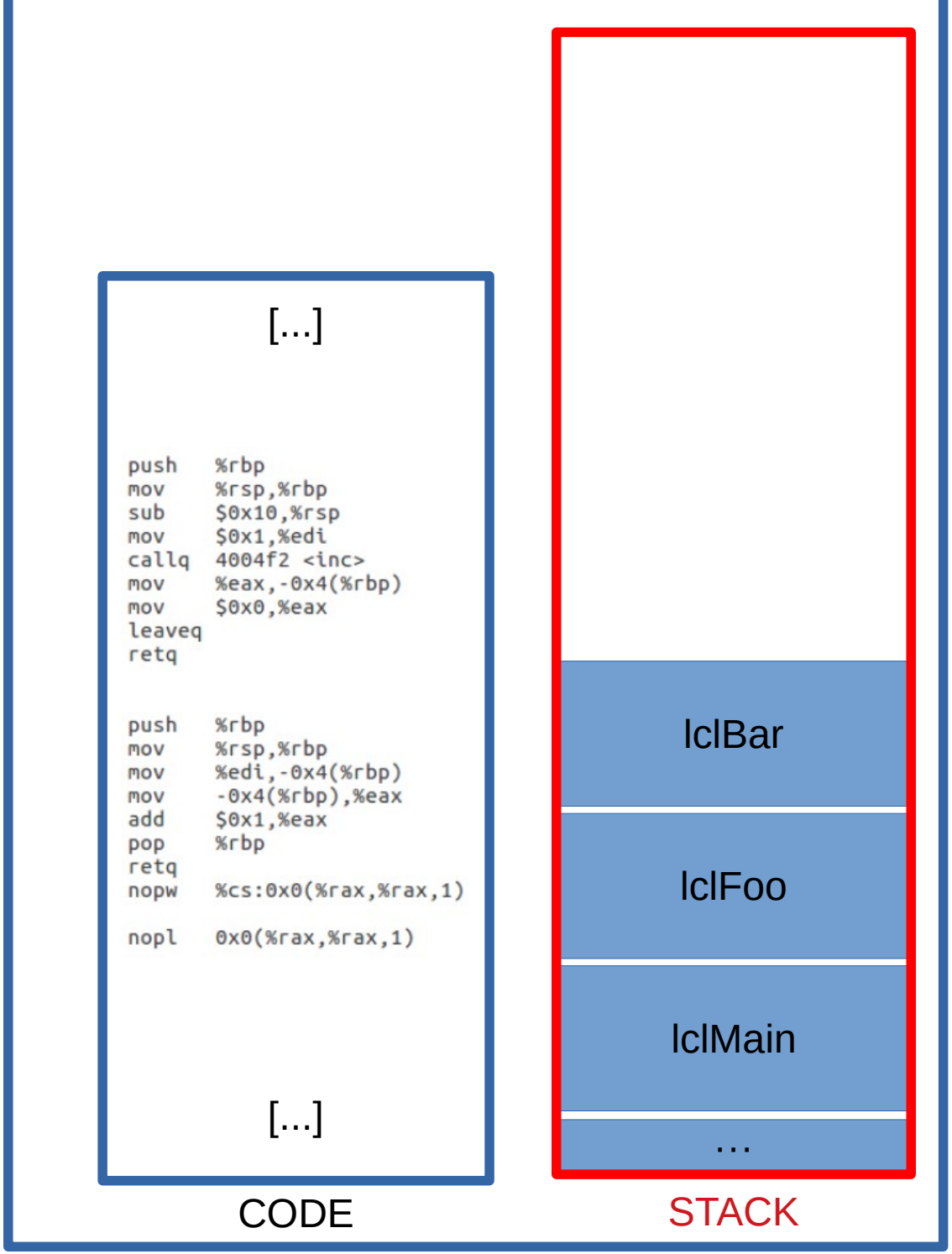
```
   int printf()
   {...}

void foo() {
   int lclFoo = 10;
   bar();
}

void bar() {
   int lclBar = 20;
   printf("OK\n");
}

int main() {
   int lclMain = 5;
   bar();
   foo();
}
```



Contexte foo()

Contexte main()

```
int printf()
{...}

void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}

int main() {
  int lclMain = 5;
  bar();
  foo();
}
```

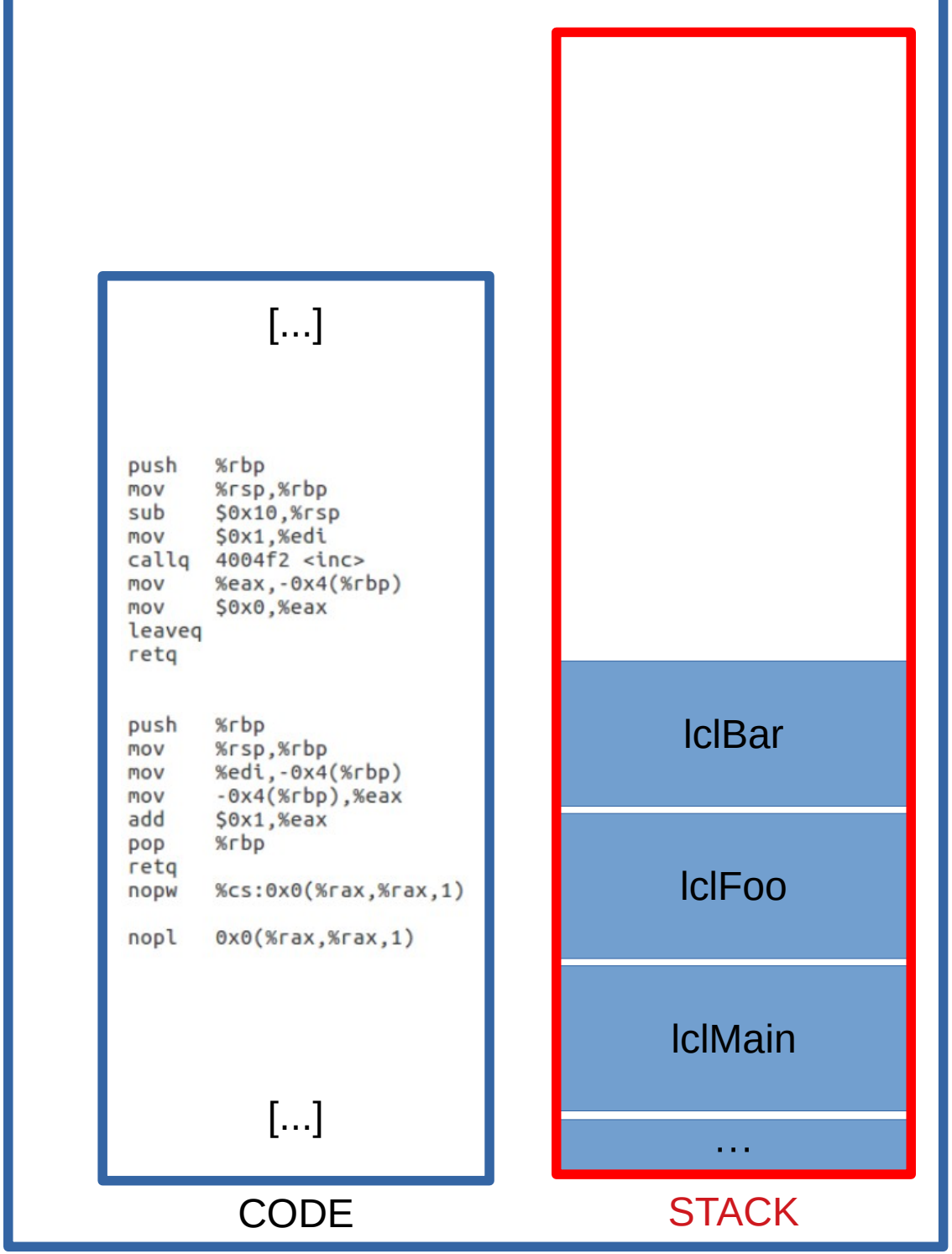


Contexte bar()

Contexte foo()

Contexte main()

```
int printf()
{...}

void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}

int main() {
  int lclMain = 5;
  bar();
  foo();
}
```

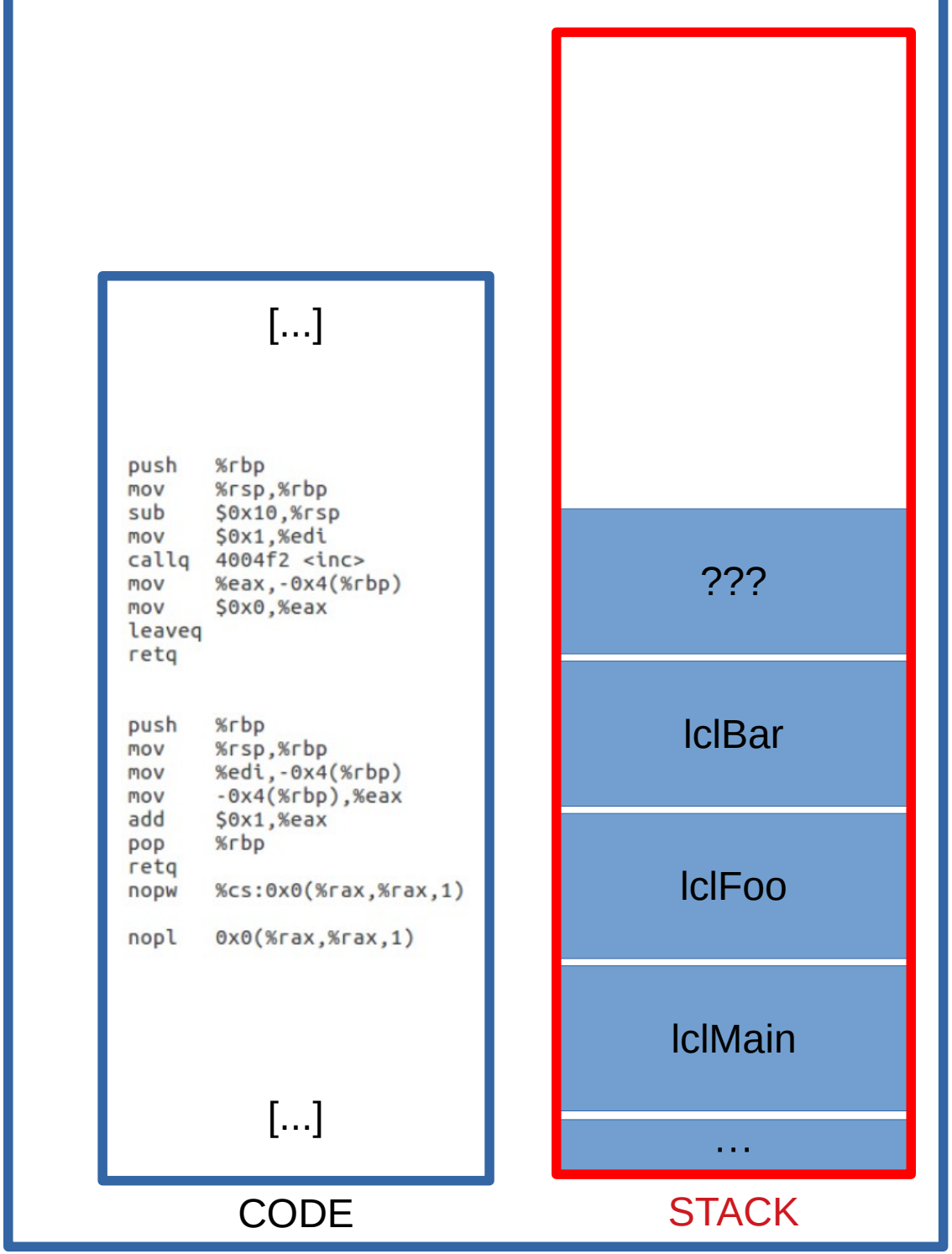


Contexte bar()

Contexte foo()

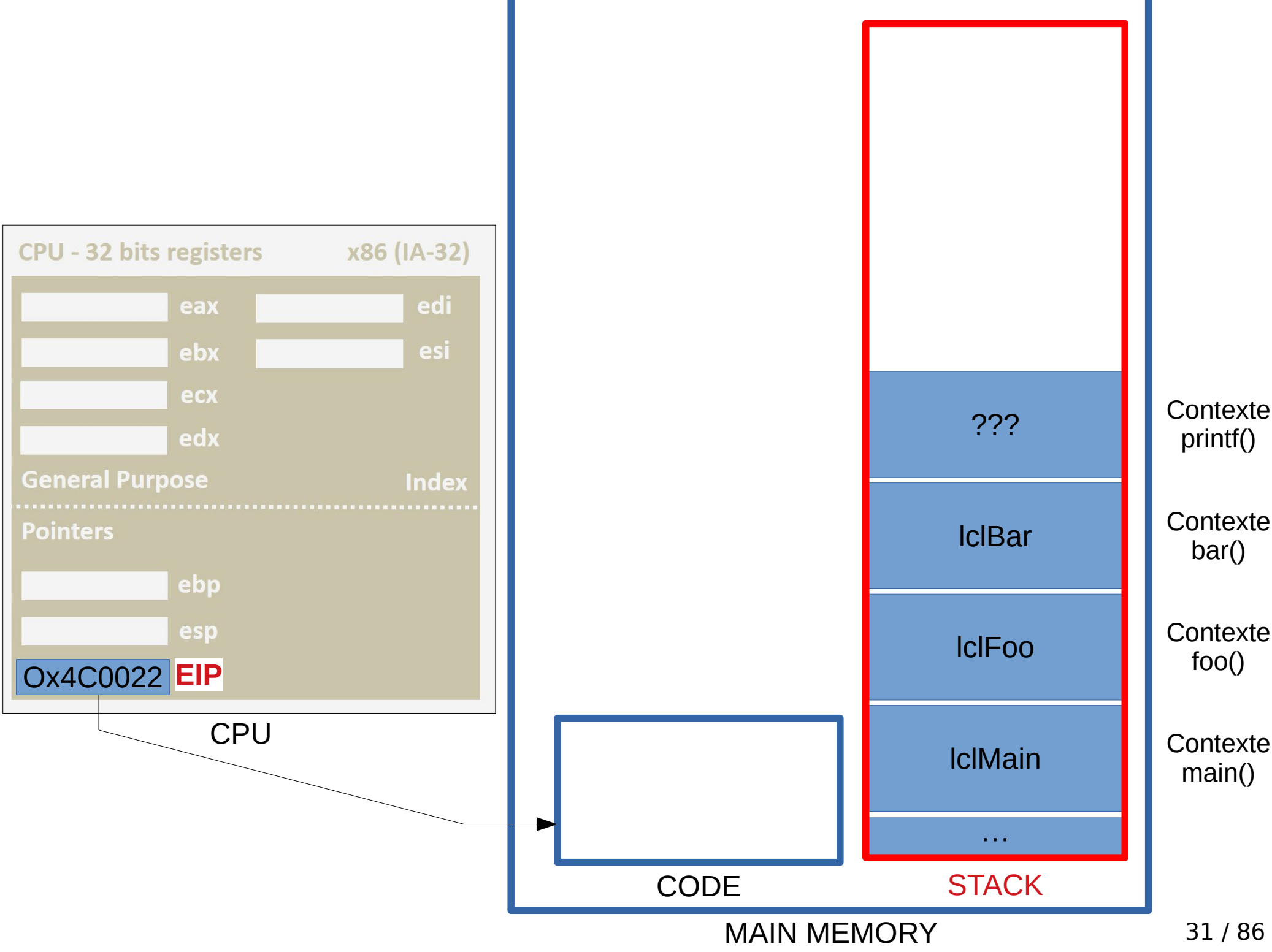
Contexte main()

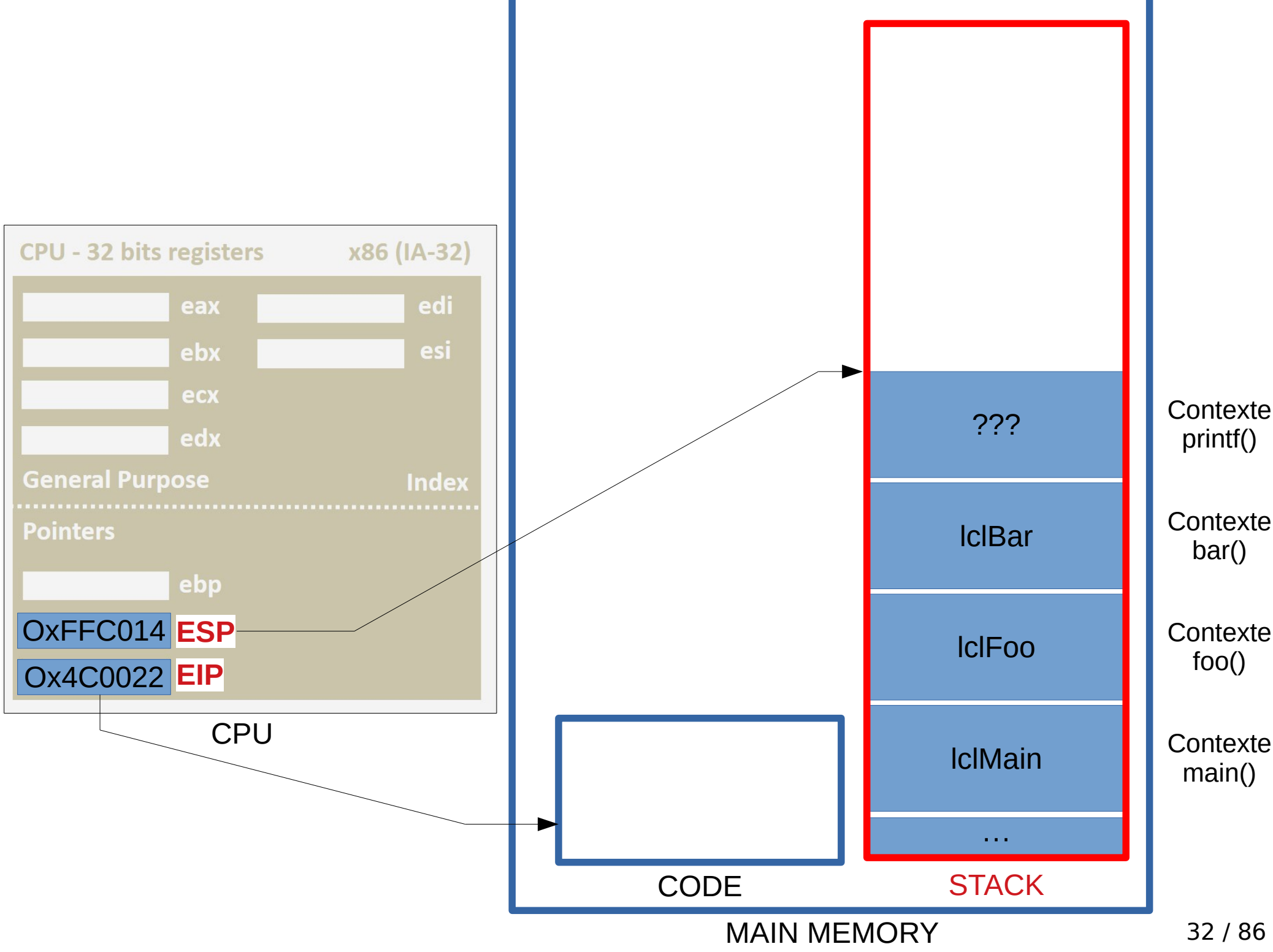
```
   int printf()
   {...}

void foo() {
   int lclFoo = 10;
   bar();
}

void bar() {
   int lclBar = 20;
   printf("OK\n");
}

int main() {
   int lclMain = 5;
   bar();
   foo();
}
```



La Pile : Question 1

- Comment connaît-on le haut de la pile ?
- Réponse : Le pointeur de pile ESP


CPU - 32 bits registers
x86 (IA-32)
eax
edi
ebx
esi
ecx
edx
General Purpose
Index
Pointers
ebp
esp
Ox4C0022
EIP
CPU
???
lclBar
lclFoo
lclMain
...
Contexte printf()
Contexte bar()
Contexte foo()
Contexte main()
CODE
STACK
MAIN MEMORY

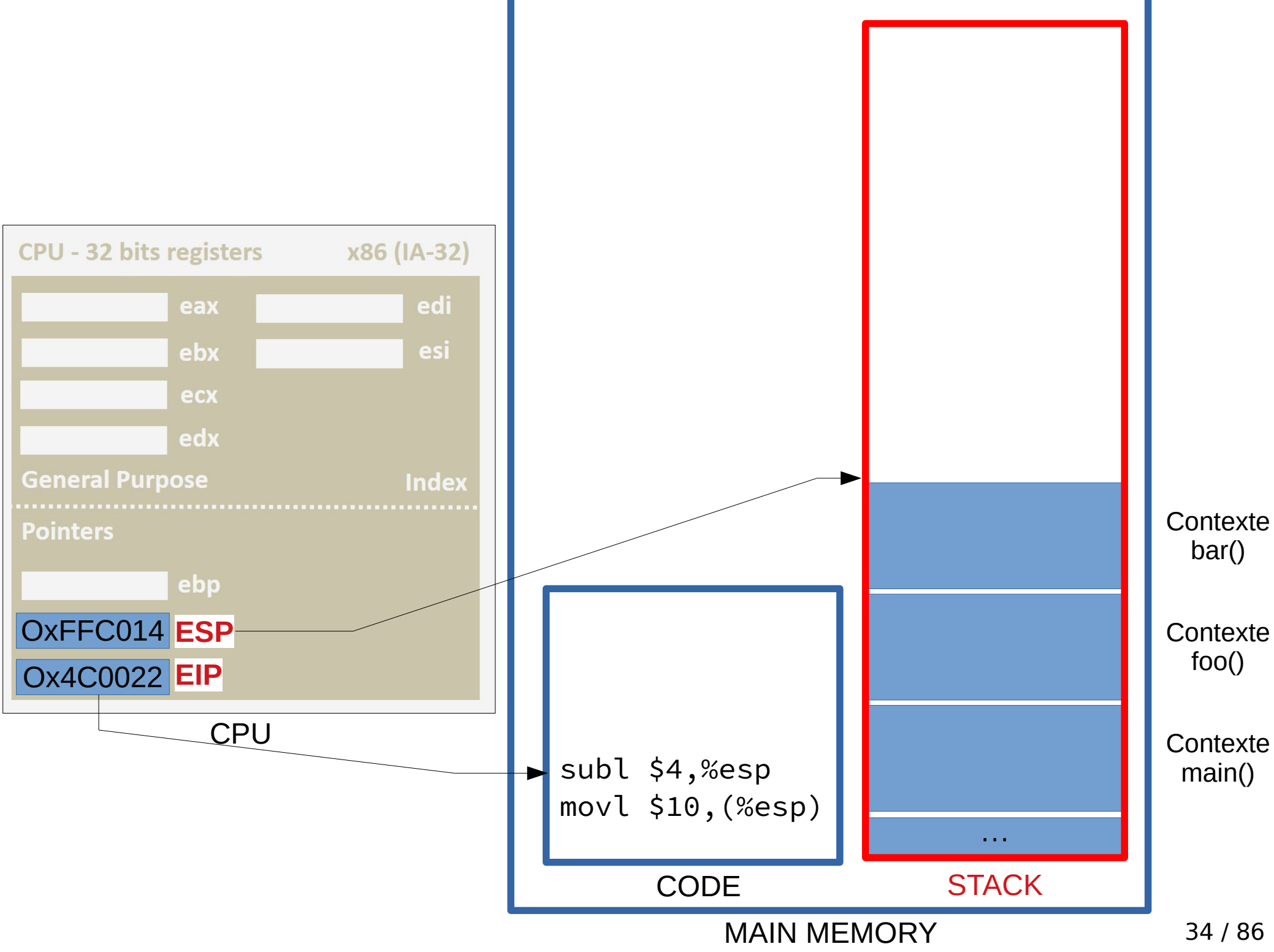

CPU - 32 bits registers
x86 (IA-32)
eax
edi
ebx
esi
ecx
edx
General Purpose
Index
Pointers
ebp
OxFFC014
ESP
Ox4C0022
EIP
CPU
???
lclBar
lclFoo
lclMain
...
Contexte printf()
Contexte bar()
Contexte foo()
Contexte main()
CODE
STACK
MAIN MEMORY

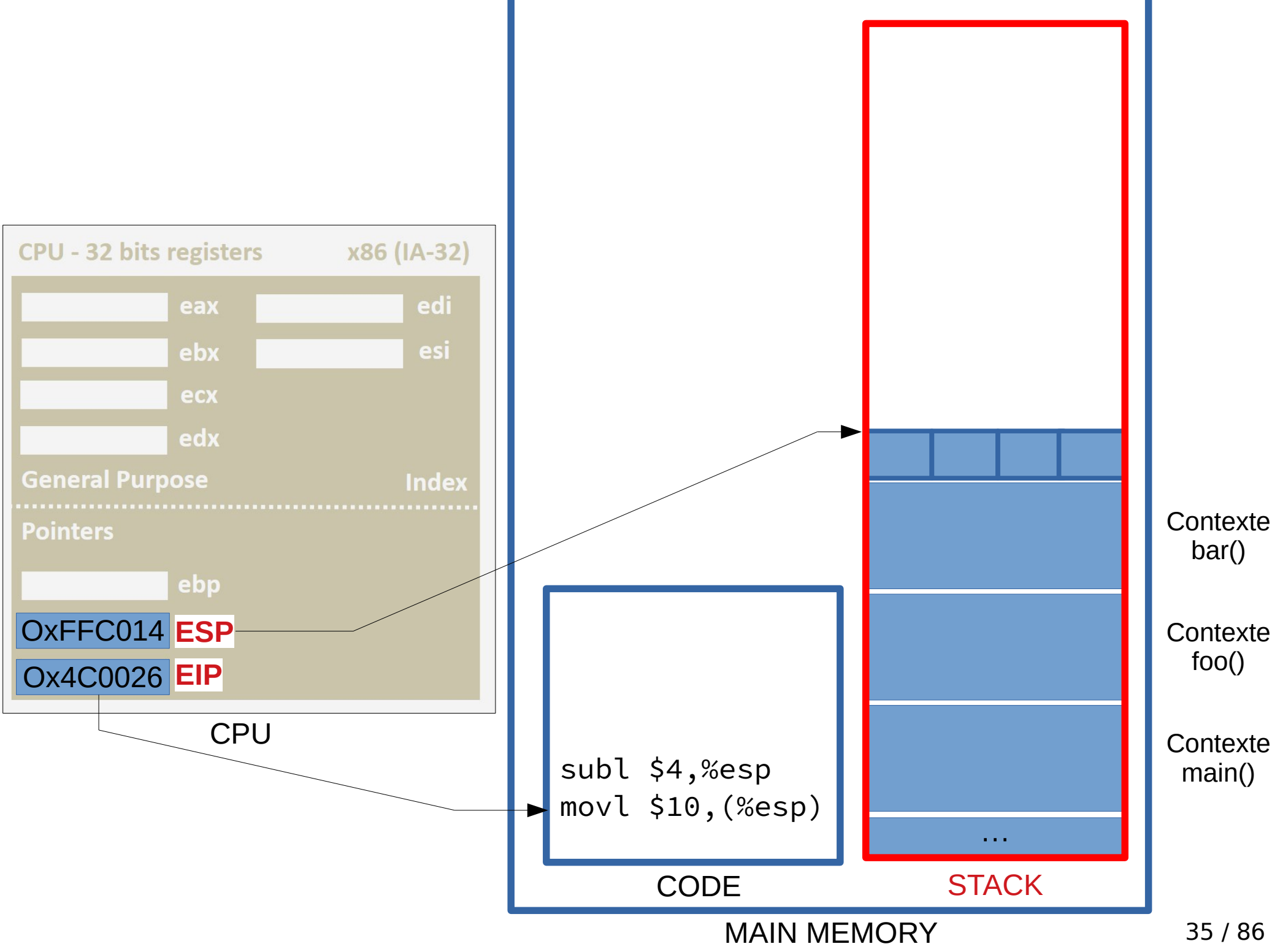
La Pile : Question 1

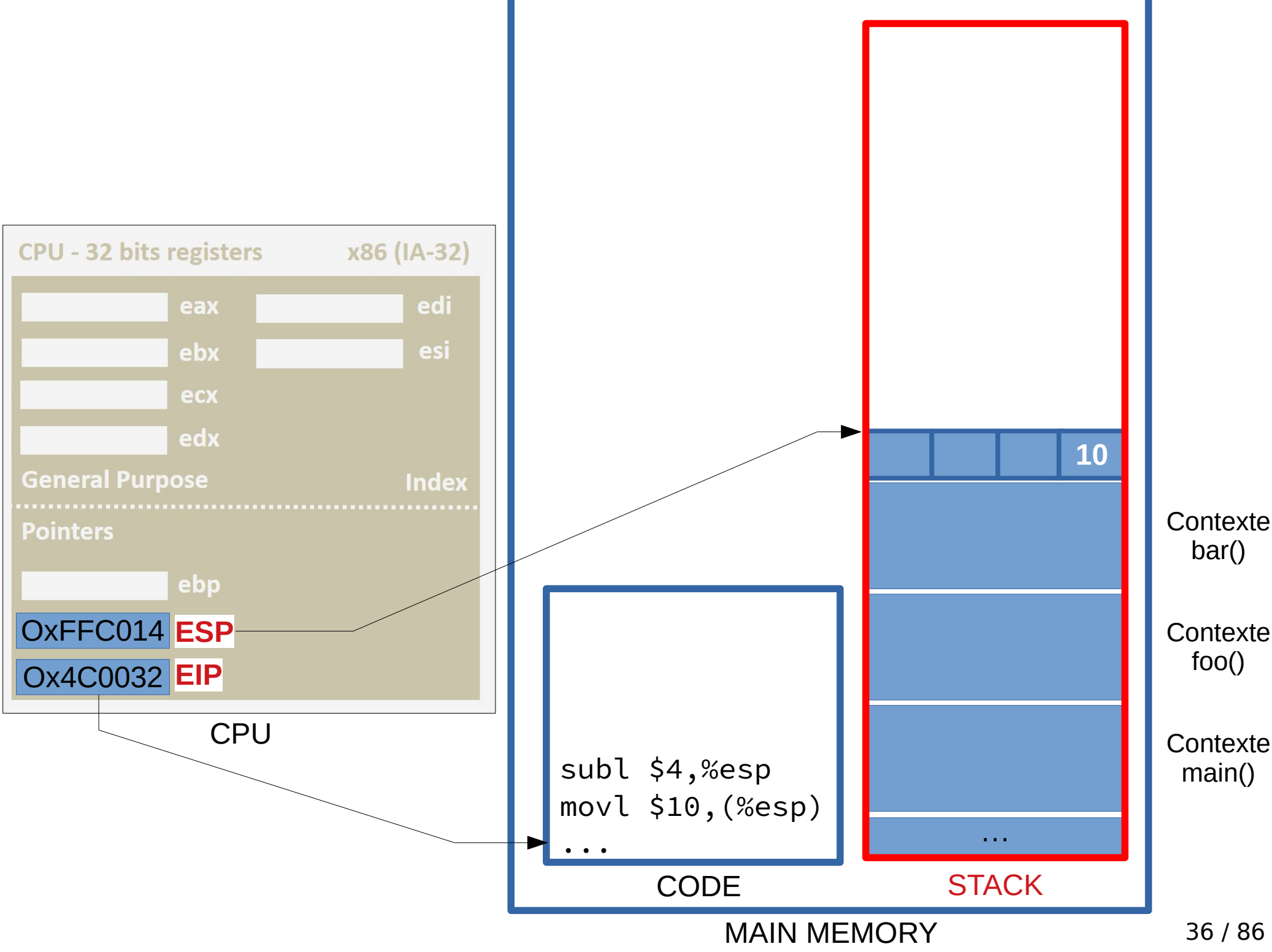
- Instructions typiques avec ESP :

- 1 : décrémenter le pointeur de pile

  → Réserver de l’espace (variables locales)


CPU - 32 bits registers
x86 (IA-32)
eax
edi
ebx
esi
ecx
edx
General Purpose
Index
Pointers
ebp
OxFFC014
ESP
Ox4C0022
EIP
CPU
subl $4,%esp
movl $10,(%esp)
CODE
STACK
...
Contexte bar()
Contexte foo()
Contexte main()
MAIN MEMORY


CPU - 32 bits registers
x86 (IA-32)
eax
edi
ebx
esi
ecx
edx
General Purpose
Index
Pointers
ebp
OxFFC014
ESP
Ox4C0026
EIP
CPU
subl $4,%esp
movl $10,(%esp)
CODE
...
STACK
Contexte bar()
Contexte foo()
Contexte main()
MAIN MEMORY

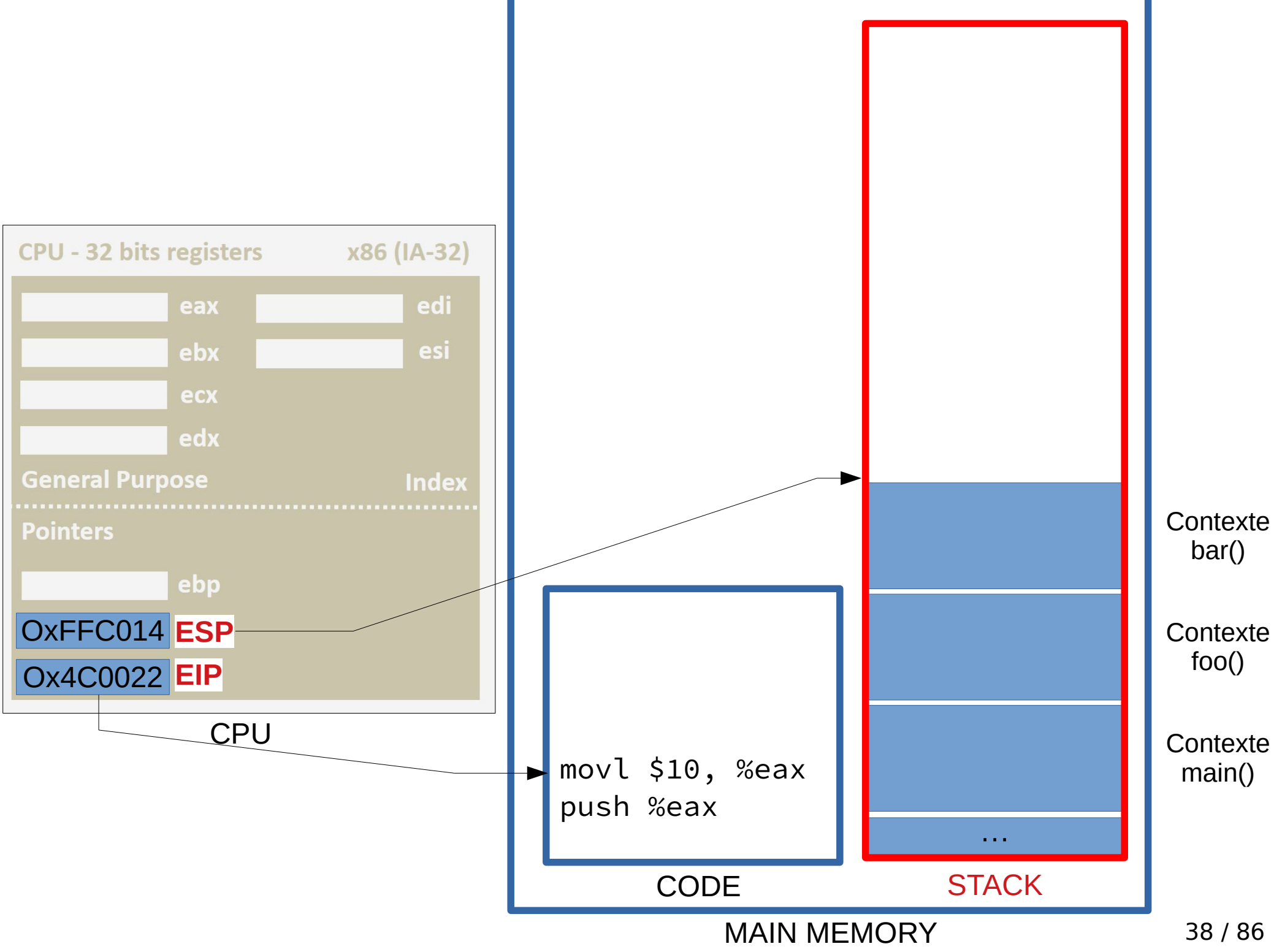

CPU - 32 bits registers
x86 (IA-32)
eax
edi
ebx
esi
ecx
edx
General Purpose
Index
Pointers
ebp
OxFFC014
ESP
Ox4C0032
EIP
CPU
subl $4,%esp
movl $10,(%esp)
...
CODE
10
...
STACK
Contexte bar()
Contexte foo()
Contexte main()
MAIN MEMORY

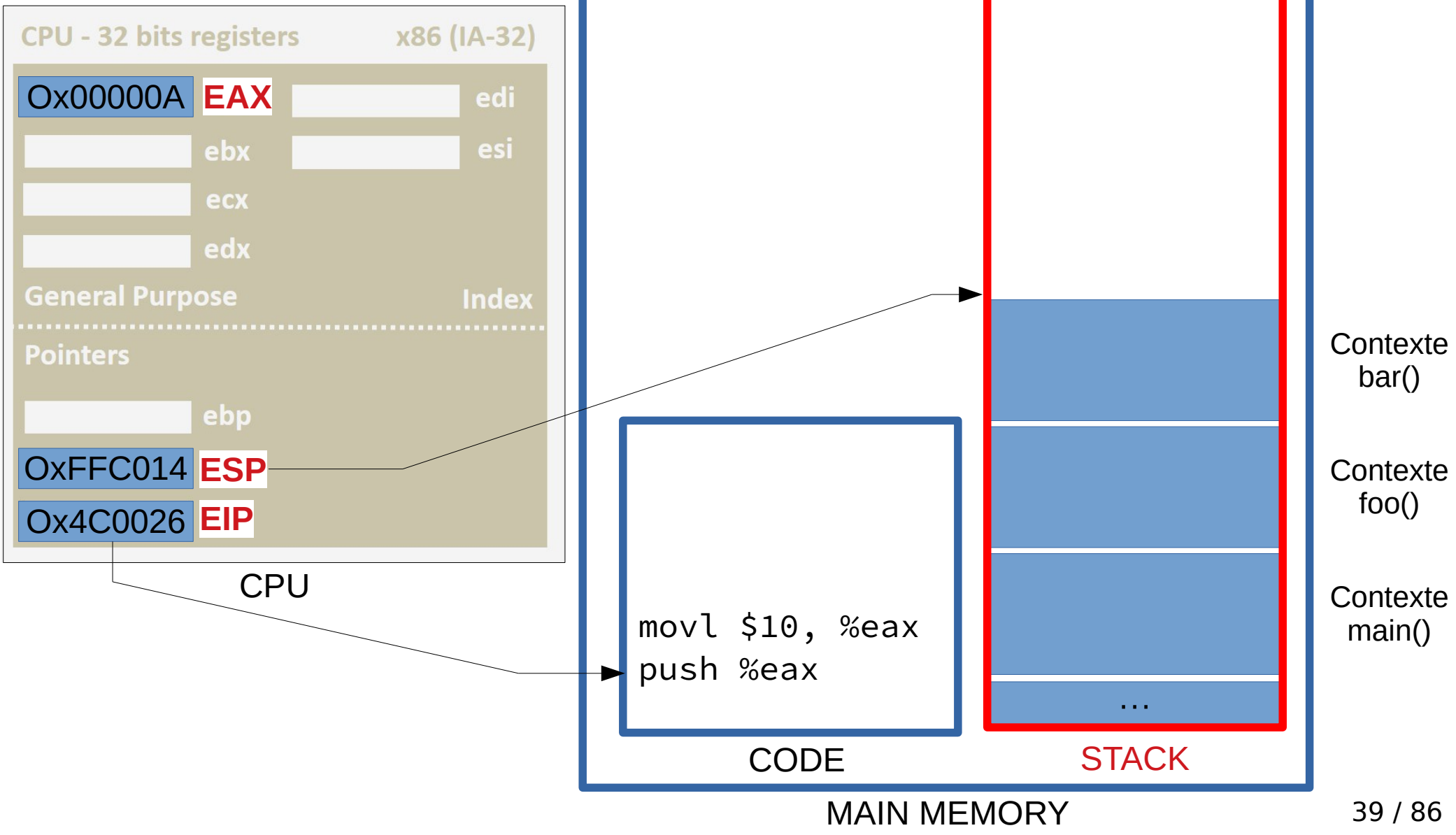
La Pile : Question 1

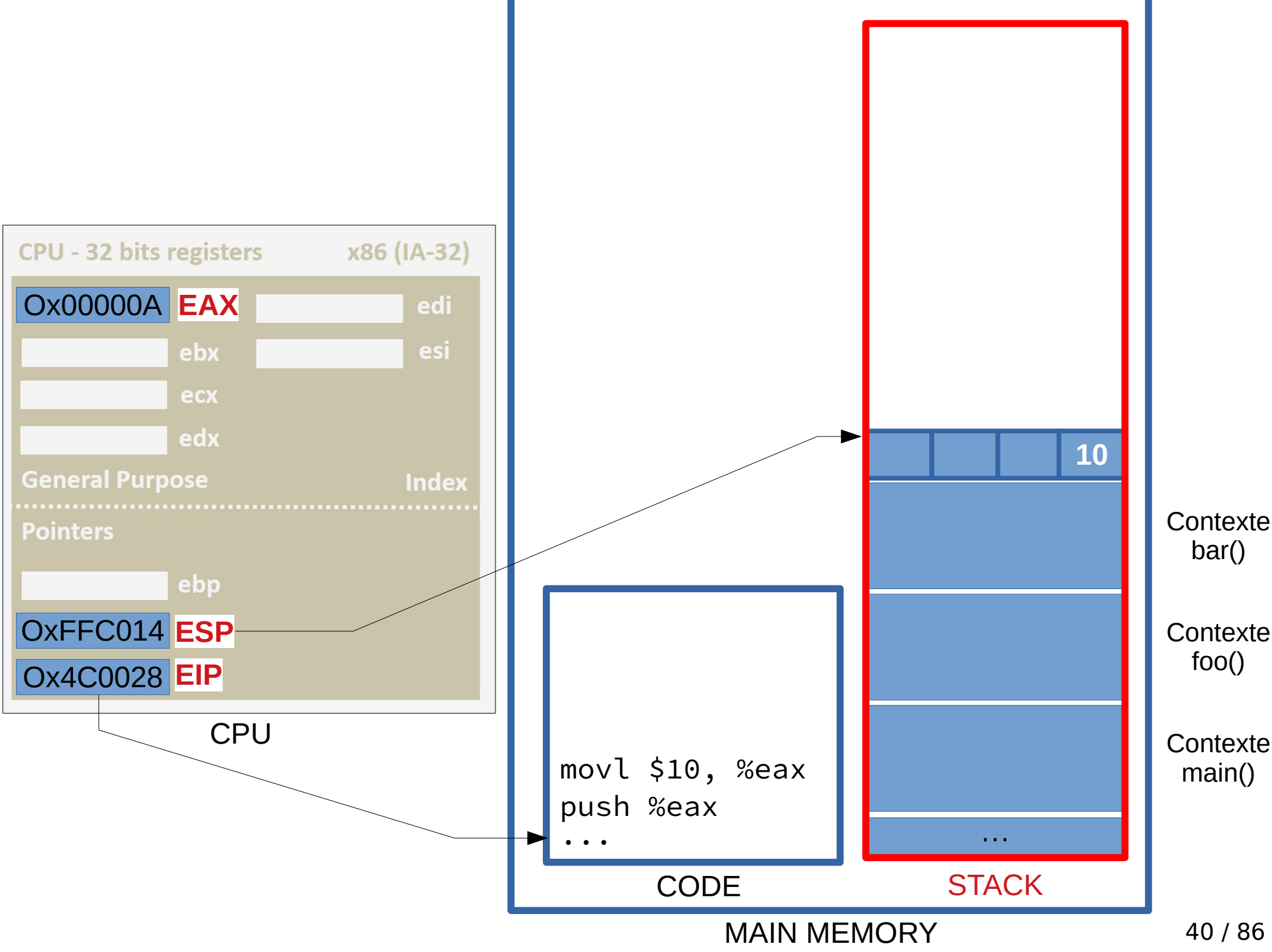
- Instructions typiques avec ESP :

- 2 : instructions PUSH / POP

  → Empiler simplement


CPU - 32 bits registers
x86 (IA-32)
eax
edi
ebx
esi
ecx
edx
General Purpose
Index
Pointers
ebp
OxFFC014
ESP
Ox4C0022
EIP
CPU
movl $10, %eax
push %eax
CODE
STACK
...
Contexte bar()
Contexte foo()
Contexte main()
MAIN MEMORY


CPU - 32 bits registers
x86 (IA-32)
Ox00000A
EAX
edi
ebx
esi
ecx
edx
General Purpose
Index
Pointers
ebp
OxFFC014
ESP
Ox4C0026
EIP
CPU
movl $10, %eax
push %eax
CODE
STACK
...
Contexte bar()
Contexte foo()
Contexte main()
MAIN MEMORY

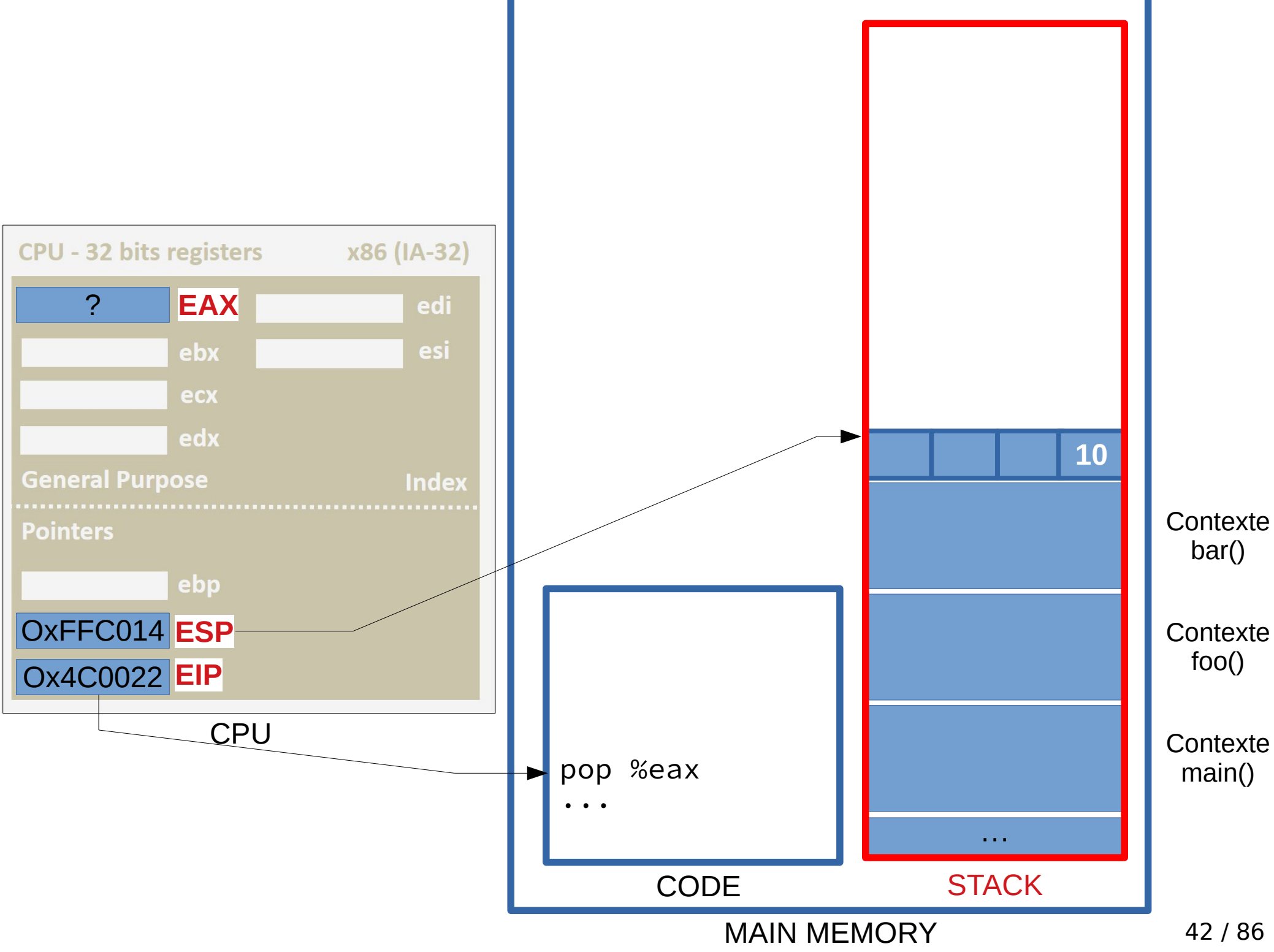

CPU - 32 bits registers
x86 (IA-32)
Ox00000A
EAX
edi
ebx
esi
ecx
edx
General Purpose
Index
Pointers
ebp
OxFFC014
ESP
Ox4C0028
EIP
CPU
movl $10, %eax
push %eax
...
CODE
10
...
STACK
MAIN MEMORY
Contexte bar()
Contexte foo()
Contexte main()

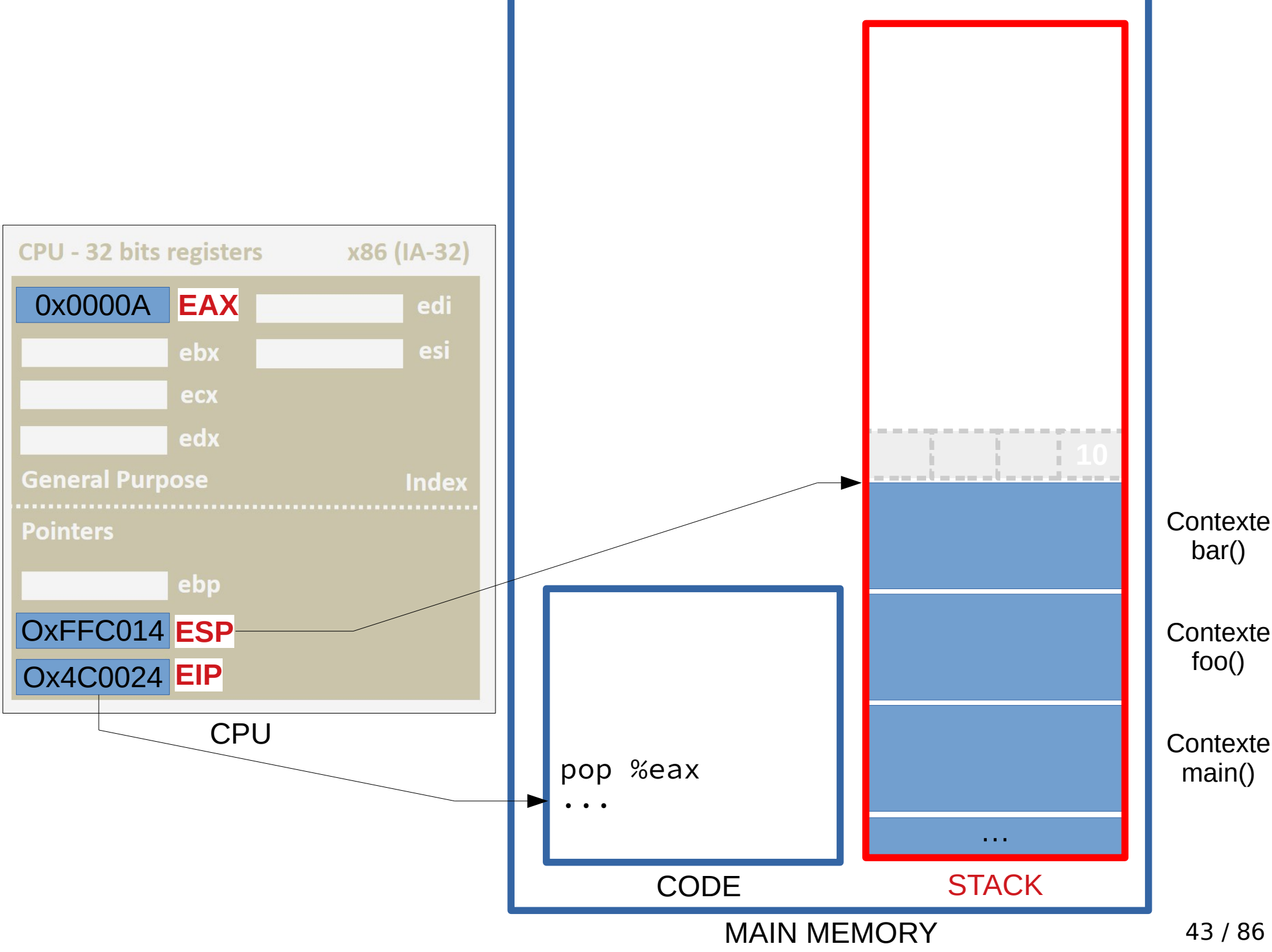
La Pile : Question 1

- Instructions typiques avec ESP :

- 2 : instructions PUSH / POP

  → Empiler simplement
  → Dépiler simplement


CPU - 32 bits registers
x86 (IA-32)
?
EAX
edi
ebx
esi
ecx
edx
General Purpose
Index
Pointers
ebp
OxFFC014
ESP
Ox4C0022
EIP
CPU
pop %eax
...
CODE
10
...
STACK
MAIN MEMORY
Contexte bar()
Contexte foo()
Contexte main()


CPU - 32 bits registers
x86 (IA-32)
0x0000A
EAX
edi
ebx
esi
ecx
edx
General Purpose
Index
Pointers
ebp
OxFFC014
ESP
Ox4C0024
EIP
CPU
pop %eax
...
CODE
10
...
STACK
Contexte bar()
Contexte foo()
Contexte main()
MAIN MEMORY

La Pile : Question 1

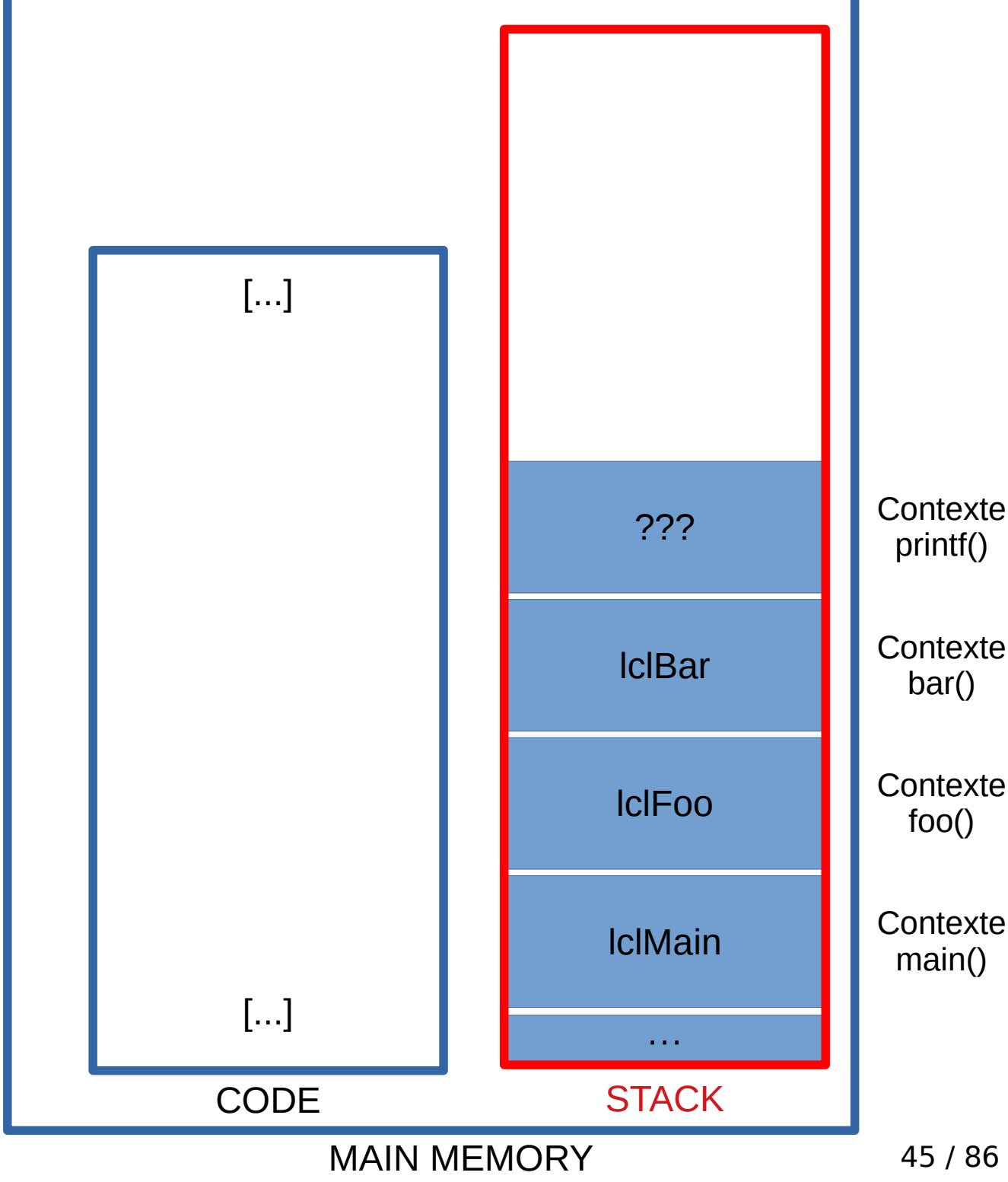
- Revenons aux contextes de fonctions...

```
int printf()
{...}
```
4

```
void foo() {
  int lclFoo = 10;
  bar();
}
```
2

```
void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```
3

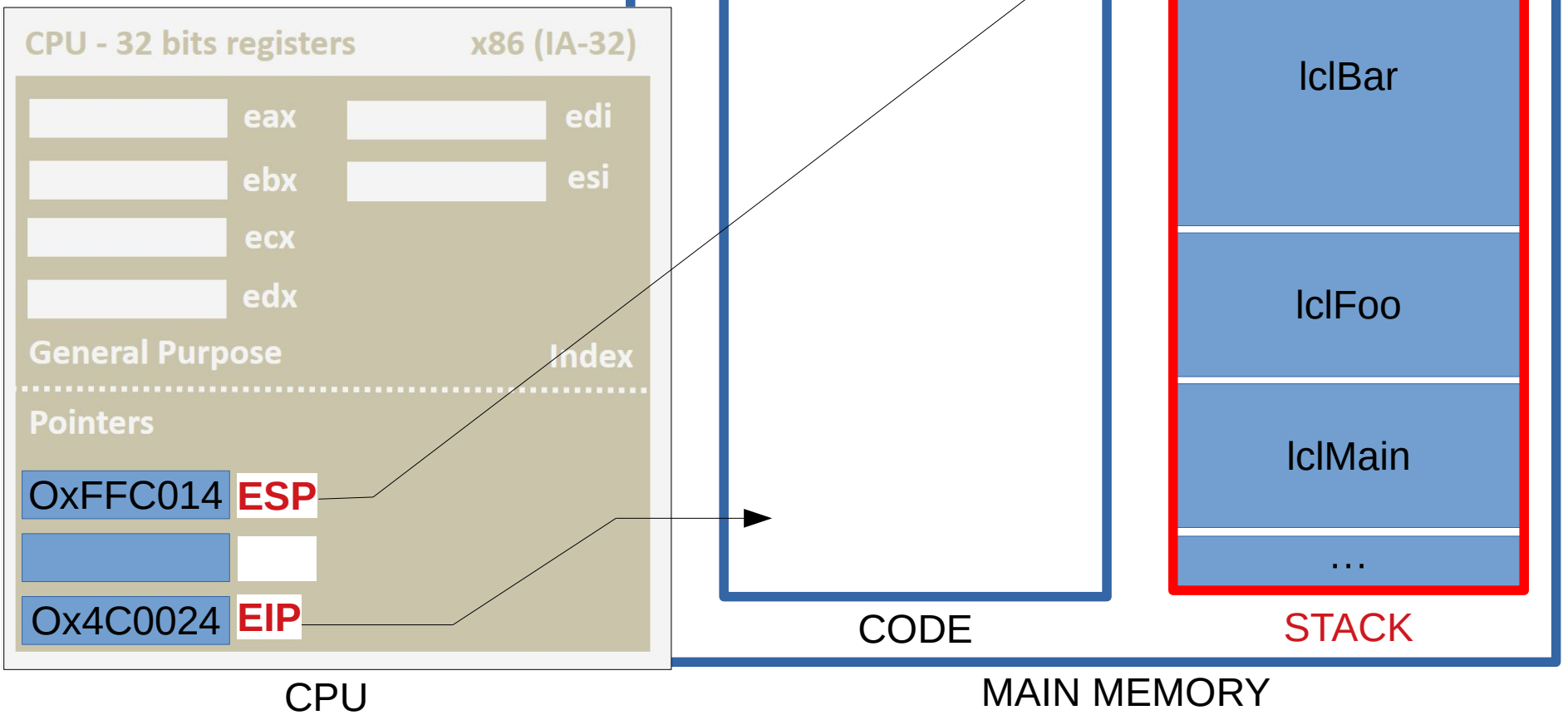
```
int main() {
  int lclMain = 5;
  bar();
  foo();
}
```
1



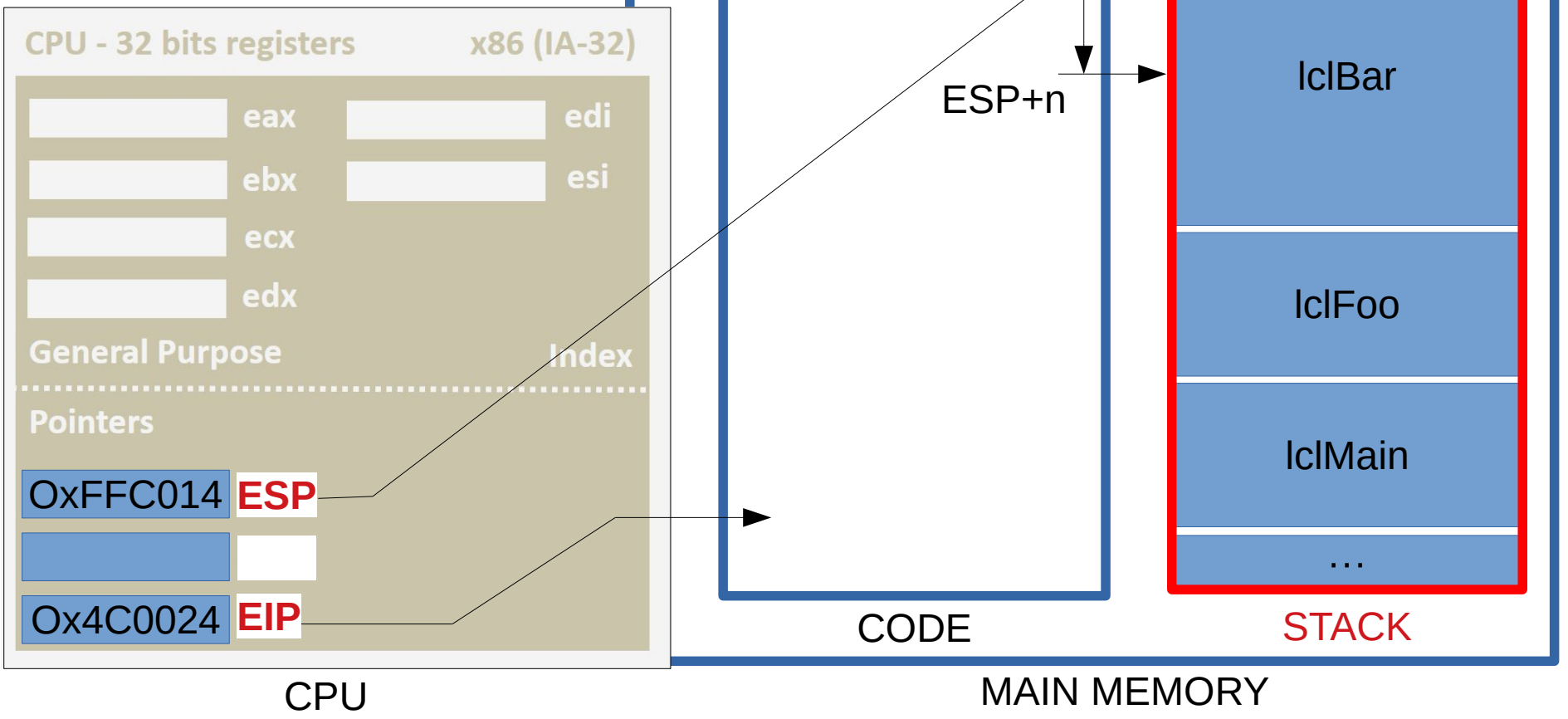
La Pile : Question 2

- Comment connaît-on l'adresse des variables locales (@ effectives inconnues à la compilation)?
- Réponse 1 : relativement à ESP !

```
void bar() {
   int lclBar = 20;
   printf("OK\n");
}
```



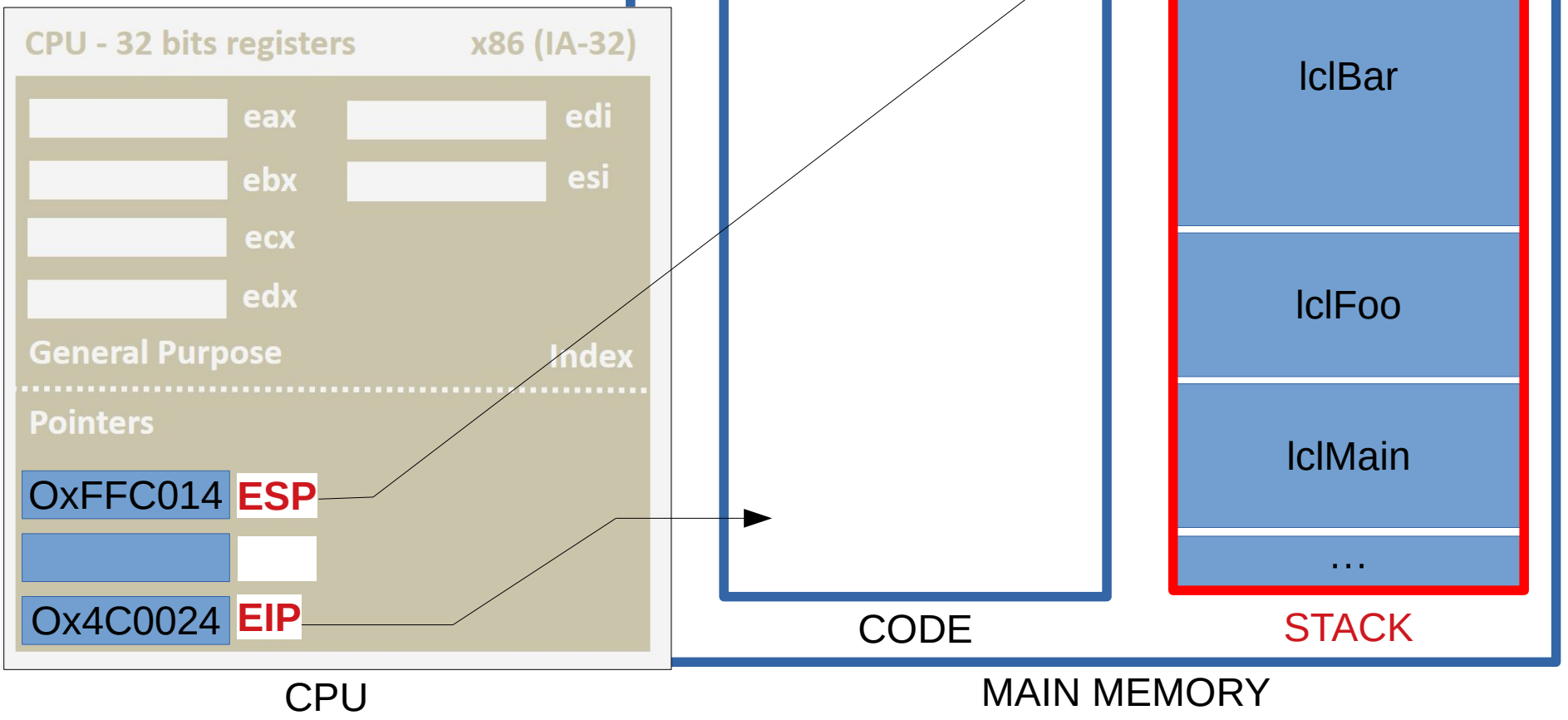
```
void bar() {
   int lclBar = 20;
   printf("OK\n");
}
```



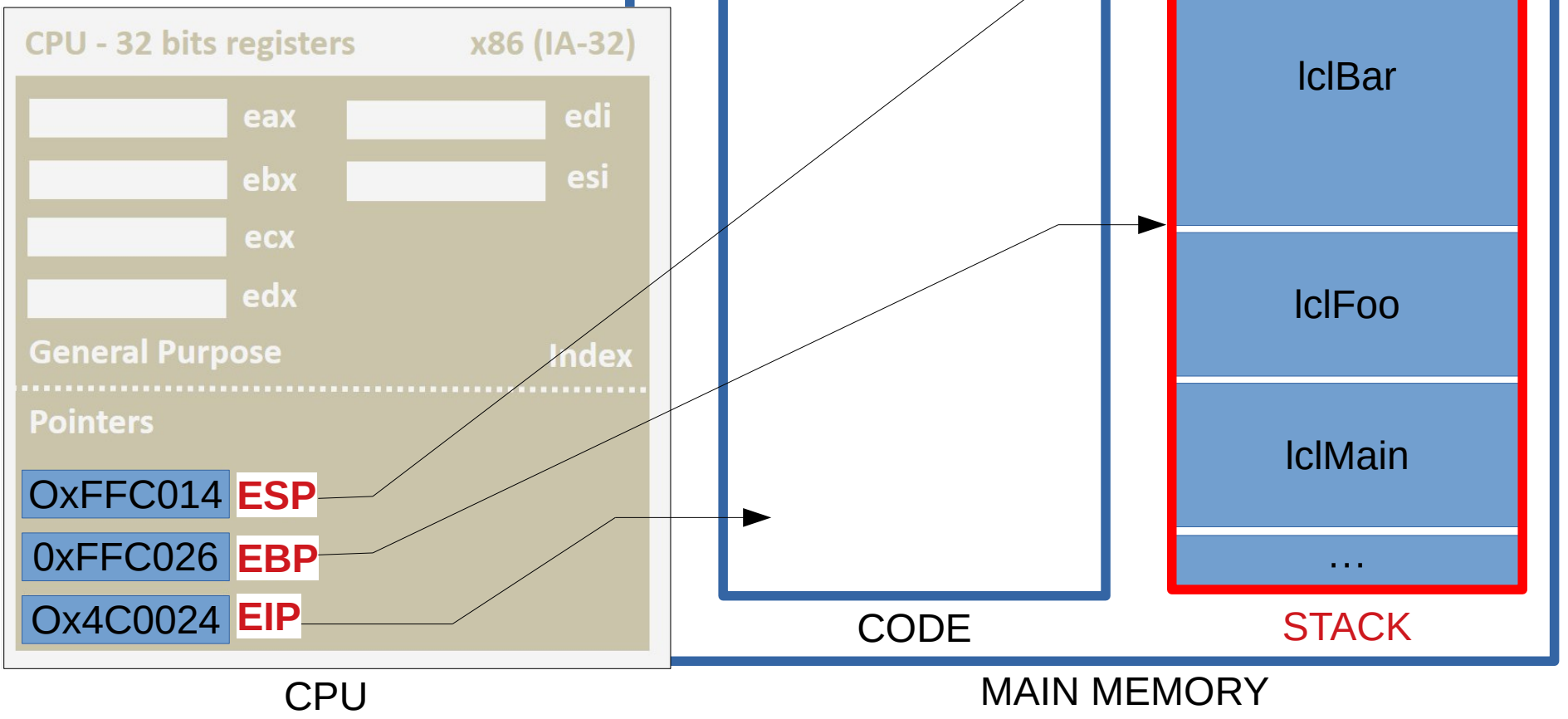
La Pile : Question 2

- Comment connaît-on l'adresse des variables locales (@ effectives inconnues à la compilation)?
- Réponse 1 : relativement à ESP !
- Possible dans certains cas.
  Mais parfois, ESP bouge ! (push, pop, sub, etc.)
- Réponse 2 : relativement à la base de son contexte → EBP (*Base Pointer*)

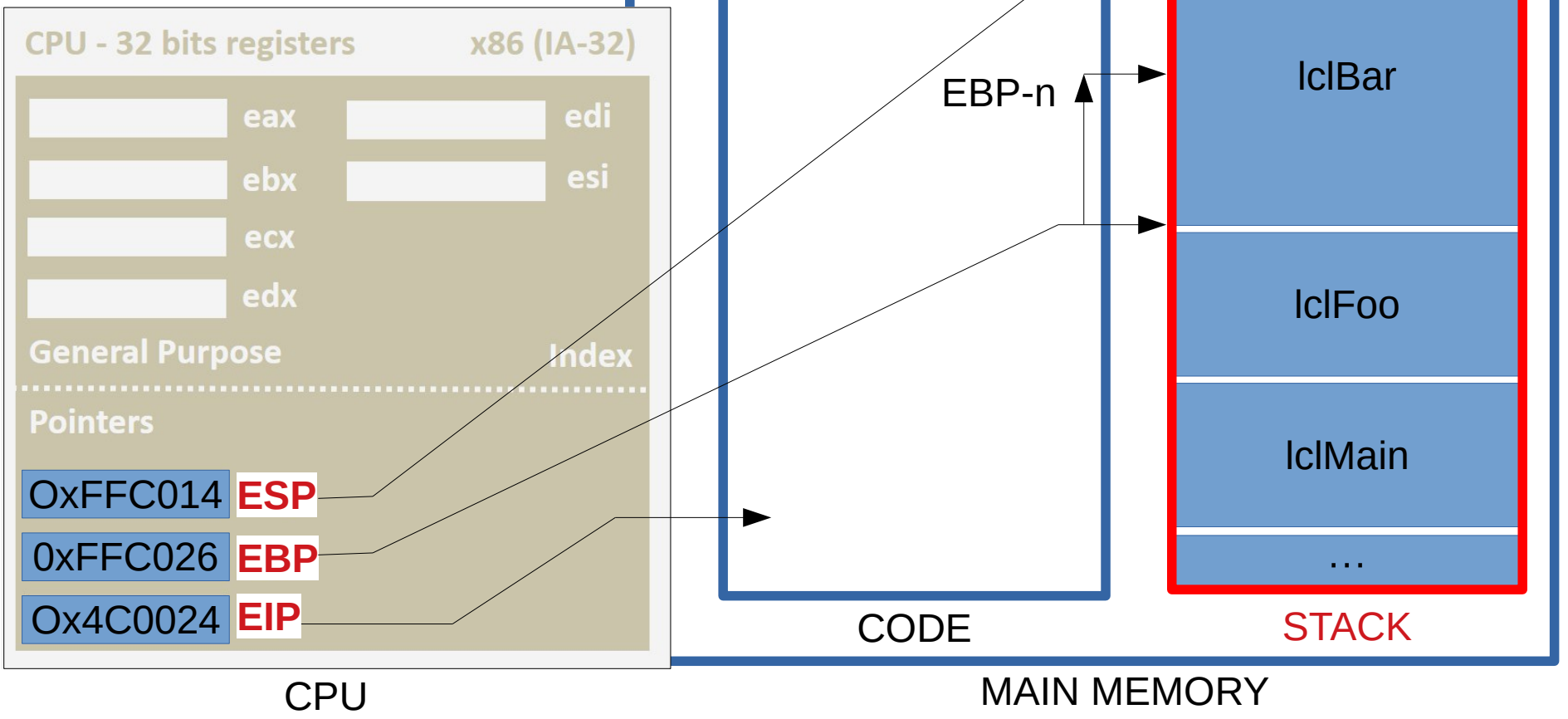
```
void bar() {
   int lclBar = 20;
   printf("OK\n");
}
```



```
void bar() {
   int lclBar = 20;
   printf("OK\n");
}
```



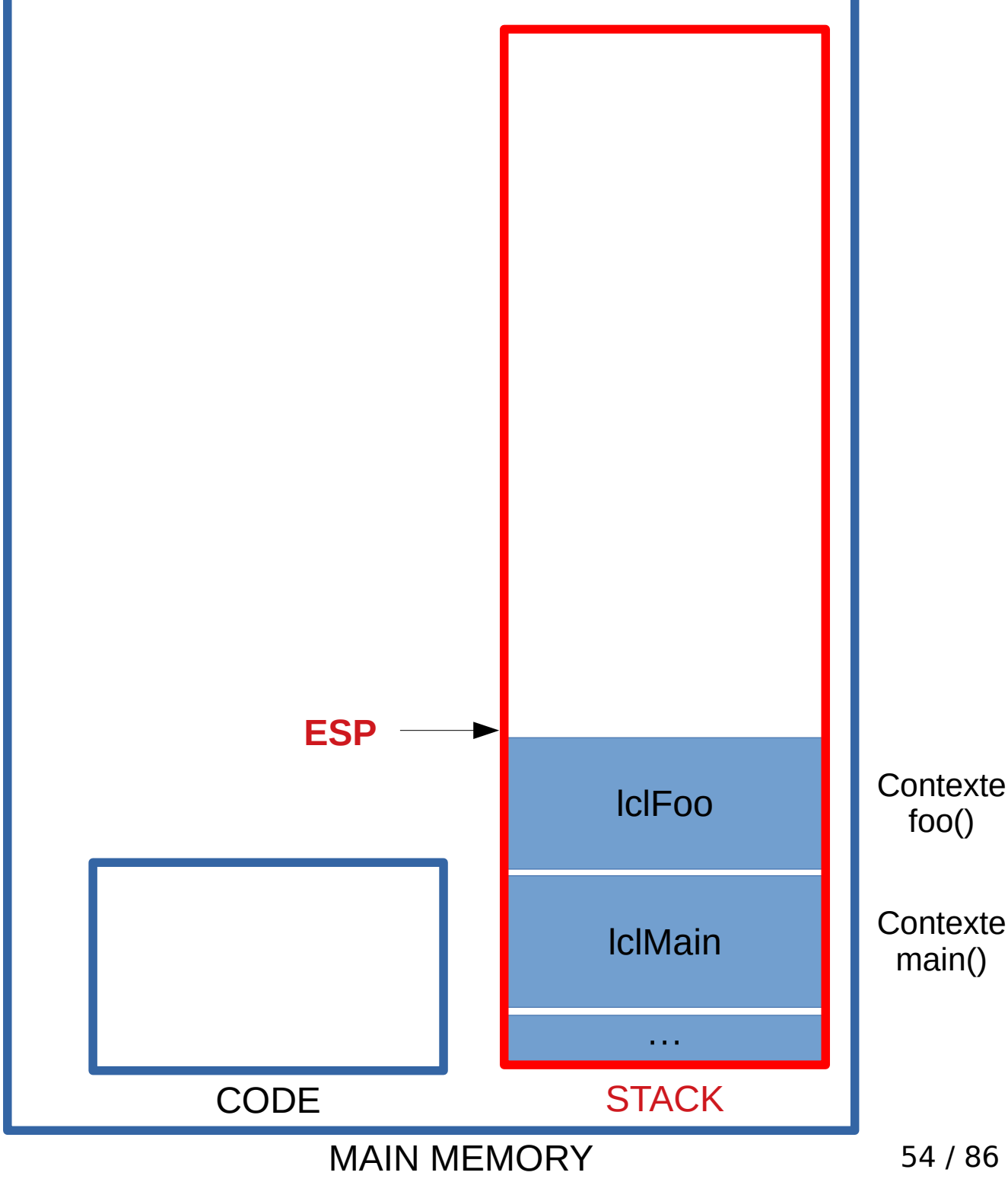
```
void bar() {
   int lclBar = 20;
   printf("OK\n");
}
```



La Pile : Question 2

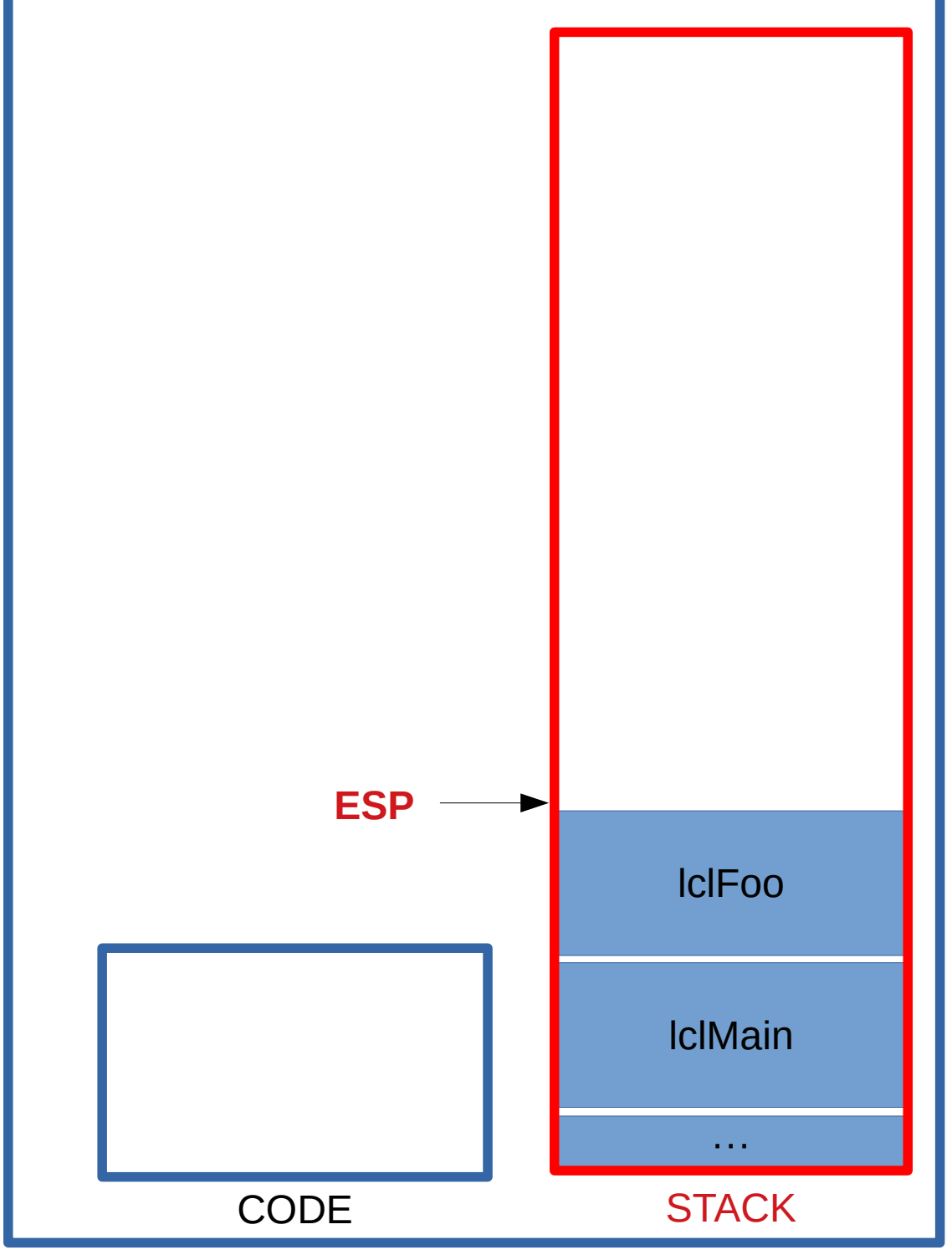
- EBP est fixe au sein de la fonction
- Les adresses relatives à EBP aussi…
- Comment est initialisé EBP ?
- En fait, c’est (quasiment) le ESP de la fonction appelante...

```c
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```



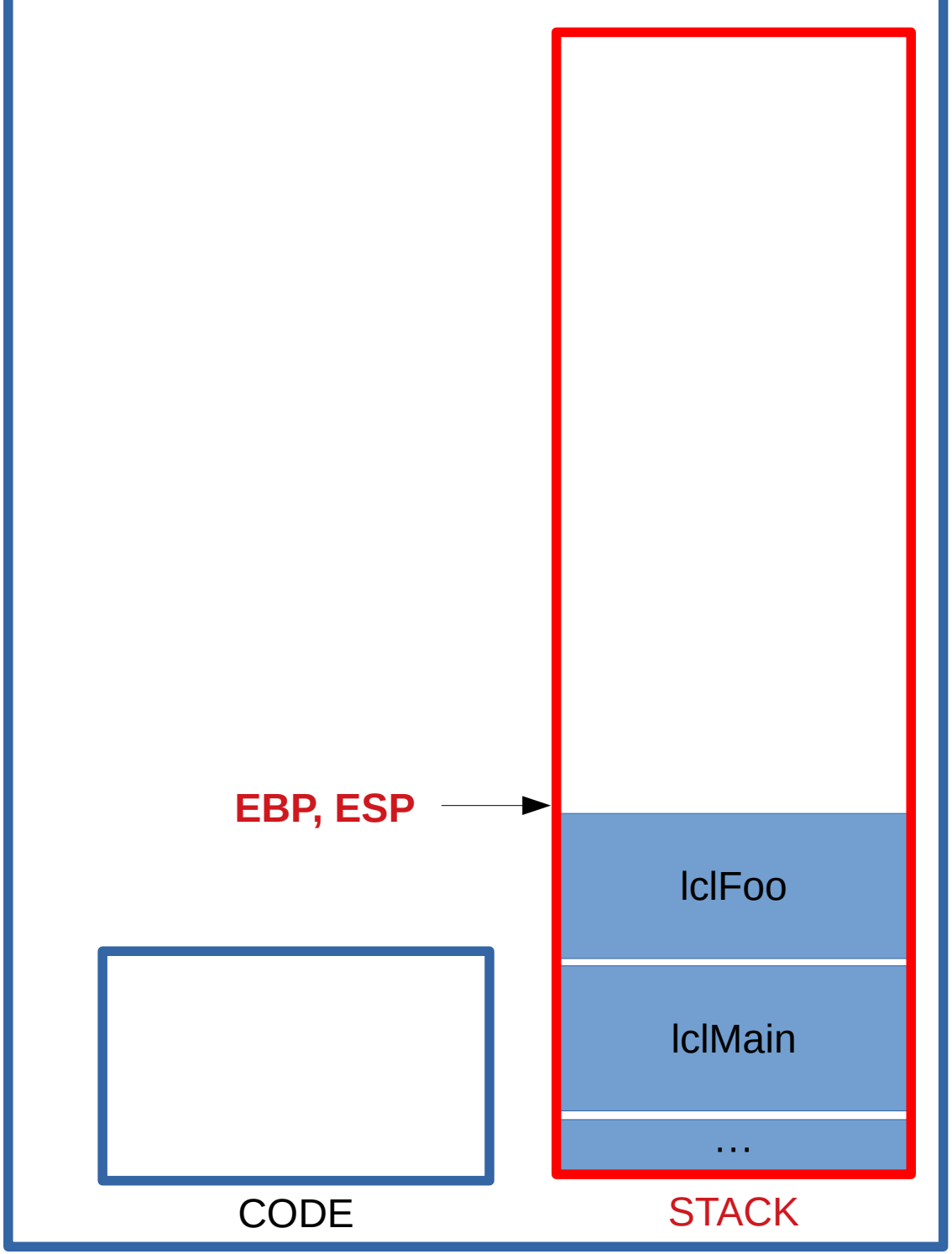
```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```

mov %esp,%ebp



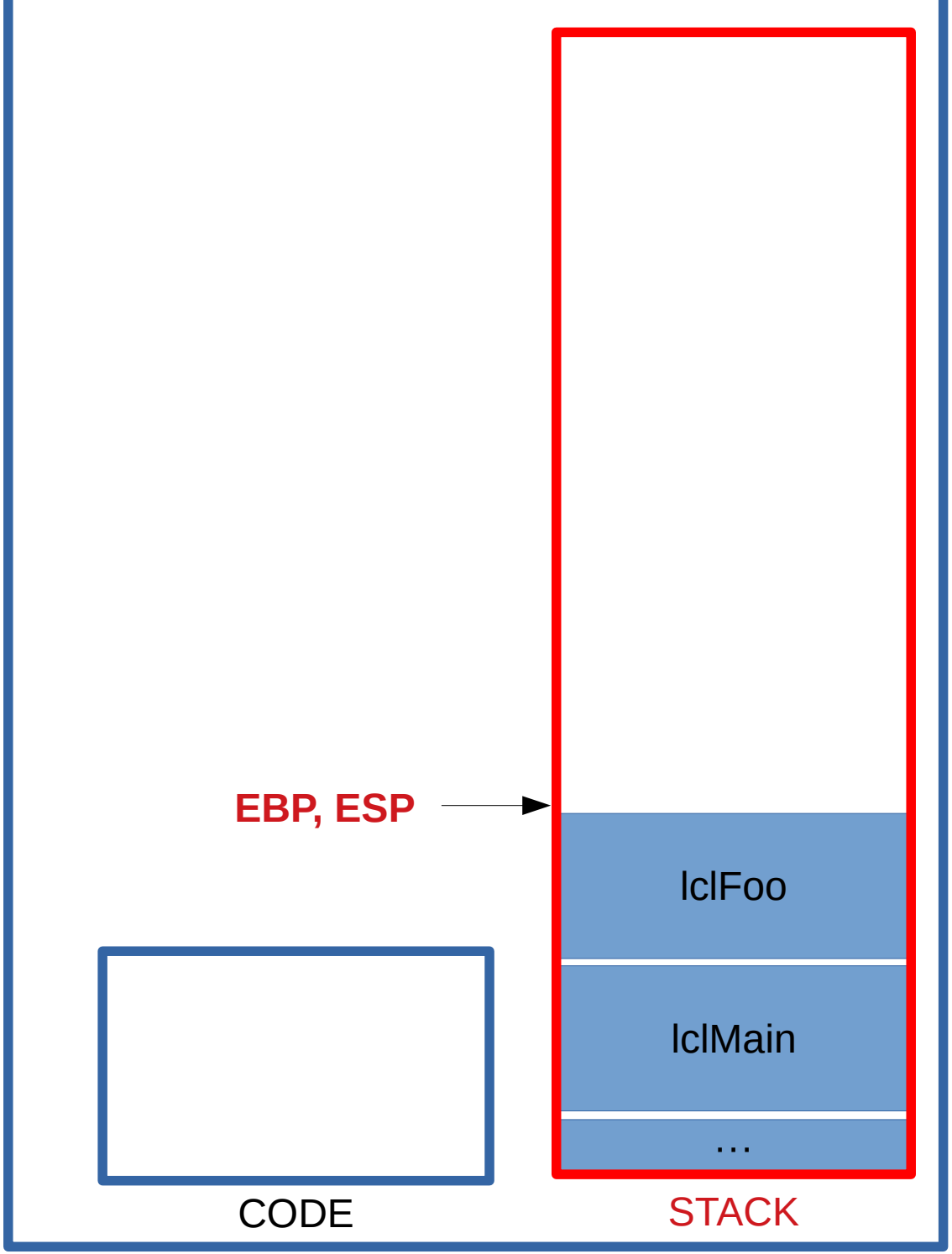
```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```

```
mov %esp,%ebp
```



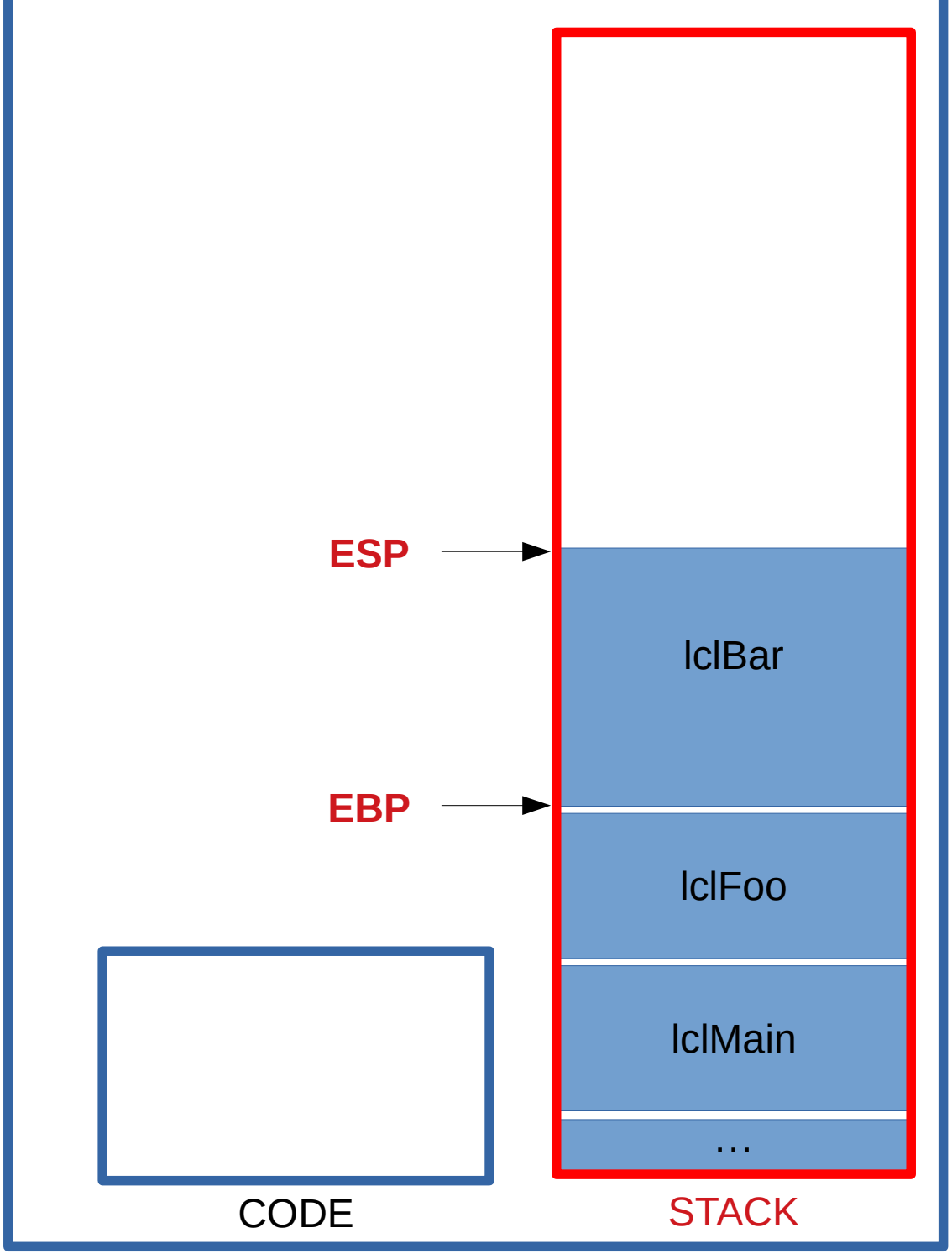
```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```

```
mov %esp,%ebp
sub $32,%esp
```



```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```

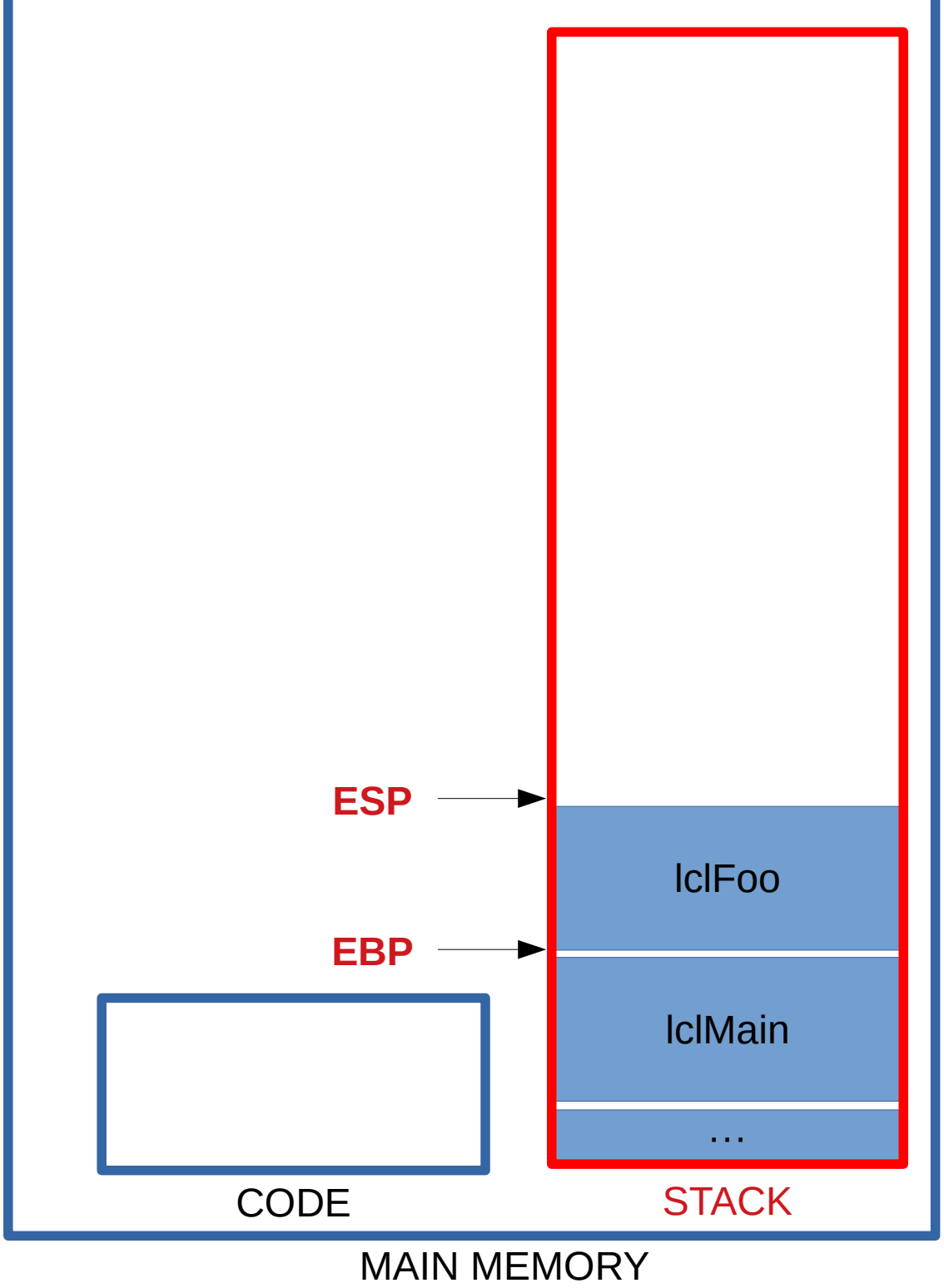
```
mov %esp,%ebp
sub $32,%esp
```



La Pile : Question 3

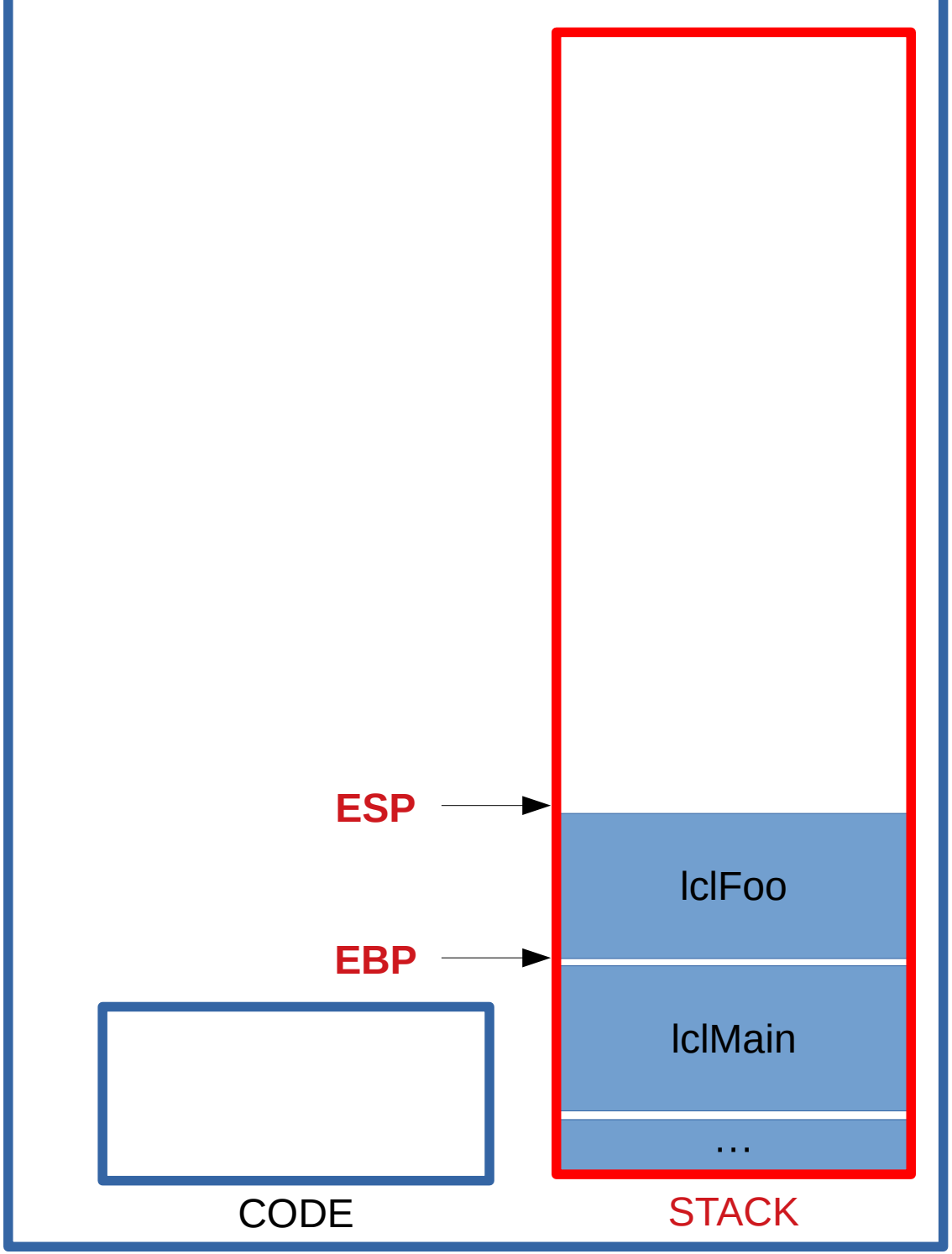
- En fin de fonction, comment retrouver/restaurer le EBP de la fonction appelante ?
- Il suffit de le sauver sur la pile !

```c
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```



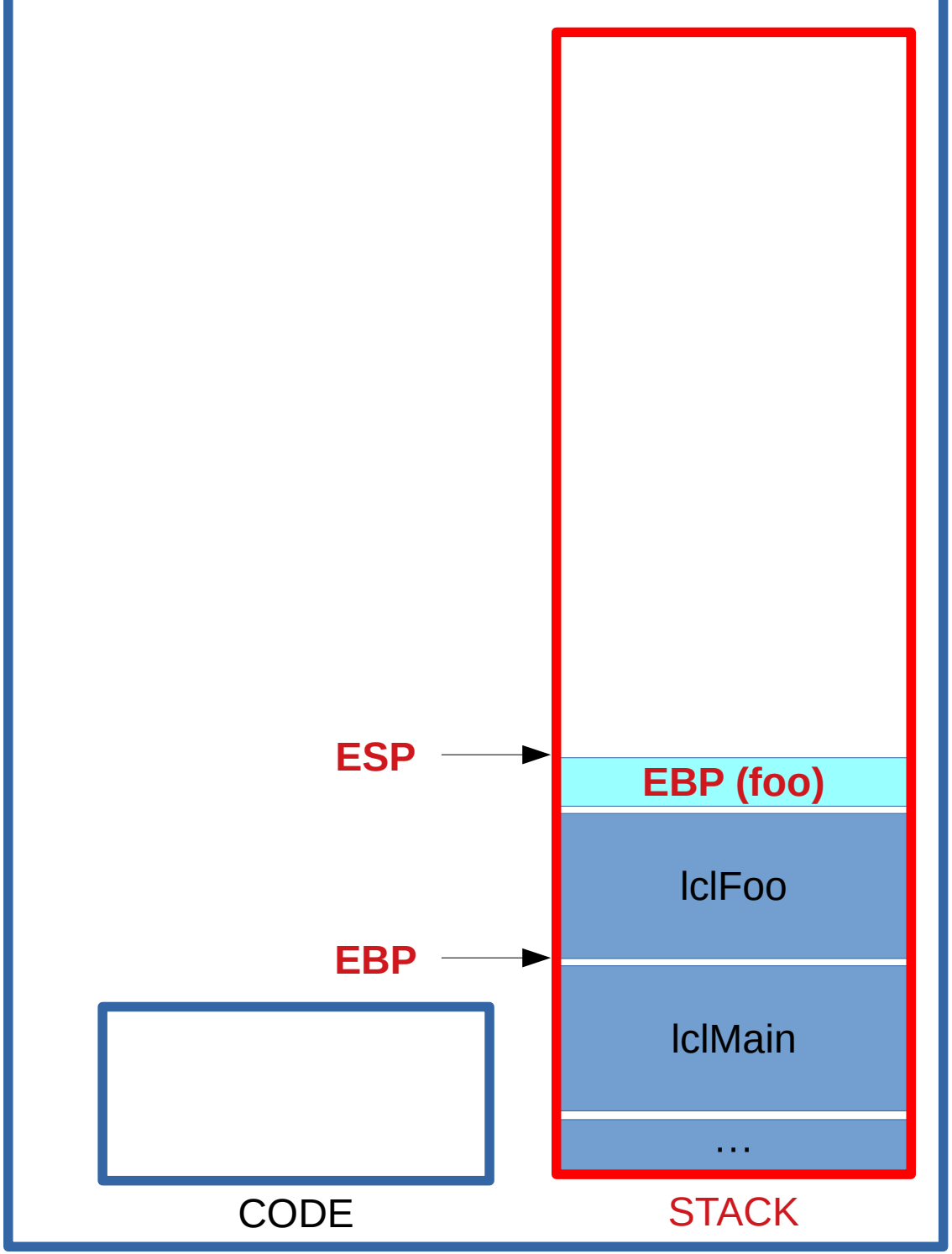
```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```

```
push %ebp
mov %esp,%ebp
sub $32,%esp
```



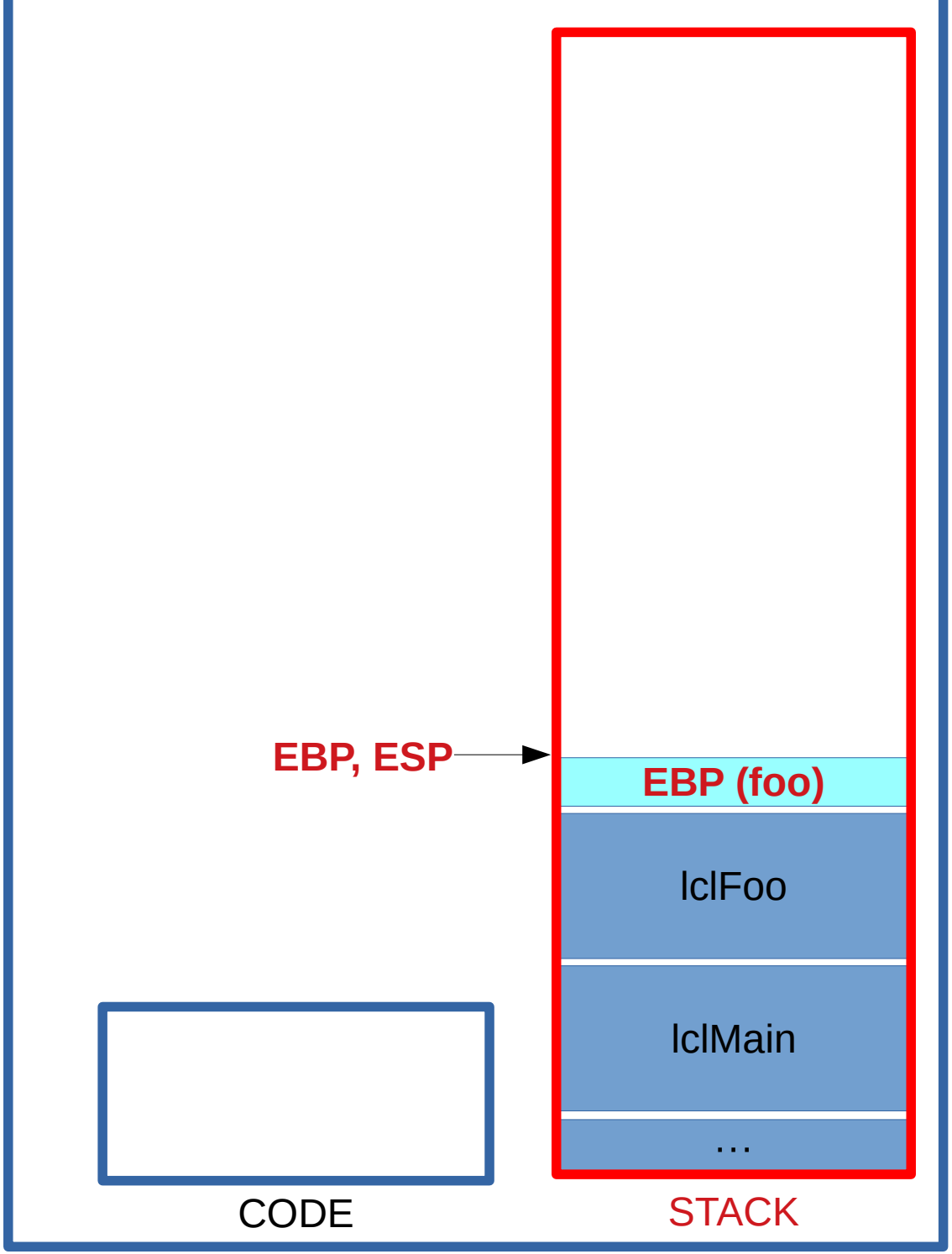
```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```

```
push %ebp
mov %esp,%ebp
sub $32,%esp
```



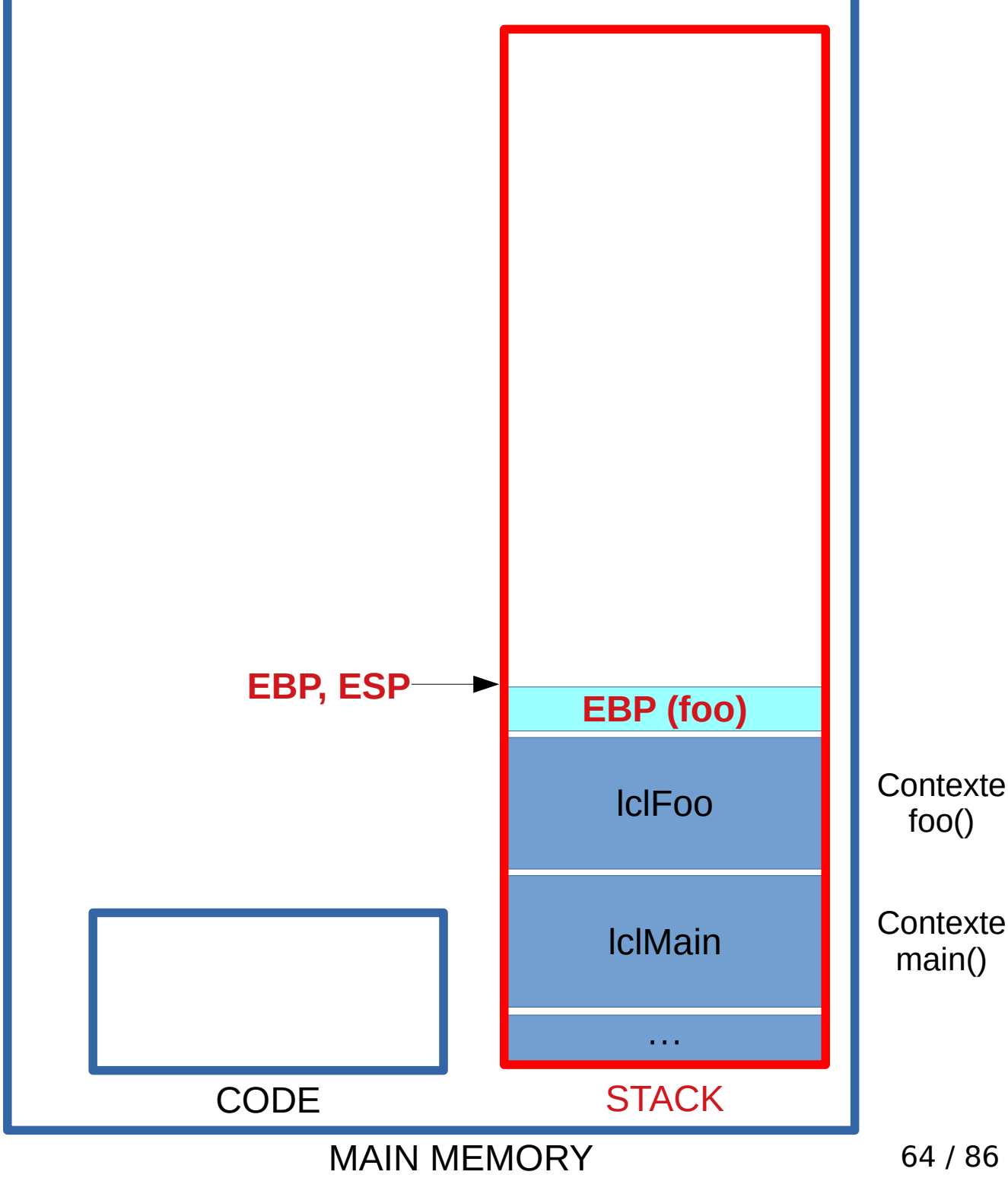
```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```

```
push %ebp
mov %esp,%ebp
sub $32,%esp
```



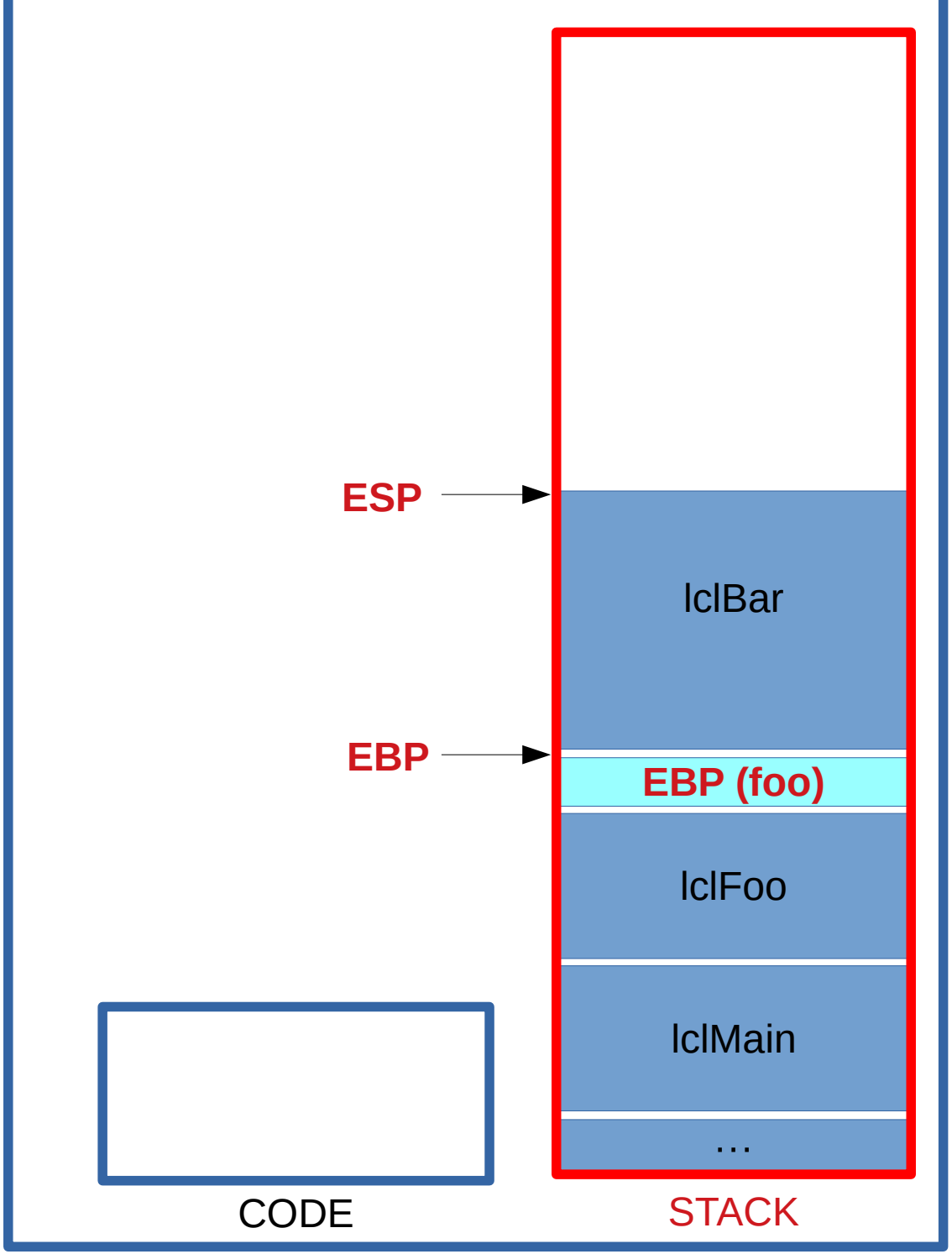
```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```

```
push %ebp
mov %esp,%ebp
sub $32,%esp
```



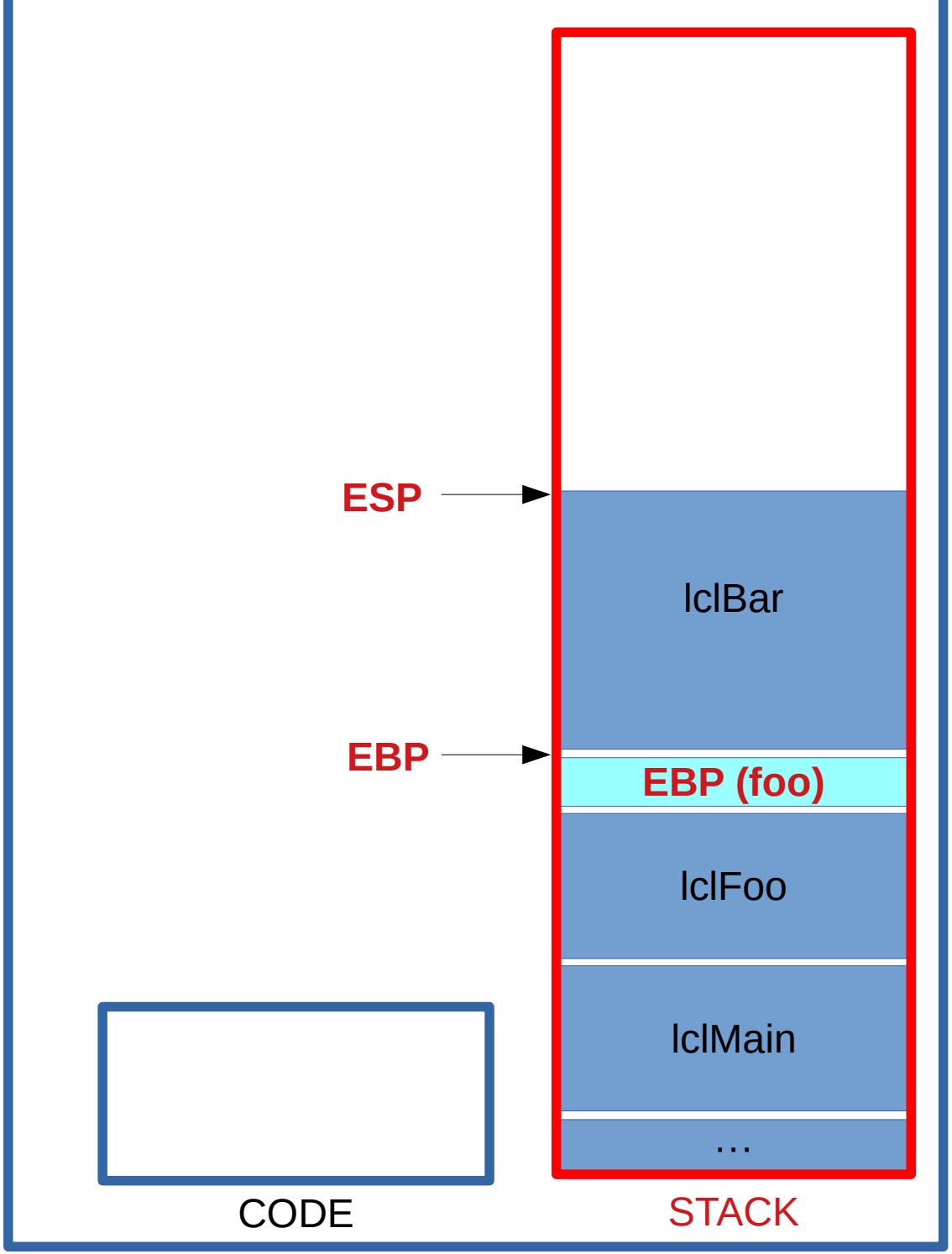
```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```

```
push %ebp
mov %esp,%ebp
sub $32,%esp
```



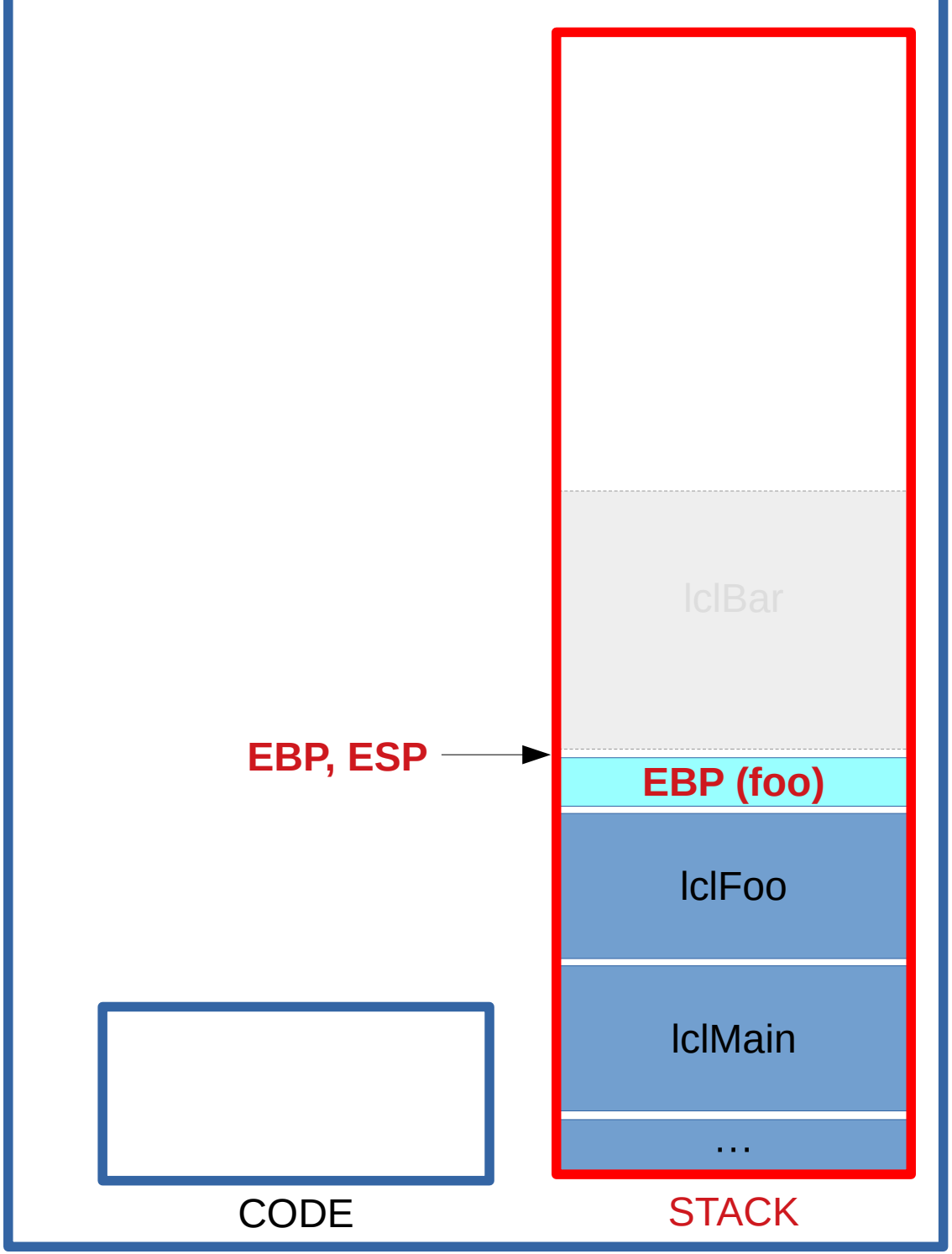
```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```

```
mov %ebp,%esp
pop %ebp
```



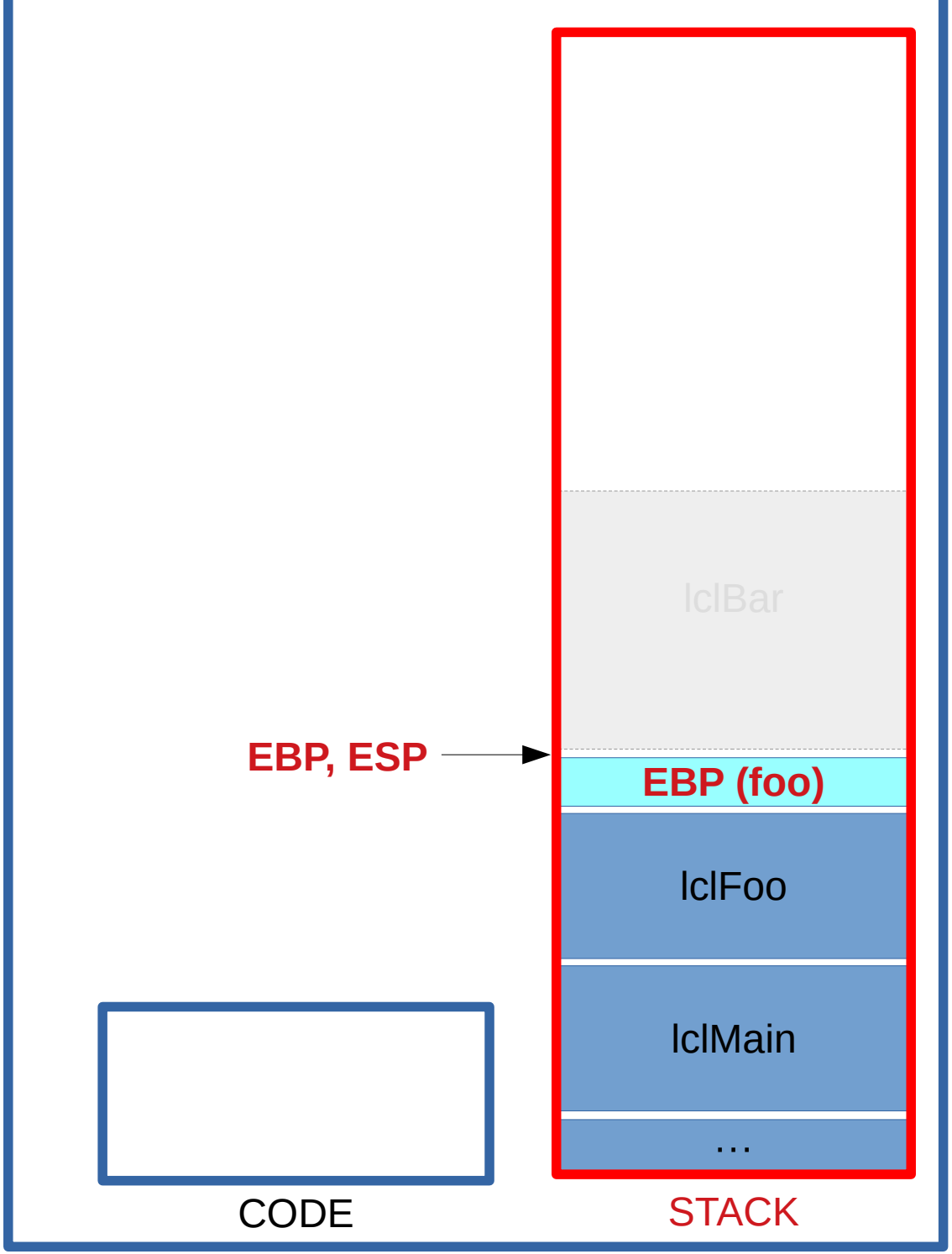
```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```

```
mov %ebp,%esp
pop %ebp
```



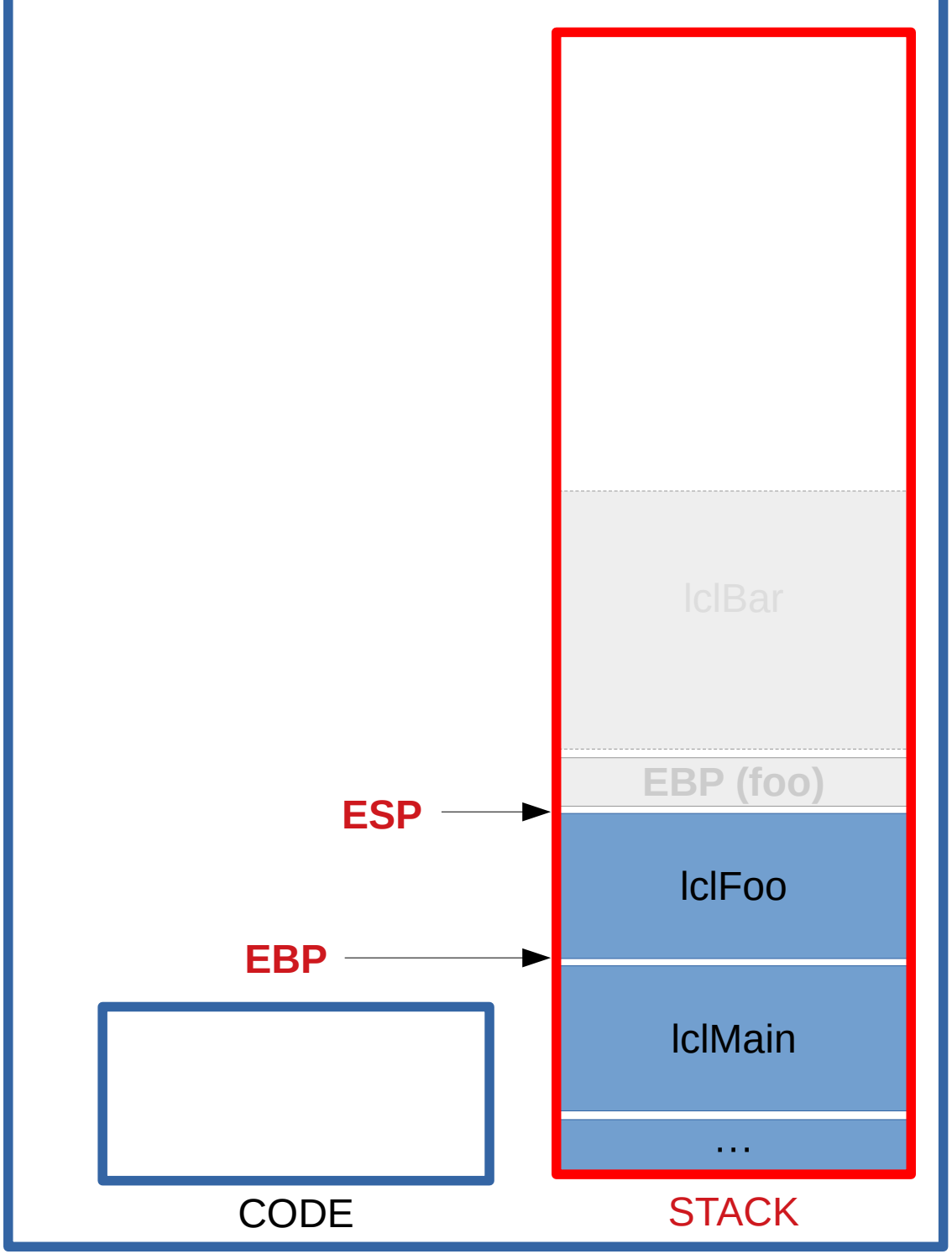
```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```

```
mov %ebp,%esp
pop %ebp
```



```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```

```
mov %ebp,%esp
pop %ebp
```

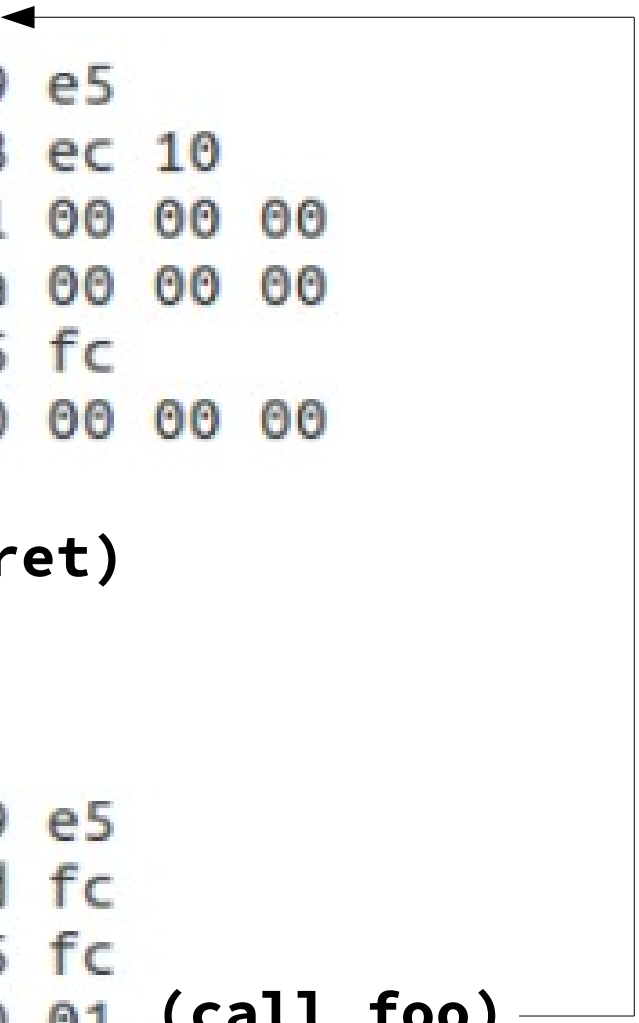


La Pile : Question 4

- Comment connaît-on l’adresse de retour d’une fonction ?

```
foo() :   55
          48 89 e5
          48 83 ec 10
          bf 01 00 00 00
          e8 0a 00 00 00
          89 45 fc
          b8 00 00 00 00
          c9
          c3 (ret)

main() :  55
          48 89 e5
          89 7d fc
          8b 45 fc
          83 c0 01 (call foo)
          5d
          c3
          66 2e 0f 1f 84 00 00
          00 00 00
          0f 1f 44 00 00
```

```
foo() :   55
          48 89 e5
          48 83 ec 10
          bf 01 00 00 00
          e8 0a 00 00 00
          89 45 fc
          b8 00 00 00 00
          c9
          c3  (ret)


main() :  55
          48 89 e5
          89 7d fc
          8b 45 fc
          83 c0 01  (call foo)
          5d
          c3
          66 2e 0f 1f 84 00 00
          00 00 00
          0f 1f 44 00 00
```



- Simple (a priori) !
- Saut en mémoire (JMP)
- Agit sur EIP

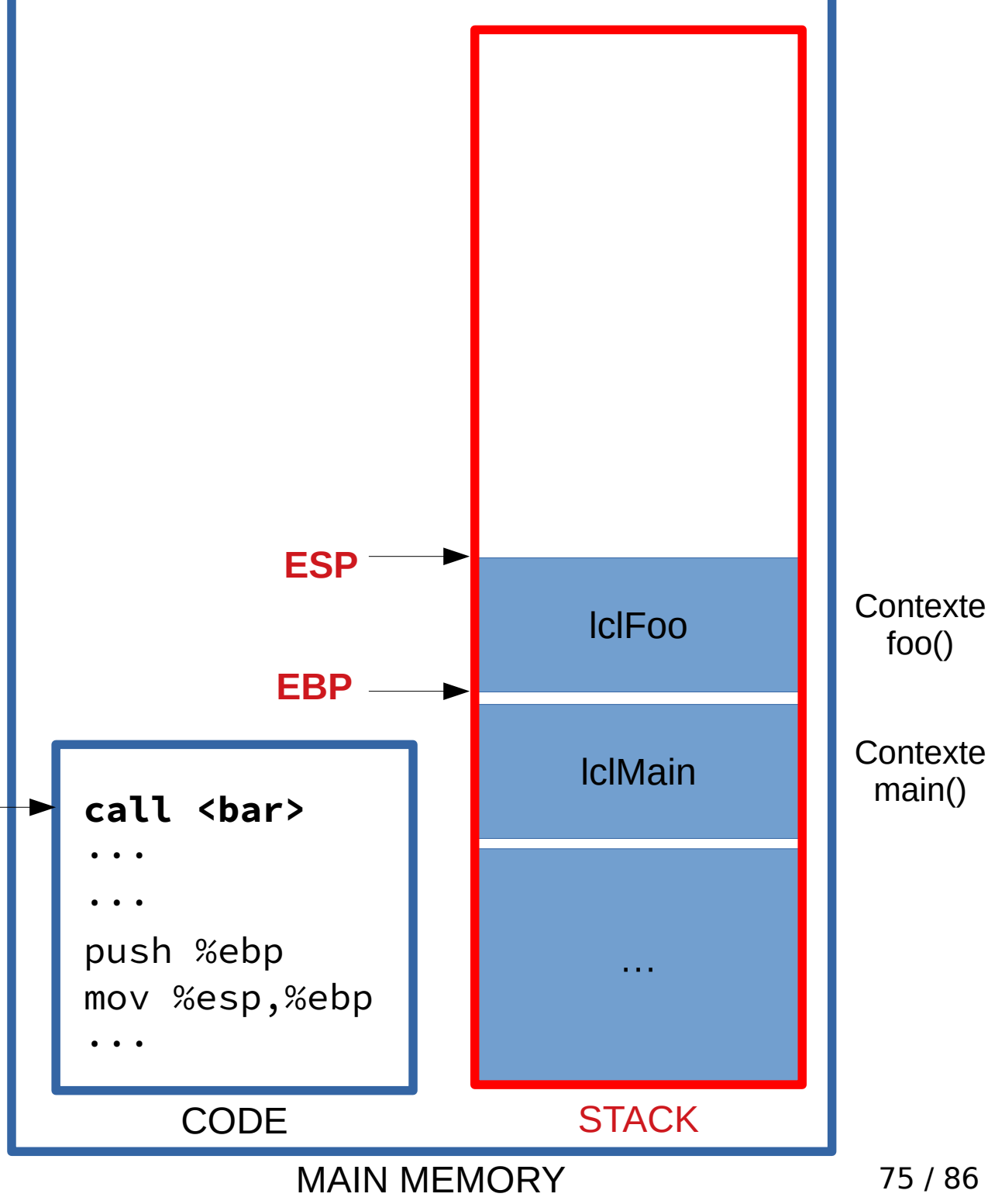
```
foo() :   55
          48 89 e5
          48 83 ec 10
          bf 01 00 00 00
          e8 0a 00 00 00
          89 45 fc
          b8 00 00 00 00
          c9
          c3  (ret)

main() :  55
          48 89 e5
          89 7d fc
          8b 45 fc
          83 c0 01  (call foo)
          5d
          c3
          66 2e 0f 1f 84 00 00
          00 00 00
          0f 1f 44 00 00
```

- Moins simple !
- D’où vient-on ?
- foo() peut être appelée de plusieurs endroits dans le code.

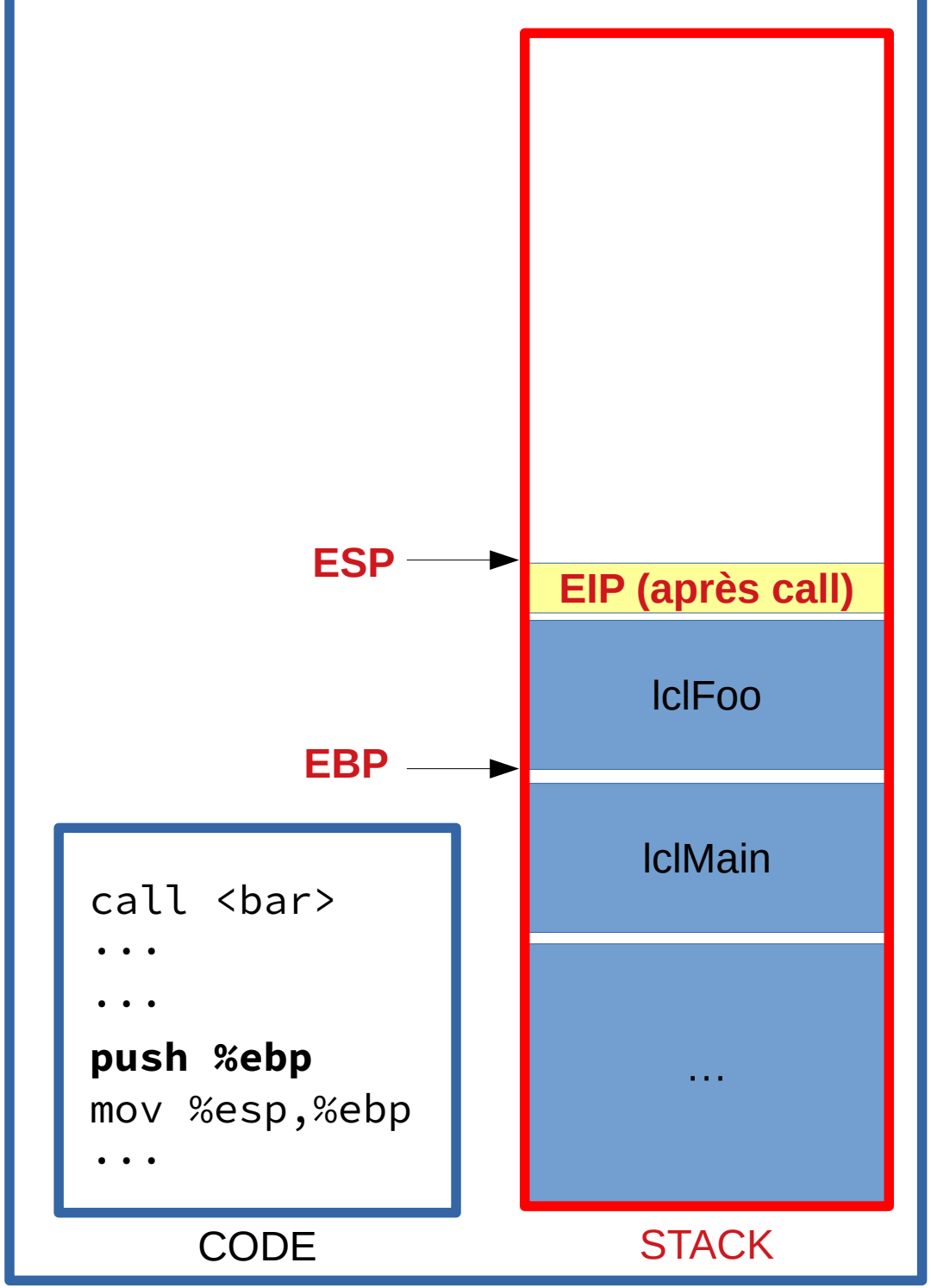
La Pile : Question 4

- Comment connaît-on l’adresse de retour d’une fonction ?
- Réponse : c’est encore grâce à la pile !

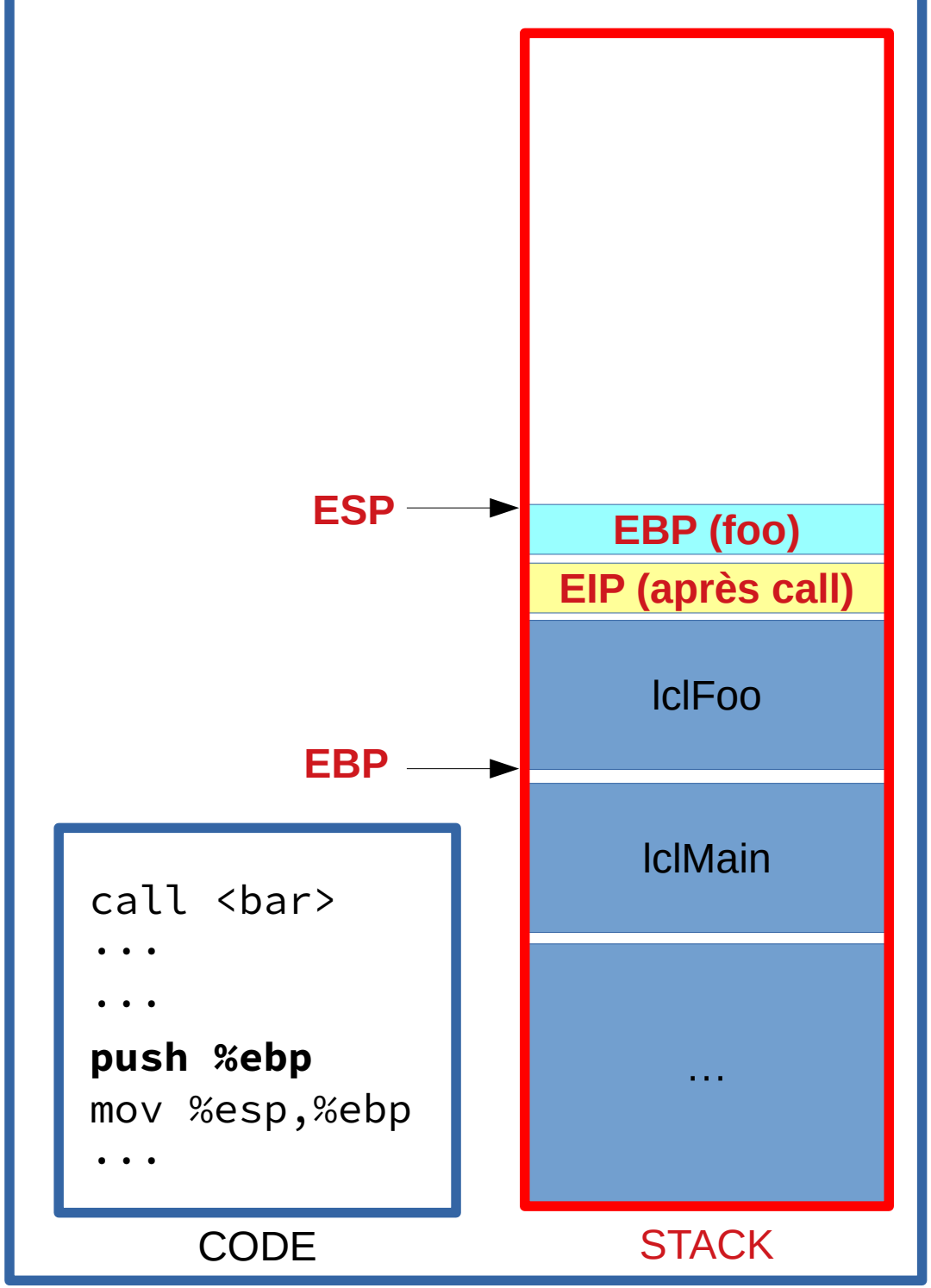
```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```



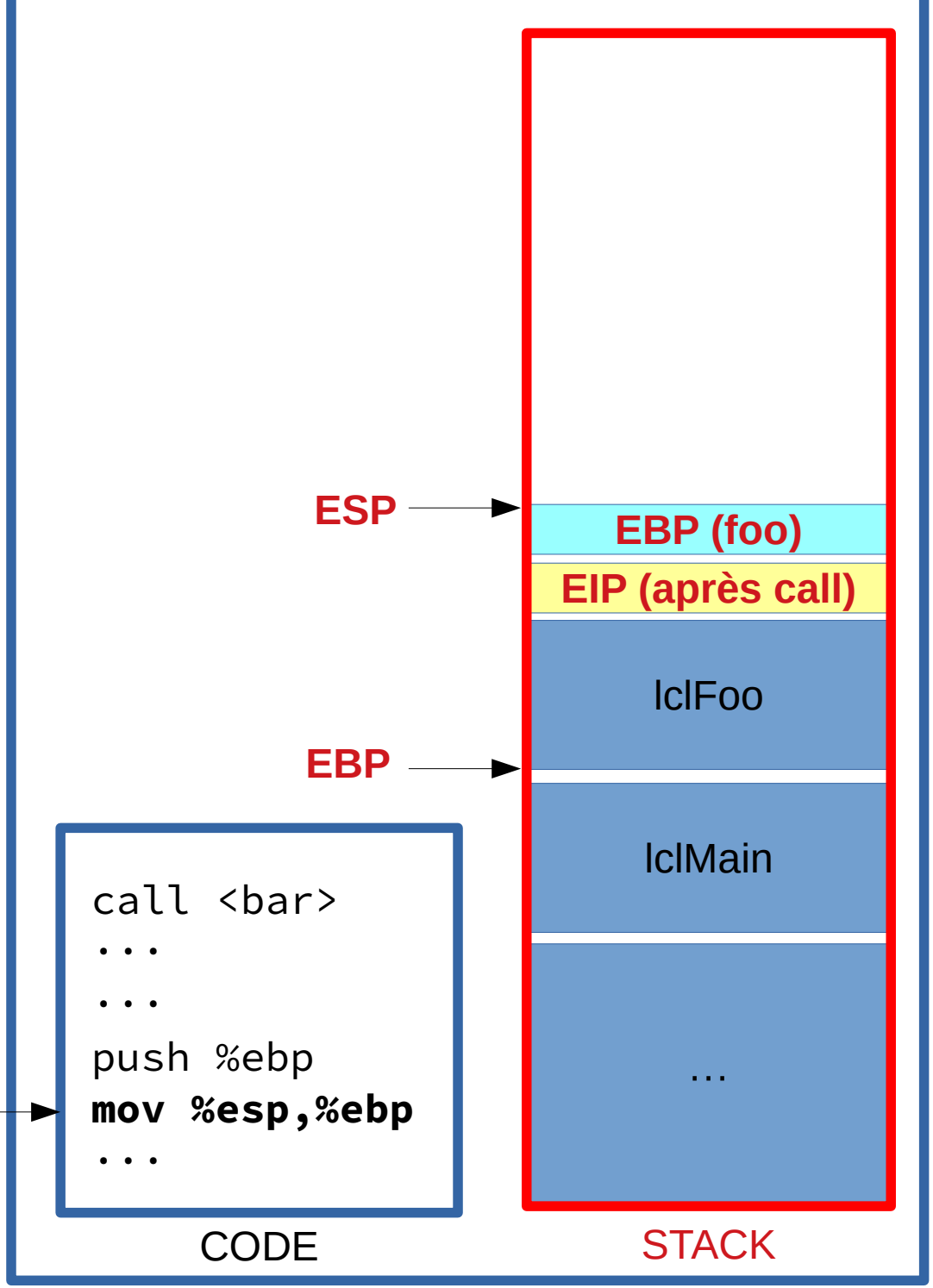
```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```



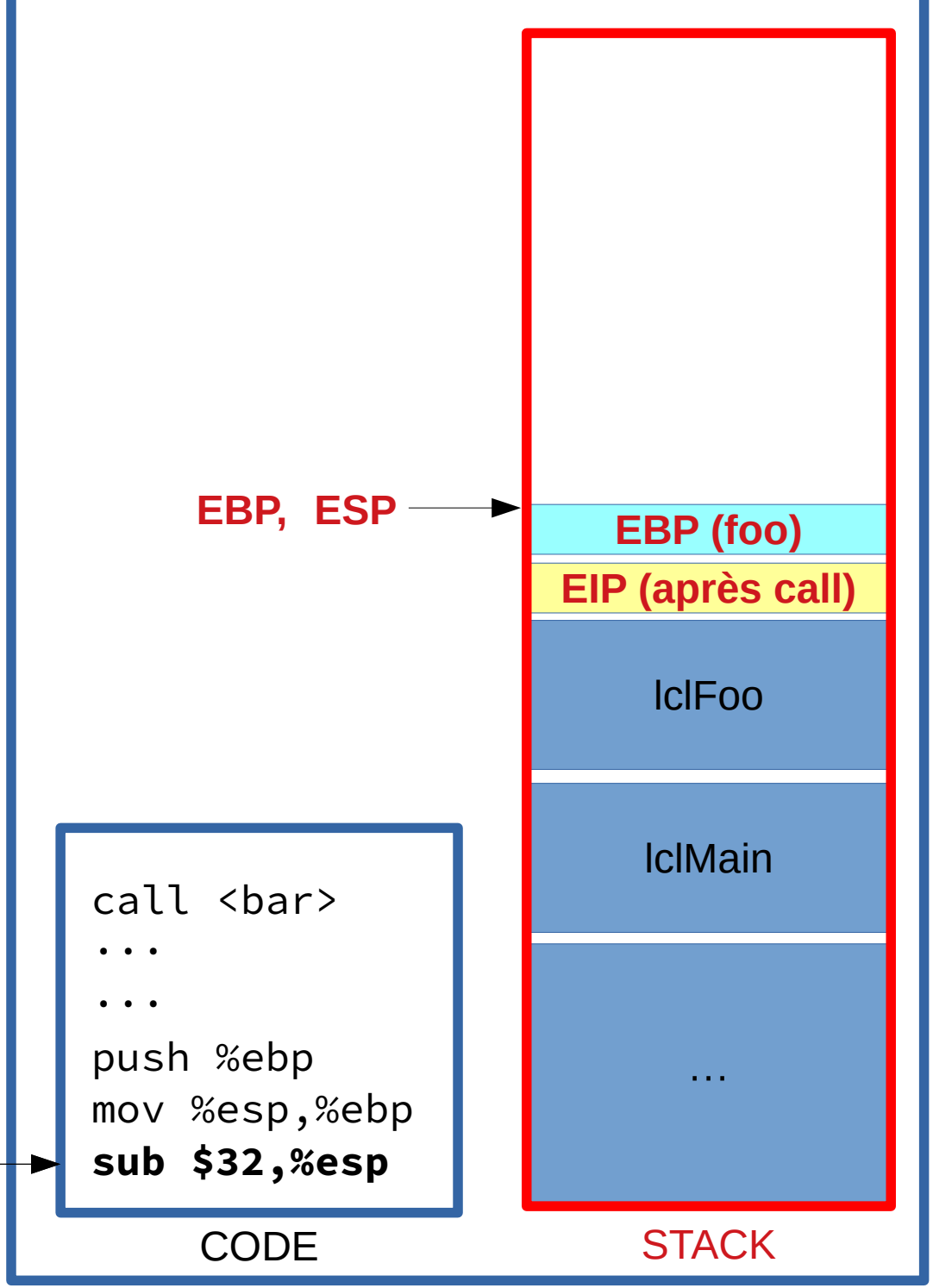
```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```



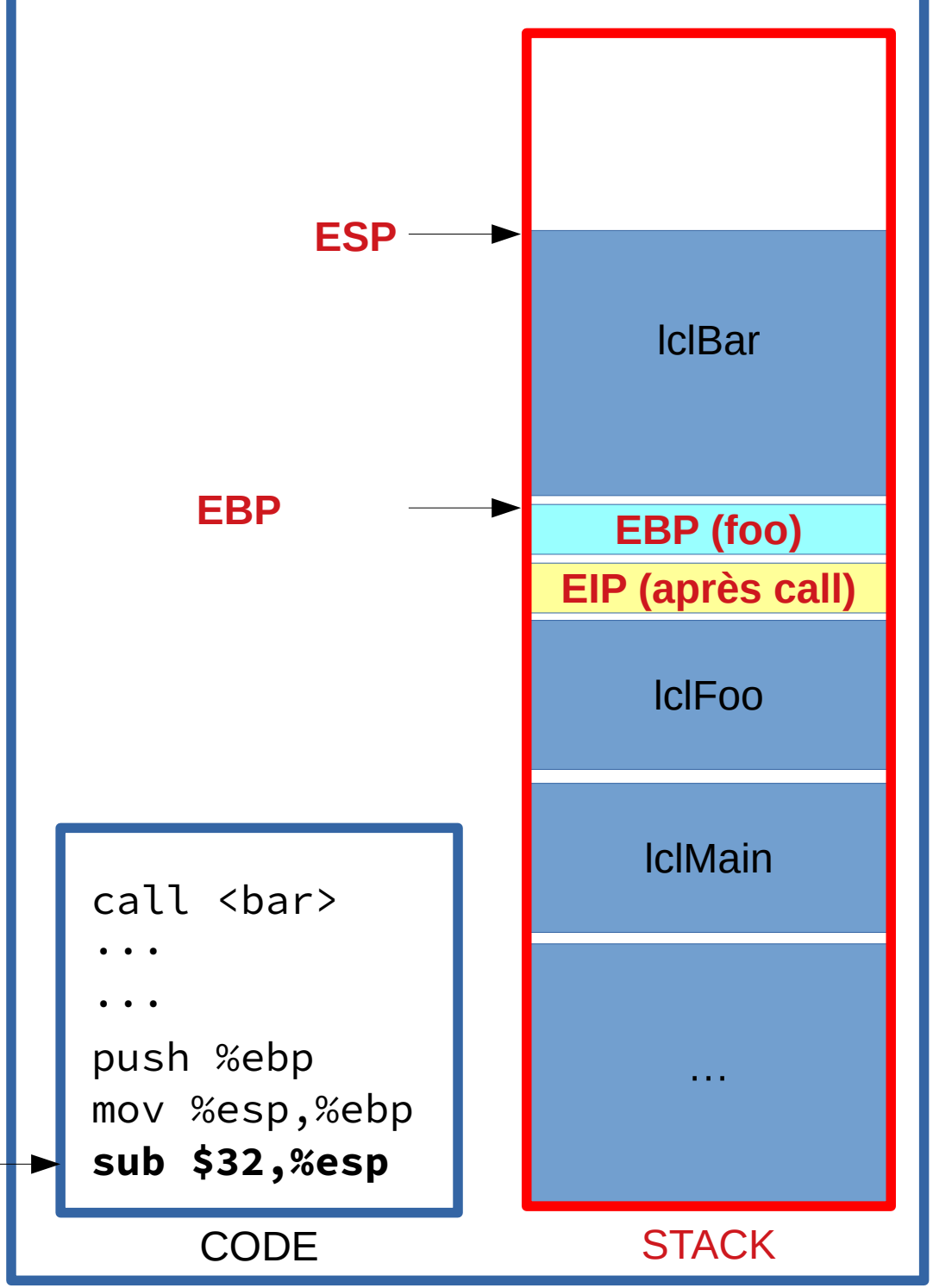
```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```



```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```



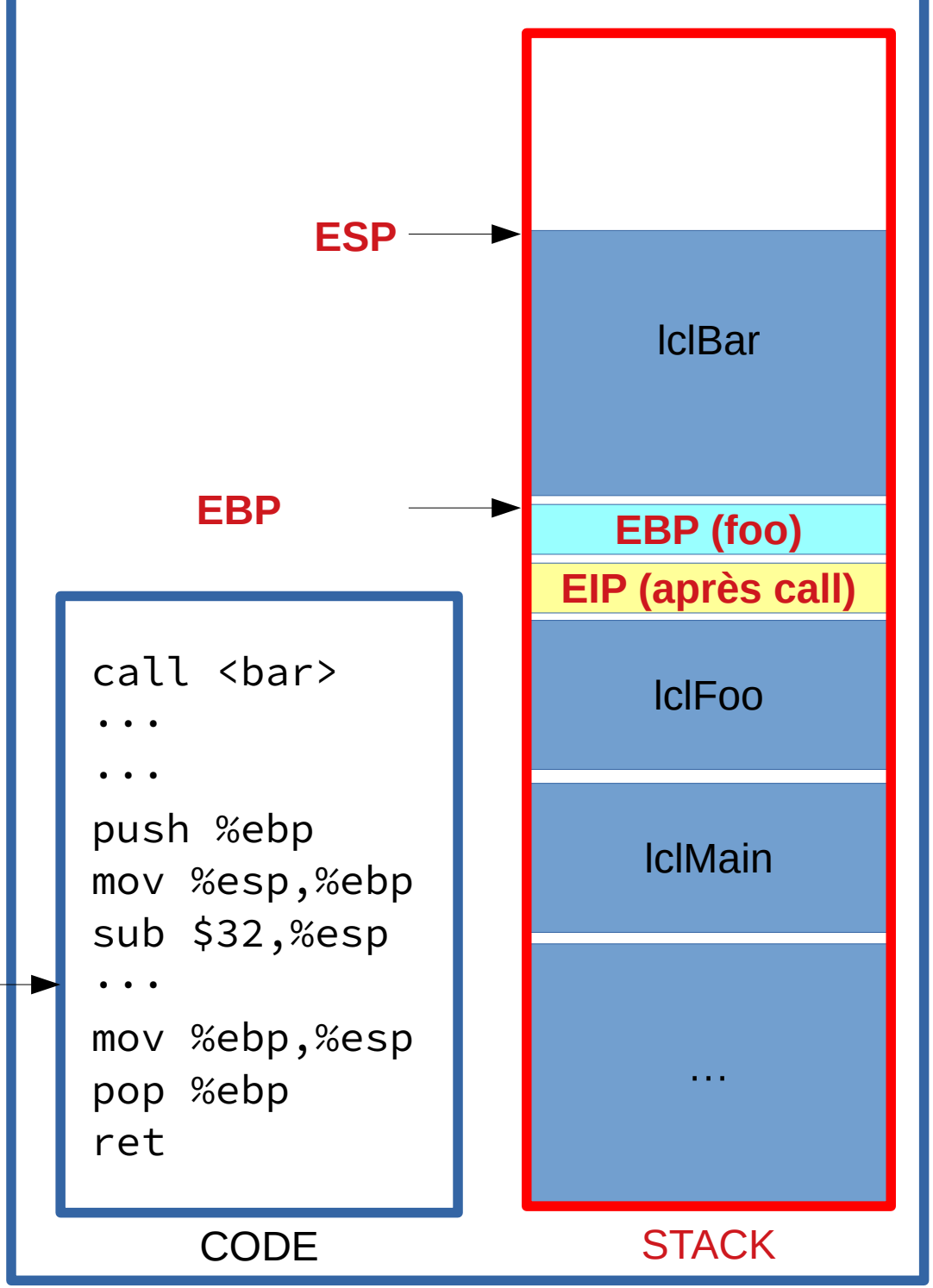
```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```



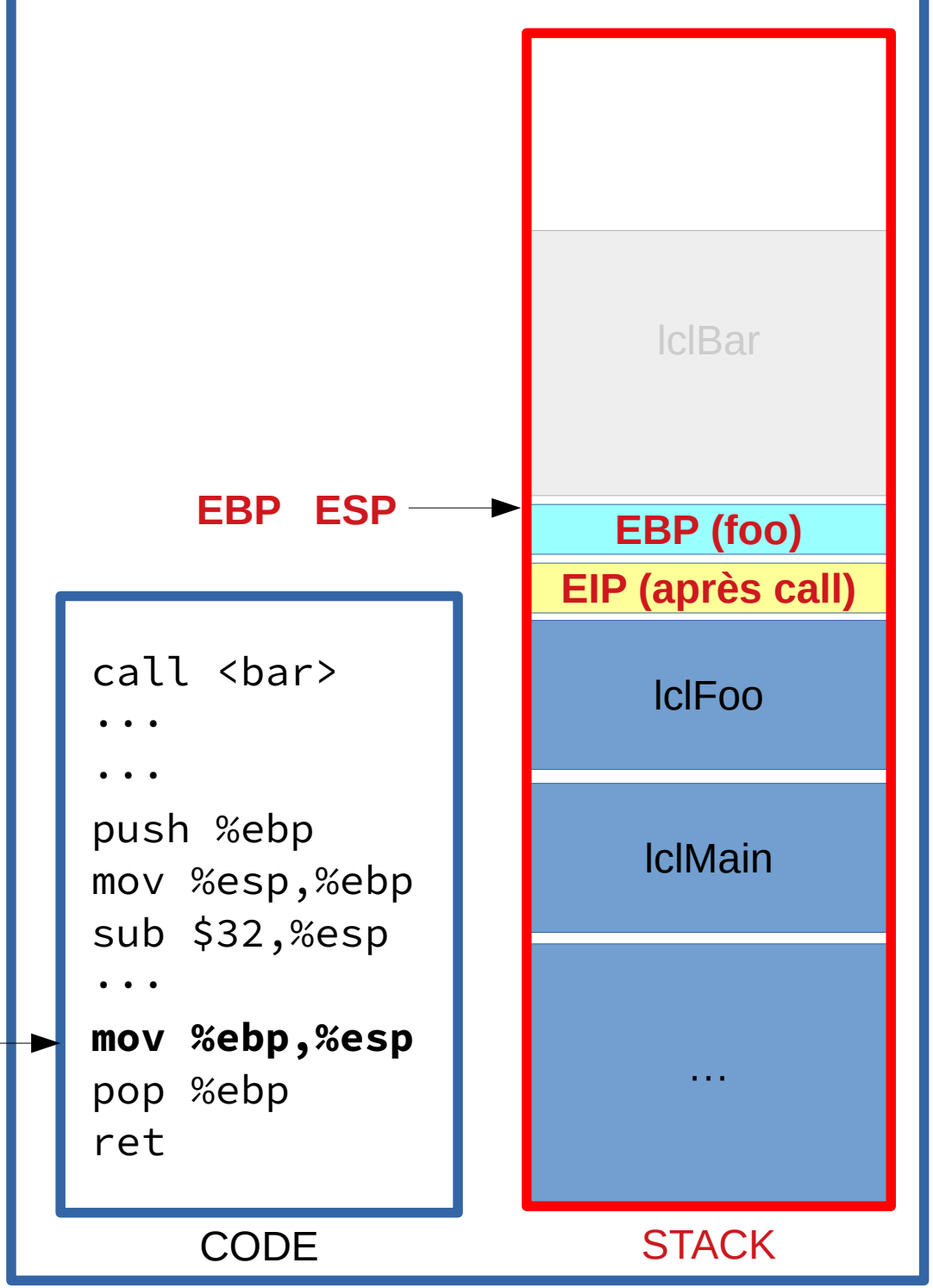
La Pile : Question 4

- Le retour...

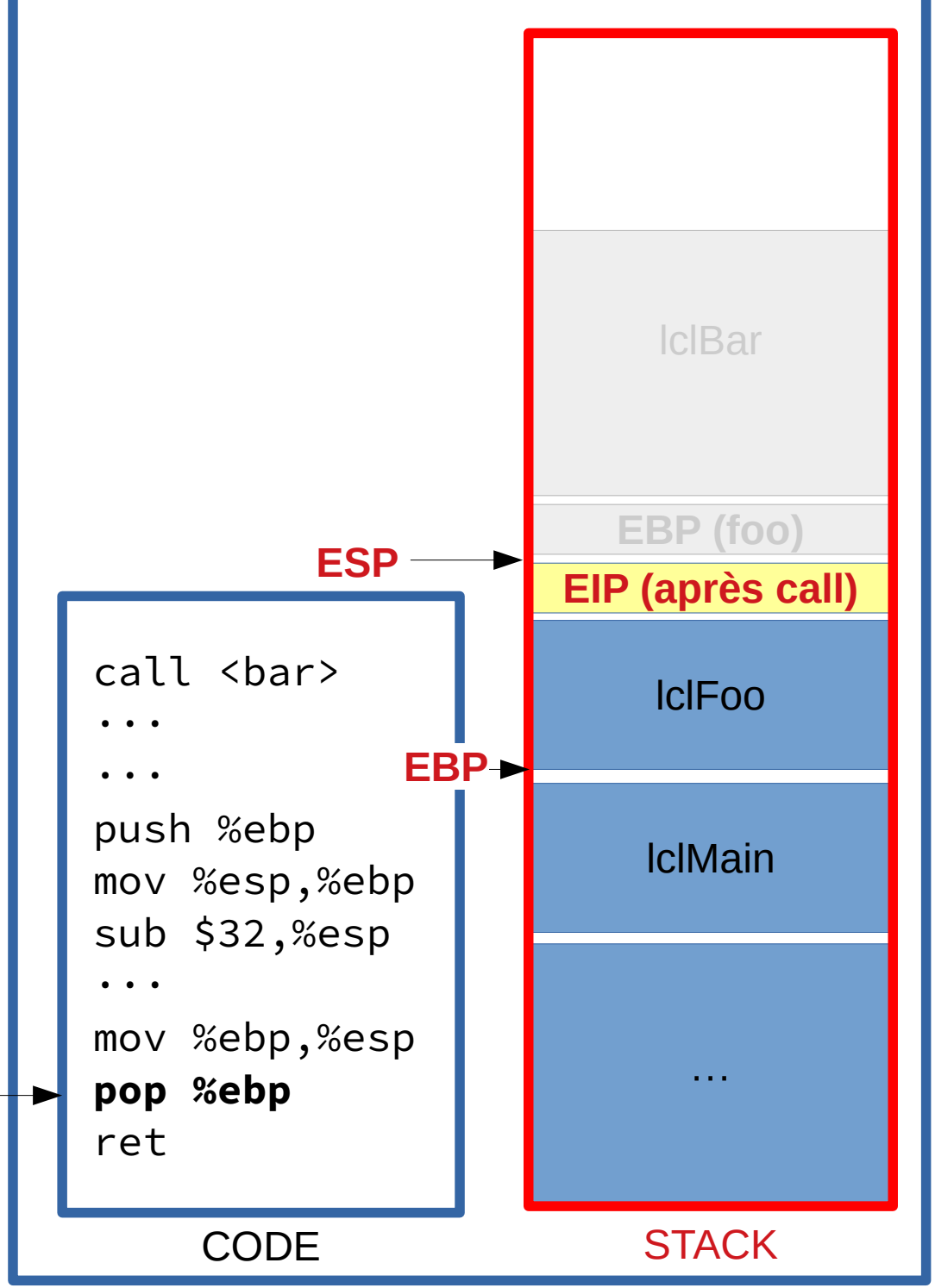
```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```



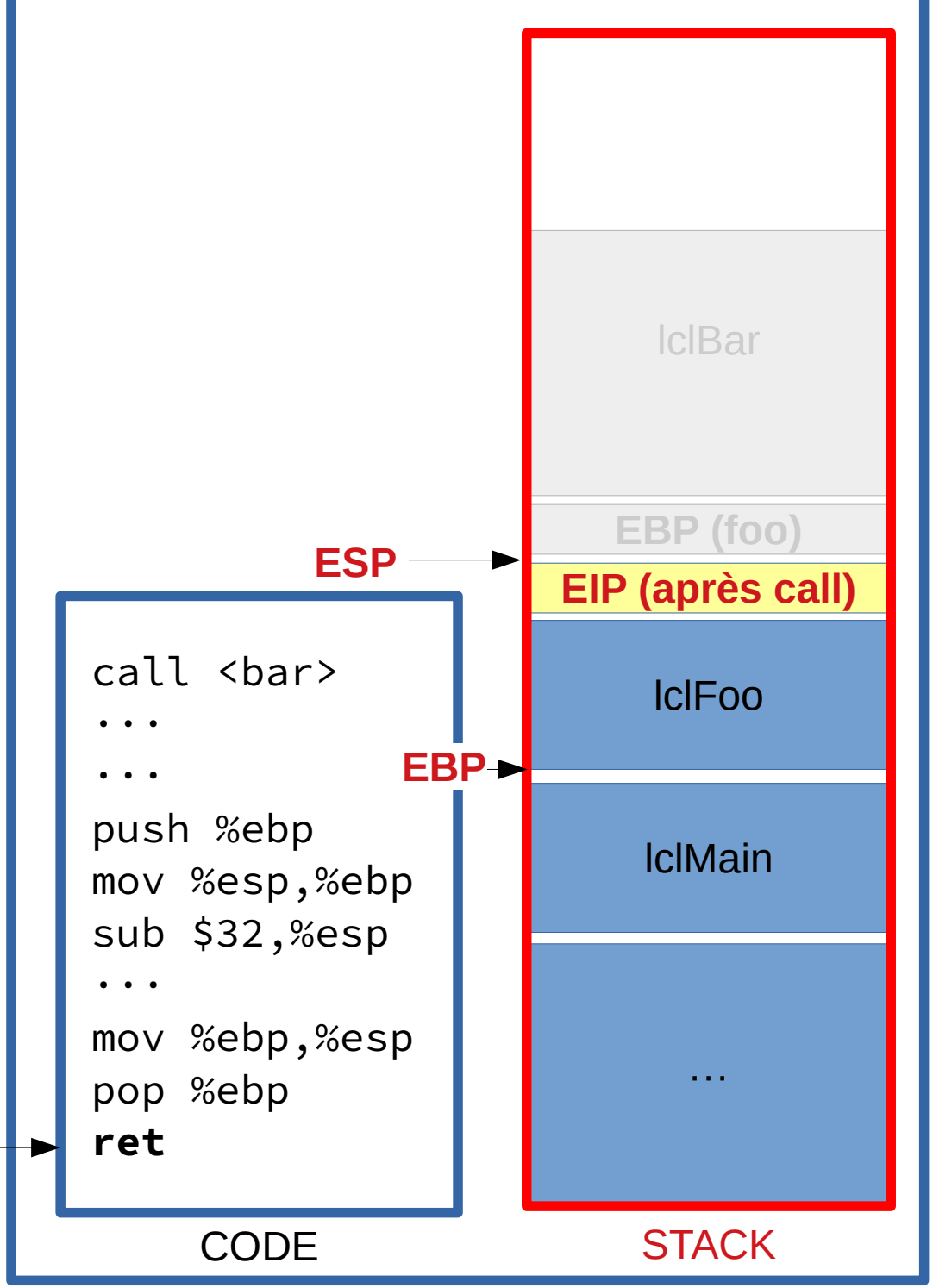
```
void foo() {
   int lclFoo = 10;
   bar();
}

void bar() {
   int lclBar = 20;
   printf("OK\n");
}
```



```
void foo() {
   int lclFoo = 10;
   bar();
}

void bar() {
   int lclBar = 20;
   printf("OK\n");
}
```



```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```



```
void foo() {
  int lclFoo = 10;
  bar();
}

void bar() {
  int lclBar = 20;
  printf("OK\n");
}
```