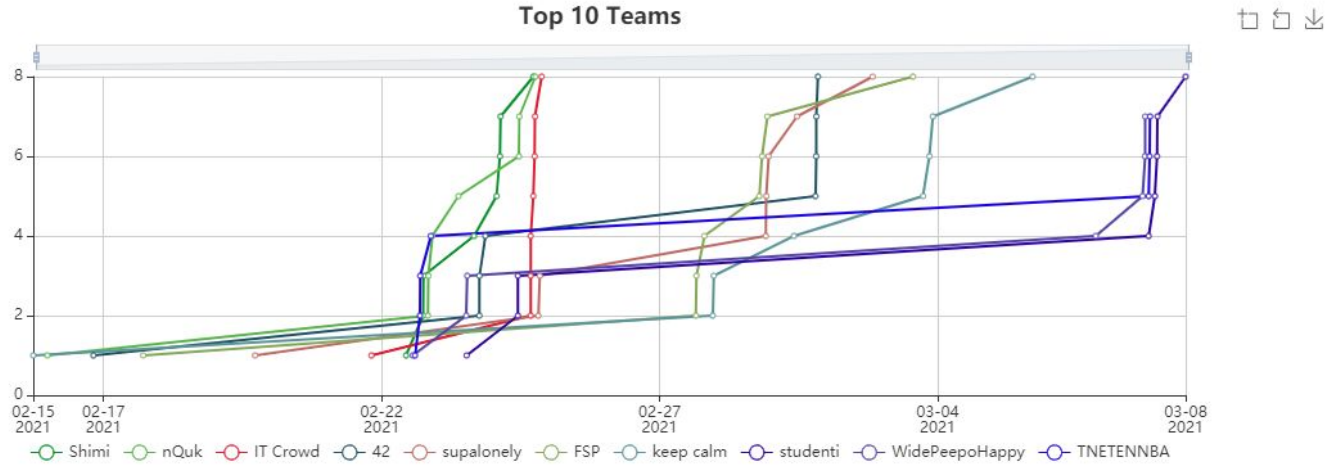
Reverzné inžinierstvo

Peter Švec
peter_svec@stuba.sk

Shellcoding výsledky



Place	Team	Score
1	Shimi	8
2	nQuk	8
3	IT Crowd	8

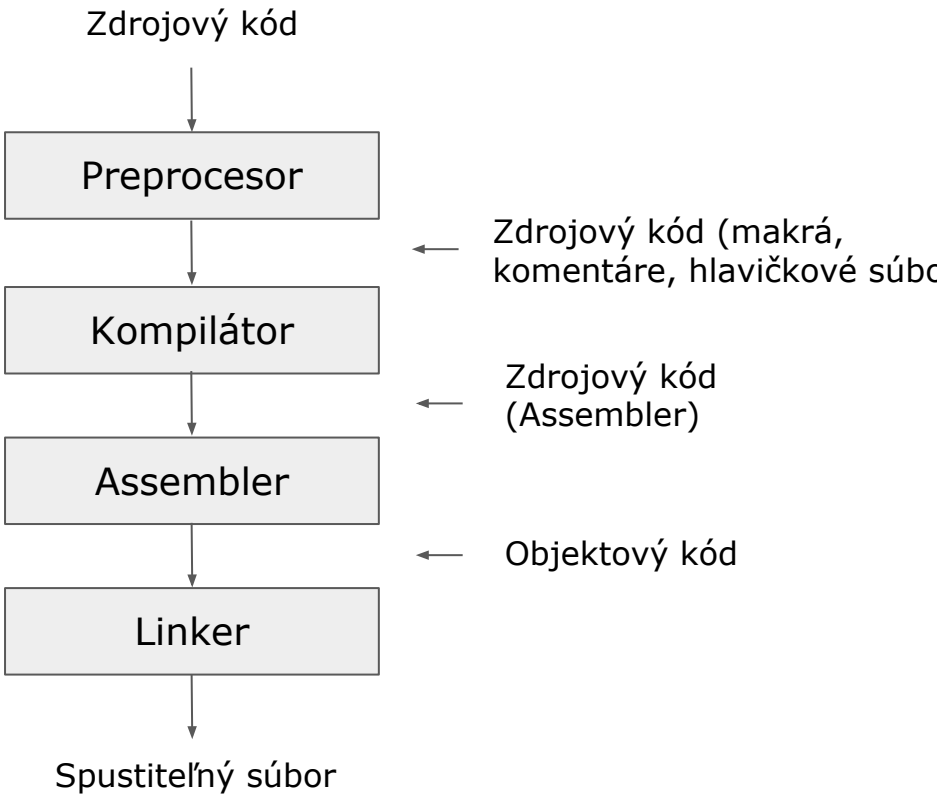
Štandardný proces

Zdrojový kód

```
#include <stdio.h>
int main(int argc, char *argv[])
{
        printf("FeiCTF\n");
        return 0
}
```

Spustiteľný súbor

```
00 0f ff 48 22 01 55 42 12 69 12 00
48 12 13 22 f5 a5 ...
```



Zostavovací proces

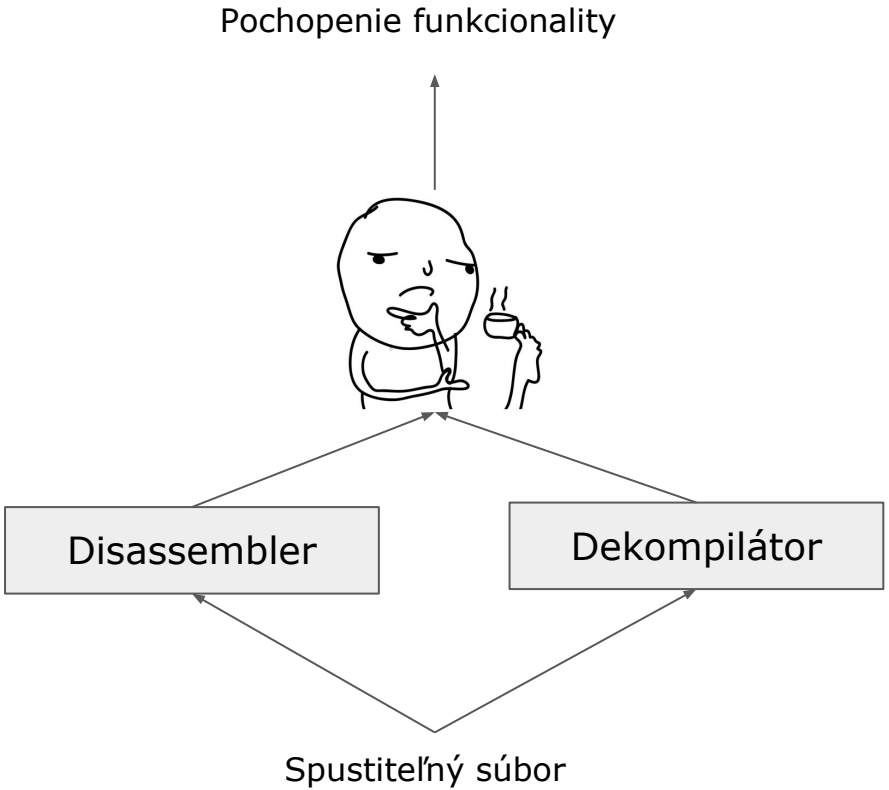
Počas zostavovacieho procesu sa množstvo informácií stráca:

- Názvy premenných
- Názvy funkcií
- Komentáre
- Štruktúry
- + Optimalizácie prekladača

Reverzné inžinierstvo

Opačný proces:

1. Máme binárny spustiteľný súbor.
2. Ako zistíme ako funguje a čo robí?



Nástroje

Statická analýza (bez spustenia):

- objdump, IDA, Ghidra, **BinaryNinja**

Dynamická analýza (so spustením):

- **strace**, **GDB**

https://cloud.binary.ninja/

Zásobník

LIFO dátová štruktúra používaná na kontrolu volania funkcie (call stack)

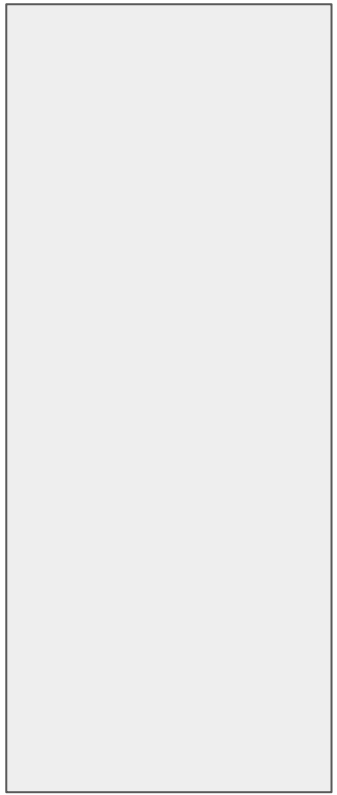
Na zásobník sa ukladajú:

- Lokálne premenné
- Návratová adresa (!!!)
- Base zásobníka z predchádzajúceho volania
- Pri veľkom množstve argumentov funkcie aj argumenty (v úlohach však budú postačovať registre)

!!! Zásobník rastie opačným smerom, od vysokých adries (0xfff...) po nižšie (0x000...) !!!

Zásobník

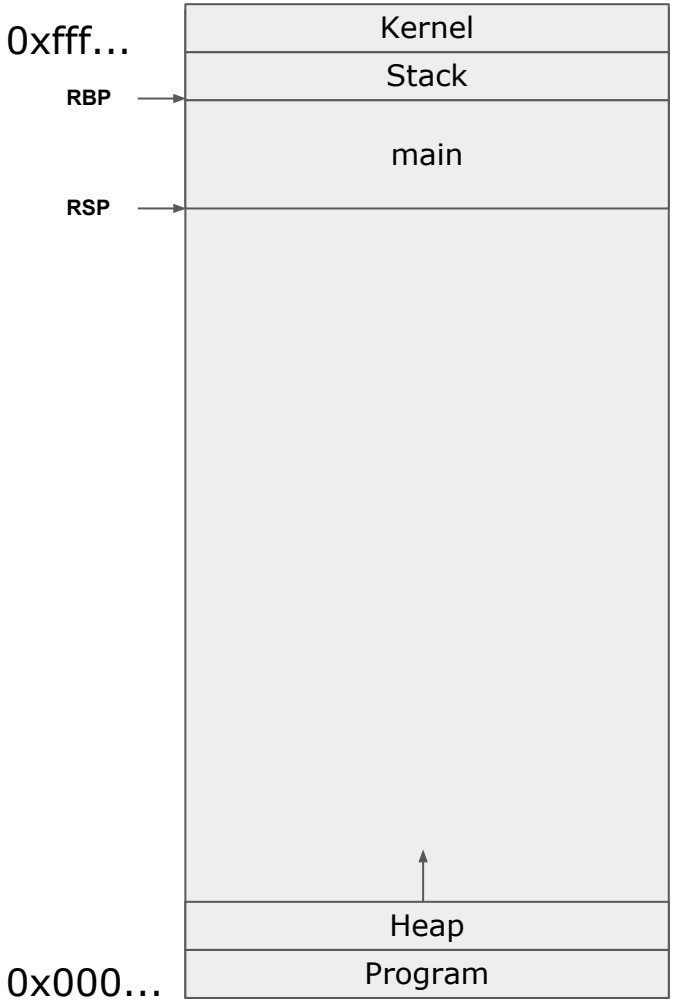
```
void foo()
{
        int a, b, c;
        b = 1; c = 2;
        a = b + c;
        return a;
}

int main(int argc, char *argv[])
{
        foo();
        return 0
}
```



Zásobník

```
void foo()
{
        int a, b, c;
        b = 1; c = 2;
        a = b + c;
        return a;
}

int main(int argc, char *argv[])
{
        foo();
        return 0
}
```

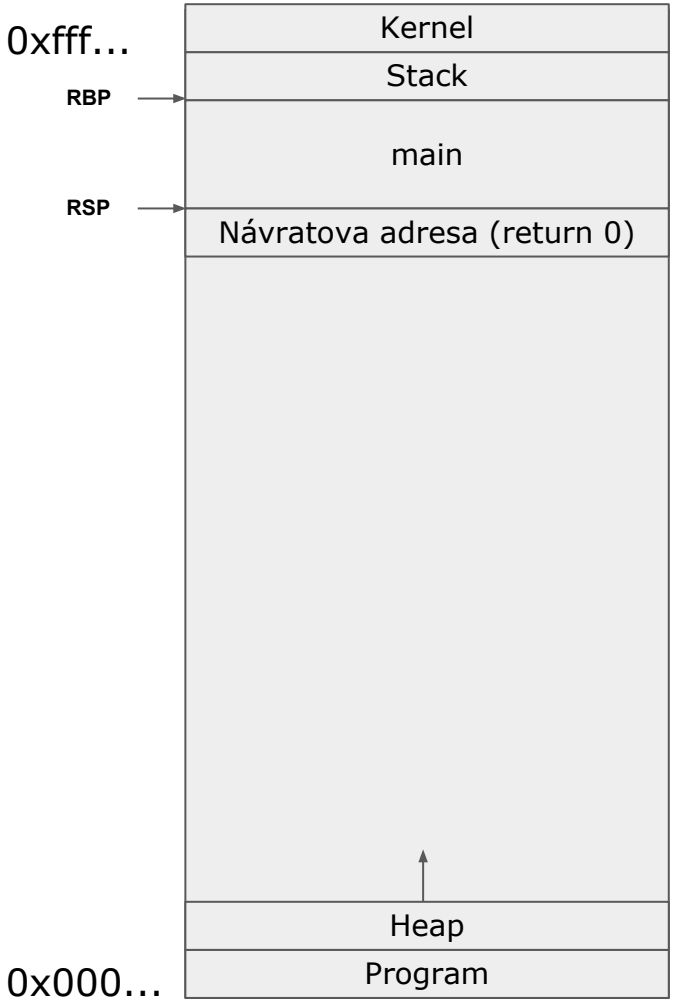
```
call foo
```



Zásobník

```
void foo()
{
	int a, b, c;
	b = 1; c = 2;
	a = b + c;
	return a;
}

int main(int argc, char *argv[])
{
	foo();
	return 0
}
```

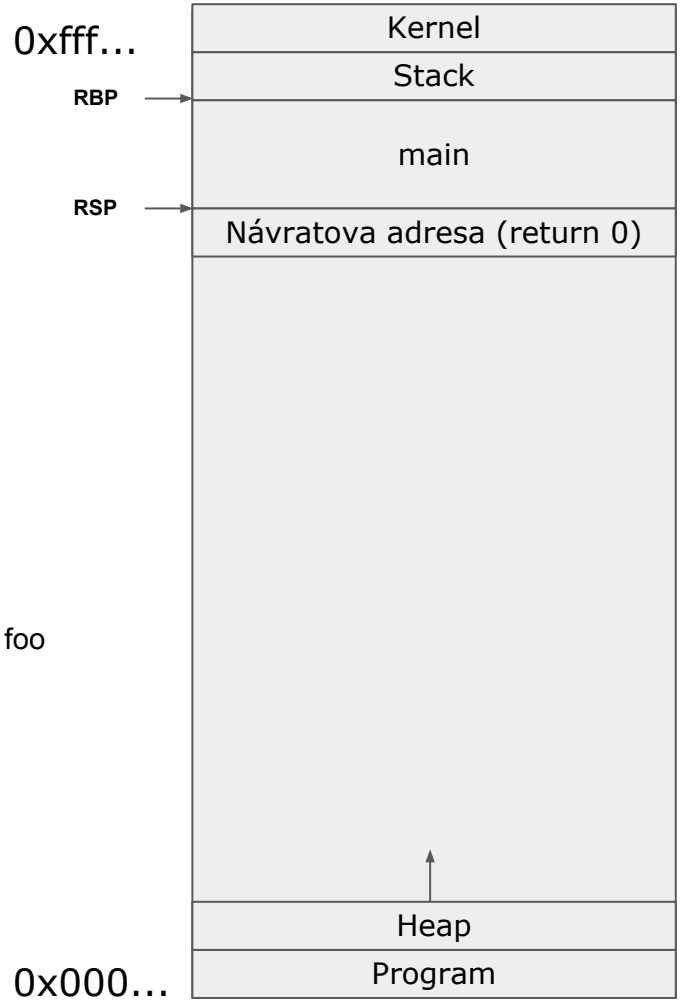
```
call foo
```



Zásobník

```
void foo()
{
        int a, b, c;
        b = 1; c = 2;
        a = b + c;
        return a;
}

int main(int argc, char *argv[])
{
        foo();
        return 0
}
```

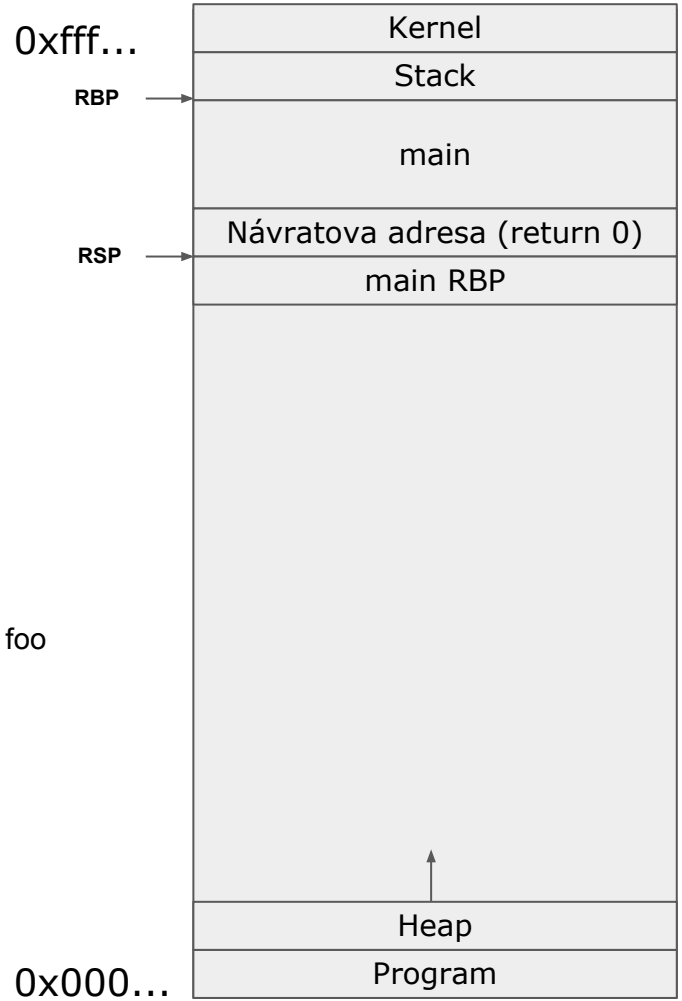
```
call foo
push rbp
```



Zásobník

```
void foo()
{
        int a, b, c;
        b = 1; c = 2;
        a = b + c;
        return a;
}

int main(int argc, char *argv[])
{
        foo();
        return 0
}
```

```
call foo
push rbp
```

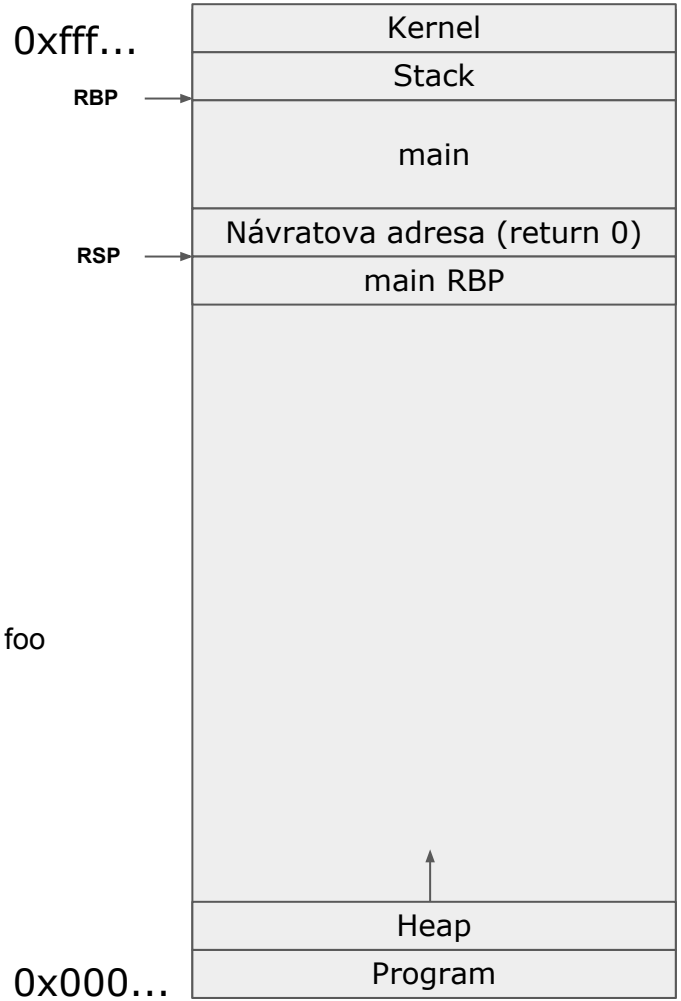


Zásobník

```
void foo()
{
        int a, b, c;
        b = 1; c = 2;
        a = b + c;
        return a;
}

int main(int argc, char *argv[])
{
        foo();
        return 0
}
```

```
call foo
push rbp
mov rbp, rsp
```

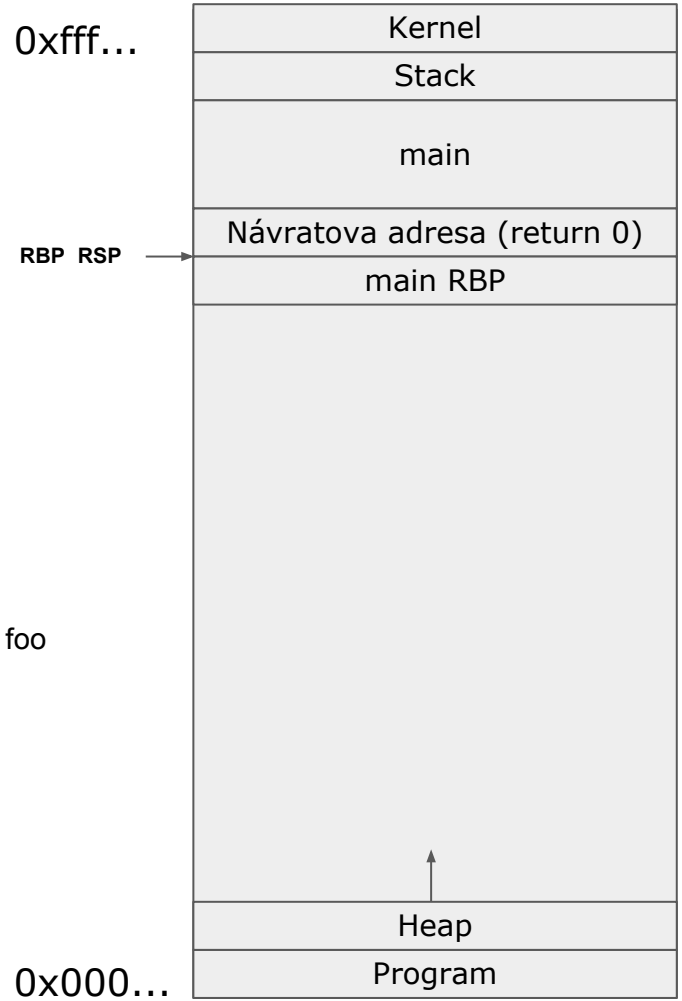
foo



Zásobník

```
void foo()
{
        int a, b, c;
        b = 1; c = 2;
        a = b + c;
        return a;
}

int main(int argc, char *argv[])
{
        foo();
        return 0
}
```

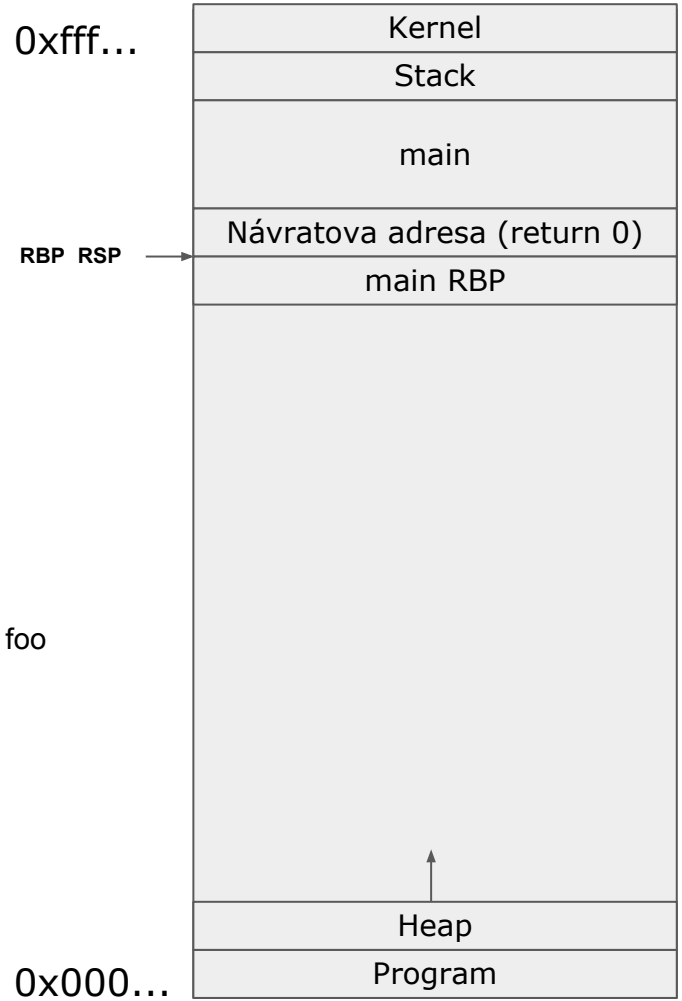
```
call foo
push rbp
mov rbp, rsp
```



Zásobník

```
void foo()
{
        int a, b, c;
        b = 1; c = 2;
        a = b + c;
        return a;
}

int main(int argc, char *argv[])
{
        foo();
        return 0
}
```

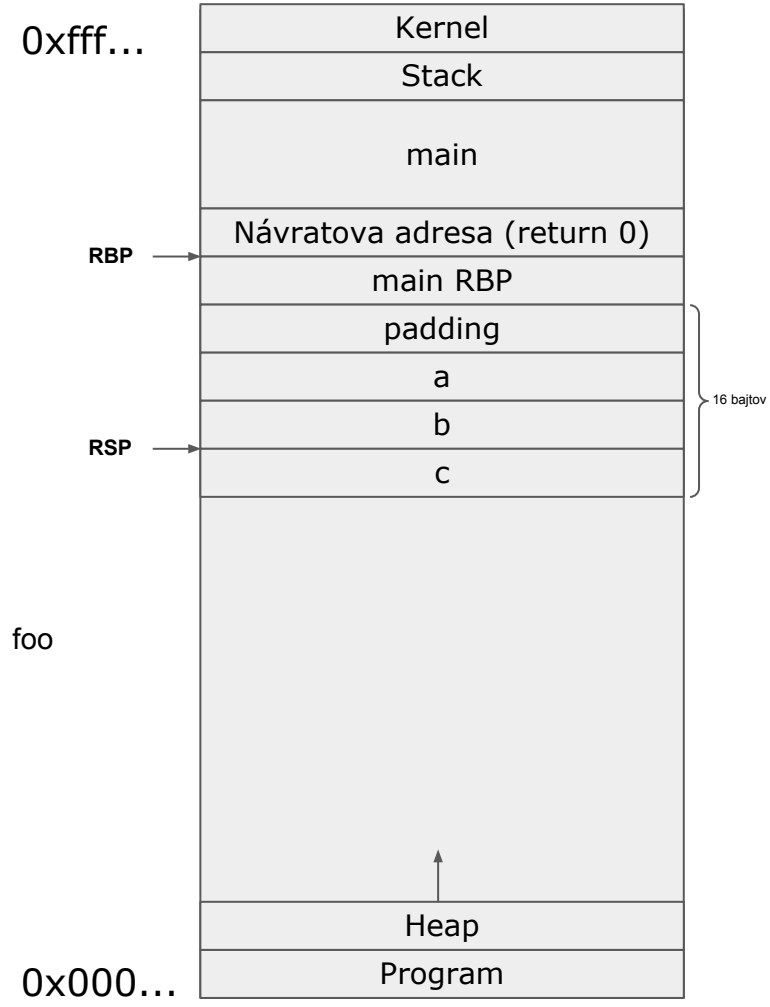
```
call foo
push rbp
mov rbp, rsp
sub rsp, 0x10
```



Zásobník

```
void foo()
{
        int a, b, c;
        b = 1; c = 2;
        a = b + c;
        return a;
}

int main(int argc, char *argv[])
{
        foo();
        return 0
}
```

```
call foo
push rbp
mov rbp, rsp
sub rsp, 0x10
```

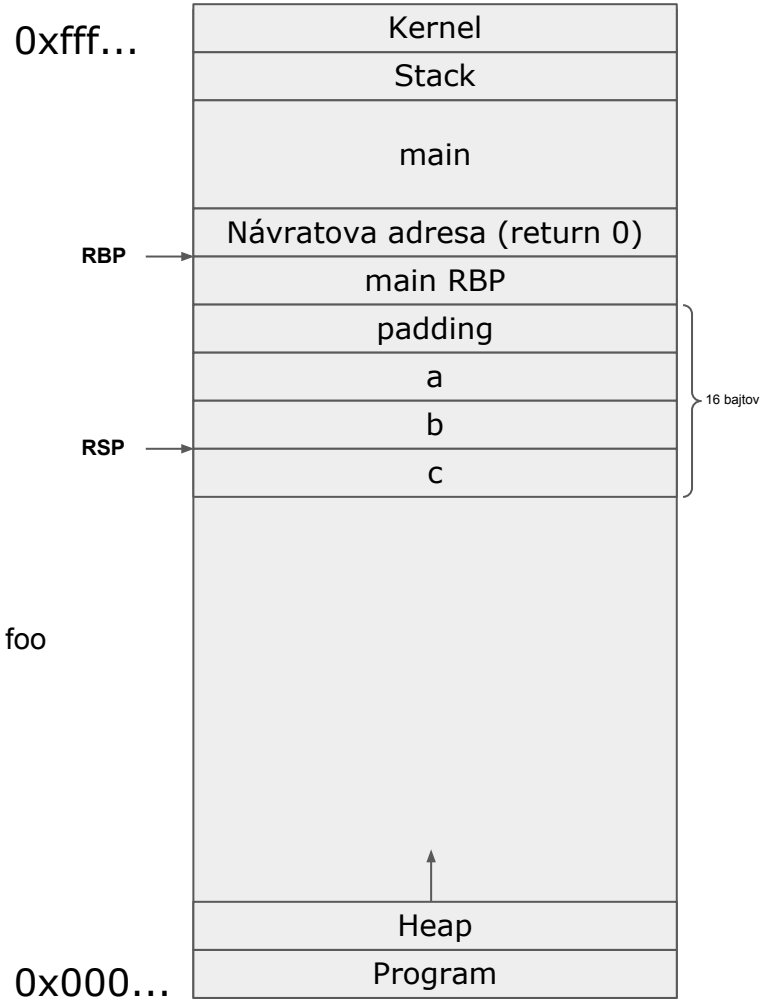


Zásobník

```
void foo()
{
        int a, b, c;
        b = 1; c = 2;
        a = b + c;
        return a;
}

int main(int argc, char *argv[])
{
        foo();
        return 0
}
```

```
call foo
push rbp
mov rbp, rsp
sub rsp, 0x10
mov [rbp-8], 1
mov [rbp-12], 2
```

foo

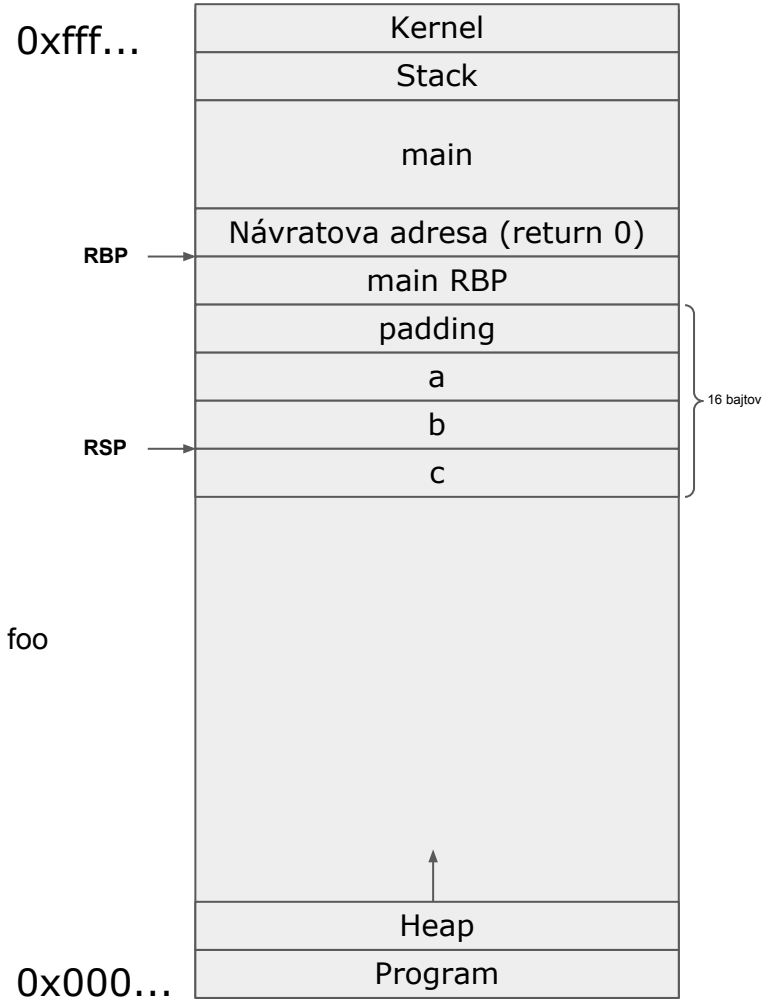


Zásobník

```
void foo()
{
        int a, b, c;
        b = 1; c = 2;
        a = b + c;
        return a;
}

int main(int argc, char *argv[])
{
        foo();
        return 0
}
```

```
call foo
push rbp
mov rbp, rsp
sub rsp, 0x10
mov [rbp-8], 1
mov [rbp-12], 2
...
výpočty
...
```

foo

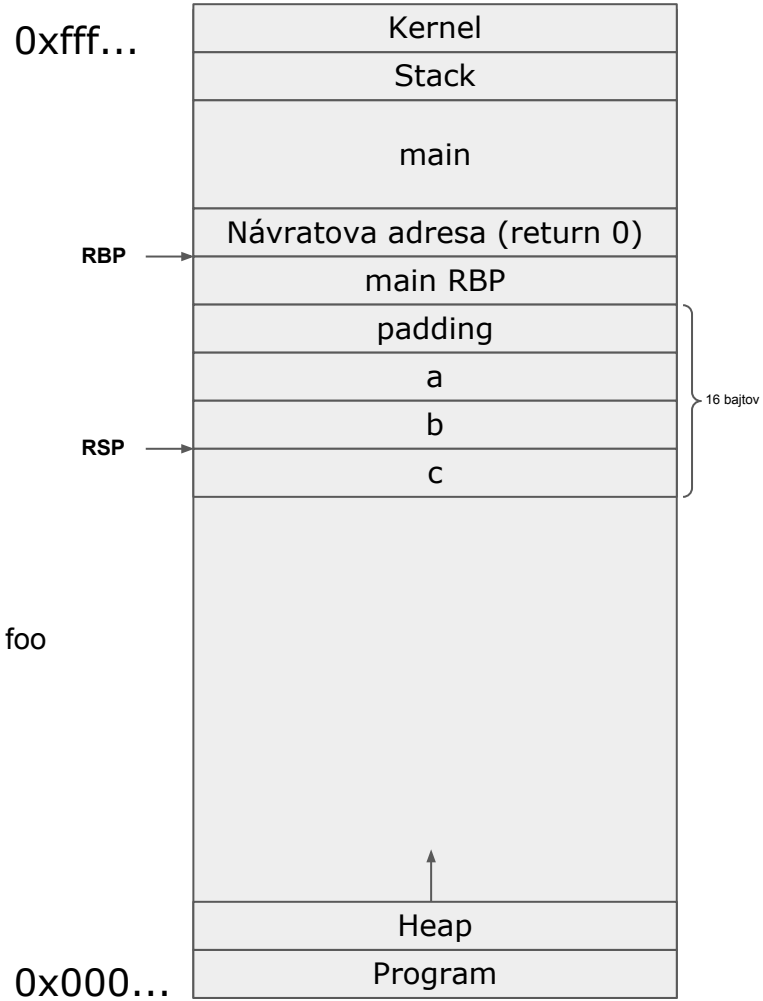


Zásobník

```
void foo()
{
	int a, b, c;
	b = 1; c = 2;
	a = b + c;
	return a;
}

int main(int argc, char *argv[])
{
	foo();
	return 0
}
```

```
call foo
push rbp
mov rbp, rsp
sub rsp, 0x10
mov [rbp-8], 1
mov [rbp-12], 2
...
výpočty
...

mov rax, [rbp-4]
```

foo

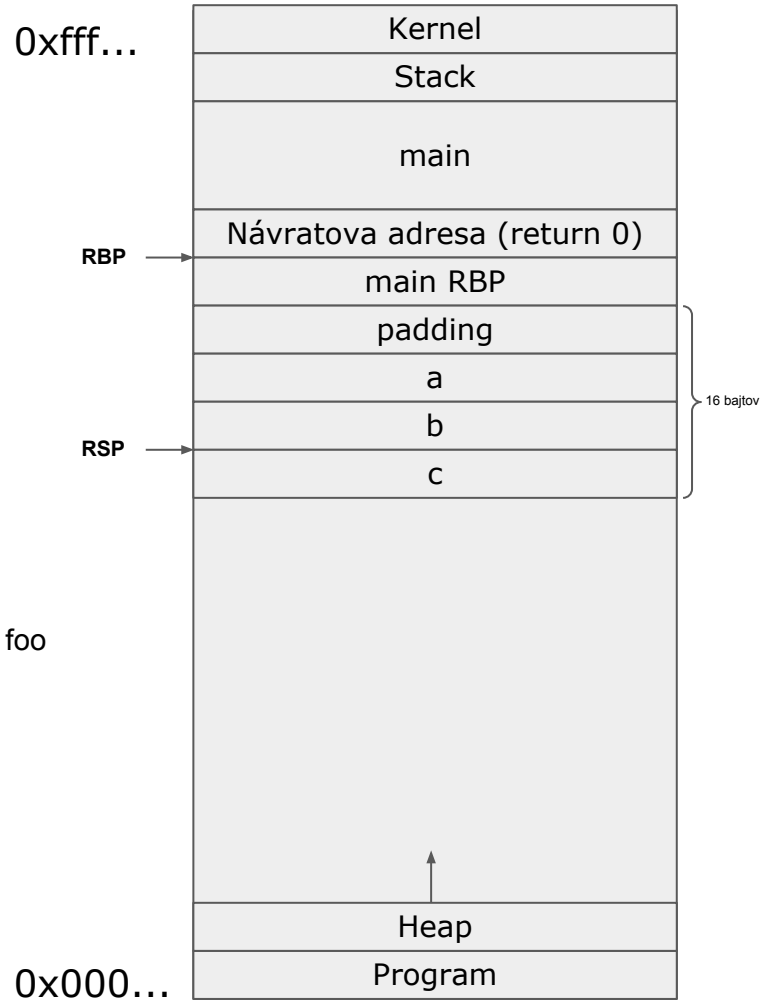


Zásobník

```
void foo()
{
        int a, b, c;
        b = 1; c = 2;
        a = b + c;
        return a;
}

int main(int argc, char *argv[])
{
        foo();
        return 0
}
```

```
call foo
push rbp
mov rbp, rsp
sub rsp, 0x10
mov [rbp-8], 1
mov [rbp-12], 2
...
výpočty
...

mov rax, [rbp-4]
mov rsp, rbp
```

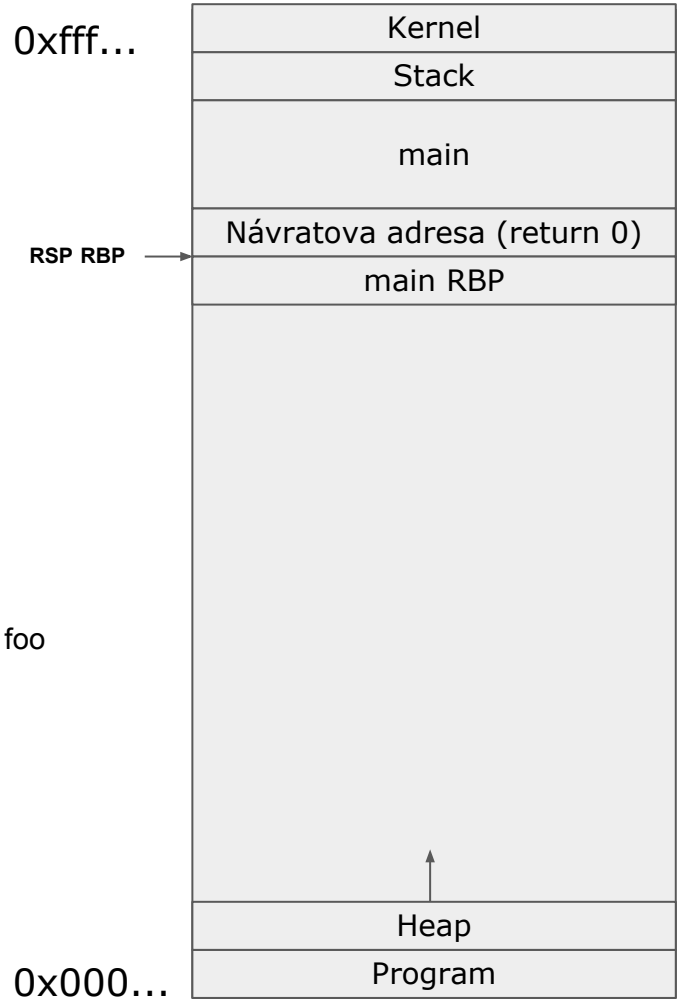
foo



Zásobník

```
void foo()
{
        int a, b, c;
        b = 1; c = 2;
        a = b + c;
        return a;
}

int main(int argc, char *argv[])
{
        foo();
        return 0
}
```

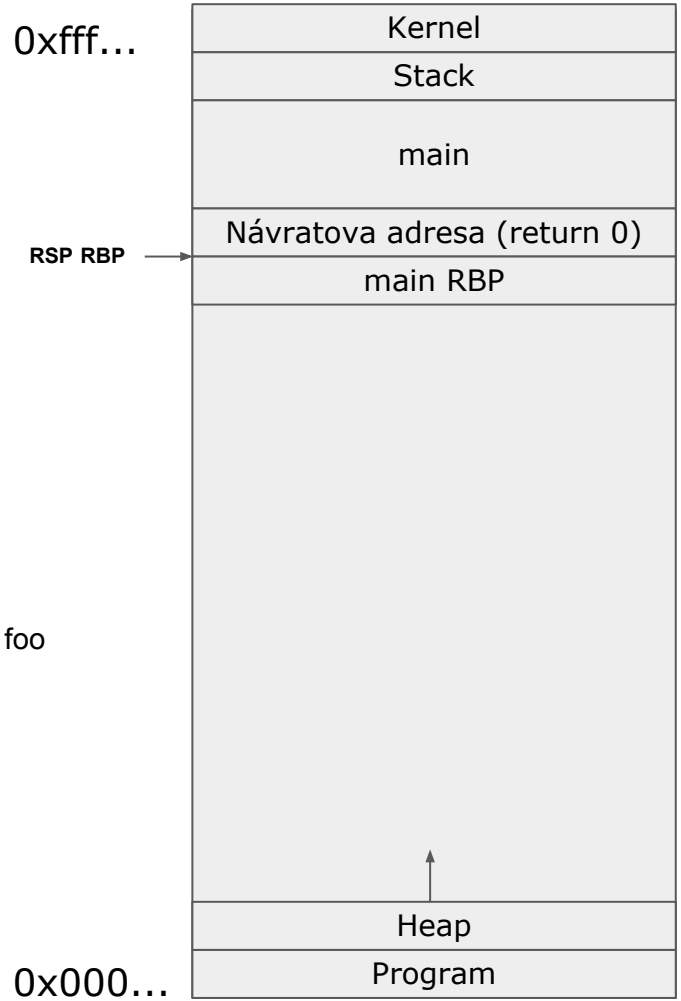
```
call foo
push rbp
mov rbp, rsp
sub rsp, 0x10
mov [rbp-8], 1
mov [rbp-12], 2
...
výpočty
...

mov rax, [rbp-4]
mov rsp, rbp
```



Zásobník

```
void foo()
{
        int a, b, c;
        b = 1; c = 2;
        a = b + c;
        return a;
}

int main(int argc, char *argv[])
{
        foo();
        return 0
}
```

```
call foo
push rbp
mov rbp, rsp
sub rsp, 0x10
mov [rbp-8], 1
mov [rbp-12], 2
...
výpočty
...

mov rax, [rbp-4]
mov rsp, rbp
pop rbp
```

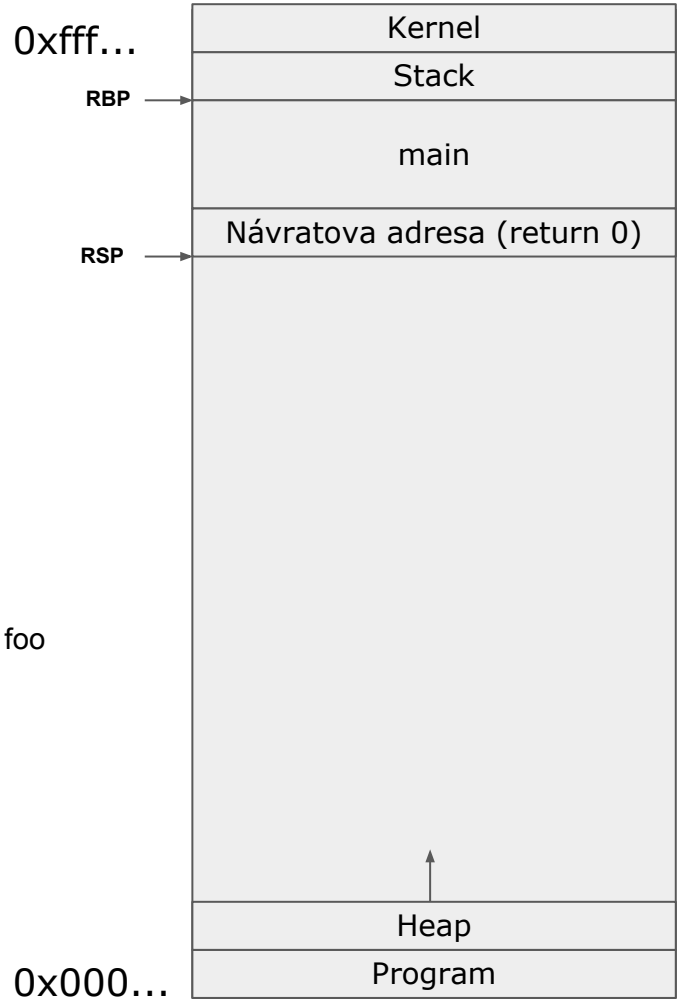


Zásobník

```
void foo()
{
        int a, b, c;
        b = 1; c = 2;
        a = b + c;
        return a;
}

int main(int argc, char *argv[])
{
        foo();
        return 0
}
```

```
call foo
push rbp
mov rbp, rsp
sub rsp, 0x10
mov [rbp-8], 1
mov [rbp-12], 2
...
výpočty
...

mov rax, [rbp-4]
mov rsp, rbp
pop rbp
```

foo



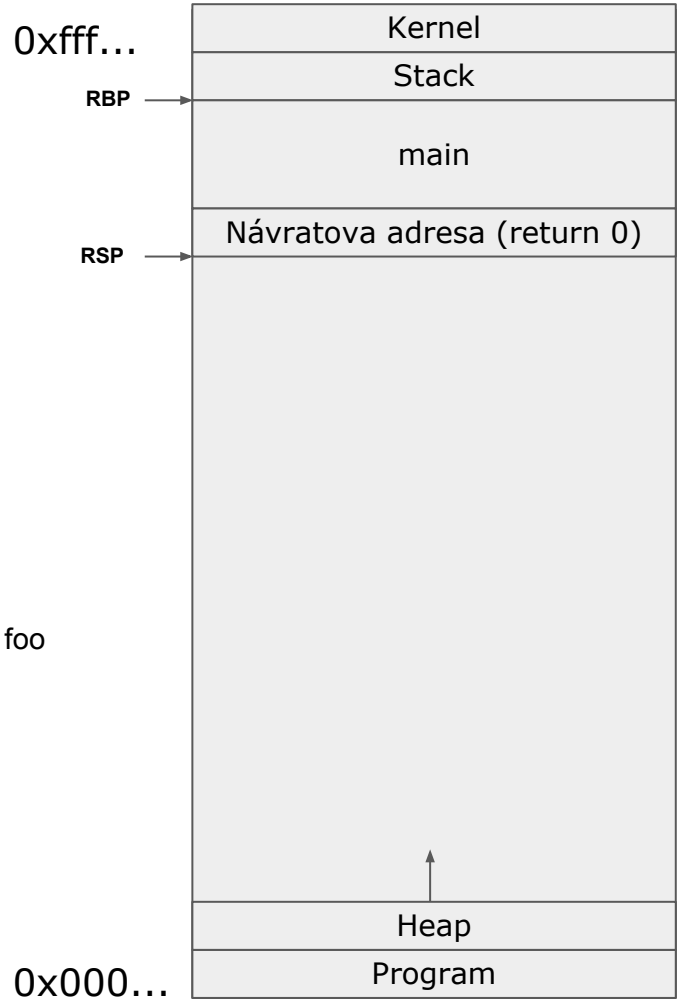
Zásobník

```
void foo()
{
        int a, b, c;
        b = 1; c = 2;
        a = b + c;
        return a;
}

int main(int argc, char *argv[])
{
        foo();
        return 0
}
```

ret - načíta do RIP vrch zásobníka

```
call foo
push rbp
mov rbp, rsp
sub rsp, 0x10
mov [rbp-8], 1
mov [rbp-12], 2
...
výpočty
...

mov rax, [rbp-4]
mov rsp, rbp
pop rbp
ret
```

foo



Zásobník

```
void foo()
{
        int a, b, c;
        b = 1; c = 2;
        a = b + c;
        return a;
}

int main(int argc, char *argv[])
{
        foo();
        return 0
}
```

ret - načíta do RIP vrch zásobníka

```
call foo
push rbp
mov rbp, rsp
sub rsp, 0x10
mov [rbp-8], 1
mov [rbp-12], 2
...
výpočty
...

mov rax, [rbp-4]
mov rsp, rbp
pop rbp
ret
```

foo