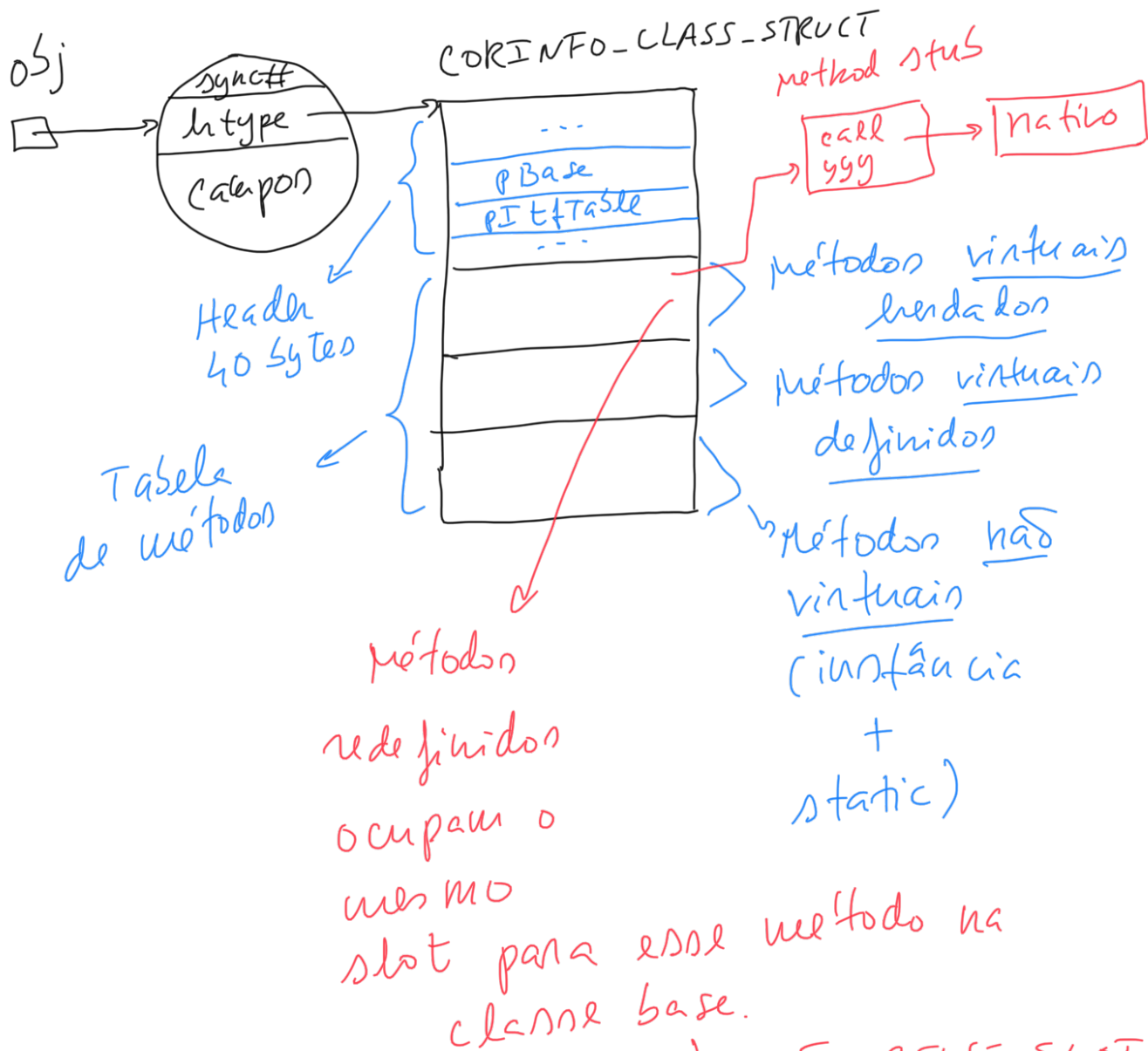


# Tabela de métodos

1.



↓ opção REUSE SLOT em Reflection.Emit

2.

Atributos do métodos em C#:

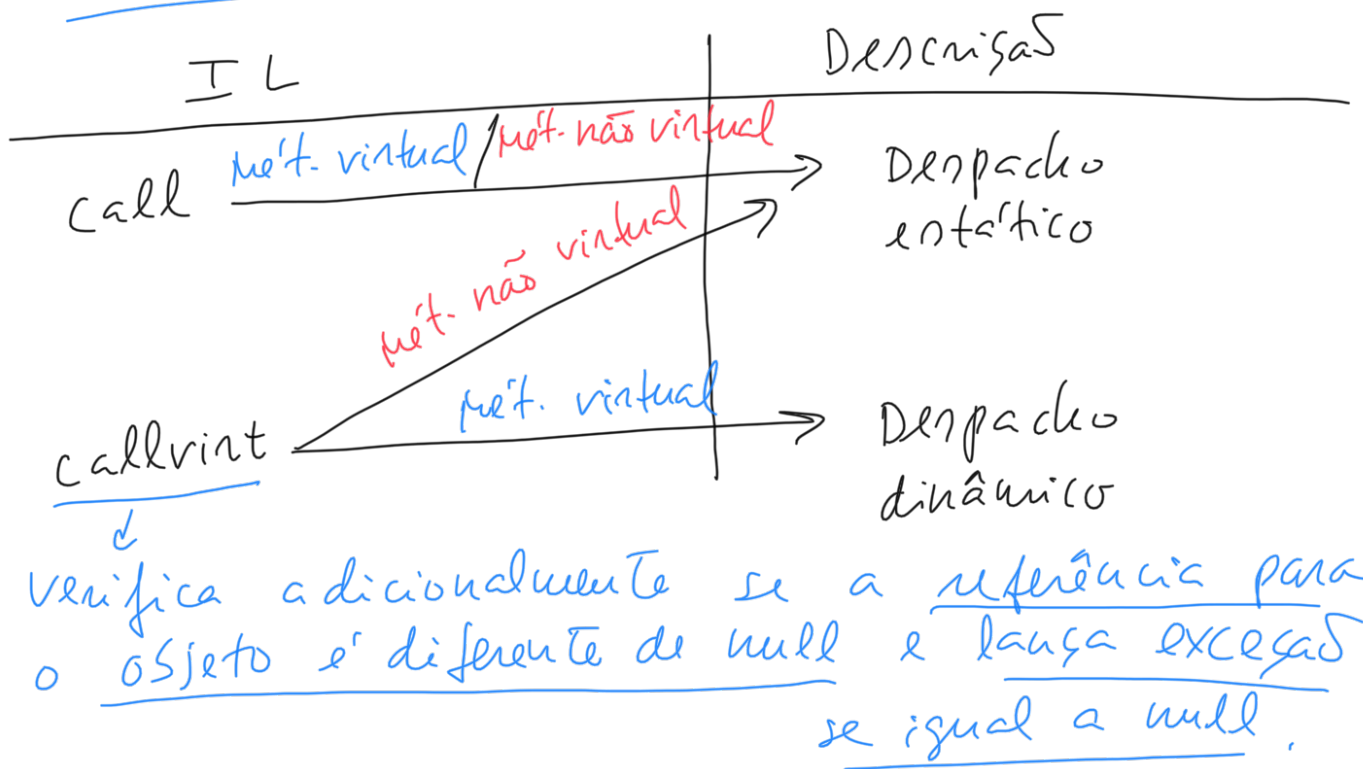
- virtual, abstract e override ⇒ o método é virtual
- Se o método não contiver um dos atributos acima, então o método é não virtual, podendo ser de instância ou estático.
- Se o método é virtual, pode ser invocado de forma polimórfica (despacho dinâmico)
- Métodos de Object:
  - virtual: ToString, Equals, GetHashCode, e Finalize
  - não virtual: GetType

## Invocações de métodos

3.

- Despacho dinâmico: chama-se a um método com comportamento polimórfico (métodos virtuais)
- Despacho estático: usado na invocação de métodos não virtuais

Contudo, um método virtual pode ser invocado de forma não virtual (despacho estático).



4.

O compilador de C# gera tipicamente um callvirt na chamada a métodos de instância (virtual e nas virtual)

⇒ Despacho dinâmico

P: Em que situações o compilador de C# gera um call na chamada a métodos de instância nas virtual?

R: Quando tem a certeza de que a referência para o objeto é diferente de null, por exemplo, na invocação de um método de instância dentro de um construtor, ou dentro de outro método de instância.

5.

```
class C {
```

```
...
```

```
public C() {
```

```
    m1(); // call m1
```

```
}
```

```
public m1() { ... }
```

```
public m2() { m1(); }
```

↳ call m1

```
}
```

Main:

```
C obj = new C();
```

```
obj.m2(); // call m2
```

↳ para verificar se obj

é ≠ null, mas

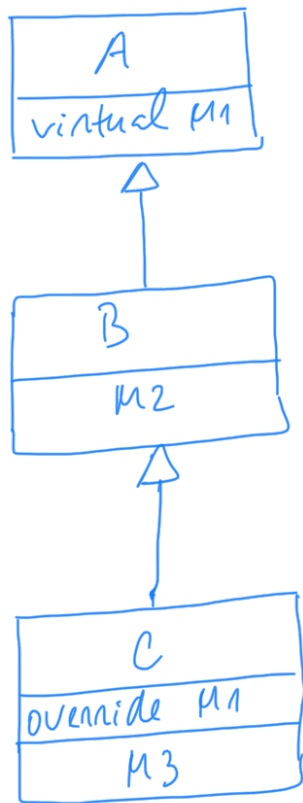
realiza despacho

entático.

6.

## Despacho dinâmico

O método virtual posto em execução é selecionado com base no tipo exato do objeto e não com base no tipo da referência.



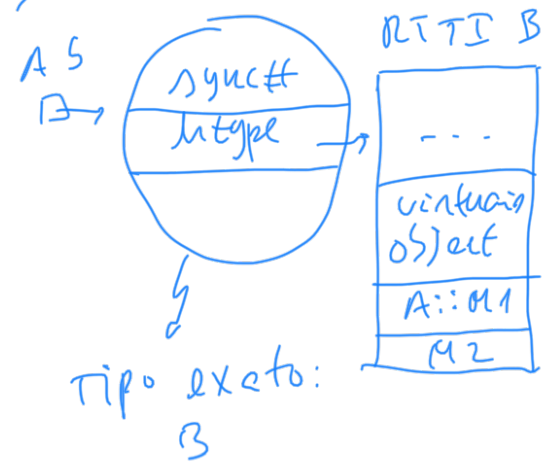
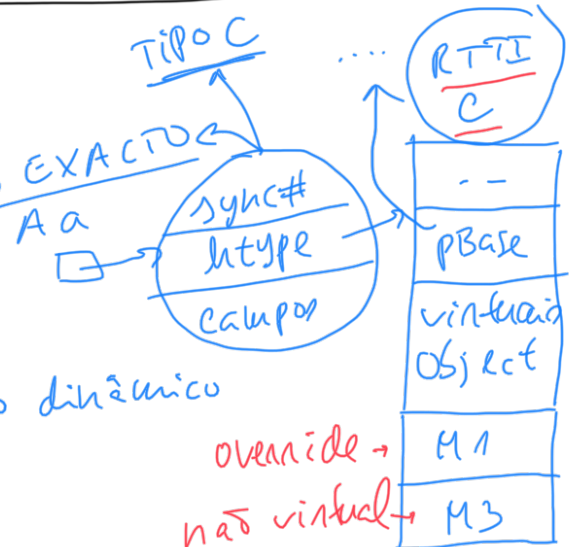
TIPO DA REFERÊNCIA  
 $A \ a = \text{new } C();$

$a.m1();$  //  $C::m1$   
 ↳ despacho dinâmico

$A \ b = \text{new } B();$

$b.m1();$

↳  $B::m1 \Leftrightarrow A::m1$



## Section 6.2 - Method Invocation and Type

7.

→ Livro Don Box, p. 157 - 158

```
class Bobs {  
    public void f() {}  
    static public void useBobs(Bobs b) {  
        b.f();  
    }  
}
```

b.f();

Nativo

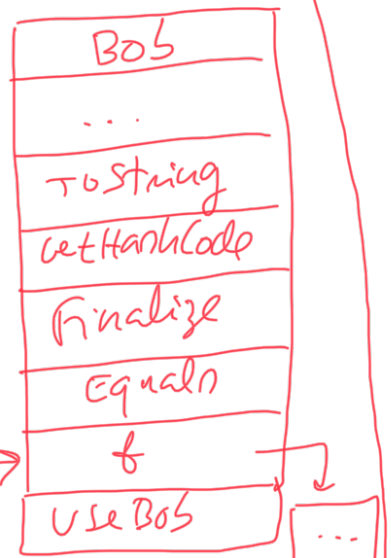
chamada mtd. não virtual

mov ecx, esi; // this  
call dword ptr ds:[352108h]

Não depende do this para  
a tabela do tipo

```
class Steve : Bobs {  
    public void f() {}  
    static public Exploit(Steve s) {  
        Bobs.useBobs(s);  
    }  
}
```

ecx = this = b = & b type



call + execute  
mensagem sendo o  
objeto um Steve,  
é chamado Bobs.f()



E se `Bob::f` e `Steve::f` fossem virtuais?

```

class Bob {
    public virtual void f() {}
    static ... UseBob(Bob b) {
        b.f();
    }
}
    
```

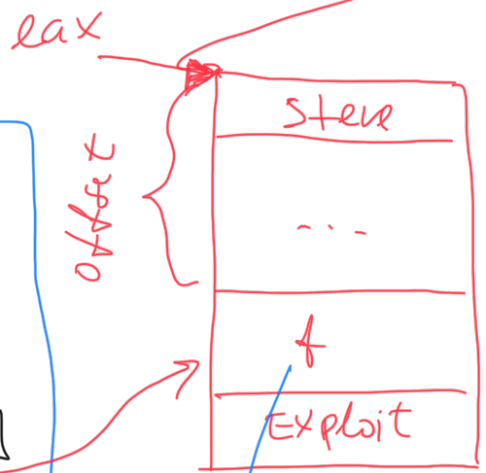
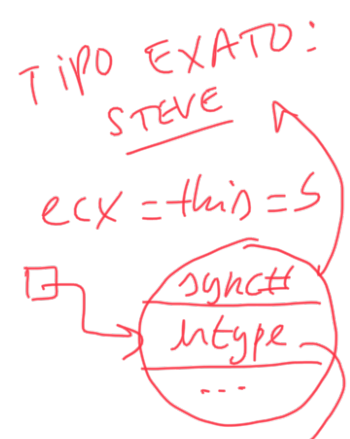
↓ nativo

chamada mtd. virtual:

```

mov ecx, esi; // this = &Intype
mov eax, dword ptr [ecx]
call dword ptr [eax + method offset]
    
```

← eax refere início da tabela de métodos



← override

```

class Steve : Bob {
    ... override void f() {}
    static ... Exploit(Steve s) {
        Bob.UseBob(s);
    }
}
    
```

← executa Steve::f()



9.  
O JIT compiler usa tipicamente a disciplina de stack -- `fastcall`: os dois primeiros parâmetros são passados nos registros `ecx` e `edx`, se possível. (D. Box, p. 158)

Dai, o `this` é passado no `ecx` no quadro anterior.