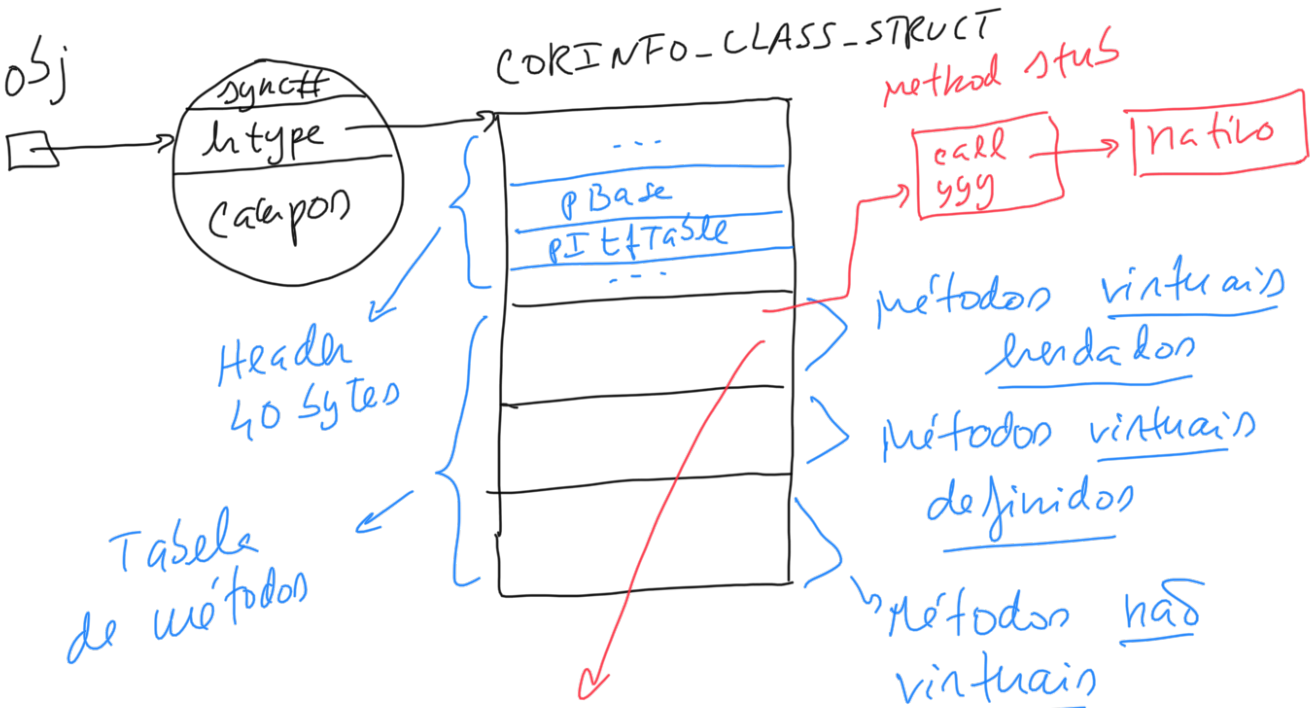
Tabela de métodos



Atributos de métodos em C#:

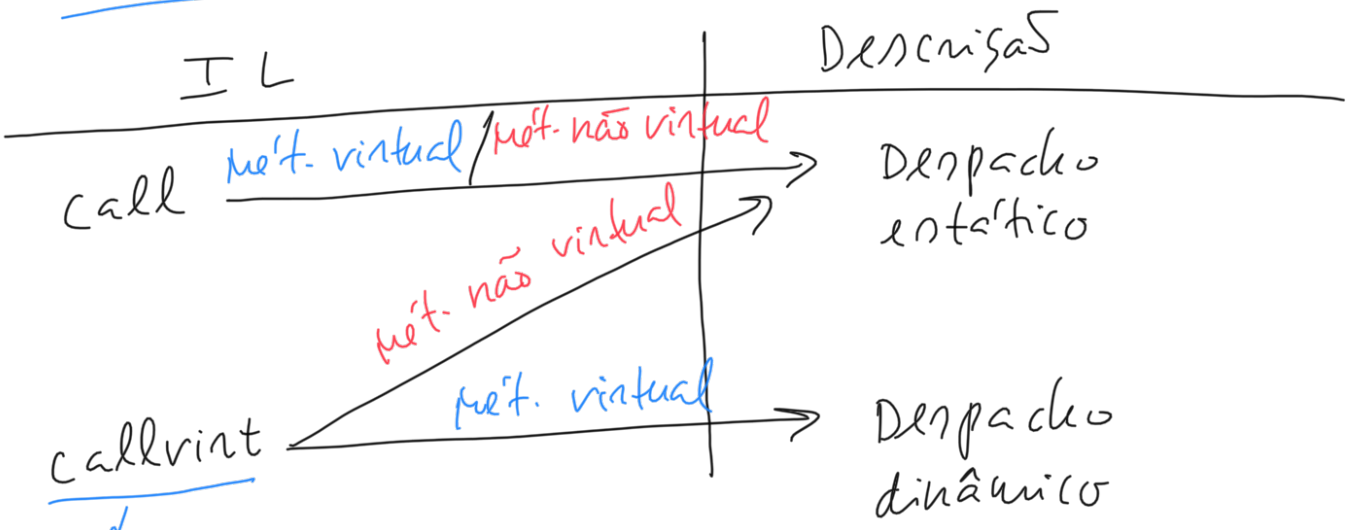
- virtual, abstract e override ⇒ o método é virtual
- Se o método não contiver um dos atributos acima, então o método é não virtual, podendo ser de instância ou estático.
- Se o método é virtual, pode ser invocado de forma polimórfica (despacho dinâmico)

- métodos de Object:
  - virtuais: ToString, Equals, GetHashCode, e Finalize
  - não virtual: GetType

Invocação de métodos

- Despacho dinâmico: chamada 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).



↓

Verifica adicionalmente se a referência para o objeto é diferente de null e lança exceção se igual a null.

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

$\not\Rightarrow$ Despacho dinâmico

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

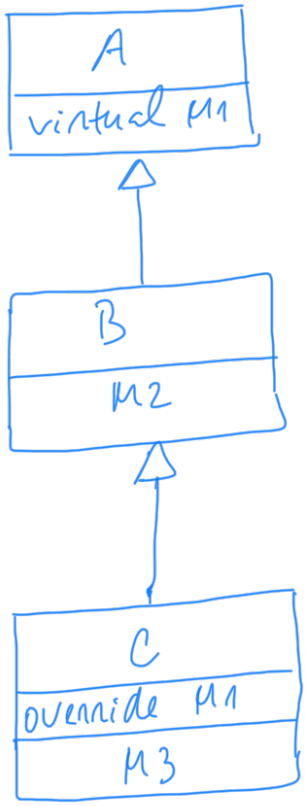
R: Quando tem a certeza de que a referência para o objeto é diferente de null, por exemplo, na invocação de um método de instância dentro dum construtor, ou dentro doutro método de instância.

```
class C {
    ...
    public C() {
        m1(); // call m1
    }
    public m1() {...}
    public m2() {  m1(); }
                   ↳ call m1
}

Main:
    C obj = new C();
    obj.m2(); // call virt m2
                ↳ para verificar se obj
                é ≠ null, mas
                realiza despacho
                estático.
```

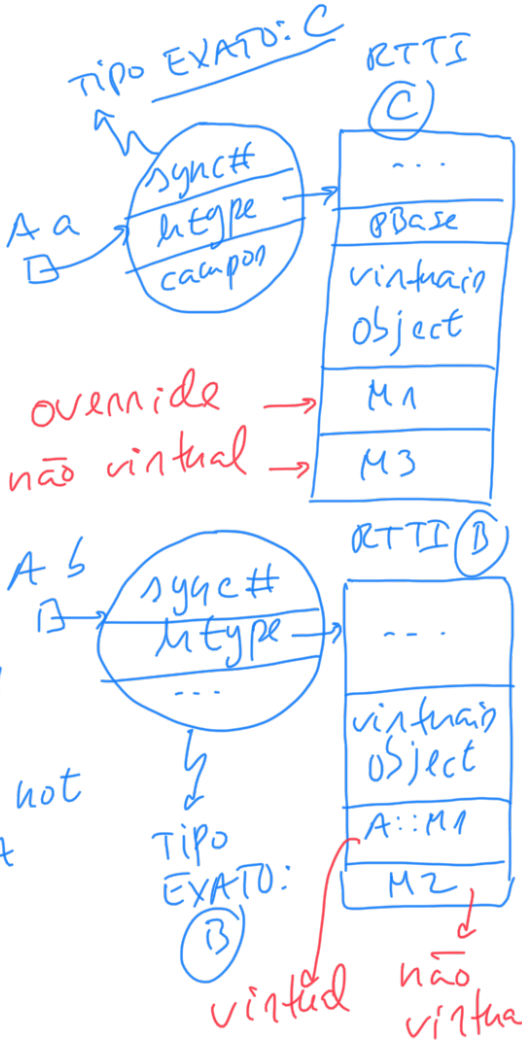
Despacho dinâmico

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
TIPO EXATO

```
A a = new C();
a.M1(); // C::M1
        ↳ despacho dinâmico
A b = new B();
b.M1();
  ↳B::M1 <=>A::M1
a.M2(); // Error, M2 not defined in A
((C)a).M2(); // B::M2
```



Section 6.2 - Method Invocation and Type

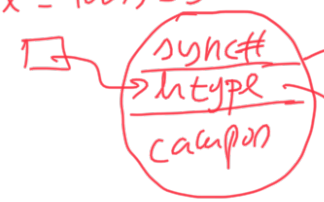
→ Livro Don Box, p. 157 - 158

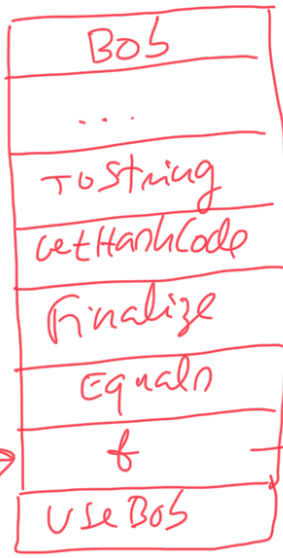
```
class Bos {
    public void f() {}
    static public void UseBos(Bos b) {
        b.f();
    }
}
```

Nativo

```
... chamar ref. não virtual
mov ecx, esi ; // this
call dword ptr ds:[352108h]
```

ecx = this = b = & in type





Não depende do this para escolher a tabela do tipo

call + executa mesmo sendo o objeto um Steve, é chamado Bos.f()

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

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

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

nativo

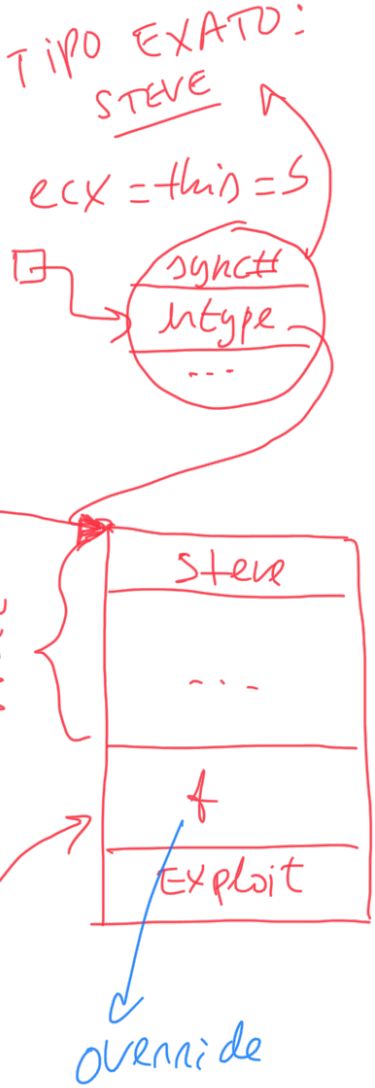
chamar mét. virtual:

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

eax refere início da tabela de métodos

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

executa Steve::f()



O JIT compilen una tipicamente a disciplina de stack -- fastcall: os dois primeiros parâmetros são passados nos registos ecx e edx, se possível. (D. Box, p. 158)

Daí, o this ser passado no ecx no quadro anterior.