

# Usare C# in process da NodeJS

Livello: 1000 (siete avvertiti, "ve l'avevo detto") ☺

Raffaele Rialdi  
Senior Software Architect  
Microsoft MVP  
Consultant - Speaker - Teacher



@raffaeler



<https://github.com/raffaeler>



<http://iamraf.net>



[raffaeler@vevy.com](mailto:raffaeler@vevy.com)



# Chi sono

@raffaeler



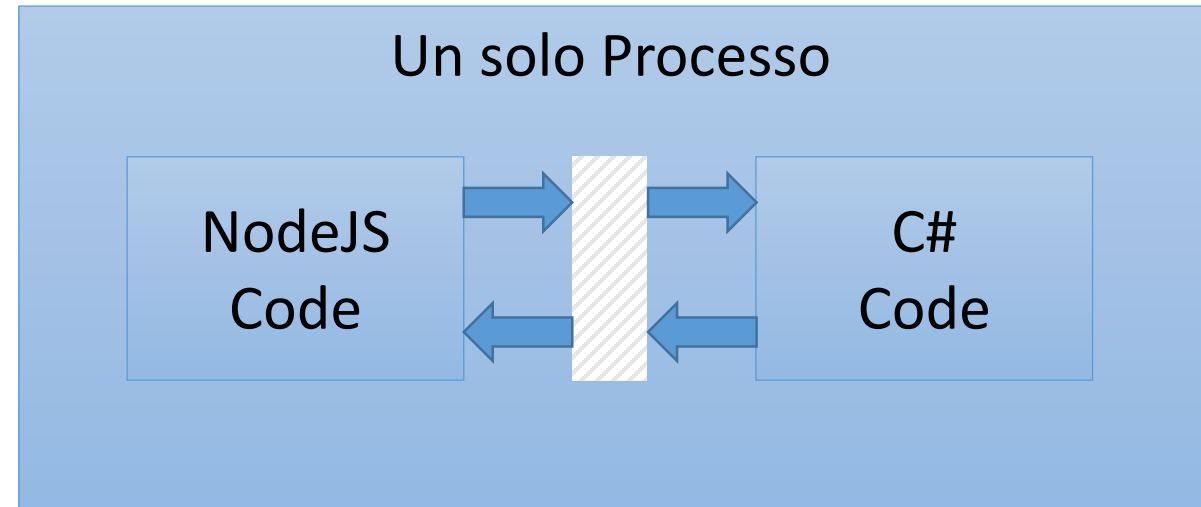
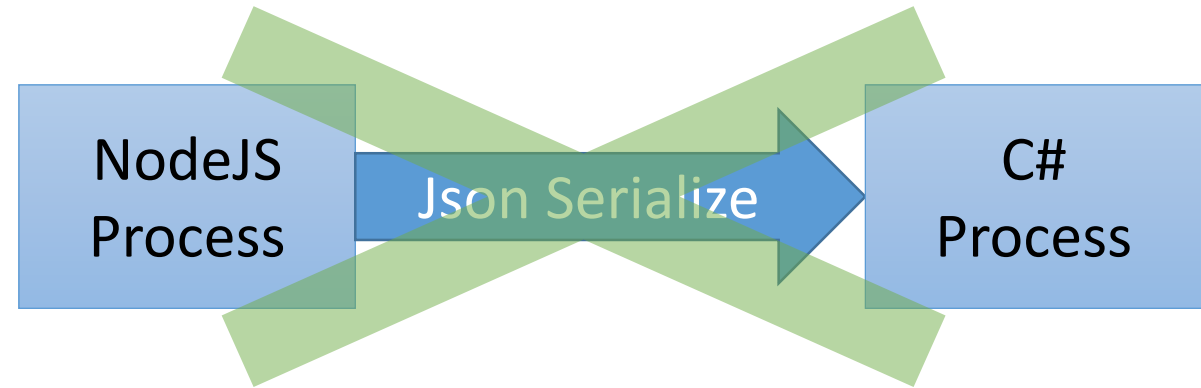
- Raffaele Rialdi, Senior Software Architect in Vevy Europe – Italy
- Senior Software Architect e consulente in diverse aree
  - Manufacturing, racing, healthcare, financial, ...
- Fiero membro della grande community MVP dal lontano 2003
- Speaker & consulente in giro per il globo (sviluppo e sicurezza)
  - Italia, Iasi, Bucharest, Cluj Napoca, Novosibirsk, Moscow, St Petersburg, Orlando, ...



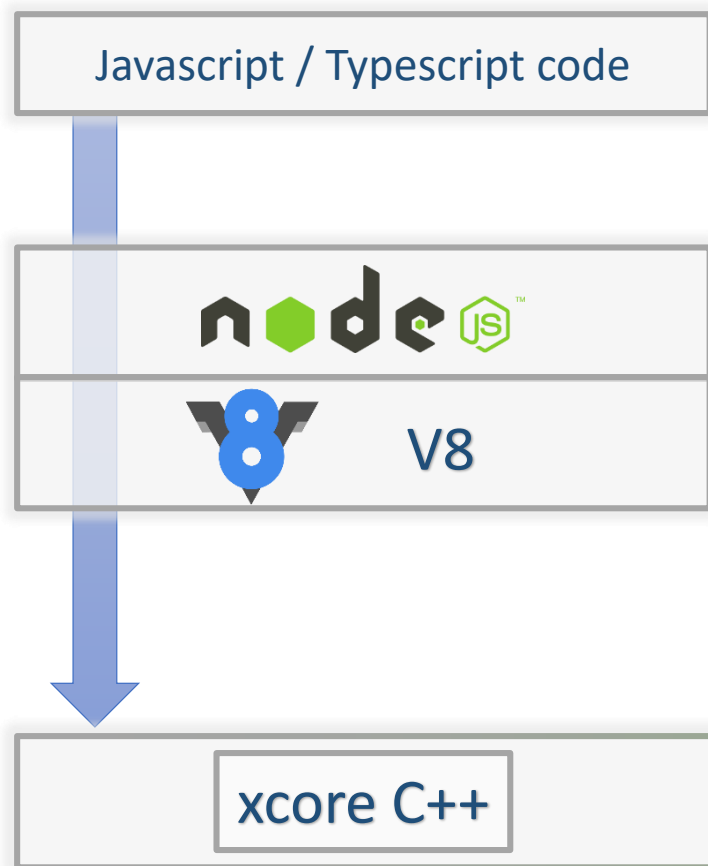
*Ignite 2017*

# NodeJS e .NET? Che significa?

- NON vogliamo serializzare i dati tra processi
  - Troppo facile 😊
  - Pessime performance 😞
- Vogliamo usare lo **stesso** processo
  - Molto complesso 😊😊
  - Ottime performance 😊



# Vi presento xcore



xcore

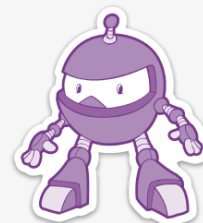
xcore .NET

[NPM, Inc. \[US\] | https://www.npmjs.com/package/xcorenode](https://www.npmjs.com/package/xcorenode)


```
npm install xcorenode
```


.NET Code


.NET Core





# xcore in azione


- Carica e inizializza xcore 

```
var xcore = require('bindings')('xcore');
xcore.initialize(__dirname + "\\Sample",
  "SampleLibrary.dll", "SampleLibrary.Initializer");
```
- Carica i metadati .net 
  - Basta la classe più alta nel grafo

```
xcore.loadClass("SampleLibrary.OrderSample.
  OrderManager, SampleLibrary");
```
- Crea un'istanza di un oggetto .NET 

```
var om = new xcore.OrderManager("raf");
```
- Chiama i metodi 

```
console.log(om.Add("Hello, ", "world"));
```
- Cammina il grafo 
  - i nuovi metadati vengono caricati automaticamente

```
console.log(om.SelectedOrder.Name);
```
- Chiama metodi asincroni (e altro ancora!) 

```
om.AddAsync(2, 3, function(res){
  console.log(res);
});
```

# Node.JS e V8

<https://nodejs.org>



- Node.js® è un noto runtime per eseguire codice javascript
- Sfrutta la potenza di "V8", il motore di Google Chrome
  - V8 è conforme allo standard ECMAScript 262
  - funziona cross-platform
- NodeJs ha un ecosistema con montagne di librerie di terze parti
- V8 può essere esteso con addons nativi sviluppati in C o C++:
  - Per ottenere migliori performance
  - Per accedere alle risorse del sistema operativo
  - Per interoperare con sistemi esterni (HW/SW)

# La struttura base di un addon V8 in C++

## Javascript

```
const addon = require('./myAddon');  
console.log(addon.hello()); // 'world'
```

Prototype

## C++

```
#include <node.h>  
#include <v8.h>
```

```
NODE_MODULE(addon, init)
```

```
void init(Local<Object> exports)  
{  
    NODE_SET_METHOD(exports, "hello", Method);  
}
```

```
void Method(const FunctionCallbackInfo<Value>& args)  
{  
    Isolate* isolate = args.GetIsolate();  
    args.GetReturnValue().Set(  
        String::NewFromUtf8(isolate, "world");  
    )  
}
```

# V8 Object Wrapping

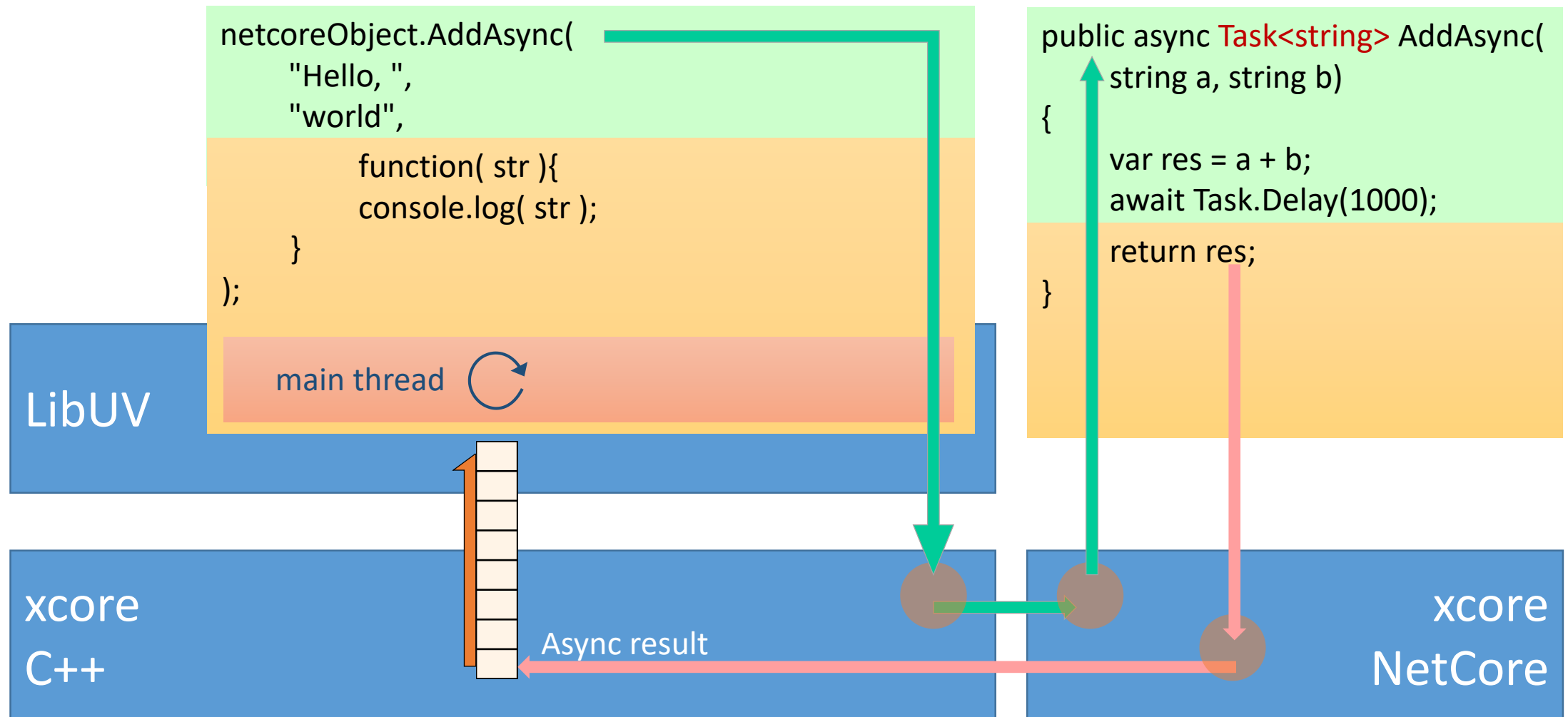
- Derivando la classe 'ObjectWrap' si può esporre oggetti javascript
- xcore aggiunge *dinamicamente* Type e membri definiti in .NET
  - Costruttori, metodi, proprietà, indexers
  - La costruzione è dinamica, indicando nome del membro e classe

```
auto constructorTemplate = FunctionTemplate::New(isolate, New, v8loadableName);  
constructorTemplate->SetClassName(v8className);  
auto classTemplate = constructorTemplate->InstanceTemplate();  
...  
constructorTemplate->PrototypeTemplate()->Set(method_name, template); // methods  
classTemplate->SetAccessor(prop_name, &GetProperty, setter, v8::Int32::New(isolate, oid)); // property
```

```
exports->Set(v8className, constructorTemplate->GetFunction()); // registrazione di un type
```



# Metodi asincroni



# *Demo: un semplice plugin V8*

# Hosting del CoreCLR

- Cosa significa fare hosting del CLR?
  - Una applicazione nativa (C++) può caricare il CLR, alcuni assemblies e avviare codice managed (.NET)
  - SQL Server ne è un buon esempio
  - Chiunque può farlo!
- Il CLR del .NET Framework può essere hosted solo su Windows
  - Si basa sull'infrastruttura COM
- NetCore può essere hosted cross-platform
  - La nuova API è stata semplificata ed è priva di riferimenti a COM



*Demo: caricare il CLR*

# Hosting del CLR

- Includere le librerie
- Caricare coreclr.dll
- Prendere l'entrypoint
- Prendere il RuntimeHost
- Avviare il CLR
- Creare l'AppDomain

```
#include <mscoree.h>
```

```
auto loader = LoadLibrary(fullCoreClr.c_str());
```

```
auto prhf = (FnGetCLRRuntimeHost)::GetProcAddress(  
    (HMODULE)loader, "GetCLRRuntimeHost");
```

```
ICLRRuntimeHost2* prh = nullptr;  
hr = prhf(IID_ICLRRuntimeHost2, (IUnknown**)&prh);  
prh->Authenticate(CORECLR_HOST_AUTHENTICATION_KEY);
```

```
hr = prh->Start();
```

# xcore wrap up

# xcore execution flow

- C++ cerca se i metadati del Type accesso da javascript sono disponibili
  - In caso negativo, C++ chiama .NET e li carica on demand
- C++ verifica se l'oggetto Javascript (prototype) è già stato creato
  - In caso negativo, lo registra con tutti i suoi membri
- Quando un membro viene invocato
  - C++ prende tutti i parametri e li passa a C# via *Reverse PInvoke*
  - I parametri indicano anche un 'instance identifier' (object reference)
- Le chiamate vengono fatte tramite delegate che invoca il membro specificato
  - Se il membro non è mai stato eseguito, viene *generato il codice via Expression*
- Il risultato dell'esecuzione viene ritornato a C++

# Performance profile

- There are still many possible improvements
- The add use-case is not realistic
  - it just measures infrastructure and marshalling
  - 4 marshalling involved: (JS → C++ → .NET → C++ → JS)

1 Million calls  
js: 6.156 ms  
c++: 56.352 ms  
.net: 1294.254 ms

## *prepare the benchmark*

```
var om = new xcore.OrderManager();

om.Add(2, 2);           // jitting
LocalAdd(2,2);          // jitting
xcore.add(2, 2);        // jitting
var loop = 1000000; // 1 Million

function LocalAdd(x, y)
{ return x + y; }
```

## *local javascript*

```
console.time("js");
for(var i=0; i<loop; i++)
{
    LocalAdd(i, i);
}
console.timeEnd("js");
```

js: 6.156ms

## *C++ only*

```
console.time("c++");
for(var i=0; i<loop; i++)
{
    xcore.add(i, i);
}
console.timeEnd("c++");
```

c++: 56.352ms

## *xcore + C#*

```
console.time("net");
for(var i=0; i<loop; i++)
{
    om.NetAdd(i, i);
}
console.timeEnd("net");
```

.net: 1294.254ms





# Use-cases

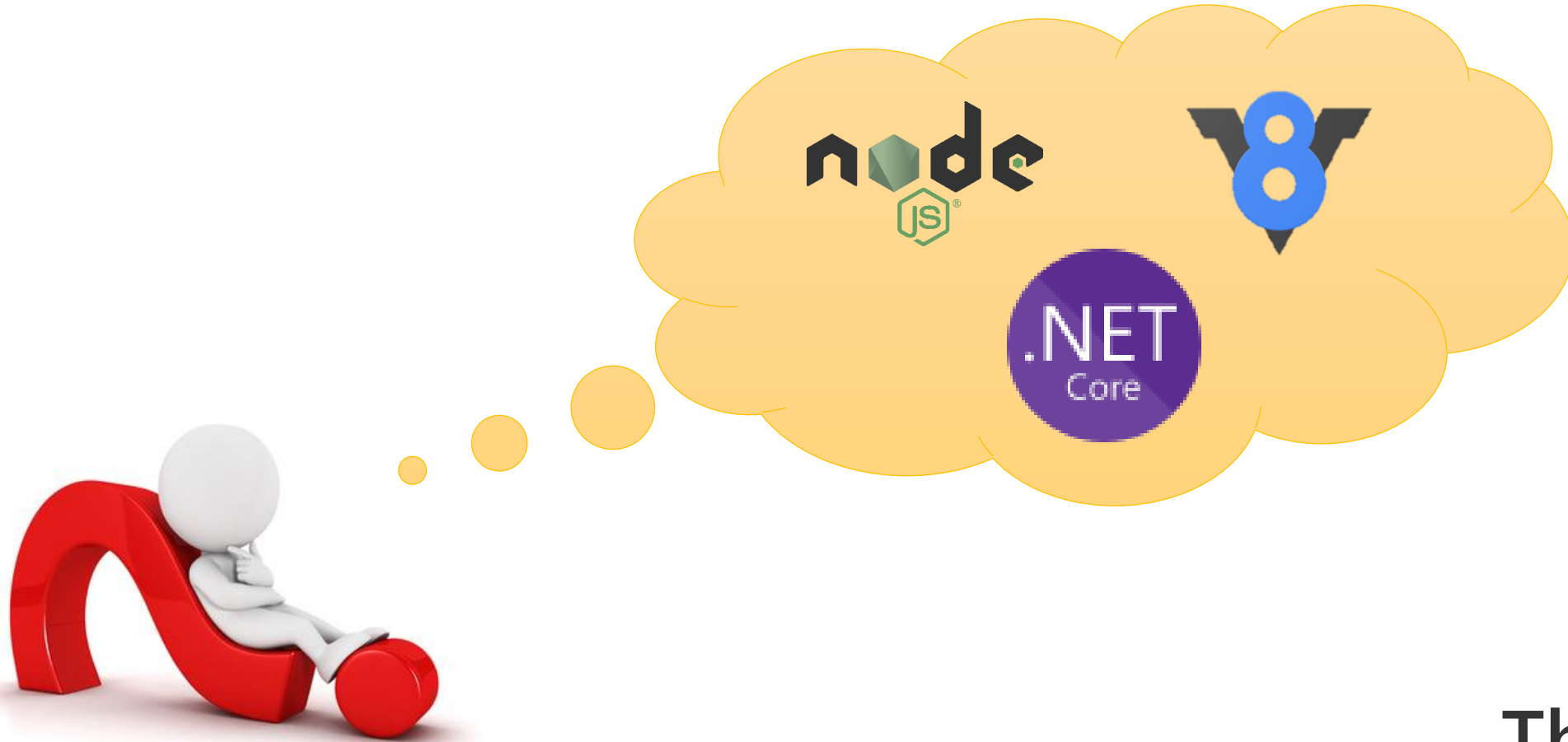


1. Node.js applications
2. UI created with the Electron framework
  - Ability to create .NET ViewModels for Angular (by Google)
3. Using JS to script Windows Powershell
4. Nativescript Mobile applications
  - Nativescript is a project by Progress Software Corporation  
<https://www.nativescript.org/>
5. Anything else based on V8

# TODO list

- Migrare la codebase C++ a N-API
- Migrare l'interop C++ a C++17 (string\_view)
- Migrare la generazione di codice C# a Span<T> e Memory<T>
- Aggiungere supporto per iterator e iterables
- Generare le definizioni di Typescript
- Bug fixing :)

# Questions?



## Thank you!

COD≡C△MP ❤️ FEEDBACK



[codecamp.ro/feedback](https://codecamp.ro/feedback)

---