Italian C++ Community

www.italiancpp.org

English video and more info here:
https://github.com/raffaeler/xcore

Costruire un bridge C++ tra NodeJS e C#

Raffaele Rialdi

Twitter: @raffaeler
Email: raffaeler@vevy.com
Website: http://iamraf.net

Italian C++ Conference 2017
17 Giugno, Milano

Thanks to the sponsors!

Italian C++ Community

Italian C++ Conference 2017 #itCppCon17

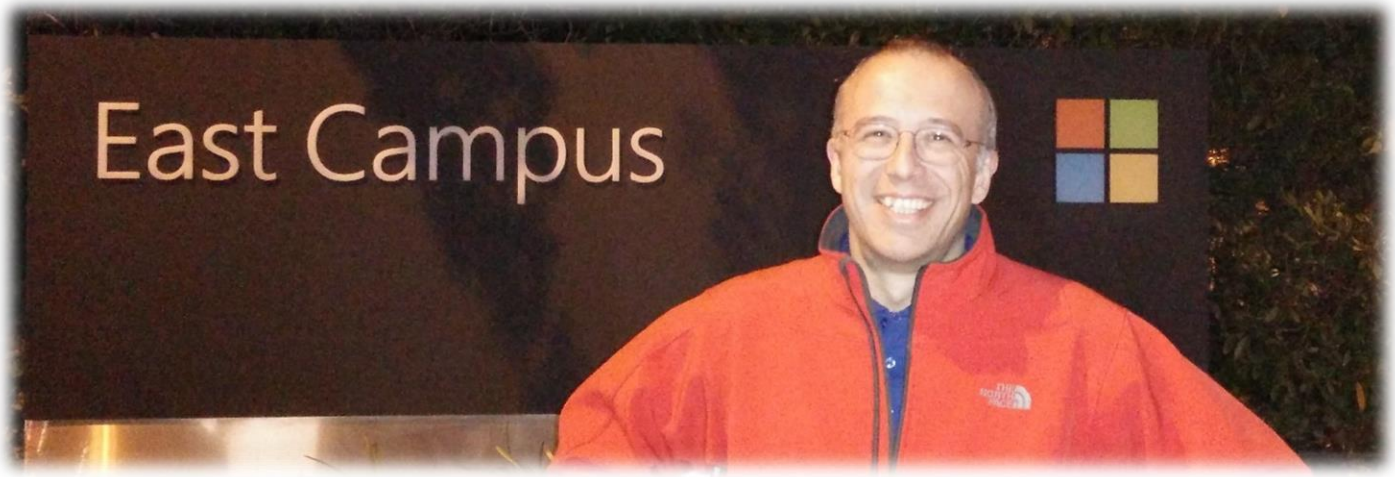
Who am I?

- Raffaele Rialdi, Senior Software Architect in Vevy Europe – Italy
- Ma anche consulente software per aziende che operano in diversi campi
  - manufacturing, healthcare, financial, racing, …
- Speaker in conferenze Italiane e internazionali
  - Trainer in corsi
- E un fiero di essere stato premiato Microsoft MVP
  - Fin dal lontano 2003

@raffaeler

raffaeler@vevy.com

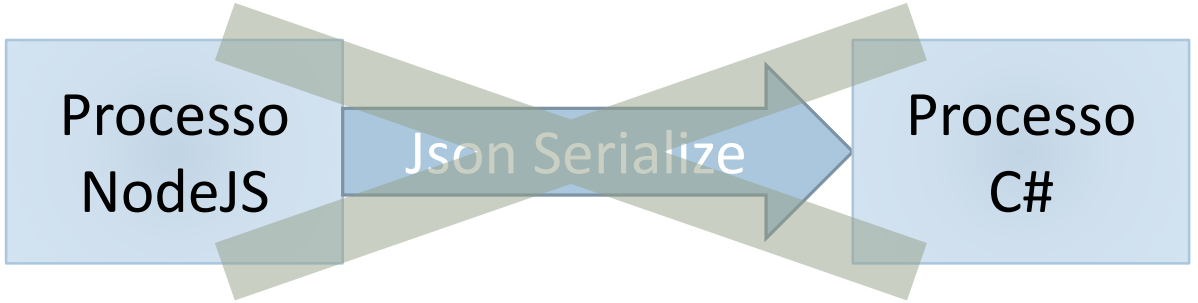
github.com/raffaeler



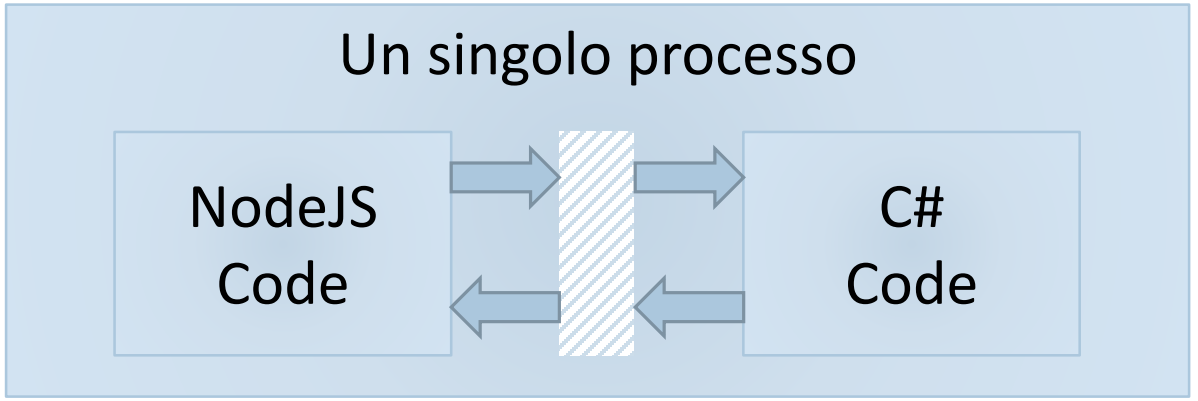
Da NodeJS a .NET?

- Rationale: unire le forze degli ecosistemi
  - La potenza di C++
  - La versatilità di C#
  - La dinamicità di NodeJS

- No alla serializzazione
  - Troppo facile ☺
  - Scarse performance ☹
  - Molti oggetti non sono serializzabili



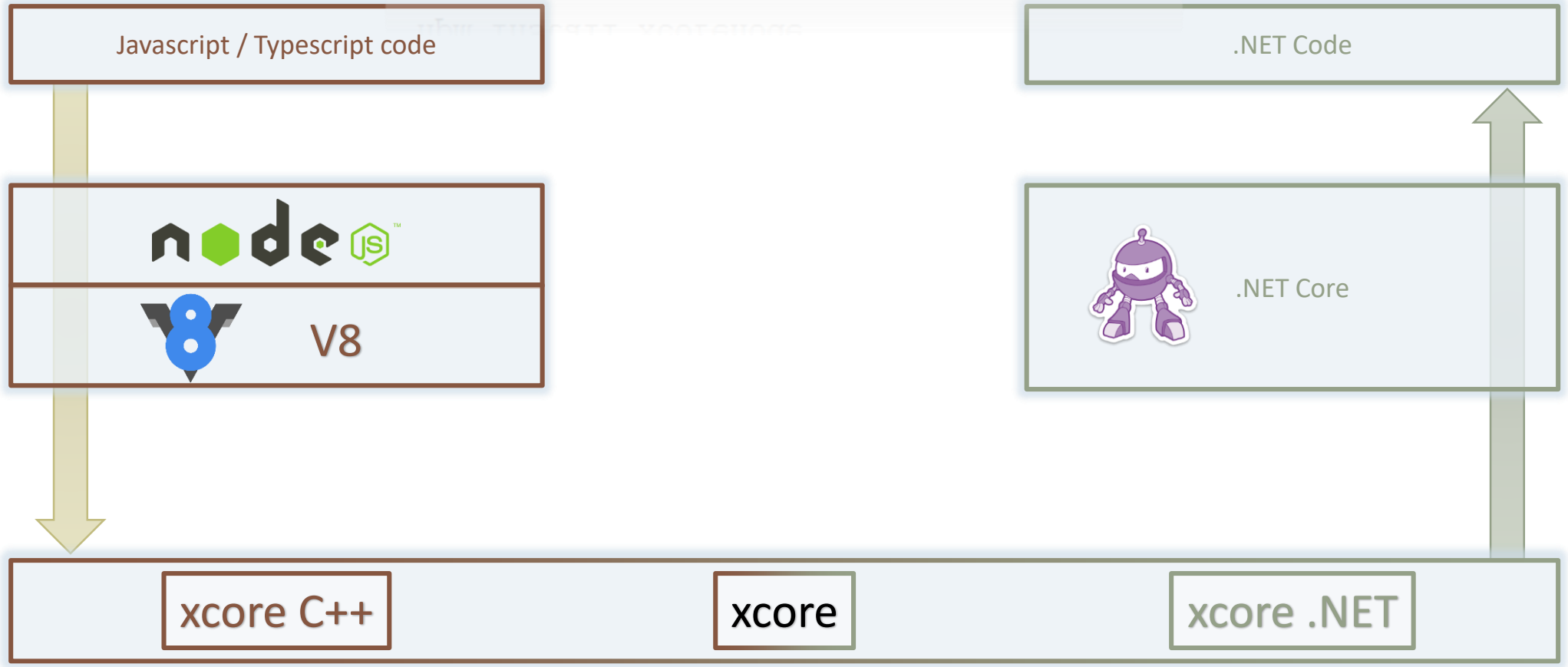
- Tutto il codice nello stesso processo
  - È molto più complesso ☺☺
  - Le migliori performance possibili ☺



Vi presento "xcore"

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

```
npm install xcorenode
```



xcore in azione

- Carico e initializzo xcore

```
var xcore = require('bindings')('xcore');
xcore.initialize(__dirname + "\\Sample",
   "SampleLibrary.dll", "SampleLibrary.Initializer");
```

- Carico i metadati .net
  - Basta la prima classe del grafo

```
xcore.loadClass("SampleLibrary.OrderSample.
   OrderManager, SampleLibrary");
```

- Creo un istanza

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

- Chiamo i metodi

```
console.log(om.Add("Hello, ", "world"));
```

- Cammino il grafo
  - i nuovi metadati vengono caricati automaticamente

```
console.log(om.SelectedOrder.Name);
```

- Chiamate asincrone (e molto di più!)

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

Node.JS e V8

https://nodejs.org

- Node.js® è un noto runtime for eseguire codice javascript
- Si avvale della potenza di "V8", il motore di Google Chrome
  - V8 è conforme allo standard ECMAScript 262
  - È cross-platform
- NodeJs ha un ecosistema molto sviluppato
  - Un'autentica immensità di librerie di terze parti
- V8 può essere esteso anche con addons nativi (C++):
  - Quando si cercano maggiori performance
  - Per accedere a risorse e servizi del sistema operativo
  - Per interoperare con sistemi esterni o hardware

La struttura base di un addon V8 in C++

- Includere le librerie
- Dichiarare l'entry-point
  - Si usa la macro NODE_MODULE
- Implementare l'entry-point e dichiarare i metodi javascript
- Il "Method" sarà disponibile con il nome "hello" in javascript
- Il suo utilizzo è molto semplice

```
#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"));
}
```

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

La toolchain

- Per compilare l'addon è necessaria una toolchain C++
  - Python, GCC for Linux, GCC / Xcode on Mac, VC++ su Windows
- Il tool node-gyp

```
npm install -g node-gyp
```

  - Evita di dover scaricare le librerie di node e i file di include
  - Astrae le operazioni di build dalle toolchain utilizzate
- Invece di usare un 'makefile' c'è un 'binding.gyp'

```
{
 "targets": [
  {
   "target_name": "myaddon",
   "sources": [ "src/myaddon.cpp" ]
  } ]
}
```

configurare gyp per utilizzare la toolchain di VC++2015

```
npm config set msvs_version 2015
```

Wrappare oggetti in V8

- Derivando la classe C++ ObjectWrap, l'oggetto viene esposto a JS
- I membri dell'oggetto vengono aggiunti dinamicamente
  - Sono supportati costruttori, metodi, proprietà e indexers
  - Basta impostare i membri specificando il nome della classe

```
exports->Set(v8className, constructorTemplate->GetFunction());
```

- Tutti i membri sono funzioni «static void»
  - I parametri di input sono tutti in «const FunctionCallbackInfo<Value>& args"
    - Args contiene i valori di runtime degli argomenti
  - L'istanza dell'oggetto "wrappato" si recupera con «ObjectWrap:Unwrap»
  - Il valore di ritorno viene impostato usando "args.GetReturnValue().Set(…);"

Marshalling: il type system di V8

- Il type system di V8 è diverso da quello di C++
  - Il motivo è perché i tipi di V8 sono gestiti dal Garbage Collector
  - Il Marshalling è necessario per copiare da/per i tipi di C++
- Ci sono più tipi V8 di quanto ci si potrebbe aspettare
  - **Int32, Number**, String, Boolean, Function, Object, Function, …
- Le stringhe sono UTF8 mentre .NET usa UTF16. Servono entrambe.

```
static std::string FromV8String(const Local<Value>& value)
{
    String::Utf8Value utf8(value);
    std::string str(*utf8);
    return str;
}
```

```
static std::wstring FromV8StringW(const Local<Value>& value)
{
    String::Utf8Value utf8(value);
    auto str = raf_helpers::Helpers::utf8_to_wstring(*utf8);
    return str;
}
```

Threading e la libreria «libuv»

- LibUV è la libreria di threading usata da V8 e NodeJs
- Il codice JS deve sempre essere eseguita nel thread principale
- Il risultato dei metodi .NET che ritornano un Task<T> devono essere invocati dal thread principale
- La libreria LibUV fornisce un supporto minimale per accodare messaggi nei suoi thread
- **È necessario creare una coda concorrente che usi il supporto di LibUV**

www.italiancpp.org

Demo: un semplice plugin V8

Italian C++ Conference 2017
17 Giugno, Milano

Eseguire l'hosting del CoreCLR

- Cosa significa fare hosting del CLR?
  - Una applicazione nativa può caricare il CLR, degli assembly e avviare del codice managed
  - SQL Server è un esempio di applicazione nativa che usa questa tecnica
- Il CLR del .NET Framework può girare solo su Windows
  - Si basa sull'infrastruttura COM
- Net Core può girare cross-platform (incluso ARM)
- Le nuove librerie "netstandard" sono compatibili (a livello binario) con .NET classico e il nuovo NET Core
  - Le librerie PCL sono state mandate in pensione.

www.italiancpp.org

Demo: loading the CLR

Italian C++ Conference 2017
17 Giugno, Milano

DIETRO LE QUINTE DI XCORE

xcore

- Reverse PInvoke è la tecnica per chiamare metodi .NET da C++
- C++ può solo chiamare metodi .NET statici
  - Non c'è semantica per il ciclo di vita degli oggetti, proprietà, eventi e async
- xcore analizza e mantiene in memoria i metadati .NET
  - Genera codice ottimizzato per accedere alle istanze di ciascun oggetto
  - Gli Expression Trees sono compilati e i delegate sono messi in cache
  - Le istanze degli oggetti sono conservate e legate al garbage collector di JS/V8
  - Il codice di marshalling è ottimizzato
- I grafi di oggetti
  - Ogni volta che un oggetto child viene acceduto, xcore dinamicamente carica i suoi metadati

Il flusso di esecuzione di xcore

- La chiamata era sincrona
  - Il valore di ritorno viene passato indietro a C++ e V8
- La chiamata era asincrona
  - In C++ il valore di ritorno è accodato nel thread principale gestito da LibUV
- La chiamata a generato un'eccezione
  - Viene propagata al chiamante

Performance profile

- C'è ancora spazio per tanti miglioramenti
- Questo è il peggiore use-case possibile
  - Serve solo a misurare l'infrastruttura e il marshalling
  - 4 operazioni di marshalling : (JS → C++ → .NET → C++ → JS)

1 Milioni di call	
js:	6.156 ms
c++:	56.352 ms
.net:	1294.254 ms

preparare il 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; }
```

solo javascript

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

js: 6.156ms

JS e C++

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

c++: 56.352ms

JS, xcore e C#

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

.net: 1294.254ms

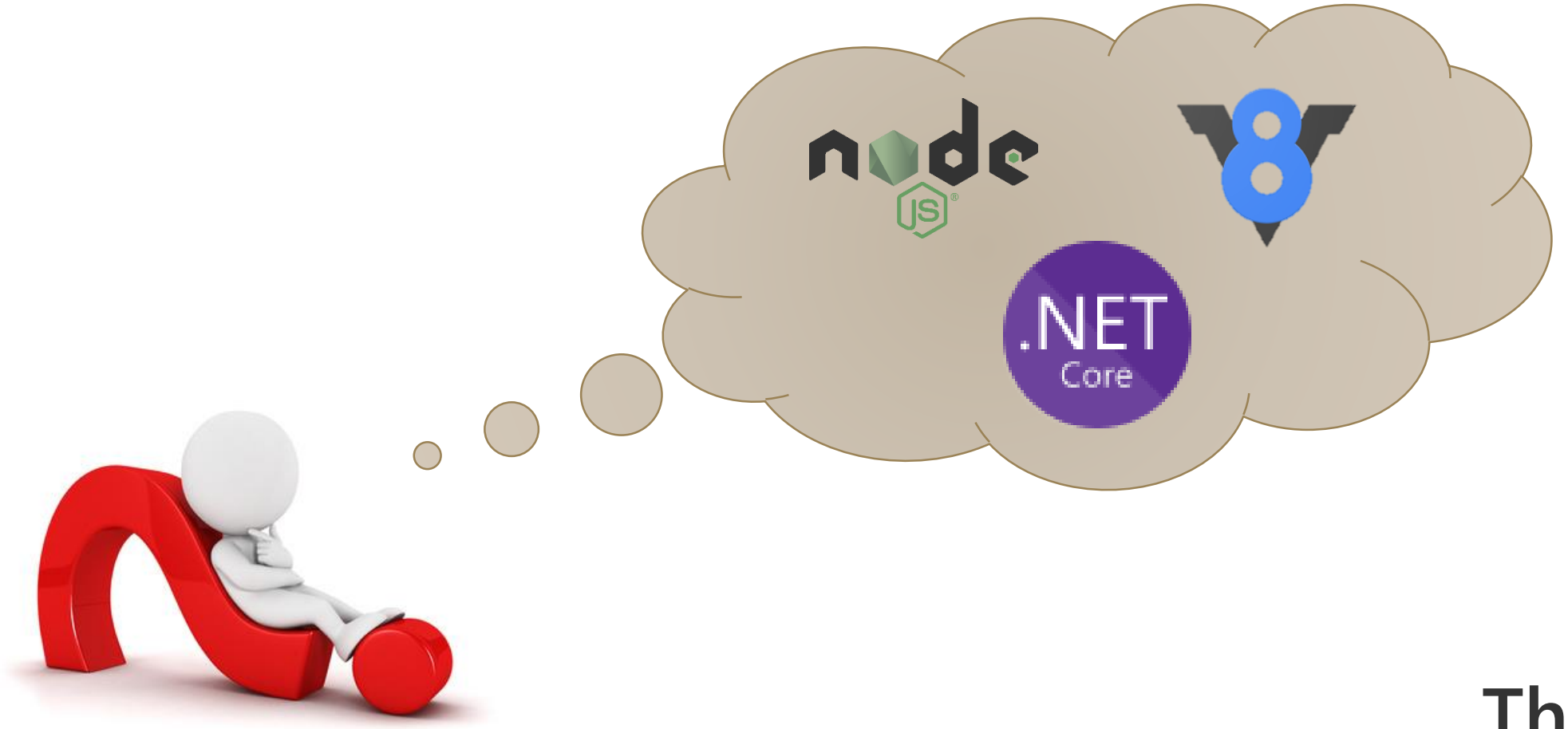
Dove può servire xcore

1. Applicazioni Node.js
2. Desktop apps che usano Electron
   - Angular view con .NET ViewModels

http://electron.atom.io

3. Usare JS per scriptare Windows Powershell
4. Applicazioni Mobile con Nativescript
   - Nativescript è un progetto di Progress Software Corporation https://www.nativescript.org/
5. Qualsiasi altra soluzione basata su V8

Domande?



Thank you!