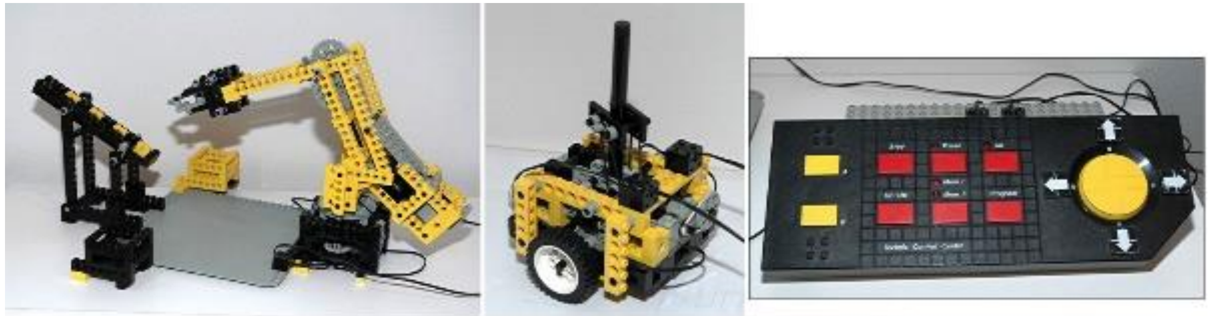
AXXES

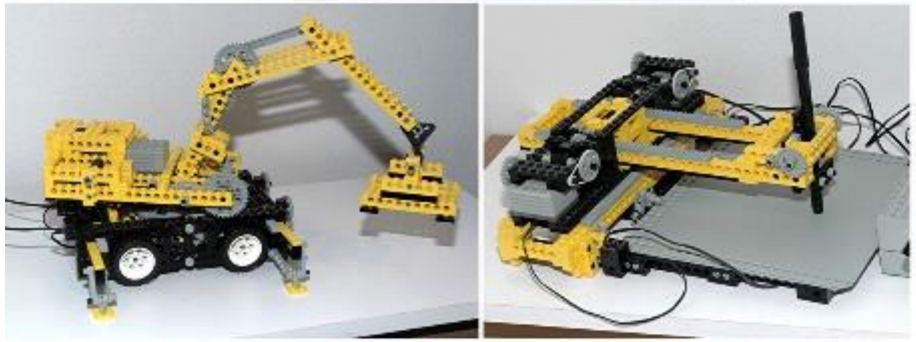
Build software like a bag of marbles, not a castle of LEGO®

–

Hannes Lowette

Disclaimer #1





Collaboration > coexistence!

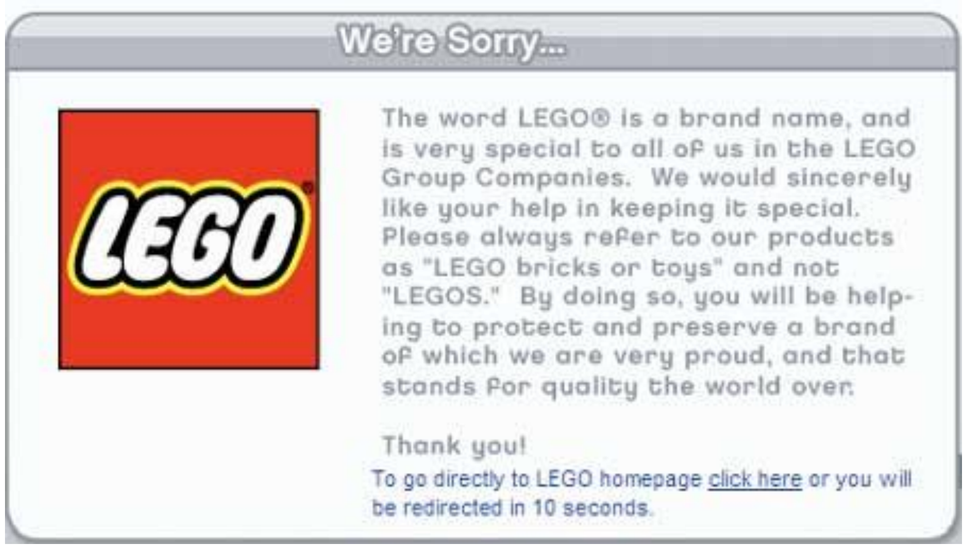
Disclaimer #2

PSA

LEGO is a brand name

used as an adjective

there is no plural, ‘LEGOs’



We're Sorry...

The word LEGO® is a brand name, and is very special to all of us in the LEGO Group Companies. We would sincerely like your help in keeping it special. Please always refer to our products as "LEGO bricks or toys" and not "LEGOS." By doing so, you will be helping to protect and preserve a brand of which we are very proud, and that stands for quality the world over.

Thank you!

To go directly to LEGO homepage click here or you will be redirected in 10 seconds.

Understanding the problem

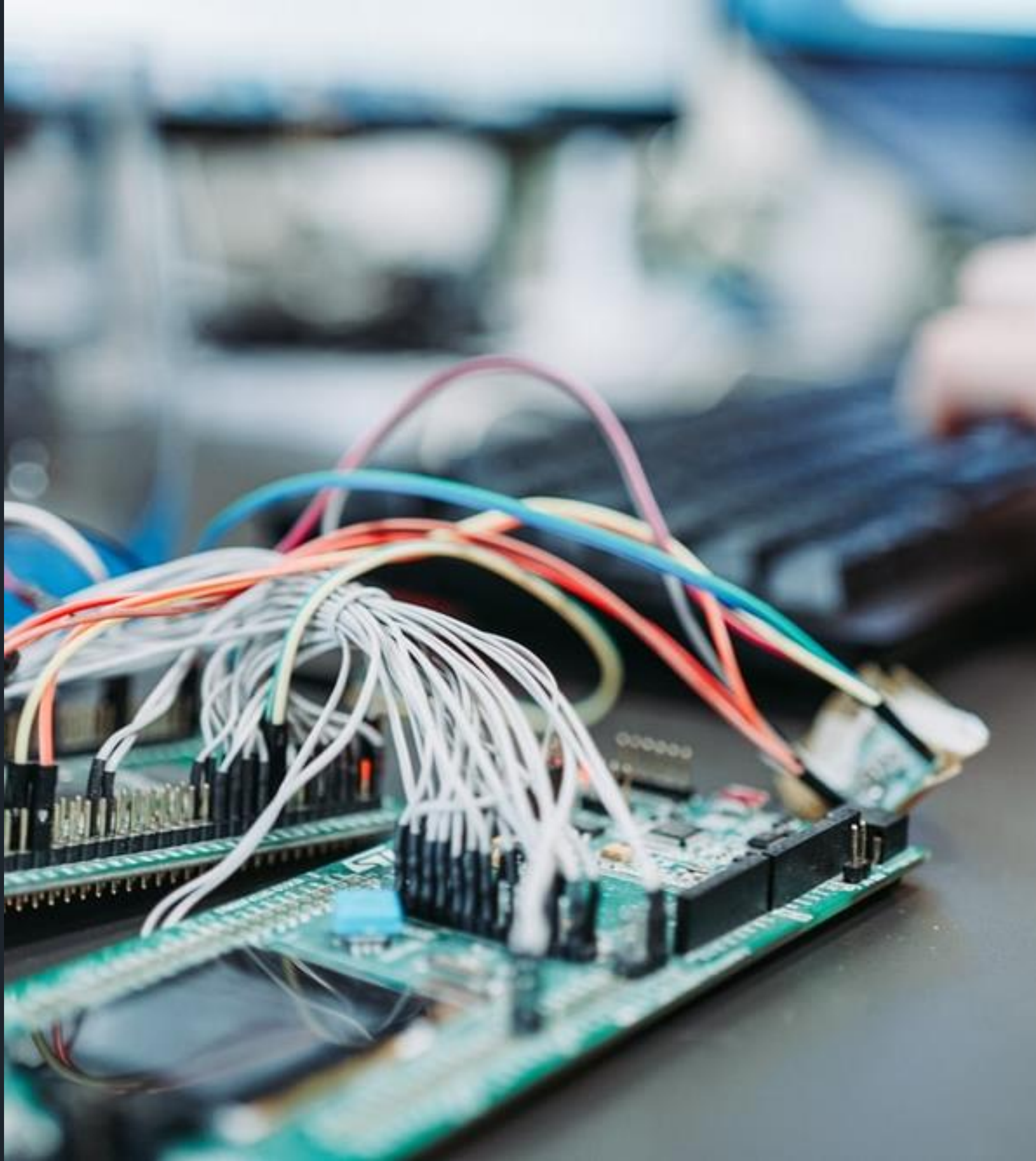
–

What were we trying to solve?

What were we working on?

- Huge product
- Goal: managing IoT devices
- A lot of **implementations** *(= specific types of devices)*
- 1 multi-tenant deployment

AXXES_

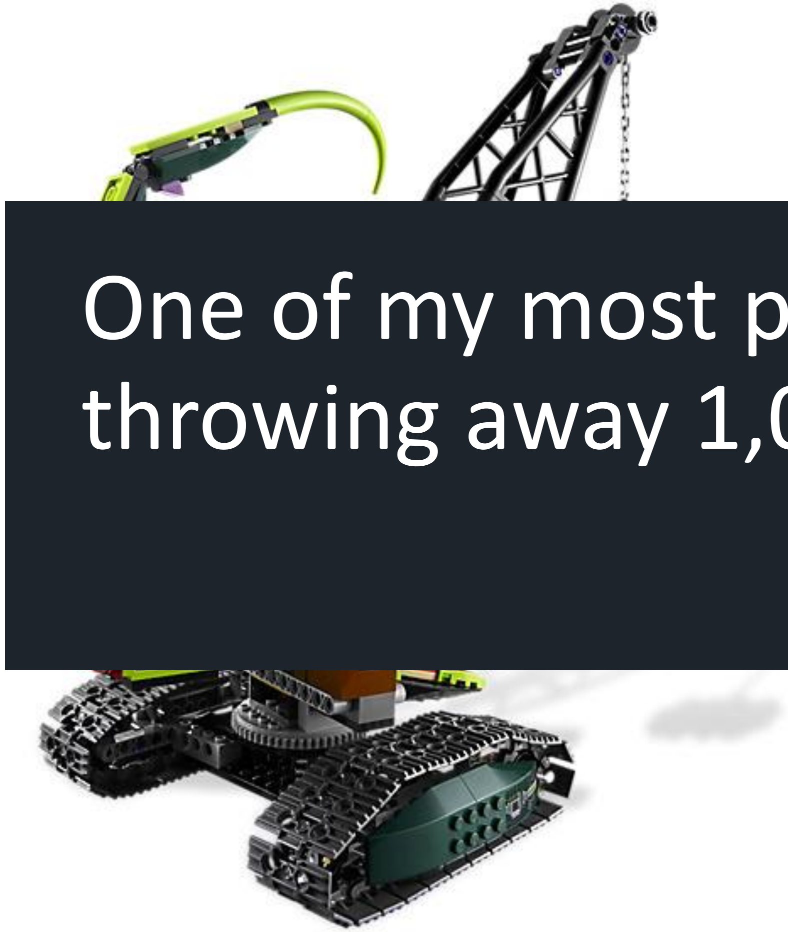


How did we roll?

- New **device types** all the time
- Most projects didn’t go live
- Code didn’t get removed because:
  - Tightly coupled
  - Reused by other devices
  - Money had been invested

→ Maintenance hell!

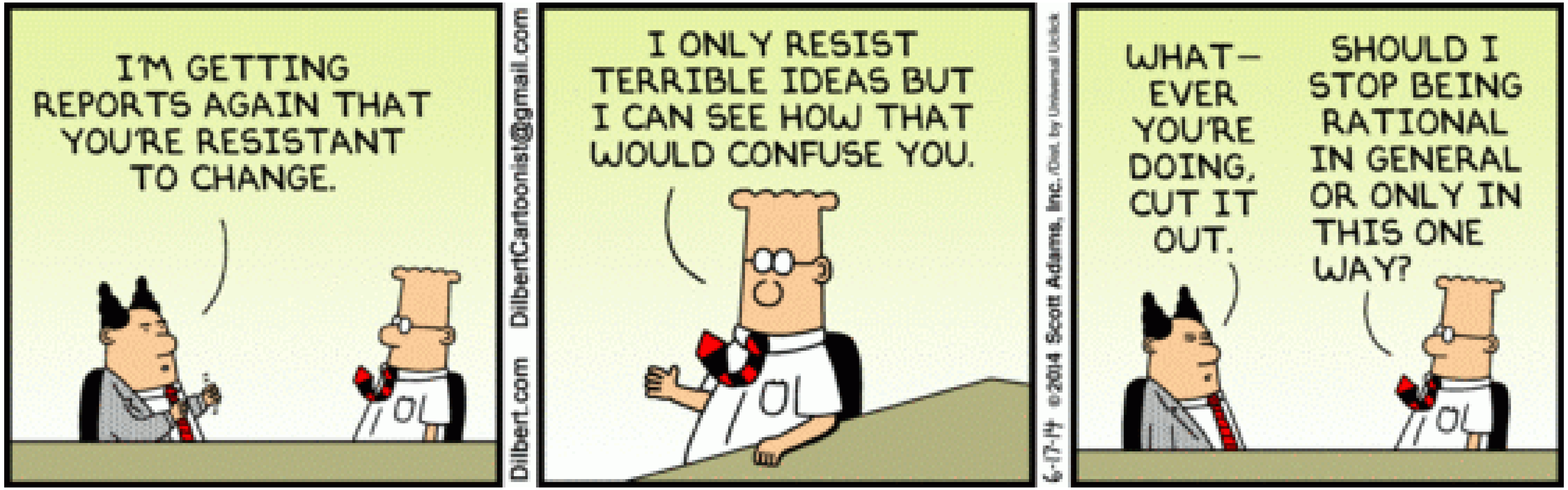
What frustrated me about this?



> One of my most productive days was throwing away 1,000 lines of code.
>
> *Ken Thompson*

Product management

- Not ready to change
- New devices would keep coming
- They expected us to keep the code


I'M GETTING REPORTS AGAIN THAT YOU'RE RESISTANT TO CHANGE.
Dilbert.com DilbertCartoonist@gmail.com
I ONLY RESIST TERRIBLE IDEAS BUT I CAN SEE HOW THAT WOULD CONFUSE YOU.
6-17-14 ©2014 Scott Adams, Inc./Dist. by Universal Uclick
WHAT-EVER YOU'RE DOING, CUT IT OUT.
SHOULD I STOP BEING RATIONAL IN GENERAL OR ONLY IN THIS ONE WAY?

What the dev team wanted

- Implement new devices quickly
- Be able to remove them easily
- Limit dependencies between devices
- Clean abstractions

→ **Stop Pollution!**

So, microservices, right?

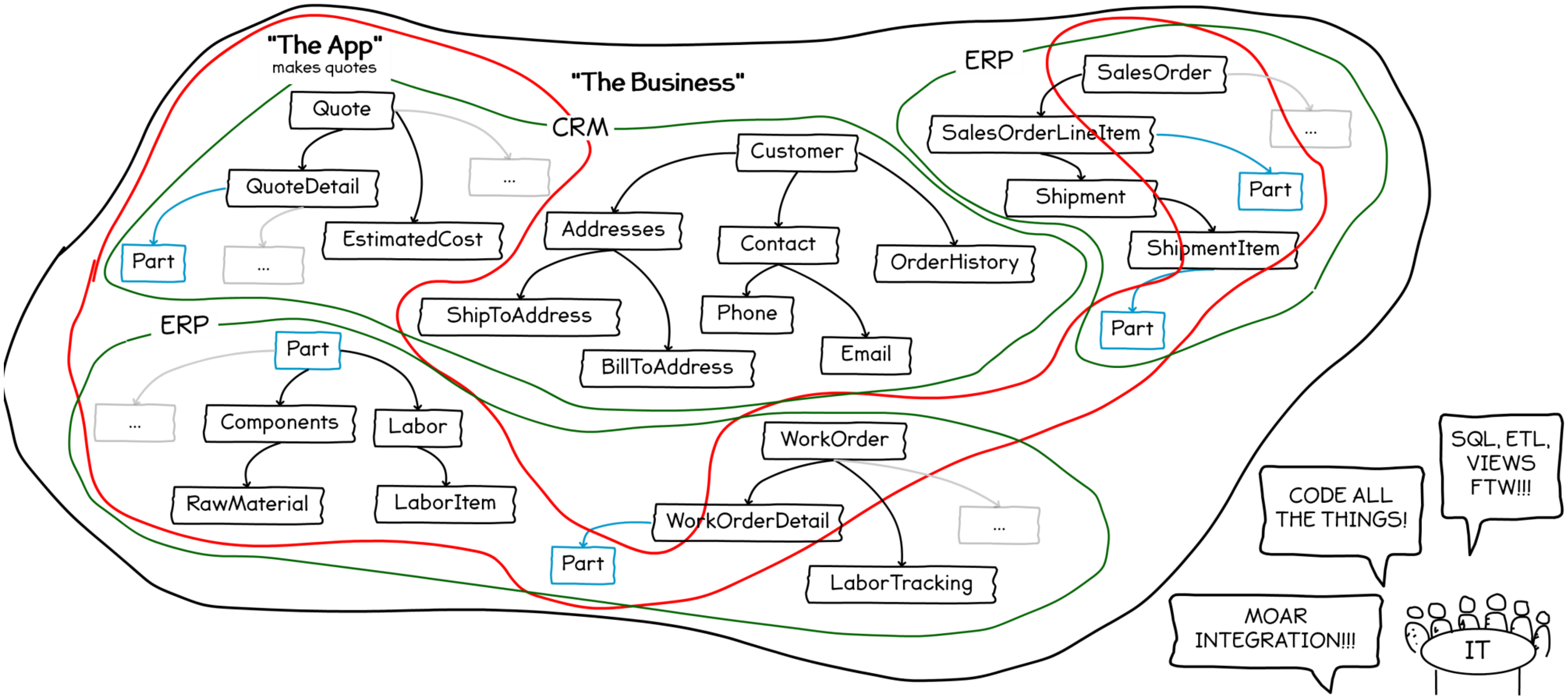
AXXES_

The OO path to success

–

Steps in our evolutions as developers

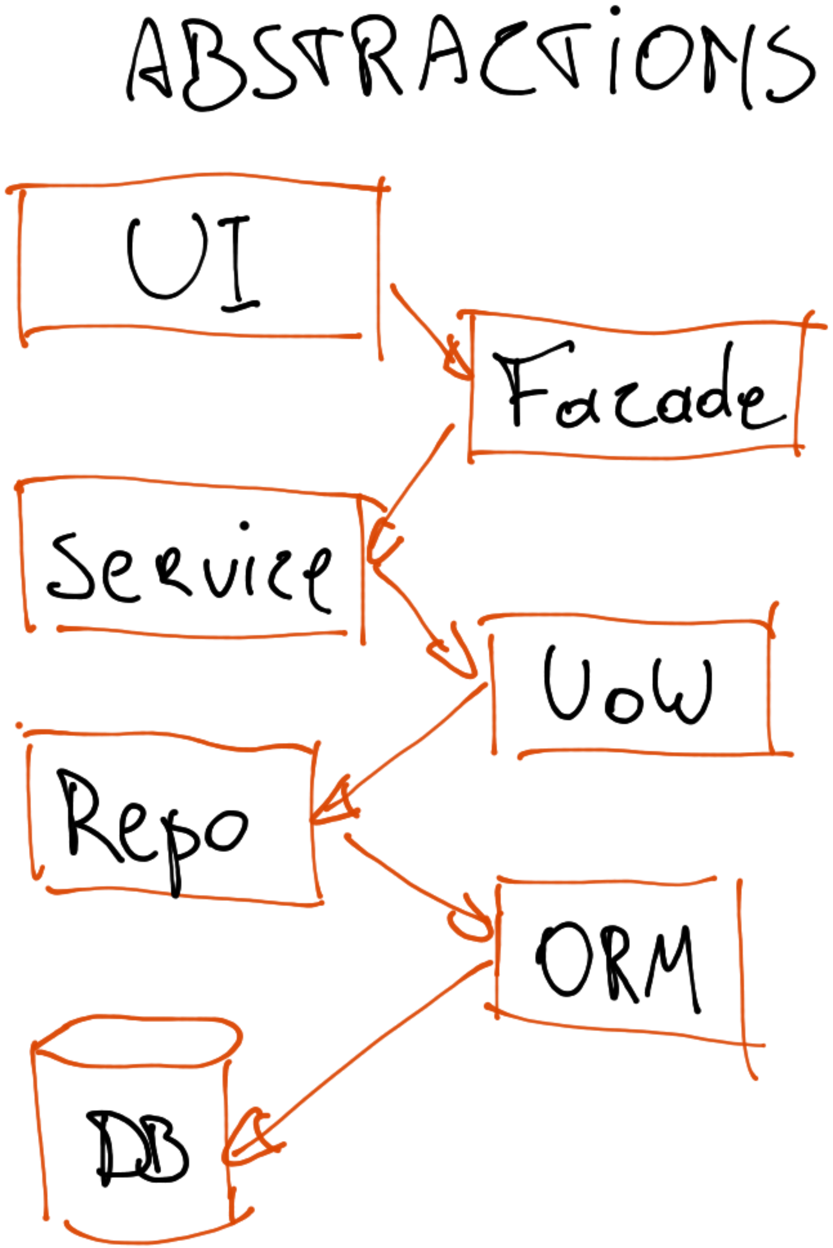
Step 1: Demoware



AXXES_

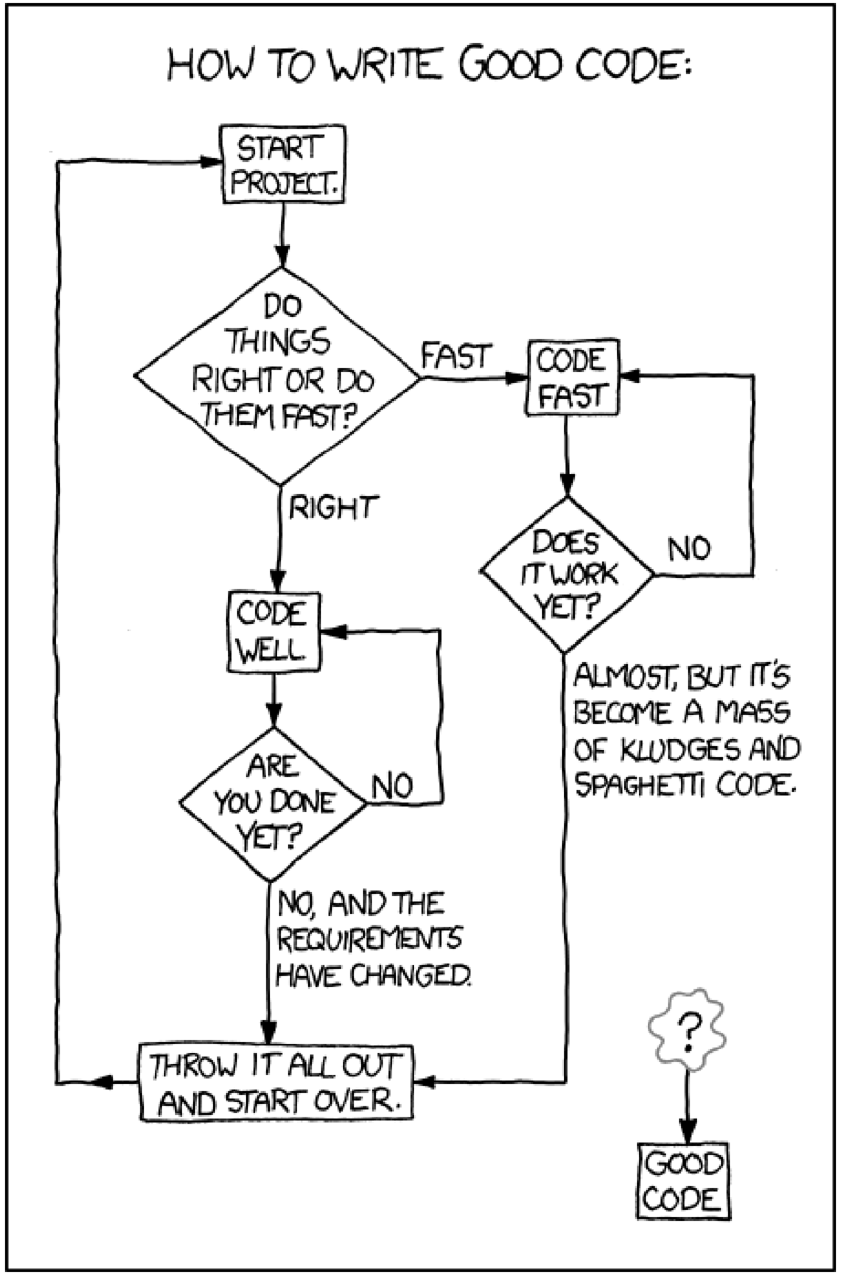
Step 2: Layers

Step 3: "SOLID"



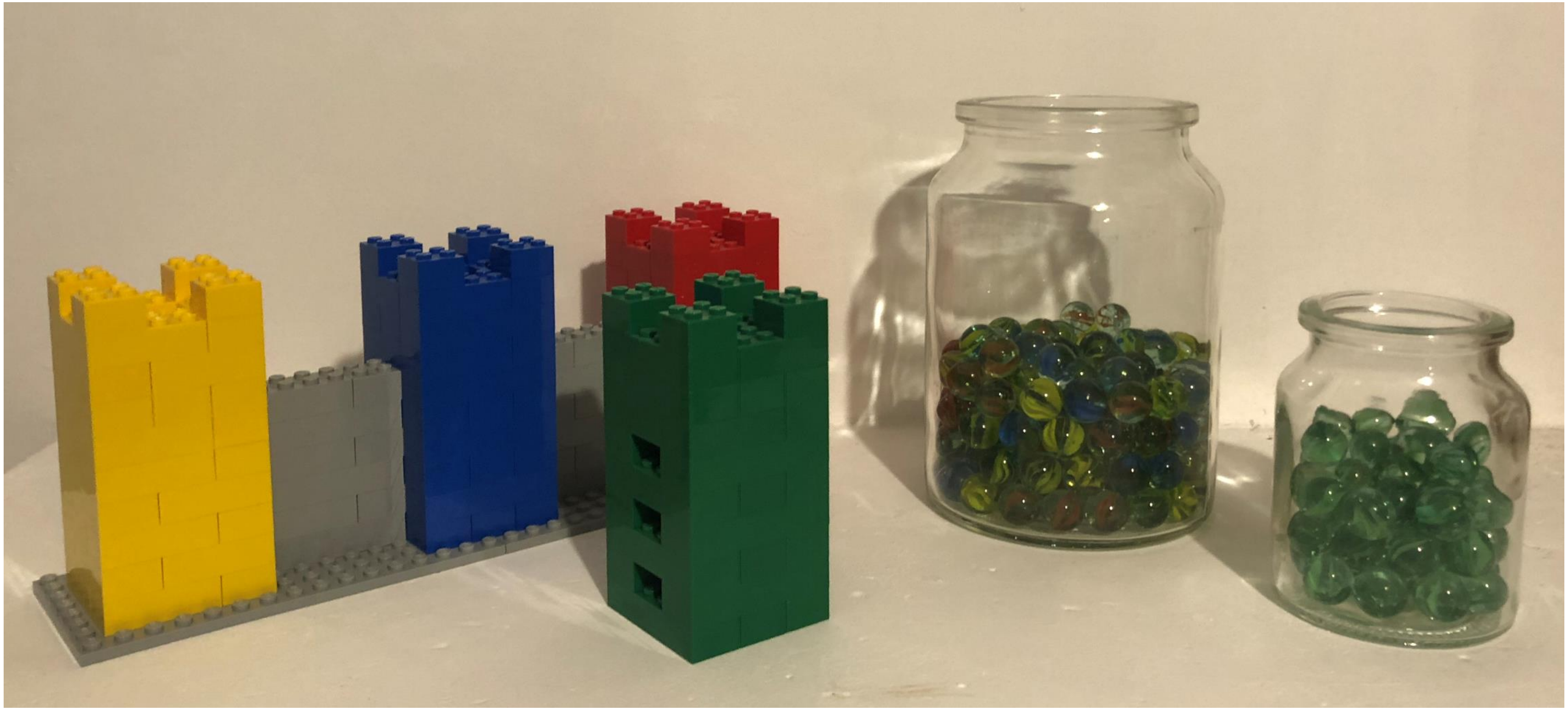
AXXES_

Step 4: SOLID


HOW TO WRITE GOOD CODE:
START PROJECT.
DO THINGS RIGHT OR DO THEM FAST?
FAST
CODE FAST
DOES IT WORK YET?
NO
ALMOST, BUT IT'S BECOME A MASS OF KLUDGES AND SPAGHETTI CODE.
RIGHT
CODE WELL
ARE YOU DONE YET?
NO
NO, AND THE REQUIREMENTS HAVE CHANGED.
THROW IT ALL OUT AND START OVER.
?
GOOD CODE

AXXES_

SOLID – A model for OO development



Step 5: Deployment models

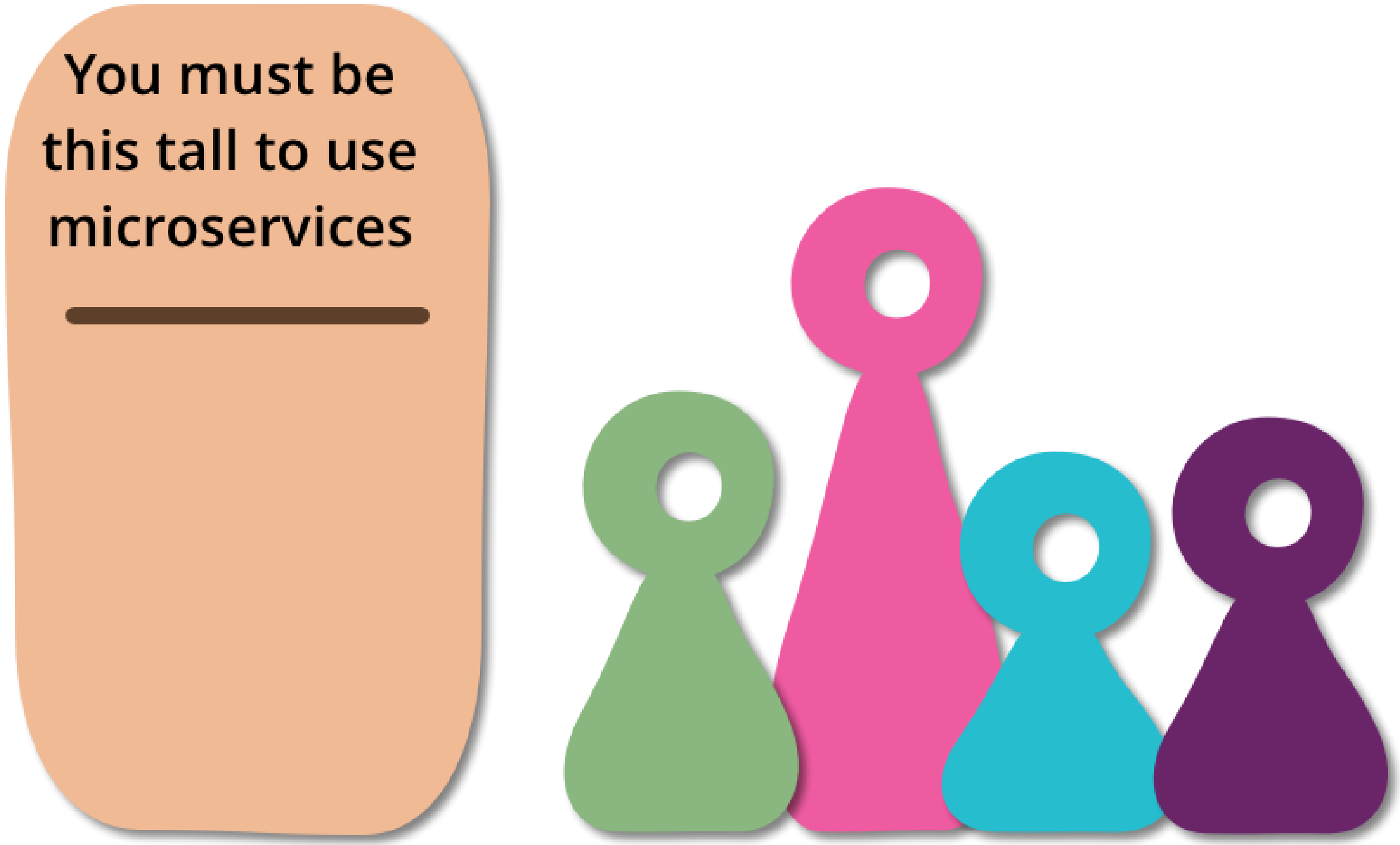
- Plugins (in this talk)
- SOA
- Bus systems
- Microservices
- …

The problem - revisited

–

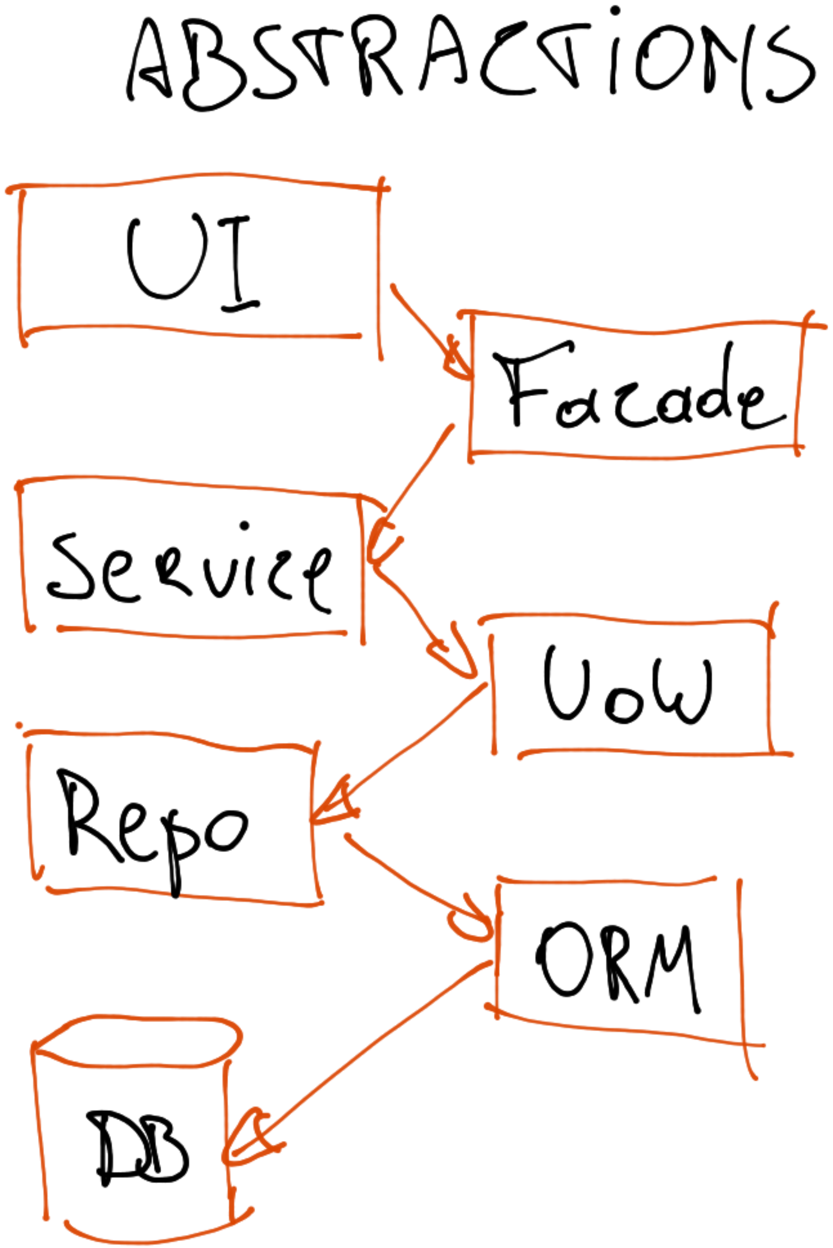
What decisions did we make?

So, microservices, right?

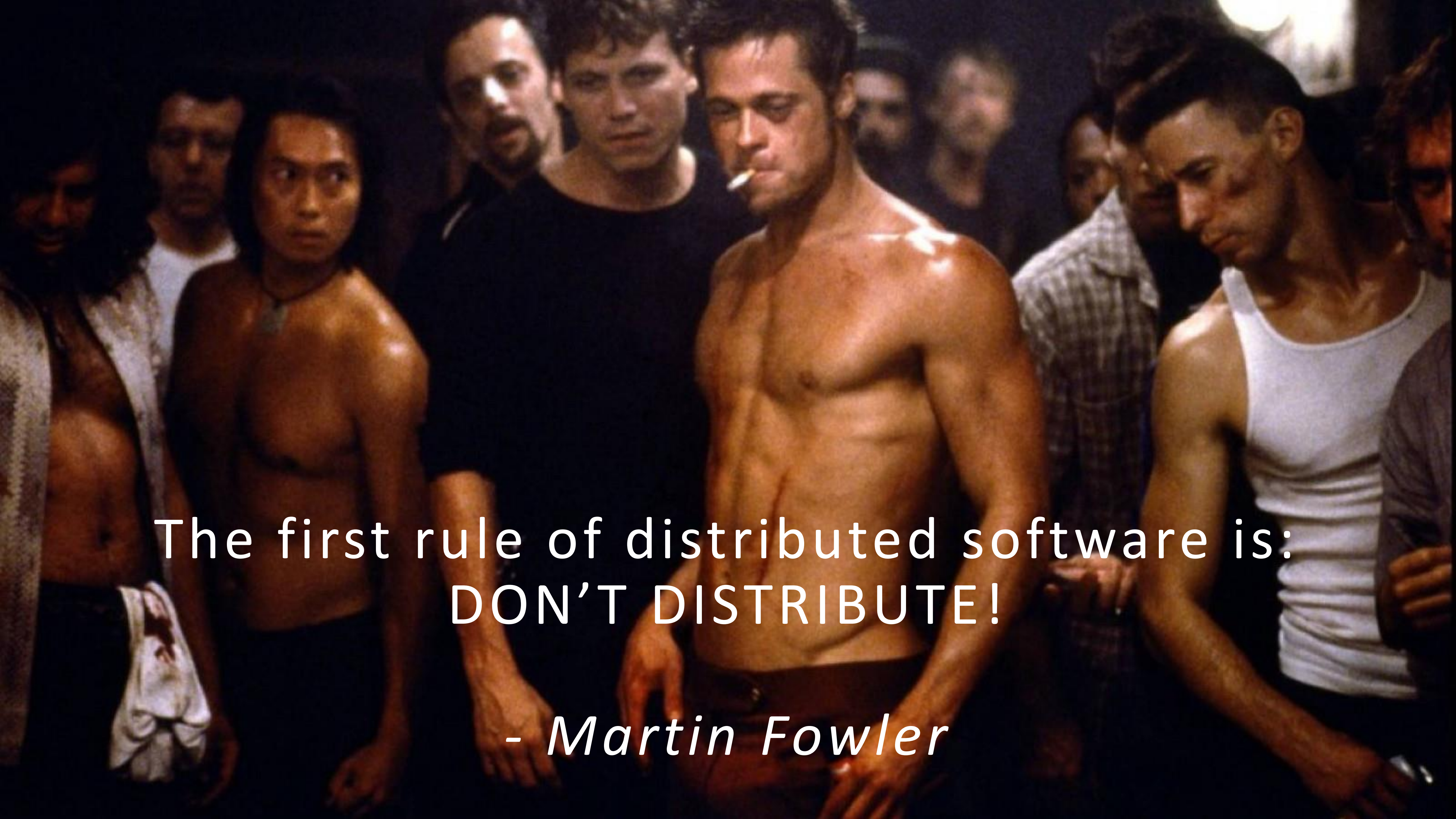


WRONG!

Step 3: "SOLID"



AXXES_



The first rule of distributed software is:
DON'T DISTRIBUTE!

- Martin Fowler

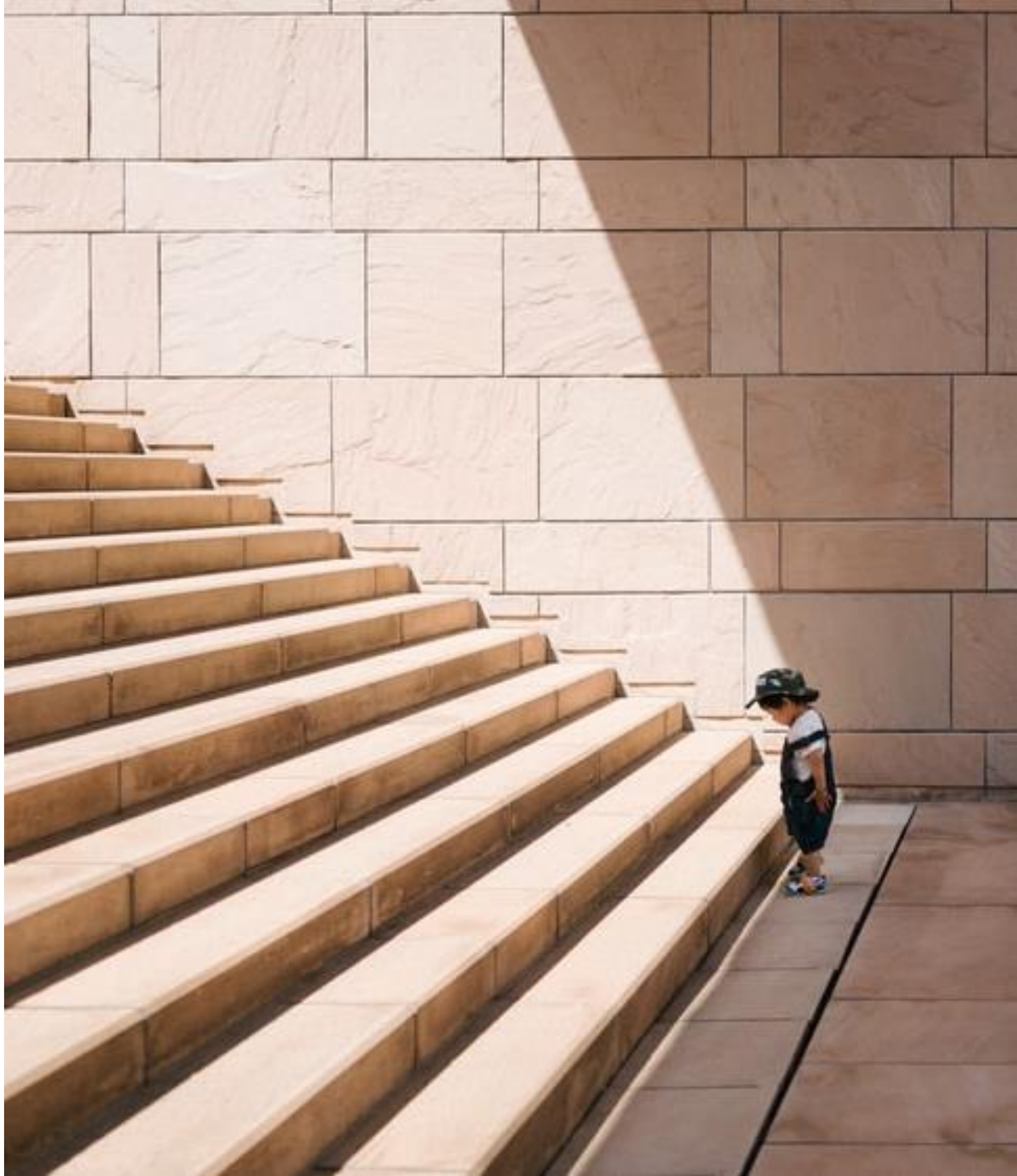
Never solve a code problem
by introducing a deployment problem!

- *Me*

Challenges

When adding a new device:

- Add controllers to ASP.NET application
- Extend API calls with derived types
  → *extend the central DbContext*
- Extend Logic in central API
- Handle DB Migrations



What happened

- We picked: **Plugins (in process)**
- We built a successful POC
- We started doing this

Onion Architecture

–

SOLID-enabled solution architecture

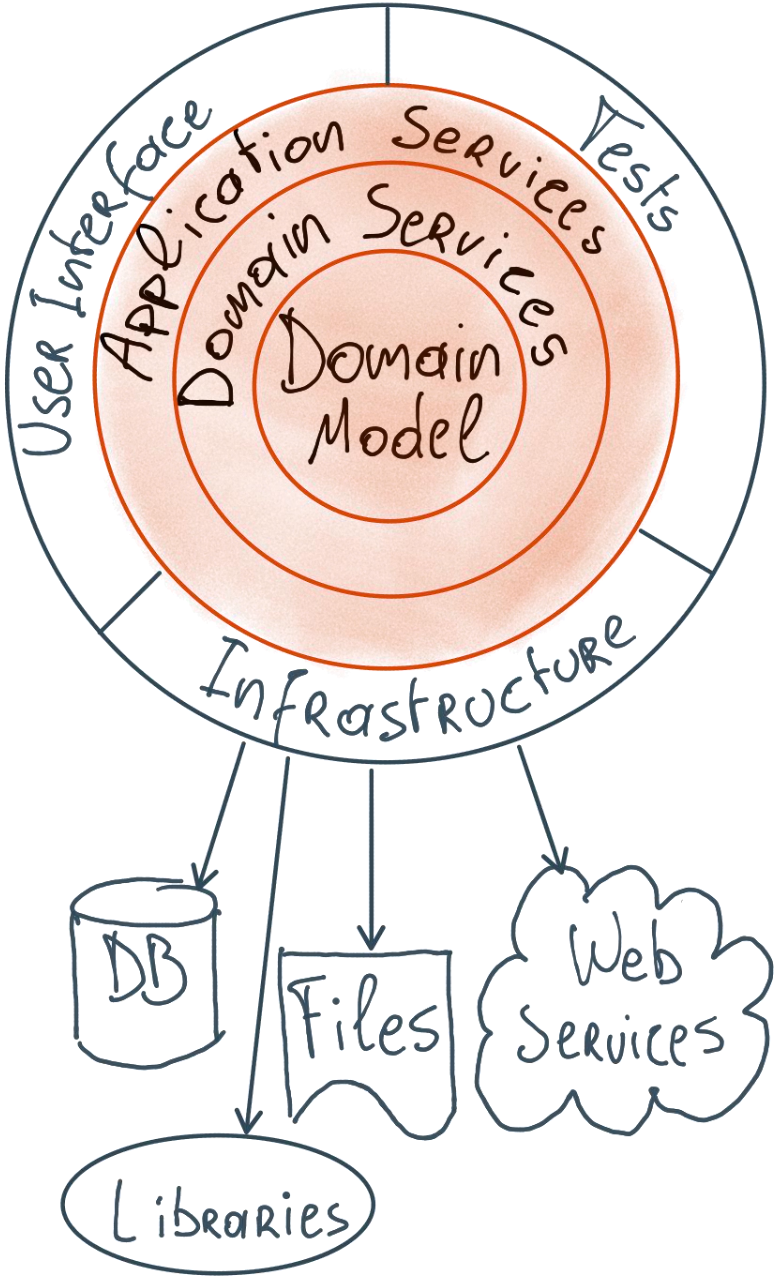
Onion Architecture

- **Jeffrey Palermo** in 2008
- Also known as:
  - ‘ports and adapters’
  - ‘hexagon architecture’
  - ‘clean architecture’
- Focus on:
  - Clean dependencies
  - Shielding abstractions
  - Testable business logic

the ONION®
America’s Finest News Source

Onion - Concepts

- References can only go ‘in’
- The Core: reference free!
- Infrastructure for integrations:
  - The DB & ORM
  - File access & logging
  - External API calls
  - Libraries & packages!

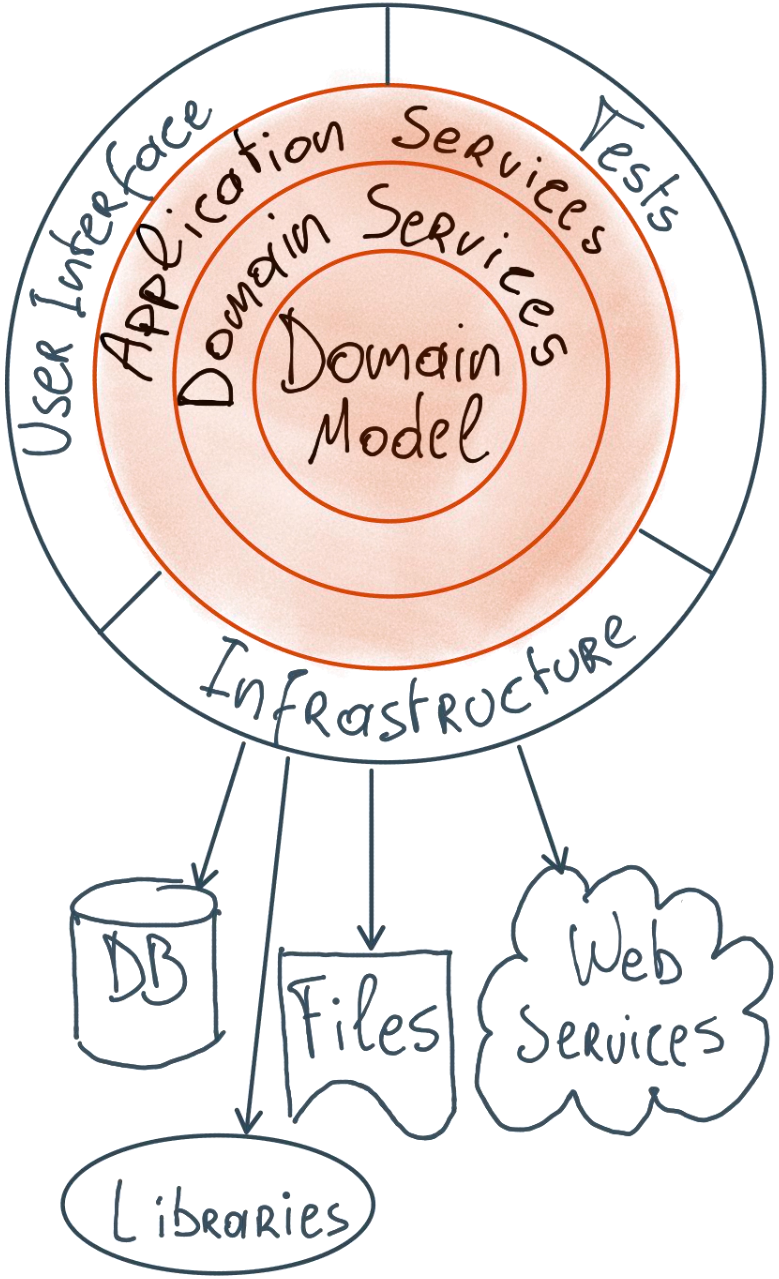


AXXES_

Onion - Concepts

Benefits:

- No leaky dependencies
- Dependencies replaceable
- Reusable Core
- Forces you to write an interface first!
  (interface owned by the consumer)

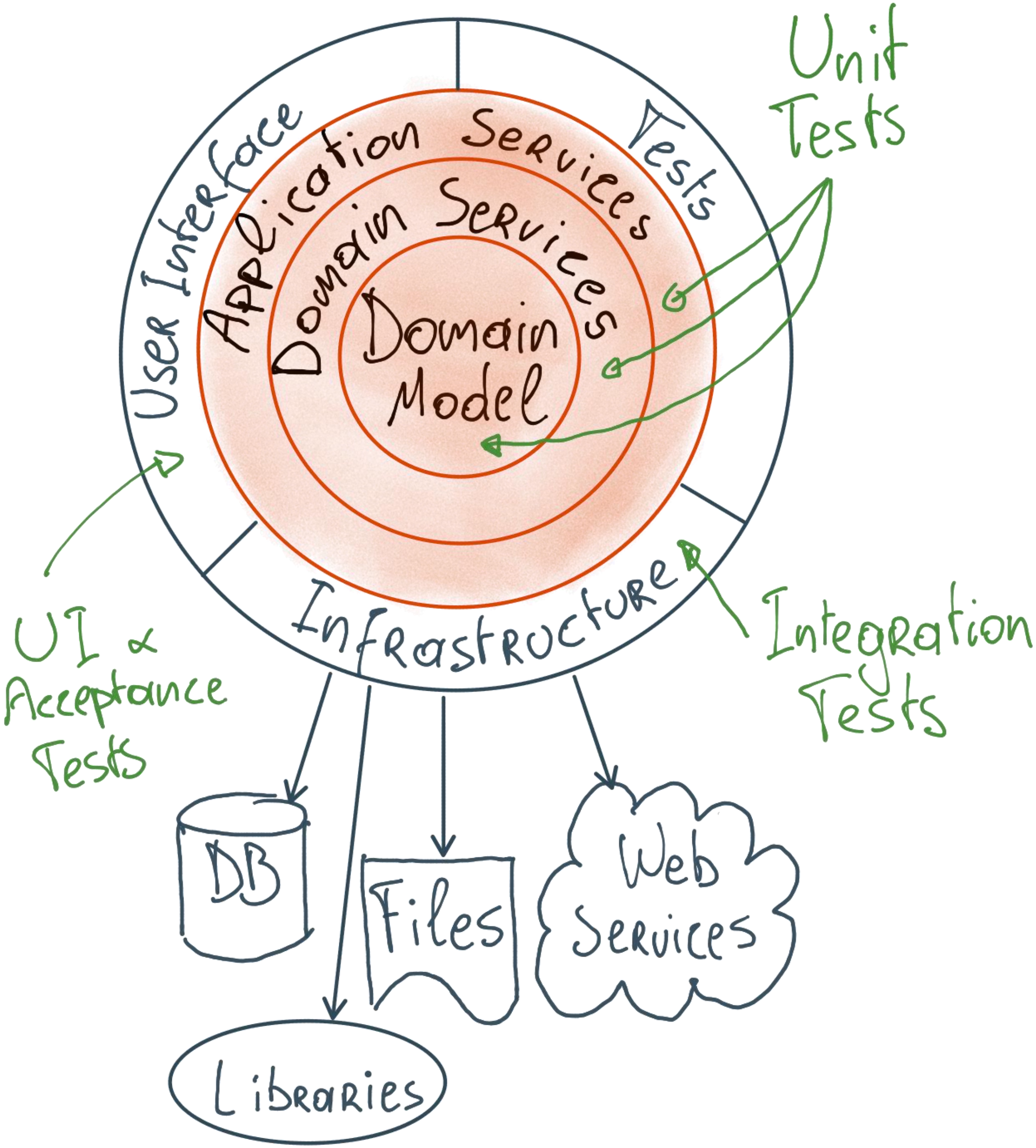


AXXES_

Onion – Testing

Easy to:

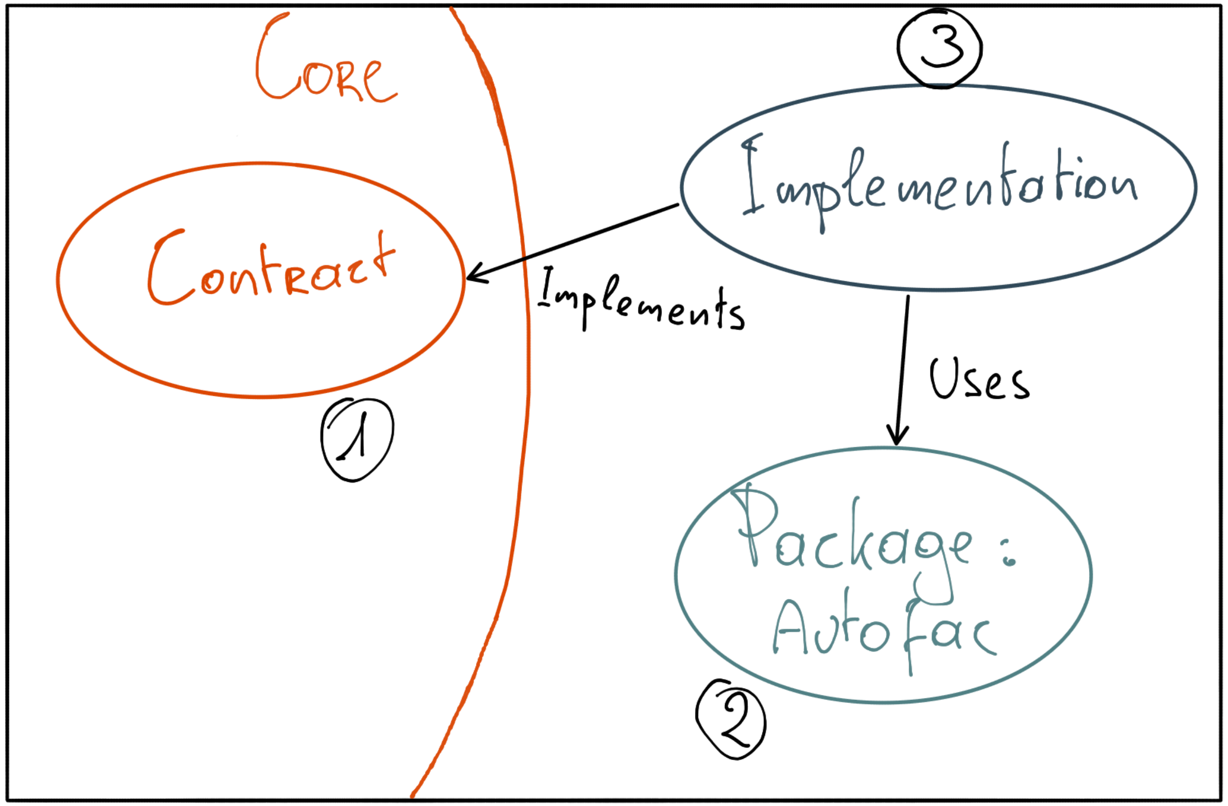
- Test all individual components
- Determine the type of tests
- Define dependencies
- Mock dependencies



AXXES_

Example: service locator

1. **Define interface(s):**
   What do I need from a DI container?
2. **Work smart:**
   Is there a package that fits this?
3. **Write an implementation**
   Use it to implement the interface.



Integrations

The resulting integrations:

- Don’t leak into your Core domain
- Easy to write
- Easy to test
- Easy to replace

Plugins

–

What do we expect from a plugin?

Plugin = assembly

- Easy to develop
- Extends our Core seamlessly
- Enable = ‘add the assemblies’
- Disable = ‘remove the assemblies’
- No references to the plugin from the Core!

AXXES_

Plugin rules

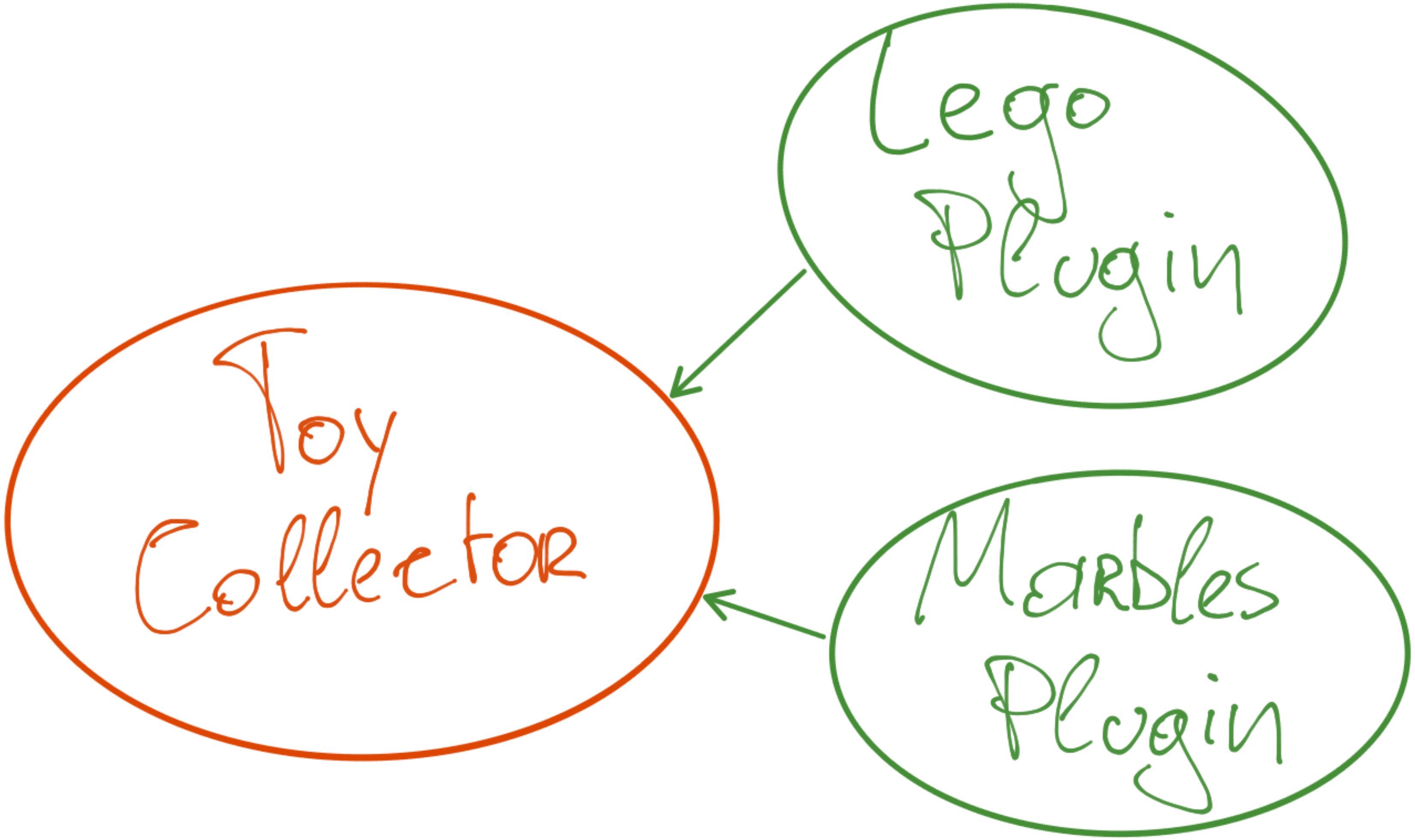
- Can only reference the Core
- Should follow some conventions
- Can be deployed with the application
- Don't break anything when removed

Implementation

–

Enough chit-chat, show us some code!

Our example



What will we discuss?

1. DI Container
2. ASP.NET controllers & views
3. JSON inheritance
4. Logic extension points
5. Extending Entity Framework
6. Migrations

AXXES_

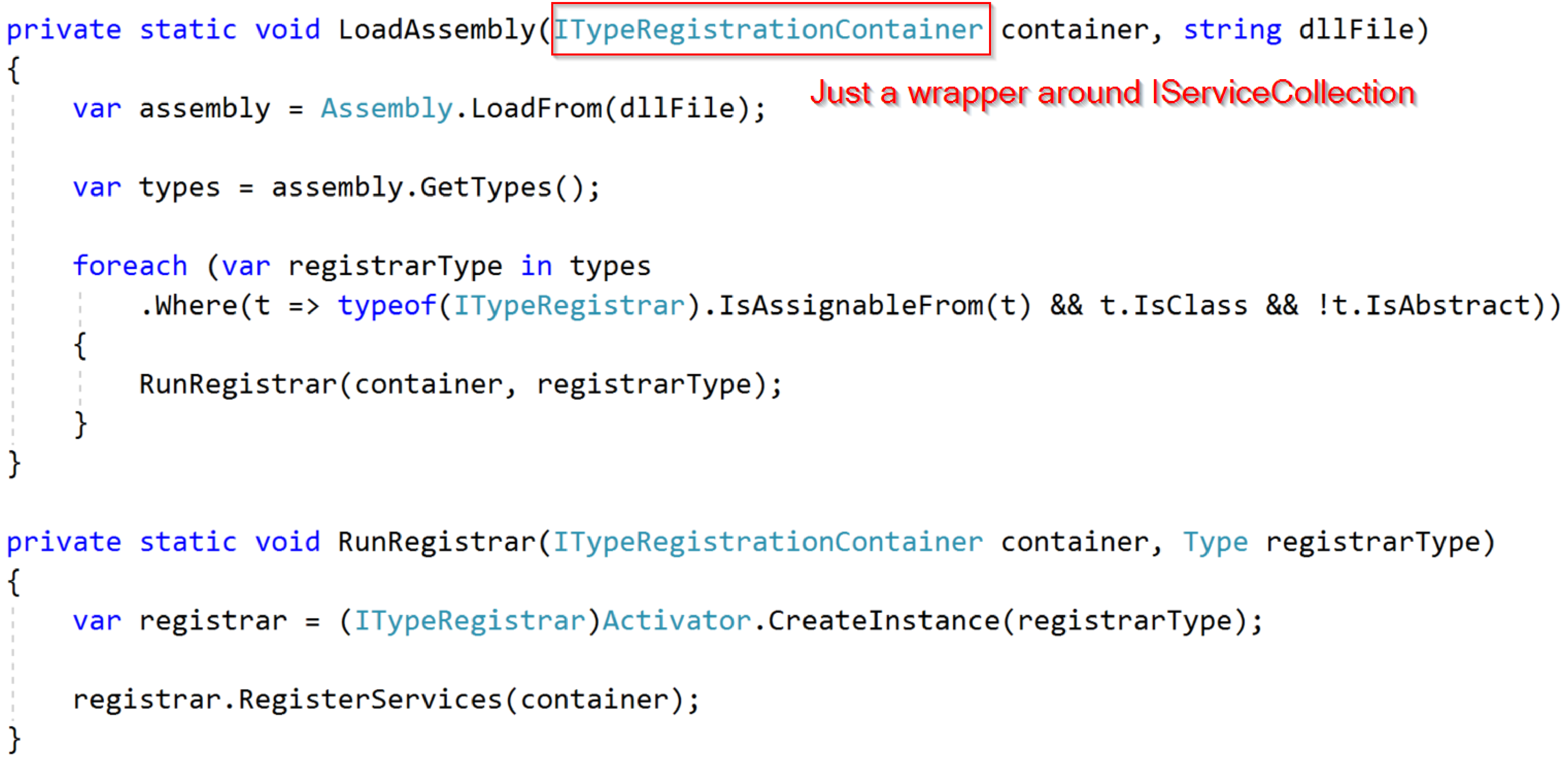
Dealing with DI

- Scan assemblies at startup
- Use reflection
- Find our type registrar in each one
- Run the type registrars

COPY YOUR ASSEMBLIES TO THE EXECUTING PROJECT
NSTRUCTO
OR YOU'RE GONNA HAVE A BAD TIME
imgflip.com

IMPORTANT for development:

- .CSPROJ: copy in ‘post build actions’ (requires manual build)
- Make a development assembly with references



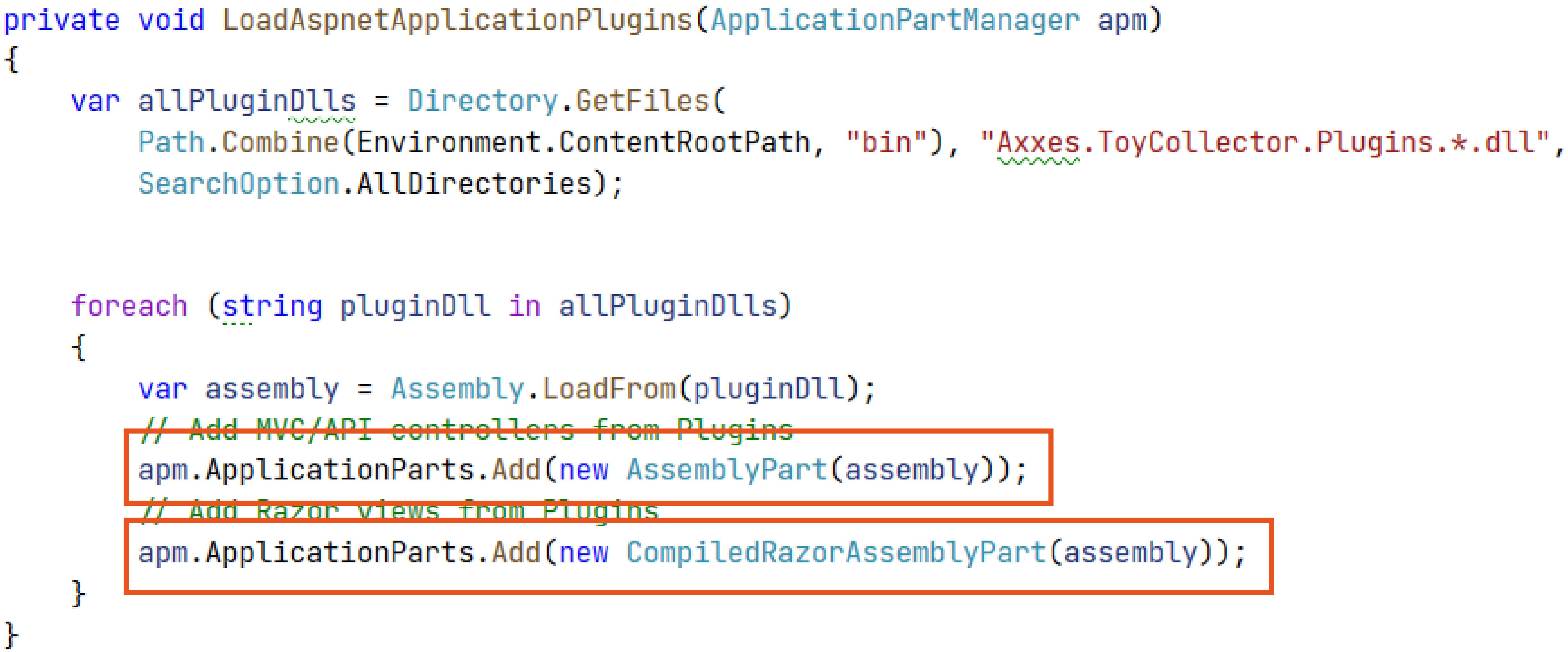
```
private static void LoadAssembly(ITypeRegistrationContainer container, string dllFile)
{
    var assembly = Assembly.LoadFrom(dllFile);

    var types = assembly.GetTypes();

    foreach (var registrarType in types
        .Where(t => typeof(ITypeRegistrar).IsAssignableFrom(t) && t.IsClass && !t.IsAbstract))
    {
        RunRegistrar(container, registrarType);
    }
}

private static void RunRegistrar(ITypeRegistrationContainer container, Type registrarType)
{
    var registrar = (ITypeRegistrar)Activator.CreateInstance(registrarType);

    registrar.RegisterServices(container);
}
```

Just a wrapper around IServiceCollection

ASP.NET

- **Controllers:**
  Use the application part manager to add them
- **Views:**
  Add the Views DLL (standard output in recent ASP.NET Core)
  using `CompiledRazorAssemblyPart`

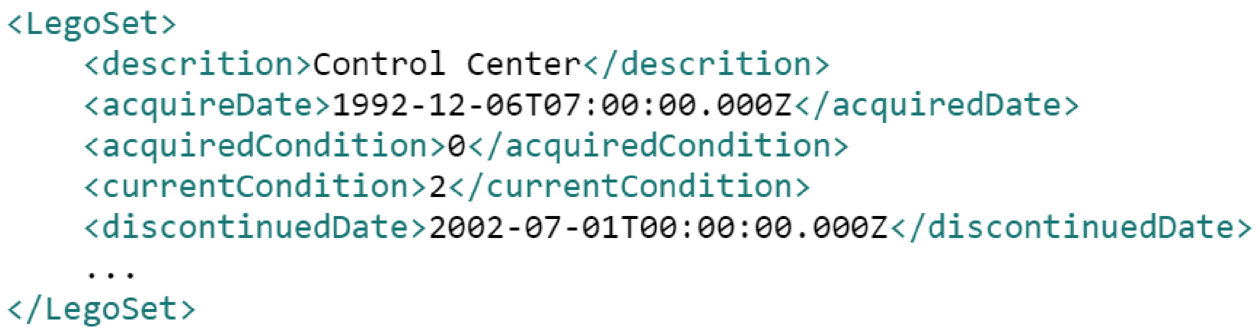
- **Pre-Core ASP.NET MVC:**
  Custom Controller Selector & View Selector



```csharp
private void LoadAspnetApplicationPlugins(ApplicationPartManager apm)
{
    var allPluginDlls = Directory.GetFiles(
        Path.Combine(Environment.ContentRootPath, "bin"), "Axxes.ToyCollector.Plugins.*.dll",
        SearchOption.AllDirectories);


    foreach (string pluginDll in allPluginDlls)
    {
        var assembly = Assembly.LoadFrom(pluginDll);
        // Add MVC/API controllers from Plugins
        apm.ApplicationParts.Add(new AssemblyPart(assembly));
        // Add Razor views from Plugins
        apm.ApplicationParts.Add(new CompiledRazorAssemblyPart(assembly));
    }
}
```

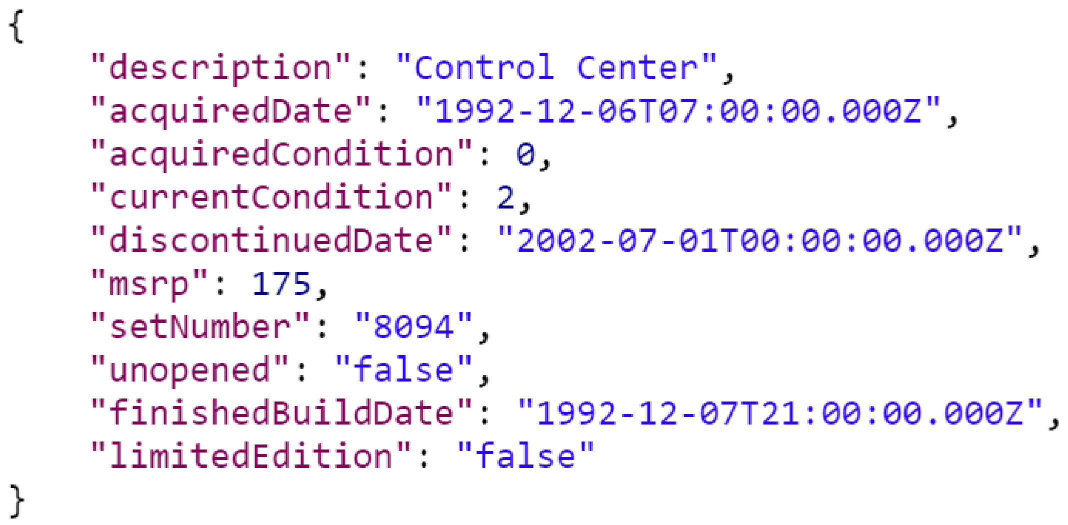
Posting inherited types

- **Easy – XML:**



```xml
<LegoSet>
    <descrition>Control Center</descrition>
    <acquireDate>1992-12-06T07:00:00.000Z</acquiredDate>
    <acquiredCondition>0</acquiredCondition>
    <currentCondition>2</currentCondition>
    <discontinuedDate>2002-07-01T00:00:00.000Z</discontinuedDate>
    ...
</LegoSet>
```

- **Hard – JSON:**



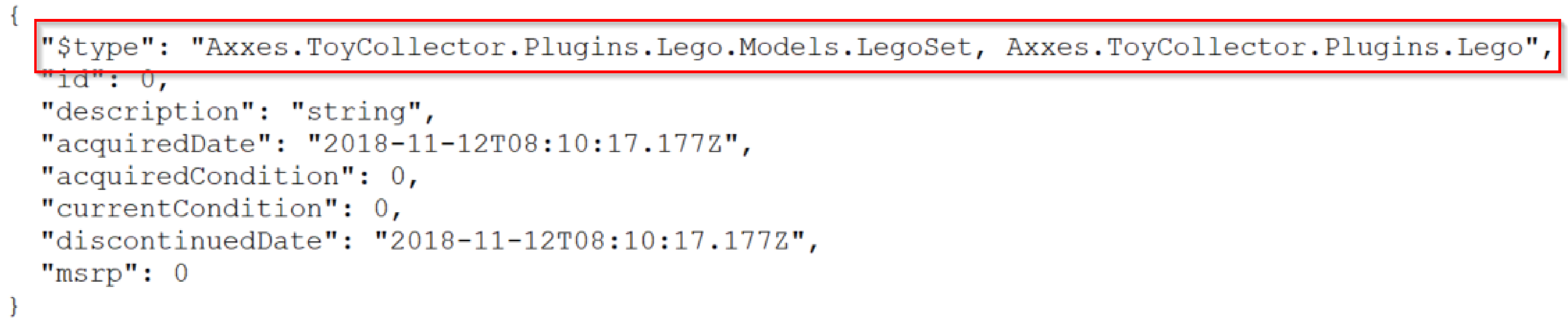
```json
{
    "description": "Control Center",
    "acquiredDate": "1992-12-06T07:00:00.000Z",
    "acquiredCondition": 0,
    "currentCondition": 2,
    "discontinuedDate": "2002-07-01T00:00:00.000Z",
    "msrp": 175,
    "setNumber": "8094",
    "unopened": "false",
    "finishedBuildDate": "1992-12-07T21:00:00.000Z",
    "limitedEdition": "false"
}
```

JSON inheritance – the risky solution

- **Registration**

```
// Allows the passing of JSON $type parameters (required for inherited types)
mvcBuilder.AddJsonOptions(
    jsonOptions => jsonOptions.SerializerSettings.TypeNameHandling = TypeNameHandling.Auto);
```

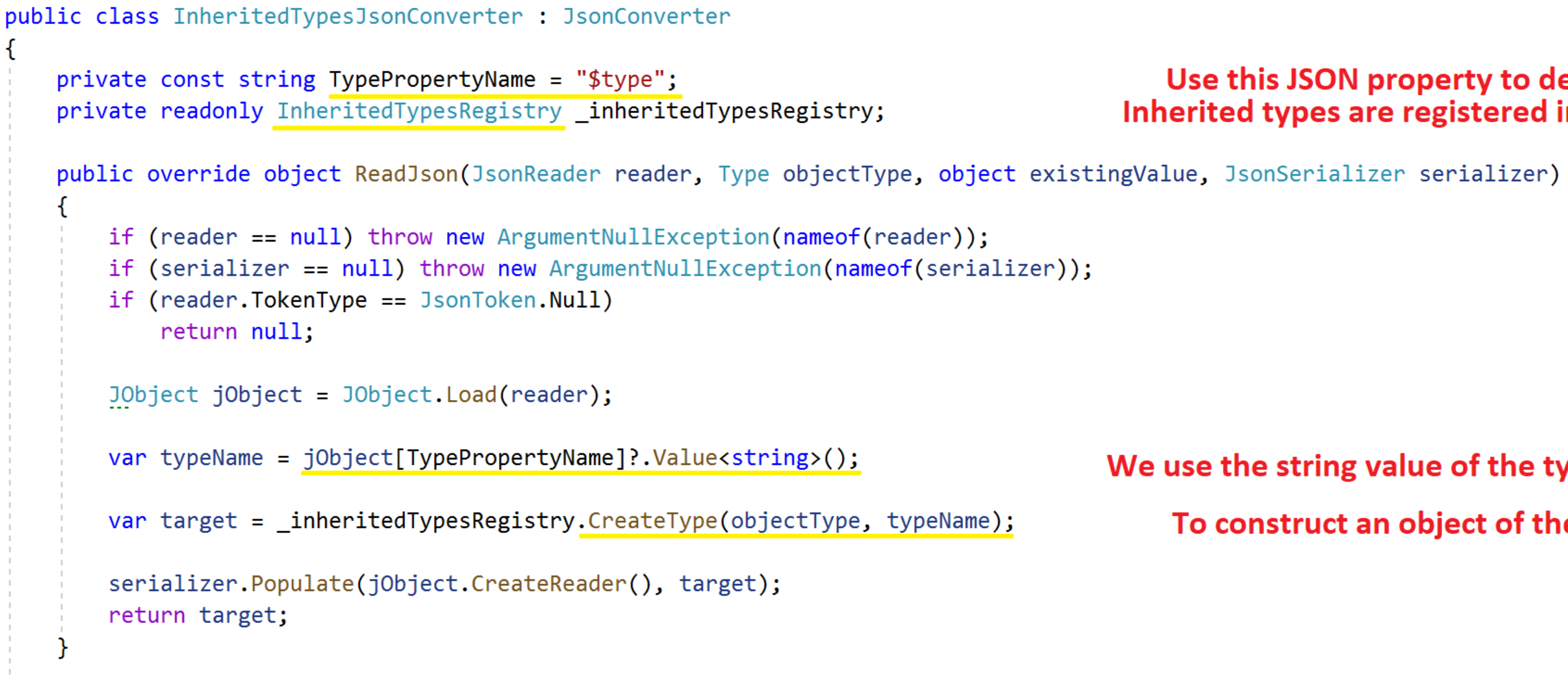
- **Usage**



```
{
  "$type": "Axxes.ToyCollector.Plugins.Lego.Models.LegoSet, Axxes.ToyCollector.Plugins.Lego",
  "id": 0,
  "description": "string",
  "acquiredDate": "2018-11-12T08:10:17.177Z",
  "acquiredCondition": 0,
  "currentCondition": 0,
  "discontinuedDate": "2018-11-12T08:10:17.177Z",
  "msrp": 0
}
```

→ Serious vulnerability if you have an object/dynamic property

JSON inheritance – the proper solution



```
public class InheritedTypesJsonConverter : JsonConverter
{
    private const string TypePropertyName = "$type";
    private readonly InheritedTypesRegistry _inheritedTypesRegistry;

    public override object ReadJson(JsonReader reader, Type objectType, object existingValue, JsonSerializer serializer)
    {
        if (reader == null) throw new ArgumentNullException(nameof(reader));
        if (serializer == null) throw new ArgumentNullException(nameof(serializer));
        if (reader.TokenType == JsonToken.Null)
            return null;

        JObject jObject = JObject.Load(reader);

        var typeName = jObject[TypePropertyName]?.Value<string>();

        var target = _inheritedTypesRegistry.CreateType(objectType, typeName);

        serializer.Populate(jObject.CreateReader(), target);
        return target;
    }
```

Use this JSON property to detect the type

Inherited types are registered in this registry

We use the string value of the type$ property

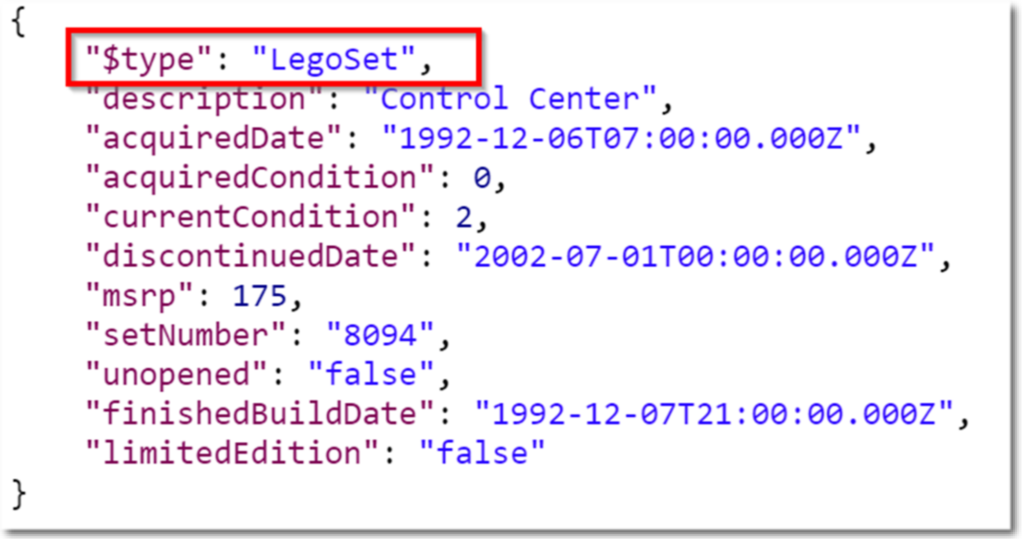
To construct an object of the proper type

JSON inheritance – the proper solution

- **Registration**

```
// Allows the passing of JSON $type parameters (required for inherited types)
mvcBuilder.AddJsonOptions(jsonOptions =>
{
    jsonOptions.SerializerSettings.Converters.Add(new InheritedTypesJsonConverter(inheritedTypesRegistry));
});
```

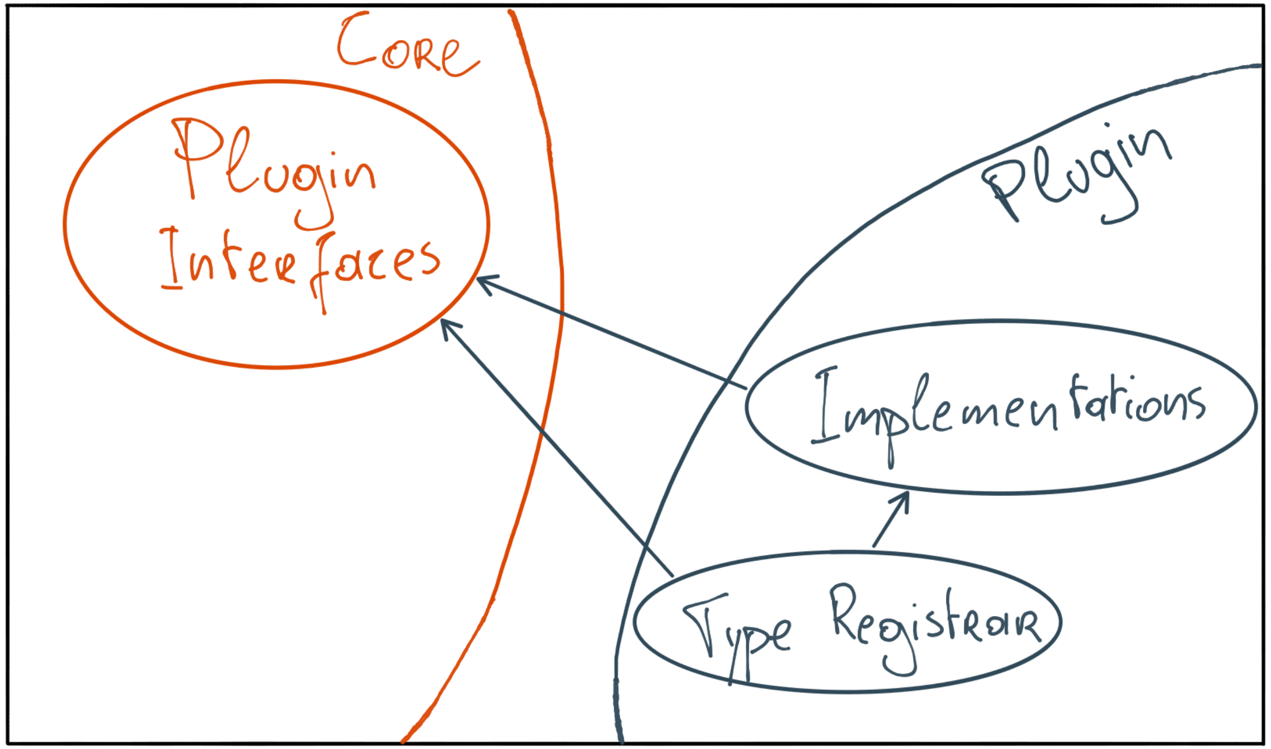
- **Usage**

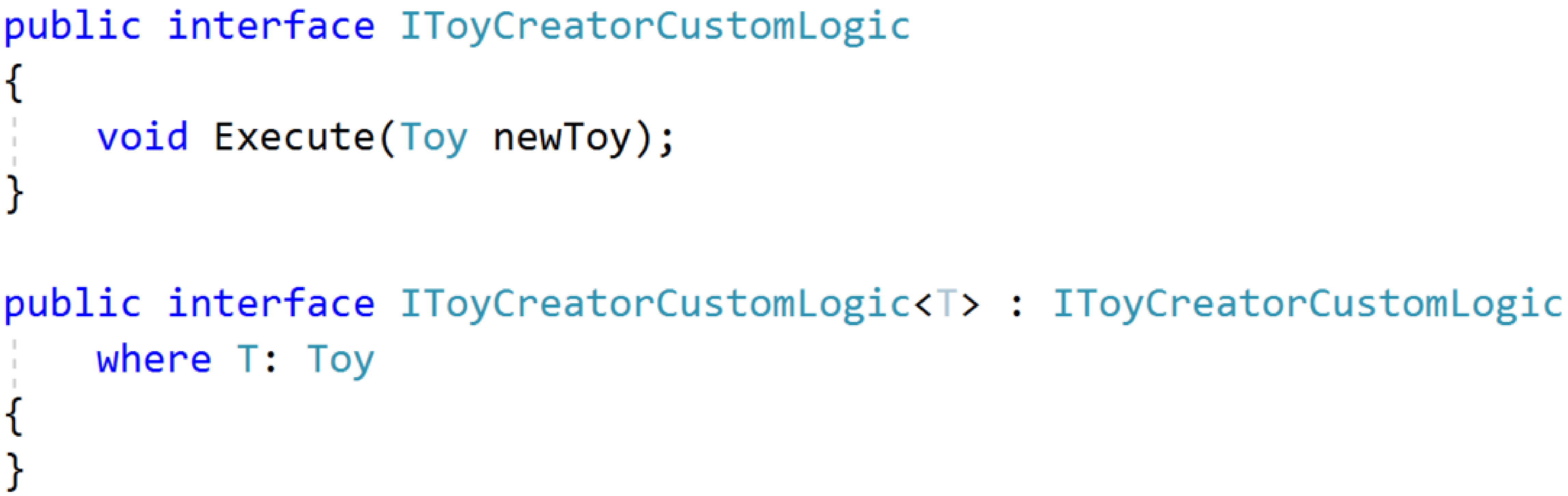


```
{
    "$type": "LegoSet",
    "description": "Control Center",
    "acquiredDate": "1992-12-06T07:00:00.000Z",
    "acquiredCondition": 0,
    "currentCondition": 2,
    "discontinuedDate": "2002-07-01T00:00:00.000Z",
    "msrp": 175,
    "setNumber": "8094",
    "unopened": "false",
    "finishedBuildDate": "1992-12-07T21:00:00.000Z",
    "limitedEdition": "false"
}
```

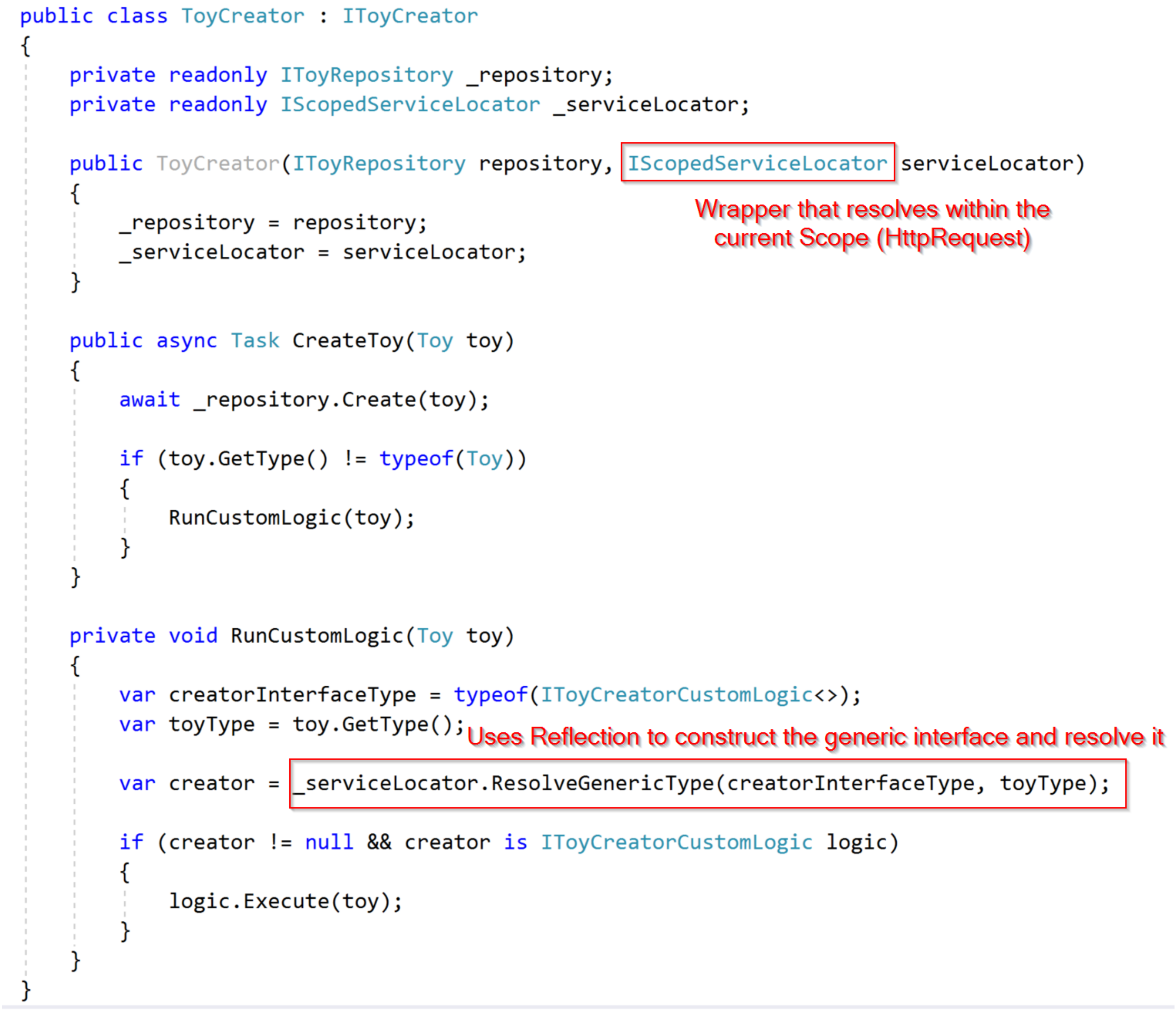
Logic extension points

1. Define generic interfaces
2. Implement them in the plugin
3. Usage:
   - Scoped service locator
   - Simply inject?





```
public interface IToyCreatorCustomLogic
{
    void Execute(Toy newToy);
}

public interface IToyCreatorCustomLogic<T> : IToyCreatorCustomLogic
    where T: Toy
{
}
```



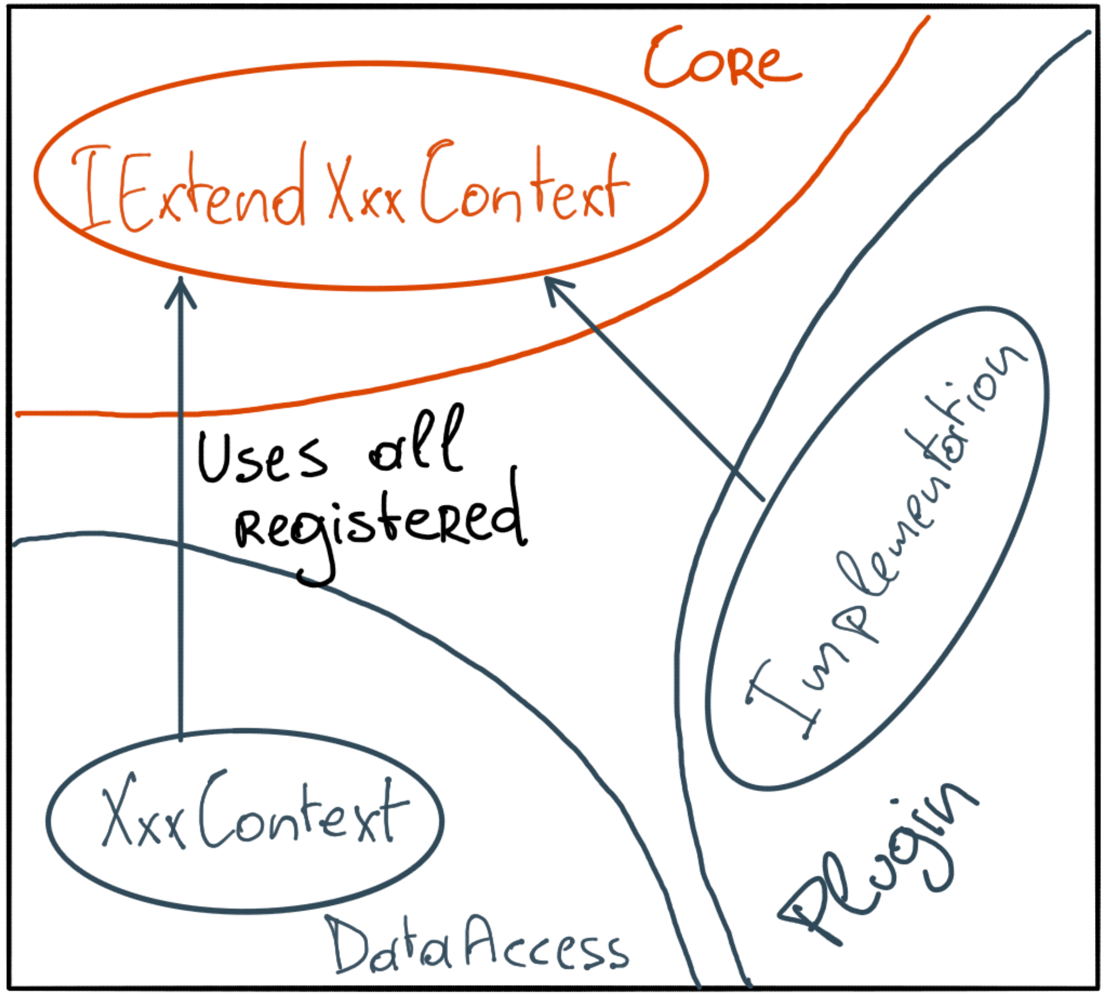
```
public class ToyCreator : IToyCreator
{
    private readonly IToyRepository _repository;
    private readonly IScopedServiceLocator _serviceLocator;

    public ToyCreator(IToyRepository repository, IScopedServiceLocator serviceLocator)
    {
        _repository = repository;
        _serviceLocator = serviceLocator;
    }

    public async Task CreateToy(Toy toy)
    {
        await _repository.Create(toy);

        if (toy.GetType() != typeof(Toy))
        {
            RunCustomLogic(toy);
        }
    }

    private void RunCustomLogic(Toy toy)
    {
        var creatorInterfaceType = typeof(IToyCreatorCustomLogic<>);
        var toyType = toy.GetType();

        var creator = _serviceLocator.ResolveGenericType(creatorInterfaceType, toyType);

        if (creator != null && creator is IToyCreatorCustomLogic logic)
        {
            logic.Execute(toy);
        }
    }
}
```

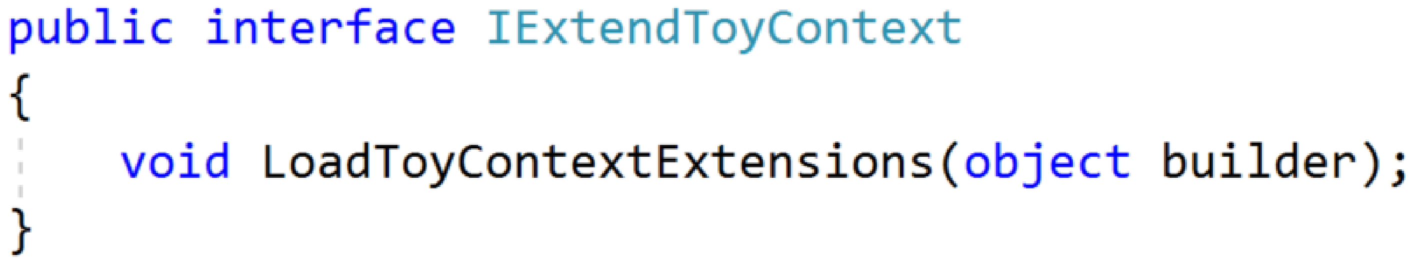
Wrapper that resolves within the current Scope (HttpRequest)

Uses Reflection to construct the generic interface and resolve it

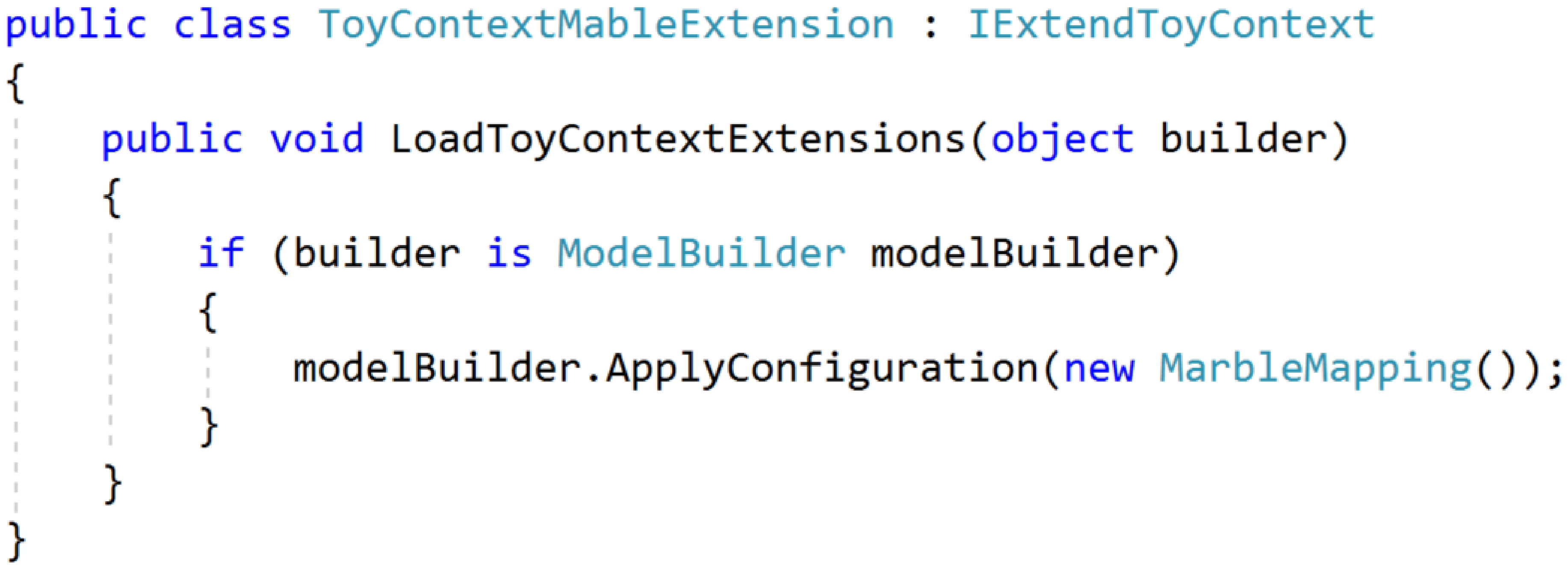
Entity Framework

- Use `OnModelCreating` to feed the `DbContext` new (inherited) types
- EF Core = only TPH inheritance
- EF adds a Discriminator where clause

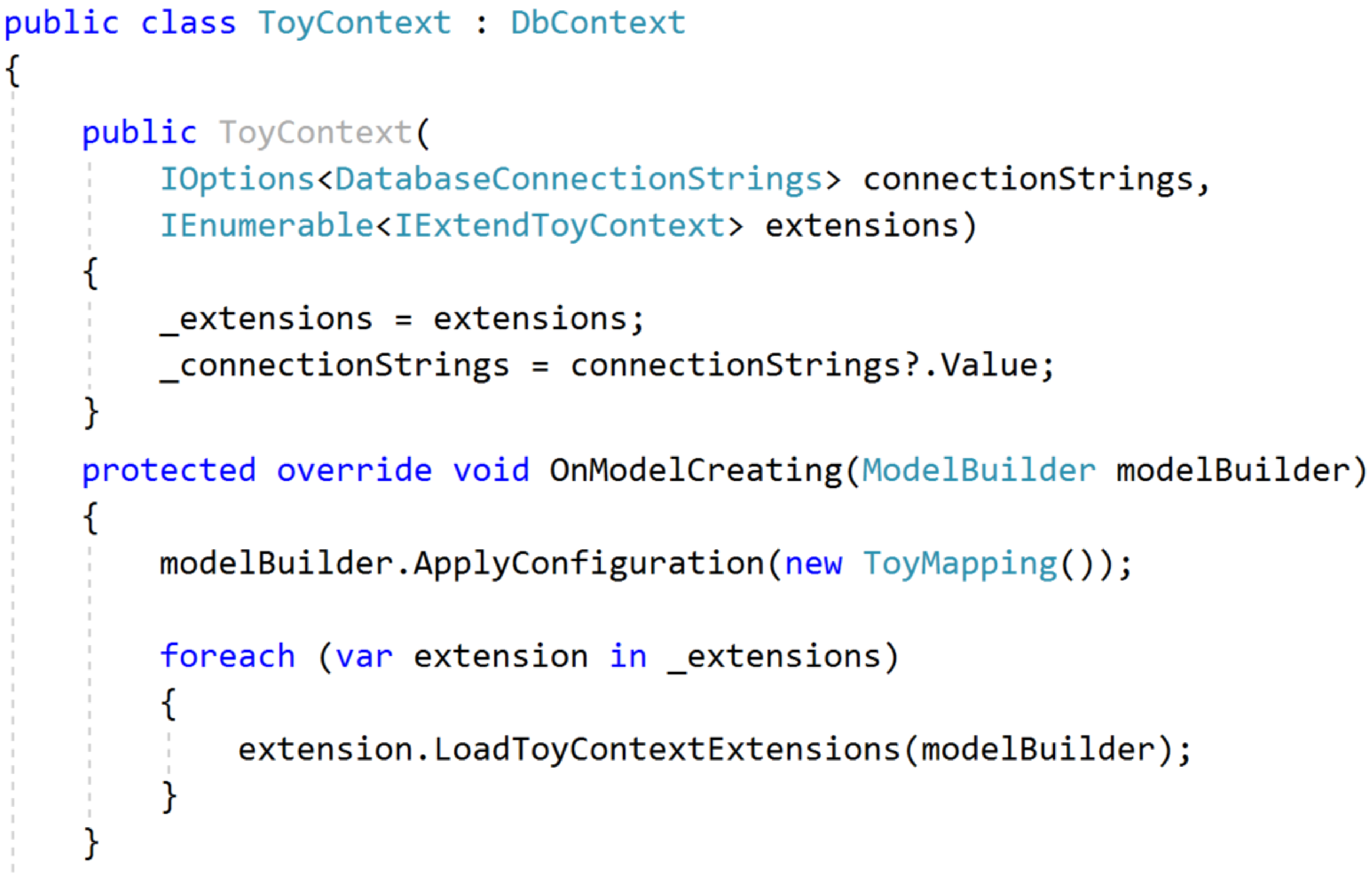




```
public interface IExtendToyContext
{
    void LoadToyContextExtensions(object builder);
}
```



```
public class ToyContextMableExtension : IExtendToyContext
{
    public void LoadToyContextExtensions(object builder)
    {
        if (builder is ModelBuilder modelBuilder)
        {
            modelBuilder.ApplyConfiguration(new MarbleMapping());
        }
    }
}
```



```
public class ToyContext : DbContext
{
    public ToyContext(
        IOptions<DatabaseConnectionStrings> connectionStrings,
        IEnumerable<IExtendToyContext> extensions)
    {
        _extensions = extensions;
        _connectionStrings = connectionStrings?.Value;
    }
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.ApplyConfiguration(new ToyMapping());

        foreach (var extension in _extensions)
        {
            extension.LoadToyContextExtensions(modelBuilder);
        }
    }
```

Database Migrations

Central Approach	Distributed approach
+ Can be generated + Easy to execute at deploy time	+ Every plugin has its own migrations + Database 100% in sync w/ plugins
- All tables/fields exist even if the plugins are not loaded - Will require you to deal with the MigrationHistory table	- Trickier to code & test - Requires runtime migrations

Central migrations: EF Core

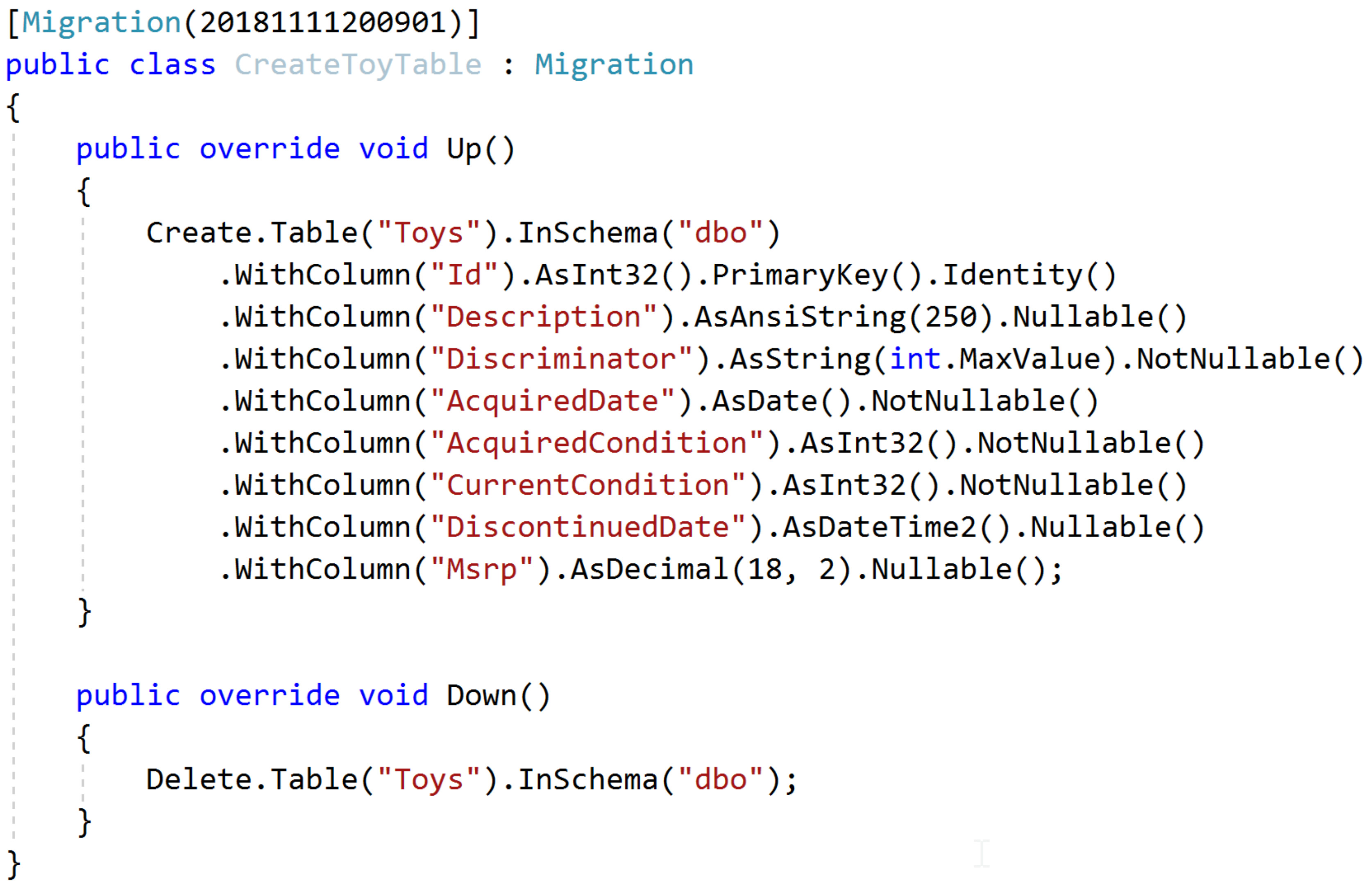
- 1 Central Migrations project
- Easy to generate migrations!
- Uses service configuration of your startup project
  BY DEFAULT

- **Without all plugins loaded, the model won't match!**
  - Generate migration scripts from the migrations
  - Rename the migration history table before/after deploy

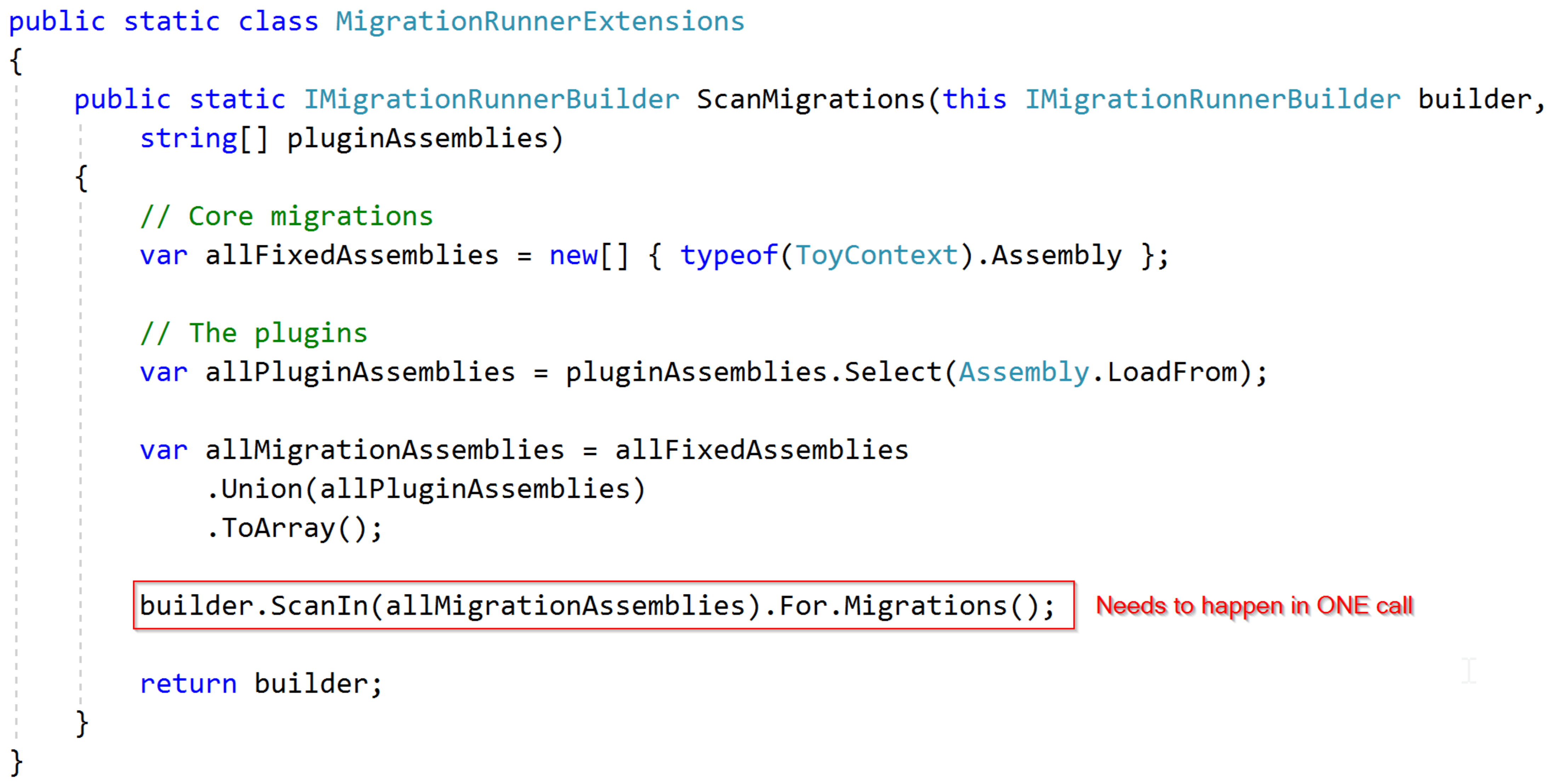
Distributed migrations: FluentMigrator

- Every Plugin its own migrations
- Either:
  - Run migrations at runtime
  - Make a runner that loads the deployed plugins

- Migrations need to be hand-coded
- The FluentMigrator API is easy to learn
- No problems with the ModelState



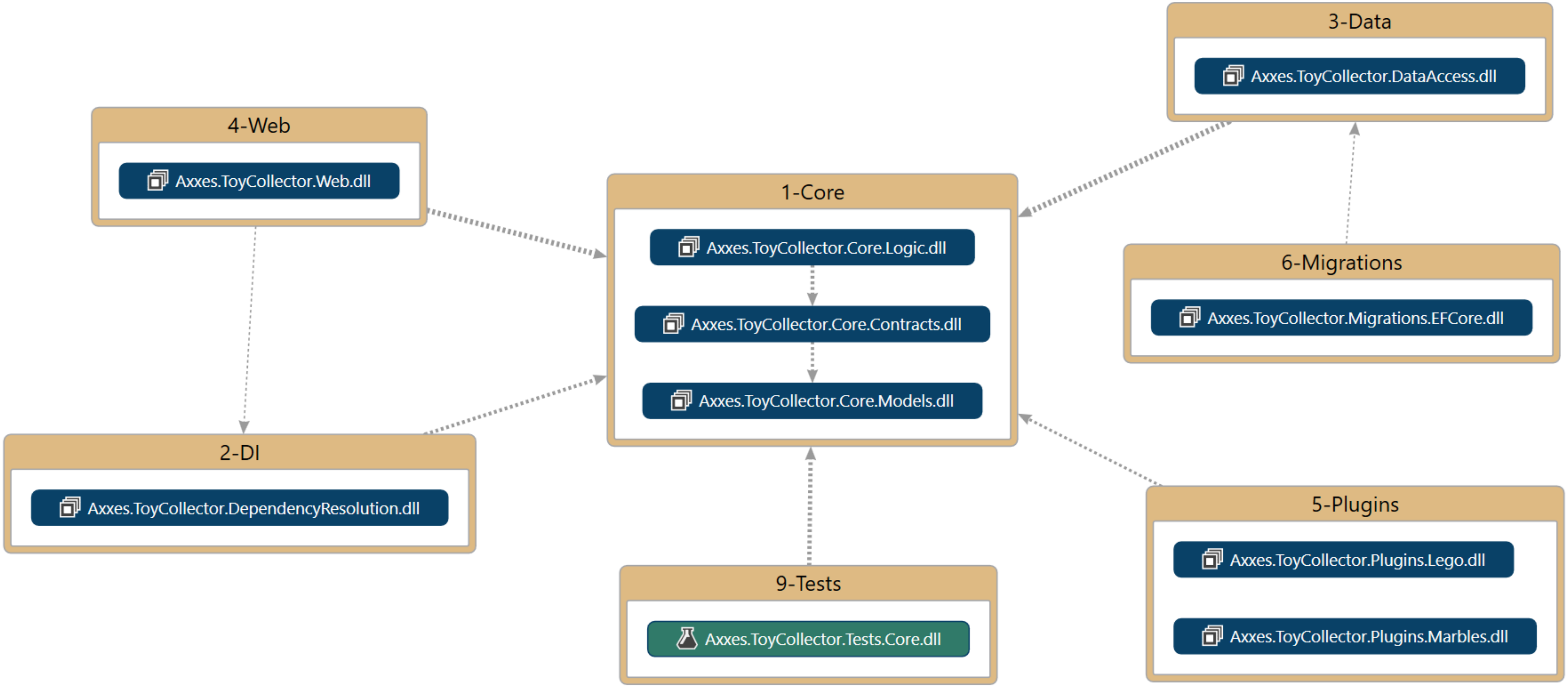
```
[Migration(20181111200901)]
public class CreateToyTable : Migration
{
    public override void Up()
    {
        Create.Table("Toys").InSchema("dbo")
            .WithColumn("Id").AsInt32().PrimaryKey().Identity()
            .WithColumn("Description").AsAnsiString(250).Nullable()
            .WithColumn("Discriminator").AsString(int.MaxValue).NotNullable()
            .WithColumn("AcquiredDate").AsDate().NotNullable()
            .WithColumn("AcquiredCondition").AsInt32().NotNullable()
            .WithColumn("CurrentCondition").AsInt32().NotNullable()
            .WithColumn("DiscontinuedDate").AsDateTime2().Nullable()
            .WithColumn("Msrp").AsDecimal(18, 2).Nullable();
    }

    public override void Down()
    {
        Delete.Table("Toys").InSchema("dbo");
    }
}
```



```
public static class MigrationRunnerExtensions
{
    public static IMigrationRunnerBuilder ScanMigrations(this IMigrationRunnerBuilder builder,
        string[] pluginAssemblies)
    {
        // Core migrations
        var allFixedAssemblies = new[] { typeof(ToyContext).Assembly };

        // The plugins
        var allPluginAssemblies = pluginAssemblies.Select(Assembly.LoadFrom);

        var allMigrationAssemblies = allFixedAssemblies
            .Union(allPluginAssemblies)
            .ToArray();

        builder.ScanIn(allMigrationAssemblies).For.Migrations();

        return builder;
    }
}
```

Needs to happen in ONE call

Project Dependencies


3-Data
Axxes.ToyCollector.DataAccess.dll
4-Web
Axxes.ToyCollector.Web.dll
1-Core
Axxes.ToyCollector.Core.Logic.dll
Axxes.ToyCollector.Core.Contracts.dll
Axxes.ToyCollector.Core.Models.dll
6-Migrations
Axxes.ToyCollector.Migrations.EFCore.dll
2-DI
Axxes.ToyCollector.DependencyResolution.dll
9-Tests
Axxes.ToyCollector.Tests.Core.dll
5-Plugins
Axxes.ToyCollector.Plugins.Lego.dll
Axxes.ToyCollector.Plugins.Marbles.dll

AXXES_

Conclusion

–

Let’s wrap up this session!

Key takeaways

- Before anything else, structure your code
- Plugins aren’t too hard to do, especially in .NET Core
- Never solve a code problem by introducing a deployment problem!
- Don’t worry if you’re not at step 4, 5 or 8 yet.

FAQ

- Isn’t this a lot harder?
- Can the plugins be Onions by themselves?
- When should I do this?

When should I use plugins?

- If you need modular deployments
(for instance: paid features per customer)
- If you want to easily retire/replace features
- If you want to be able to test features (A/B)
- To make smaller build pipelines in a large product

And if your business OK's the 10/100 rule …

Questions?

About me

Hannes Lowette

Head of Learning & Development at @Axxes_IT

@hannes_lowette

#20086521

Code samples and slides at :

https://github.com/Belenar/Axxes.ToyCollector

MVP Microsoft® Most Valuable Professional

AXXES_

Thank you!

Images from

https://www.habausa.com/
http://technicopedia.com/
https://bricks.stackexchange.com/
https://tenor.com/
https://www.reddit.com/
https://pixabay.com/
https://martinfowler.com/
https://wastelessfuture.com/
https://me.me/
https://buildplease.com/

https://noisebreak.com/
https://www.theonion.com/
https://www.psychologicalscience.org/
http://www.gillsonions.com/
https://www.toyotaofplano.com/
https://imgflip.com/
https://www.asp.net/
https://www.amazon.com/
https://xkcd.com/
https://unsplash.com/