Sezione 09

Servizi e dependency injection

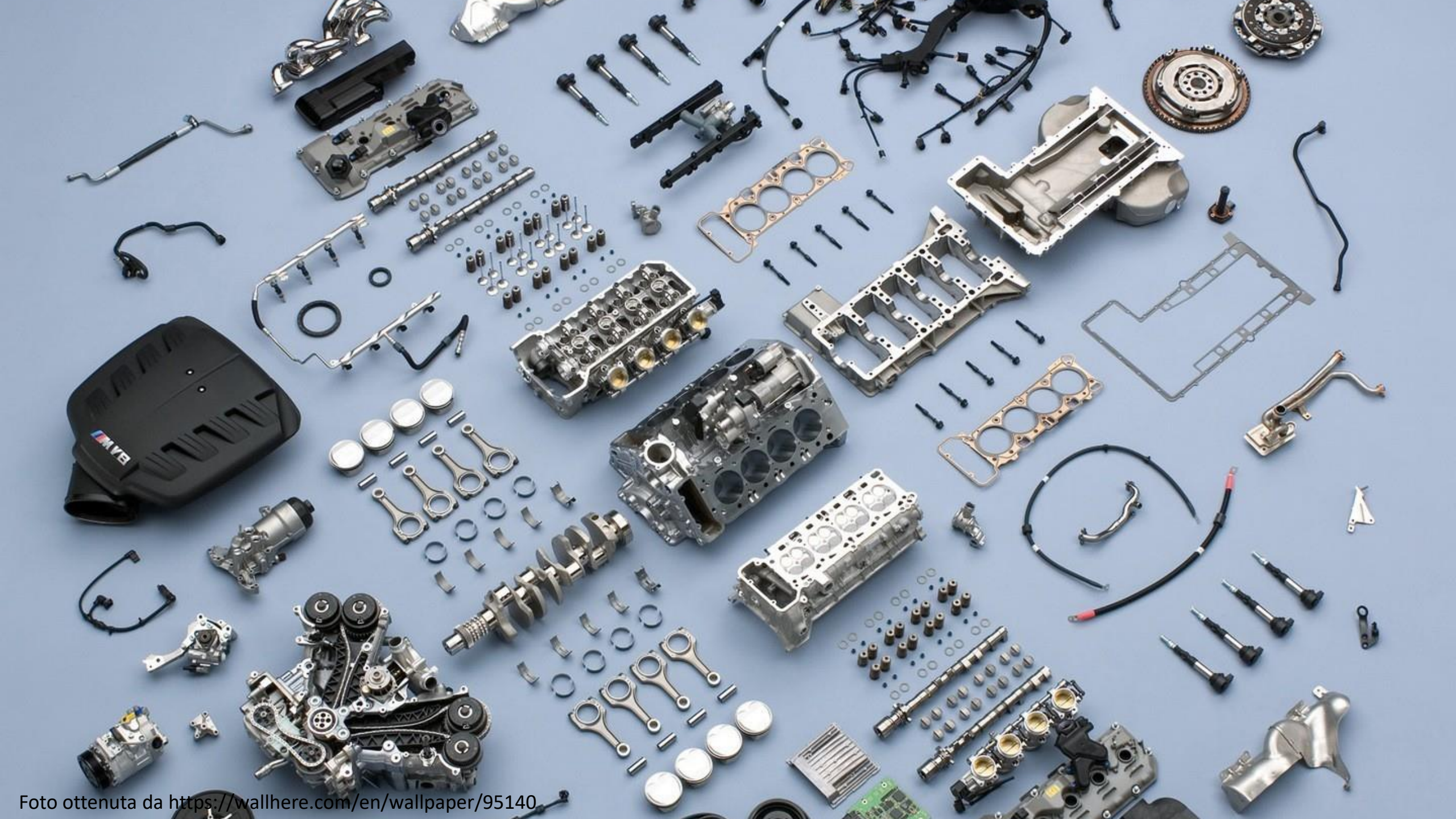


Foto ottenuta da https://wallhere.com/en/wallpaper/95140

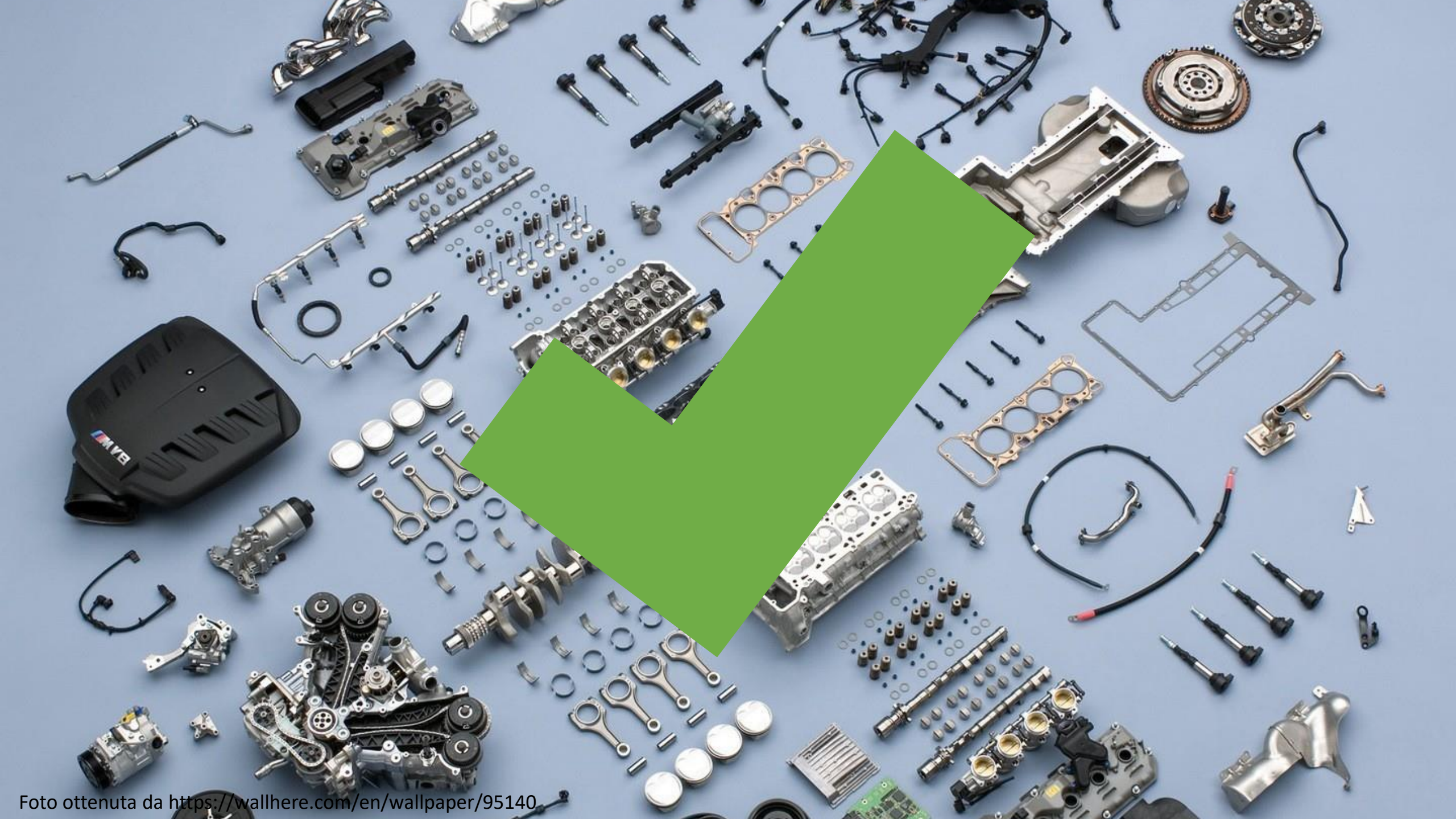


Foto ottenuta da https://wallhere.com/en/wallpaper/95140

Foto ottenuta da https://wallhere.com/en/wallpaper/95140

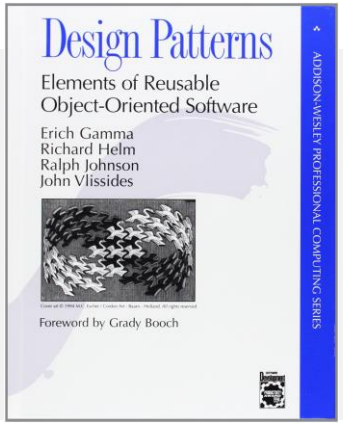
Dependency injection

- Uno dei modi corretti di esprimere la dipendenza di un componente da un altro componente è quello di definire un parametro nel costruttore: `public class CoursesController(CourseService courseService)`

- ASP.NET Core "risolve le dipendenze" cioè crea le istanze dei servizi e le passa al costruttore;
  - Affinché ciò avvenga, dobbiamo registrare i nostri servizi all'interno del metodo `ConfigureServices` della classe `Startup`.

Foto ottenuta da http://uniting4climate.net/tag/00-honda-civic-front-bumper/

Accoppiamento debole



"Program to an interface, not an implementation."

~~`public class CoursesController(CourseService courseService)`~~
```
public class CoursesController(ICourseService courseService)
```

- Migliora la testabilità dei componenti

Video di Patrick Mikolajczak da https://www.youtube.com/watch?v=xbvC20obbbI

Ciclo di vita dei servizi

Abbiamo 3 metodi per registrare servizi in `ConfigureServices`.
Ciascuno di essi influenza il ciclo di vita del servizio in modo diverso.

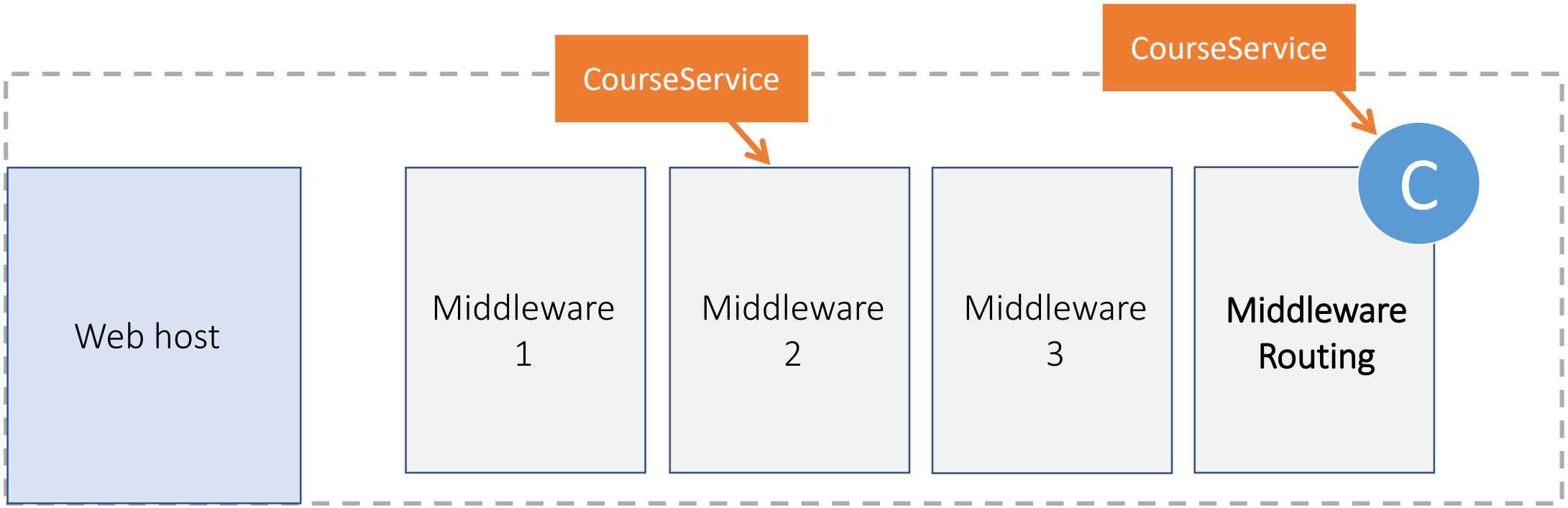
```
services.AddTransient<ICourseService, CourseService>
```

```
services.AddScoped<ICourseService, CourseService>
```

```
services.AddSingleton<ICourseService, CourseService>
```

AddTransient

- ASP.NET Core crea una nuova istanza del servizio ogni volta che un componente ne ha bisogno, e poi la distrugge dopo che è stata usata.

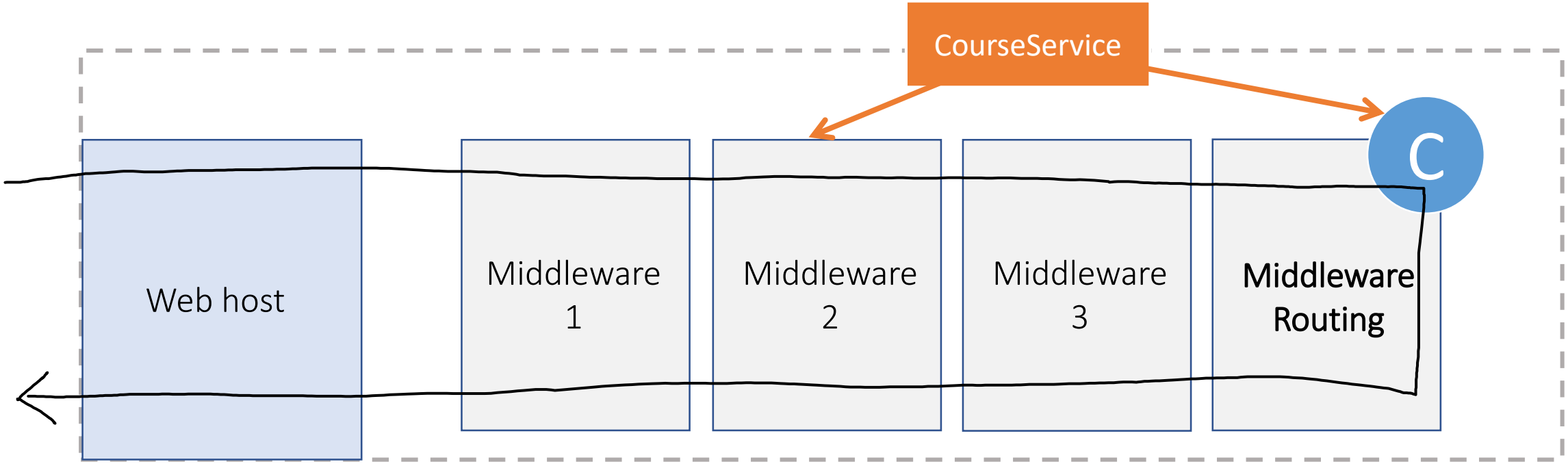


Usiamo AddTransient quando…

- I nostri servizi sono "veloci da costruire" e quindi non ci sono problemi prestazionali se ne vengono costruite più istanze.

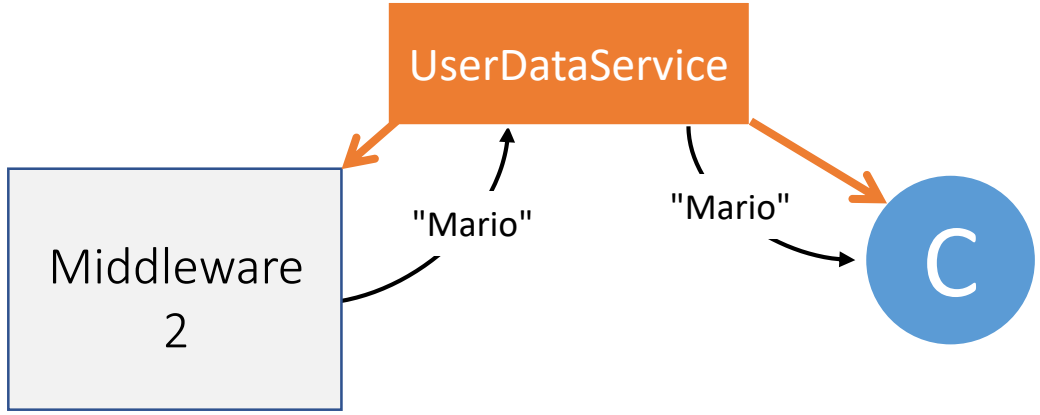
AddScoped

- ASP.NET Core crea una nuova istanza e la riutilizza finché siamo nel contesto della stessa richiesta HTTP. Al termine, la distrugge.



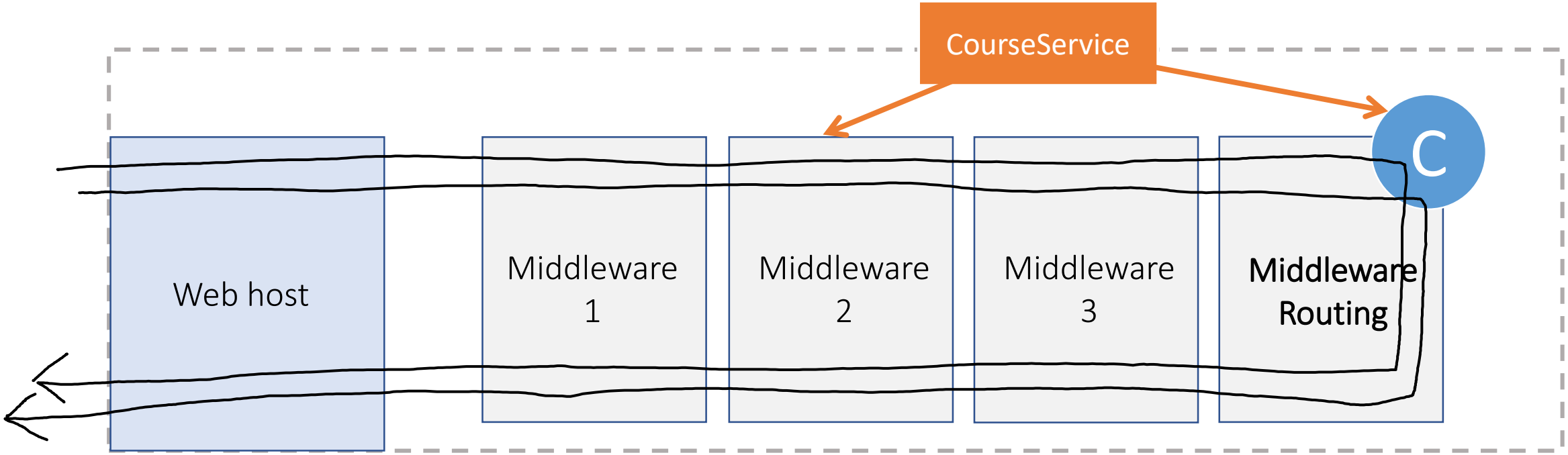
Usiamo AddScoped quando…

- Il servizio è "costoso da costruire" e perciò non vogliamo "pagare" più volte i tempi di costruzione;
  - Come il DbContext di EntityFramework Core che vedremo più avanti;
- Vogliamo usare il servizio per scambiare informazioni tra componenti.



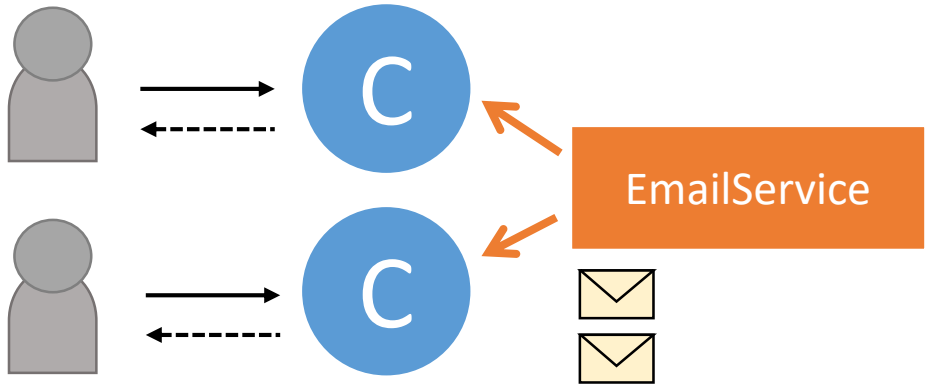
AddSingleton

- ASP.NET Core crea un'istanza e la inietta in tutti i componenti che ne hanno bisogno, anche in richieste HTTP diverse e concorrenti.



Usiamo AddSingleton quando…

- Abbiamo servizi che funzionano al di fuori della singola richiesta HTTP.
  - Un servizio che spedisce e-mail, ma le deve spedire una alla volta;



Questo EmailService tiene le email in una propria coda e così poi le può spedire una alla volta coi suoi tempi e modi

Usiamo AddSingleton quando…

- Abbiamo servizi che funzionano al di fuori della singola richiesta HTTP.
  - Un servizio che spedisce e-mail, ma le deve spedire una alla volta;
  - Un servizio che conteggia il numero di richieste HTTP.

- **ATTENZIONE!** Un servizio singleton, se ha uno stato interno, deve essere **thread-safe** perché verrà usato da più thread contemporaneamente!

Per l'approfondimento: evitare il problema delle race condition.
https://docs.microsoft.com/it-it/dotnet/standard/threading/managed-threading-best-practices#race-conditions

```
//Questa classe NON è thread-safe. NON usare con AddSingleton.
public class RequestCounterService : IRequestCounterService
{
    private int count = 0;

    public void IncrementCount()
    {
        count += 1; //Qui c'è un problema di race condition
    }
}
```

```
//Ora la classe è thread-safe e può essere registrata con AddSingleton
public class RequestCounterService : IRequestCounterService
{
    private int count = 0;

    public void IncrementCount()
    {
        Interlocked.Increment(ref count); //Ora l'accesso a count è coordinato
    }
}
```