Implementing Dependency Injection

Maintainable, Loosely Coupled Actors

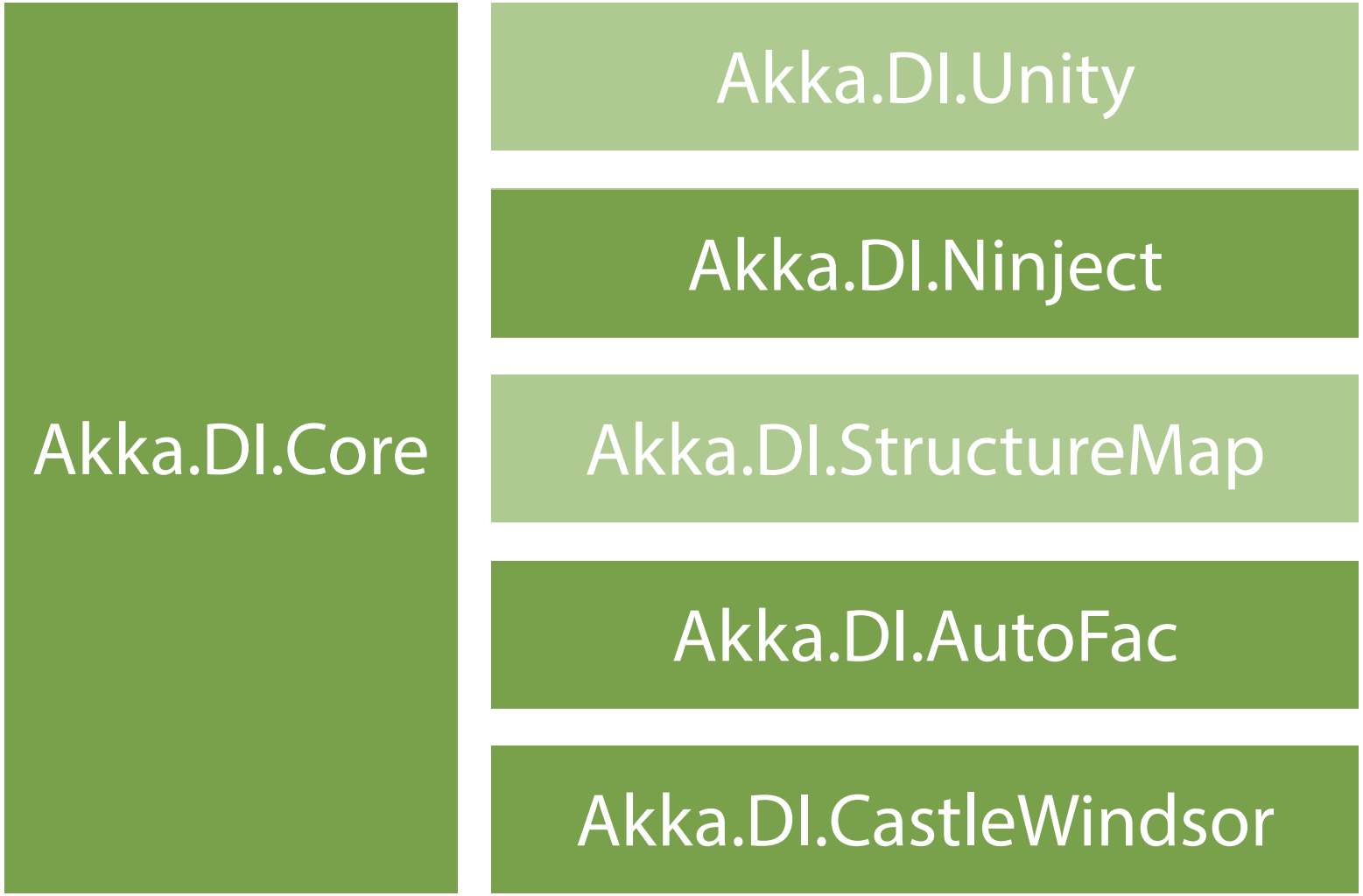
Jason Roberts

@robertsjason | dontcodetired.com

Overview

- DI Container Choices
- Overview of using DI in Akka.NET
- Refactor an actor to take a constructor dependency
- AutoFac, Castle Windsor, Ninject
- Lifecycle scopes and shared mutable state

DI Container Choices



Overview of DI in Akka.NET

- Install Akka.NET DI NuGet package
- Add dependencies to actors
  - public TrendingMoviesActor(ITrendingMovieAnalyzer analyzer)
- Configure DI container
  - Create instance of container
  - Set up any mappings/resolutions for types
- Create actor system instance
- Plug container into the actor system instance
- Context.DI().Props
- actorSystem.ActorOf(actorSystem.DI().Props<PlaybackActor>()

Refactoring Actors to Use DI

~~new SimpleTrendingMovieAnalyzer();~~

Add an ITrendingMovieAnalyzer constructor parameter

Run system and notice failure

Implementing DI with AutoFac

Install Akka.DI.AutoFac NuGet Package

Configure and build AutoFac container

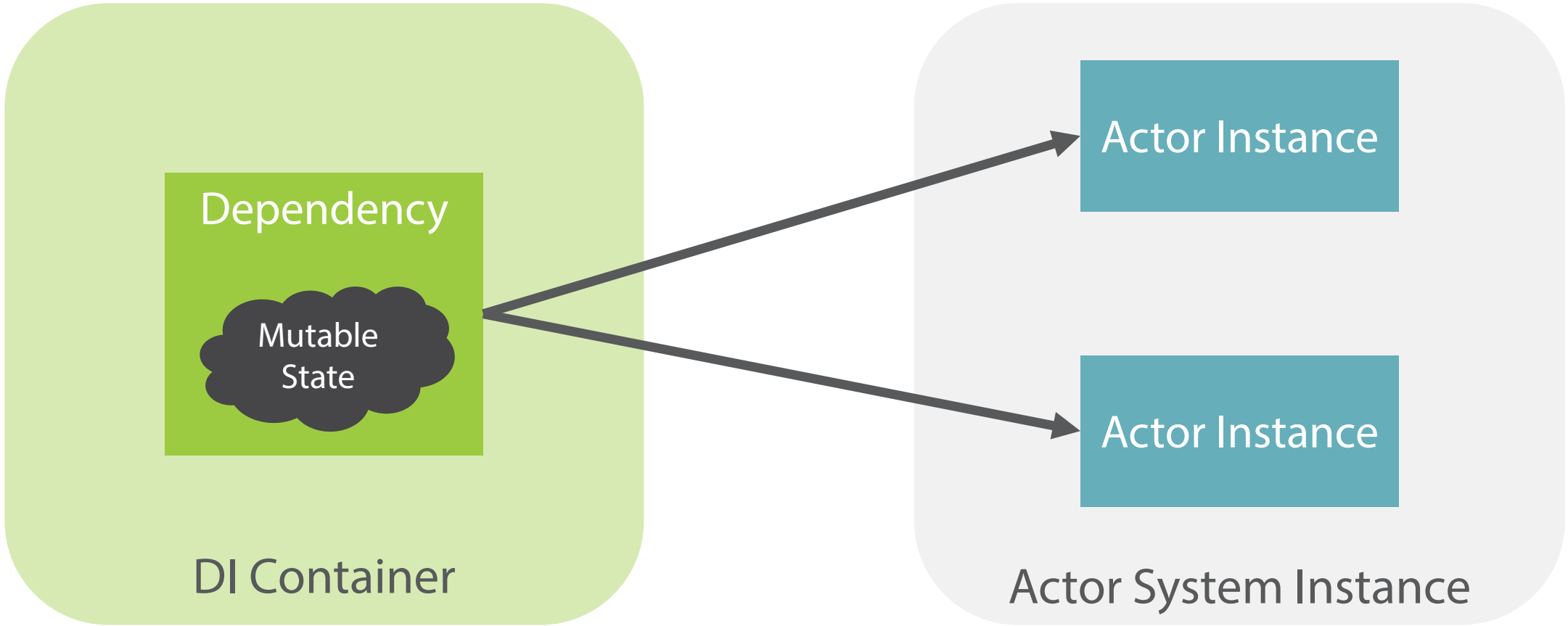
AutoFacDependencyResolver

Context.DI().Props

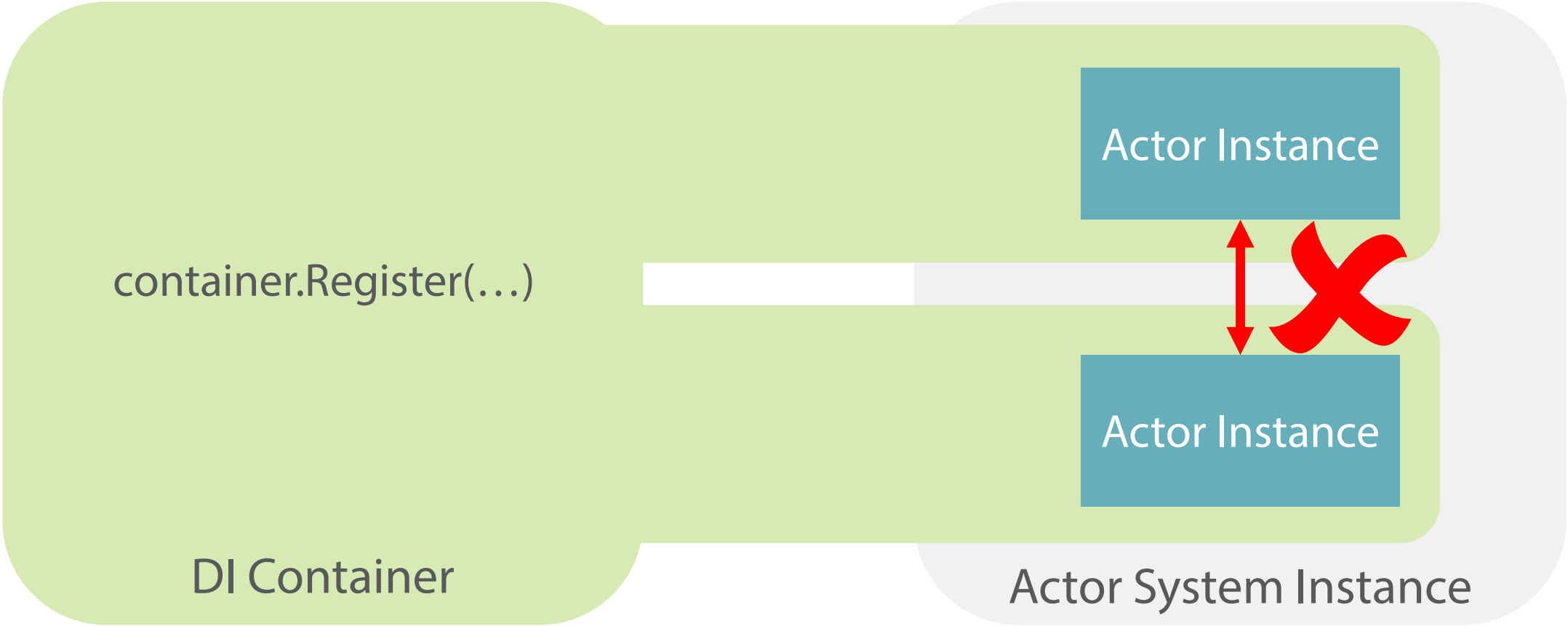
Implementing DI with CastleWindsor

Implementing DI with Ninject

Lifecycle Scopes and Shared Mutable State



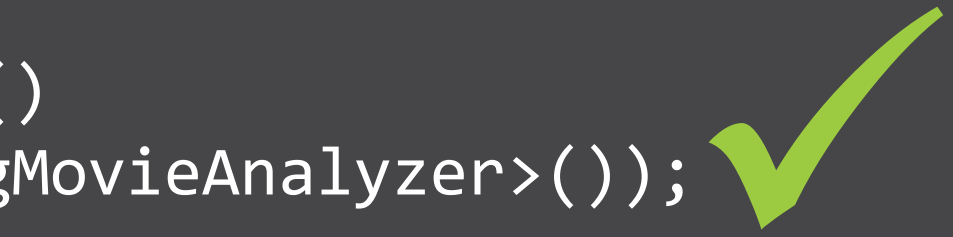
Lifecycle Scopes and Shared Mutable State



When using dependency injection in Akka.NET we need to ensure we don't get shared instances resolved for actors or for dependencies that have shared mutable state.

Lifecycle Scopes and Shared Mutable State

```
container.Register(Component
    .For<ITrendingMovieAnalyzer>()
    .ImplementedBy<SimpleTrendingMovieAnalyzer>());
```



```
container.Register(Component
    .For<TrendingMoviesActor>());
```

```
container.Register(Component
    .For<TrendingMoviesActor>()
    .LifestyleTransient());
```

Summary

DI container choices

Overview of using DI in Akka.NET

ITrendingMovieAnalyzer analyzer

AutoFac, Castle Windsor, Ninject

Lifecycle scopes and shared mutable state

LifestyleTransient()