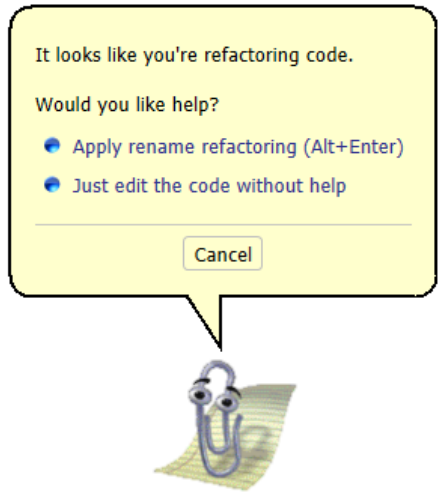
Deconstructing Clippy for ReSharper

Common patterns for an uncommon extension



Just an April Fools' joke

Linus Rörstad
@TheRealPipecity

Follow

After using Clippy for @resharper for two weeks I must say it's not quite as annoying as I remember the old thing but now it must go

Reply Retweet Favorite More

9:15 PM - 15 Apr 2014

Not your usual extension

Doesn’t use standard extension points

Doesn’t look at source code, or syntax tree

Doesn’t provide standard UI commands

But still an extension

Same structure as traditional plugin

Same concerns as traditional plugin

- Components
- Lifetime management
- Threading
- Data flow
- Versioning
- Settings

Lifetime

```
public class Lifetime
{
    Lifetime AddDispose(IDisposable item);
    Lifetime AddAction(Action F);
    Lifetime AddBracket(Action FOpening, Action FClosing);
    Lifetime AddRef(object @object);
    bool IsTerminated { get; }
}
```

Plus lots of extension methods, e.g. on `ICollection`:

```
public static void Add<TItem>(this ICollection<TItem> this,
    Lifetime lifetime, TItem item);
public static void AddRange<TItem>(this ICollection<TItem> this,
    Lifetime lifetime, IEnumerable<TItem> items);
```

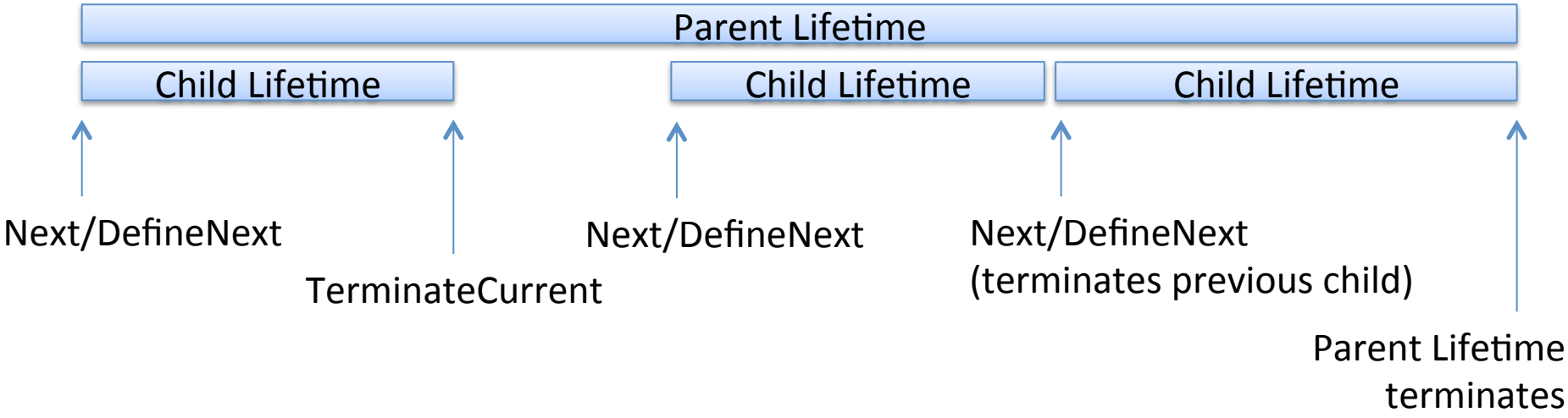
Creating and Terminating Lifetime

Usually passed into constructor by Component Model

```
public static class Lifetimes
{
    public static void Using(Action<Lifetime> F);
    public static TRetVal Using<TRetVal>(Func<Lifetime, TRetVal> F);

    public static LifetimeDefinition Define(Lifetime lifetime, string id = null,
        Action<LifetimeDefinition, Lifetime> FAtomic = null, ILogger logger = null);

    public static LifetimeDefinition CreateIntersection2(params Lifetime[] lifetimes);
    public static LifetimeDefinition CreateIntersection2(ILogger logger,
        params Lifetime[] lifetimes);

    public static void Synchronize(params LifetimeDefinition[] definitions);
}

public sealed class LifetimeDefinition
{
    public void Terminate();
    public bool IsTerminated { get; }
    public Lifetime Lifetime { get; }
}
```

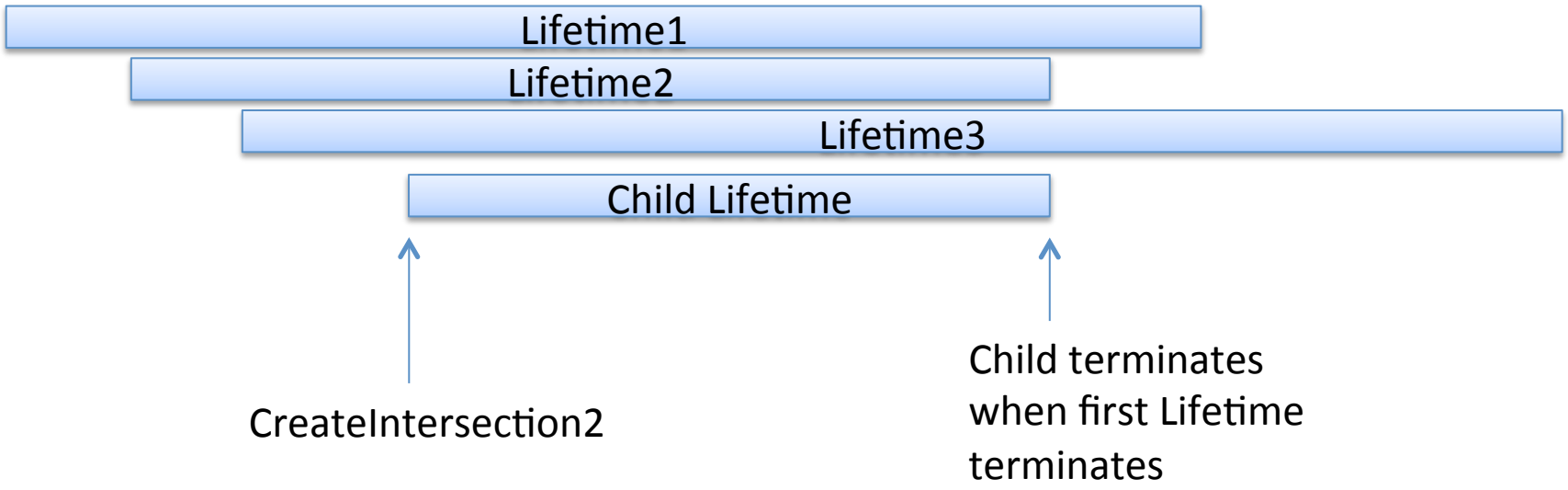
Sequential Lifetime

```
public class SequentialLifetimes
{
    public SequentialLifetimes(Lifetime parentLifetime);
    public void Next(Action<Lifetime> FNext);
    public void DefineNext(Action<LifetimeDefinition, Lifetime> FNext);
    public void TerminateCurrent();
}
```



Intersecting Lifetimes

```
public static class Lifetimes
{
    public static LifetimeDefinition CreateIntersection2(params Lifetime[] lifetimes);
    public static LifetimeDefinition CreateIntersection2(ILogger logger,
        params Lifetime[] lifetimes);
}
```



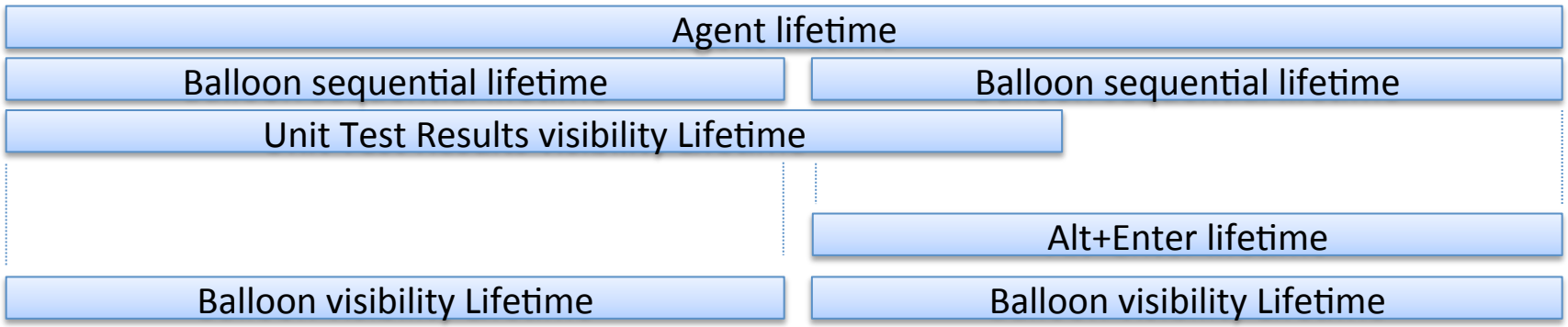
Balloon Intersecting Lifetimes

Balloon visibility is controlled by a Lifetime, created when balloon is due to be shown

Lifetime is an intersection of:

- Client's lifetime (e.g. build animation, terminated when build completes)
- Balloon's sequential lifetime (terminated when new balloon is required, or character is hidden)

Balloon's sequential lifetime also terminates when the parent (agent component) lifetime terminates



Dataflow

```
public interface ISignal<TValue> : IDisposable
{
    // [snip]
    void Advise(Lifetime lifetime, Action<TValue> handler);
    void Fire(TValue value);
    void Fire(TValue value, object cookie);
}


public interface IProperty<TValue> : IHaveUntypedProperty, IDisposable
{
    ISignal<BeforePropertyChangedEventArgs<TValue>> BeforeChange { get; }
    ISignal<PropertyChangedEventArgs<TValue>> Change { get; }
    PropertyId<TValue> Id { get; }
    TValue Value { get; set; }
    bool IsNullValueAllowed { get; }

    TValue GetValue();
    bool SetValue(TValue value);
    // [snip]
}
```

IThreading

```
public interface IThreading
{
    // Run on UI thread
    JetDispatcher Dispatcher { get; }

    // Run on UI thread, with re-entrancy protection
    ReentrancyGuard ReentrancyGuard { get; }

    // Throttle events
    GroupingEventHosts GroupingEvents { get; }

    // Run an action in a given timespan
    TimedActionsHost TimedActions { get; }
}
```