Case Study: Fluent Testing of INotifyPropertyChanged

Floyd May
SOFTWARE CRAFTSMAN

@softwarefloyd https://medium.com/@floyd.may

What to Expect

Coding

Coding

More coding!

Pauses for exposition

- Patterns
- Concepts
- Context graphs

Simplest Case of Fluent Testing API

T : INotifyPropertyChanged

PropertyChangeExpectation<T>

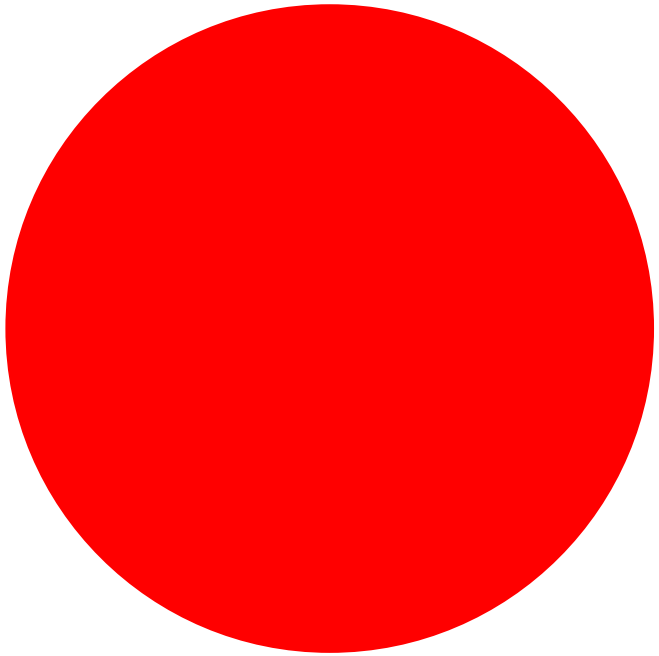
void

Void Context

End a sentence

Exit the API

Use extension methods to “break in” to an existing API



Failing tests are important

- Proof that your code truly solves the problem at hand

Strict TDD? Maybe...

- Red, green, refactor
- Psychology of small successes

```
var person = new Person();

Assert.Throws<AssertionException>(() =>

    person.ShouldNotifyFor(x => x.FirstName)
        .When(() => { /* do nothing */ })

);
```

Tests Should Reveal Intent

Simple and clean

Short distance from:

- Real-world code
- Feature request

```
var person = new Person();

person.ShouldNotifyFor(x => x.FirstName)
      .When(() => { /* do nothing */ });
      // this should throw an assert failure
```

This Could Be a Bug Report or a Code Sample

Remember the relationship between

- User feedback
- Automated tests
- Documentation

Throwing Exceptions for Invalid Cases

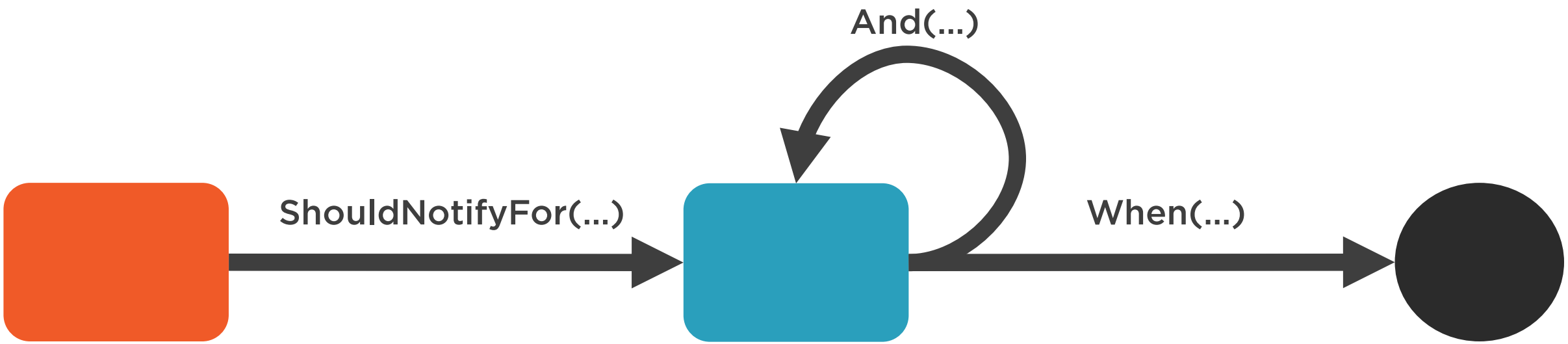
Avoid if possible

- Use the type system

Make your API easy to use, hard to misuse

- Throw early
- Suggest solutions

Adding a Simple Fluent Call



T : INotifyPropertyChanged

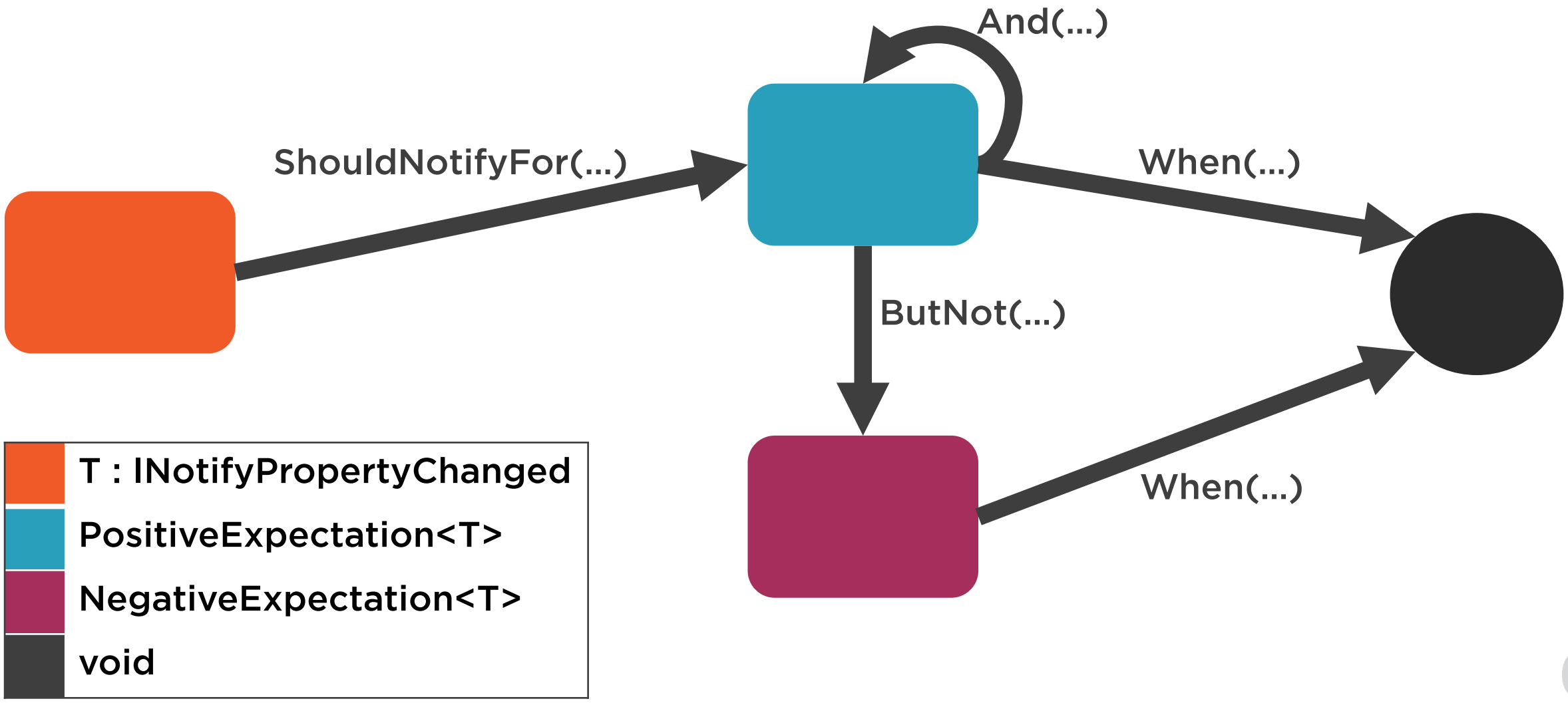
PropertyChangeExpectation<T>

void

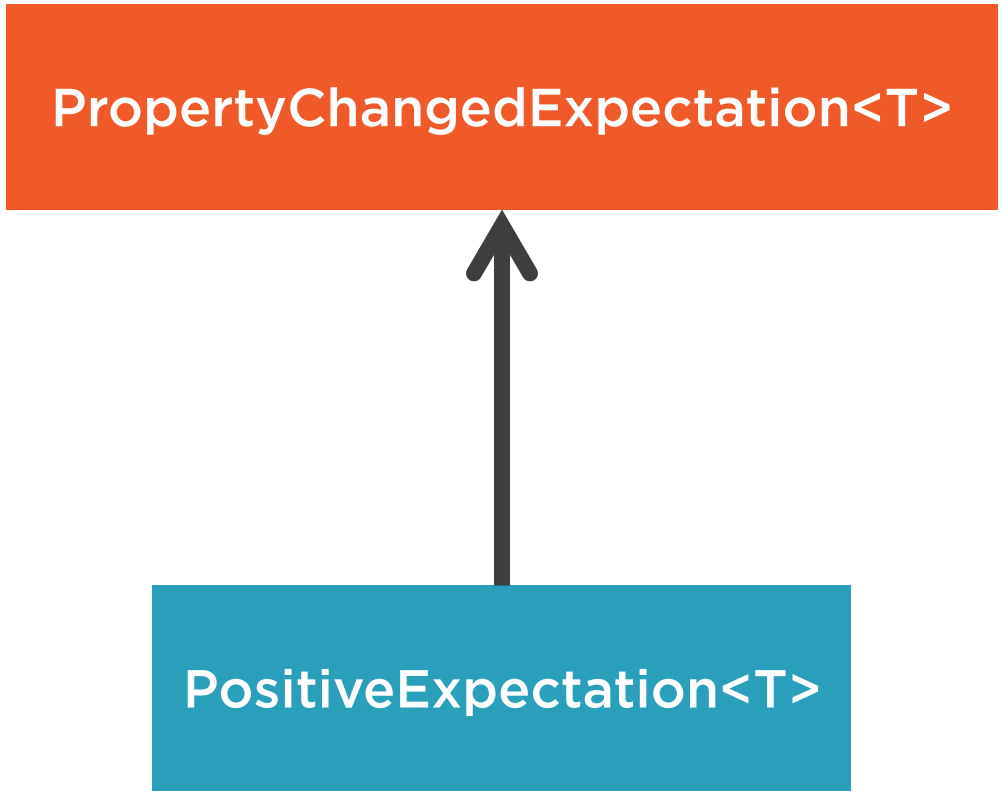
We Don’t Want to Allow This

```
person.ShouldNotifyFor(x => x.FirstName)
    .ButNot(x => x.LastName)
    .And(x => x.FullName) // positive or negative?
    .When(x => x.FirstName = "John“);
```

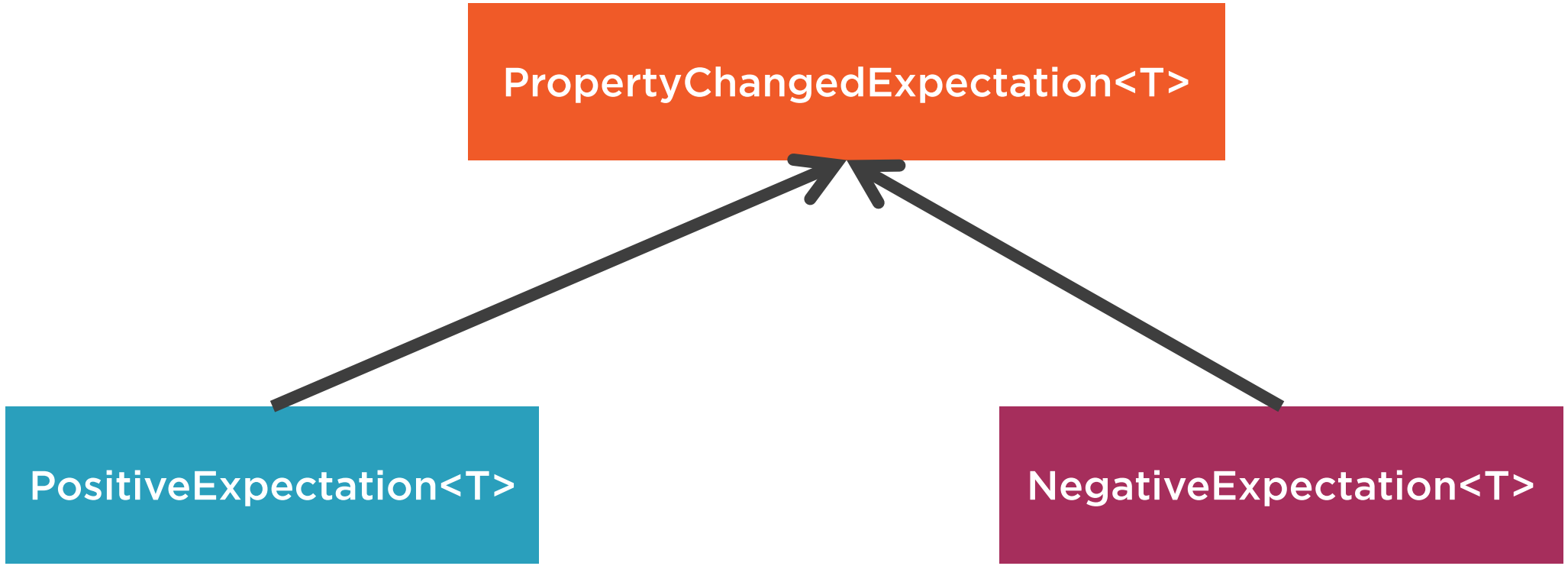
Negative Testing

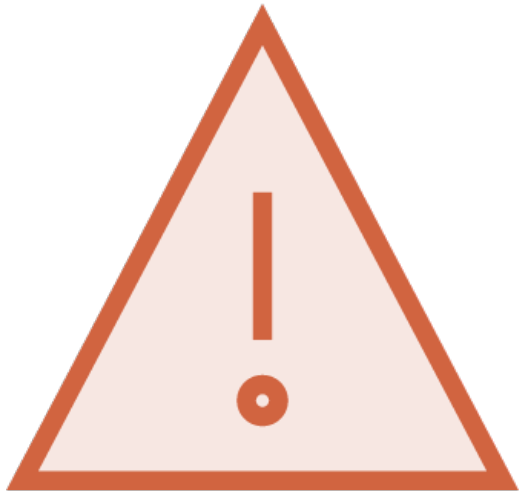


Refactoring: Step 1



Refactoring: Step 2





Exception messages are critical

- Test assertions even more so

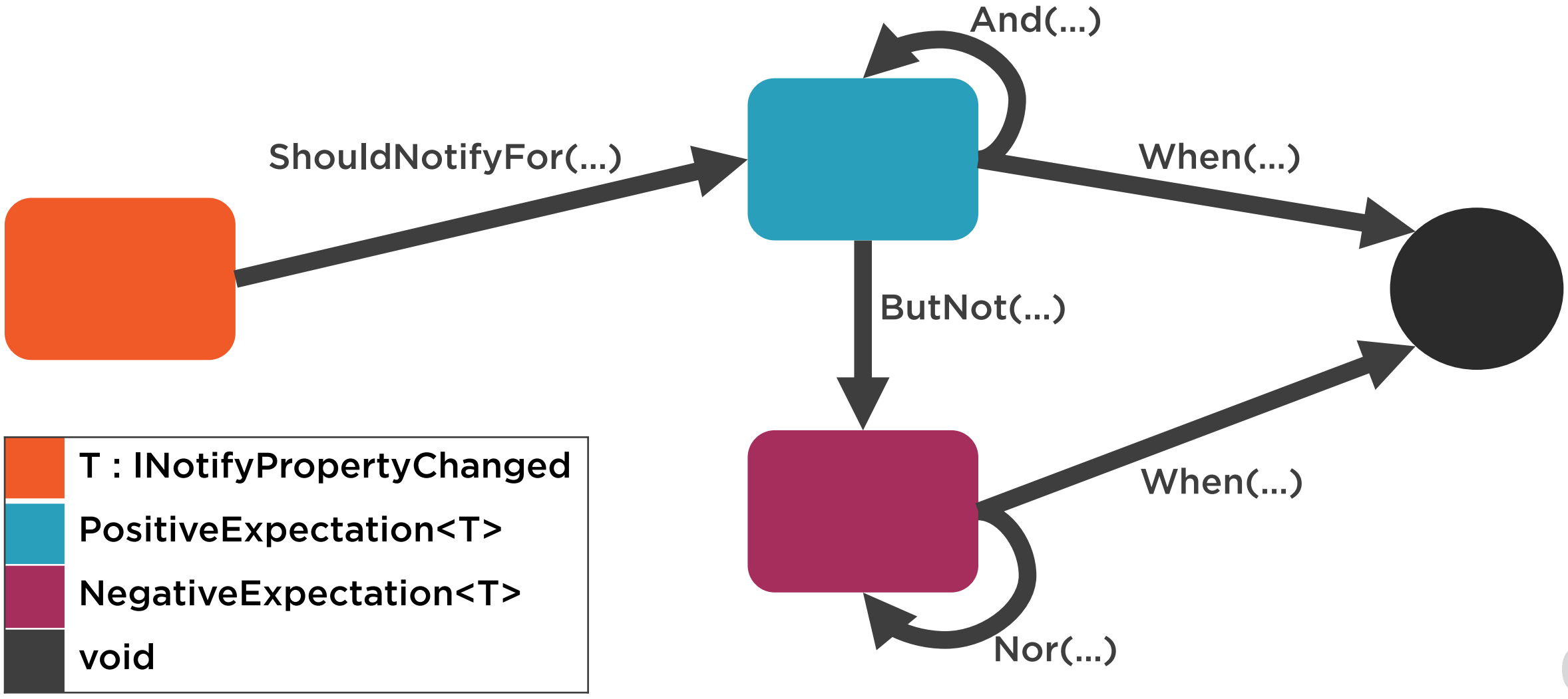
Exception messages should contain:

- What went wrong
- What was expected
- Suggestions for correction, if possible

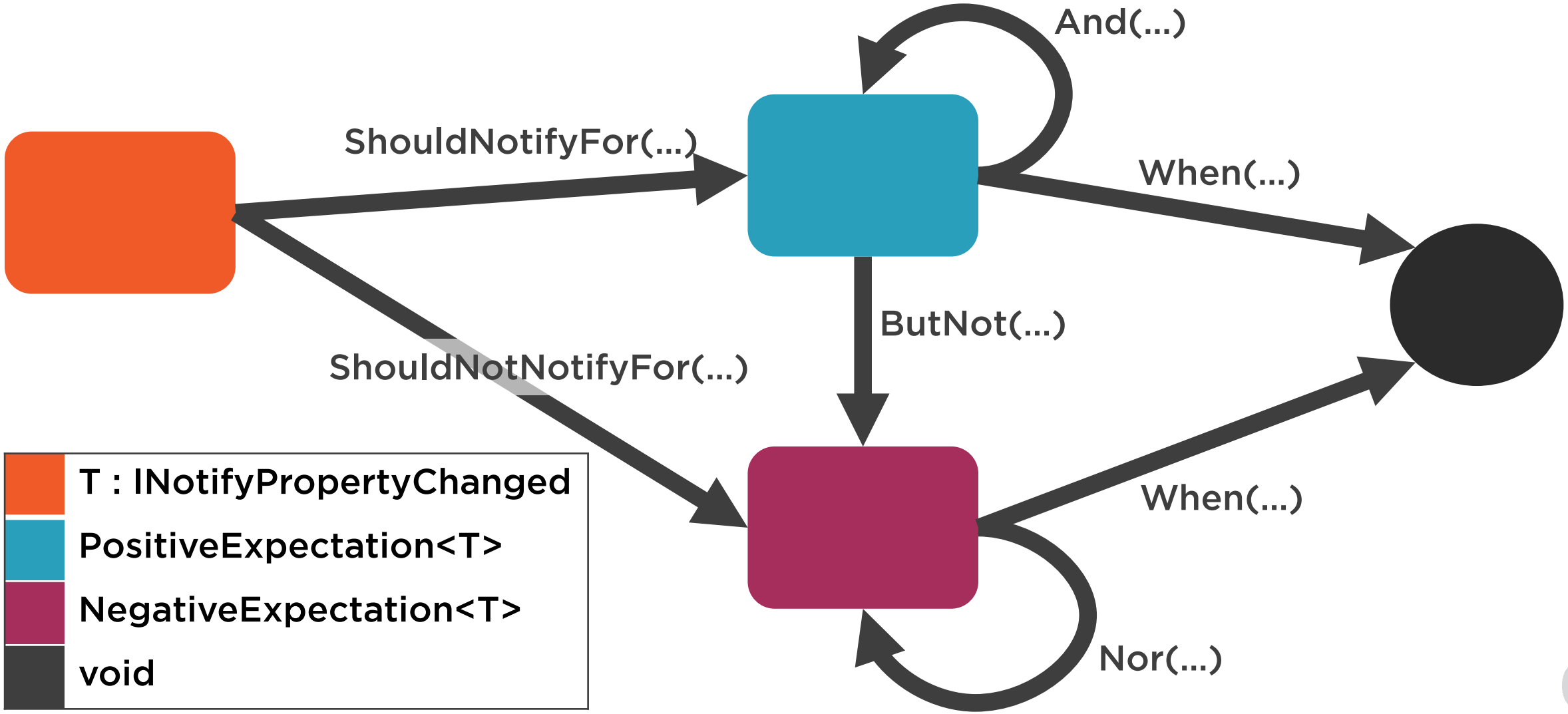
Consistency is important

- Similar messages across the API

Another Simple Fluent Call



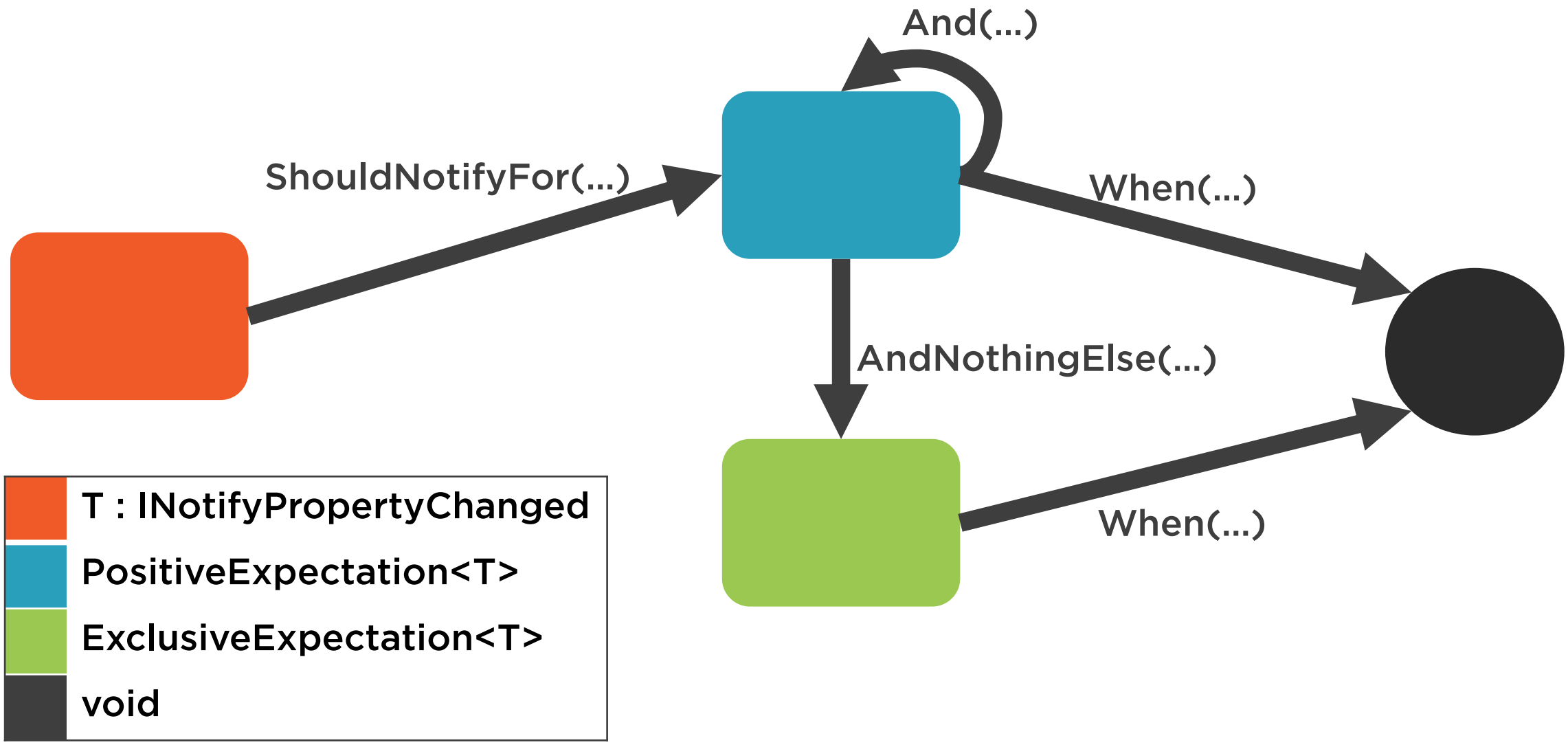
Negative Tests Only



Constrained Context

A context that exposes a subset of a previous context's public methods.

Constrained Context



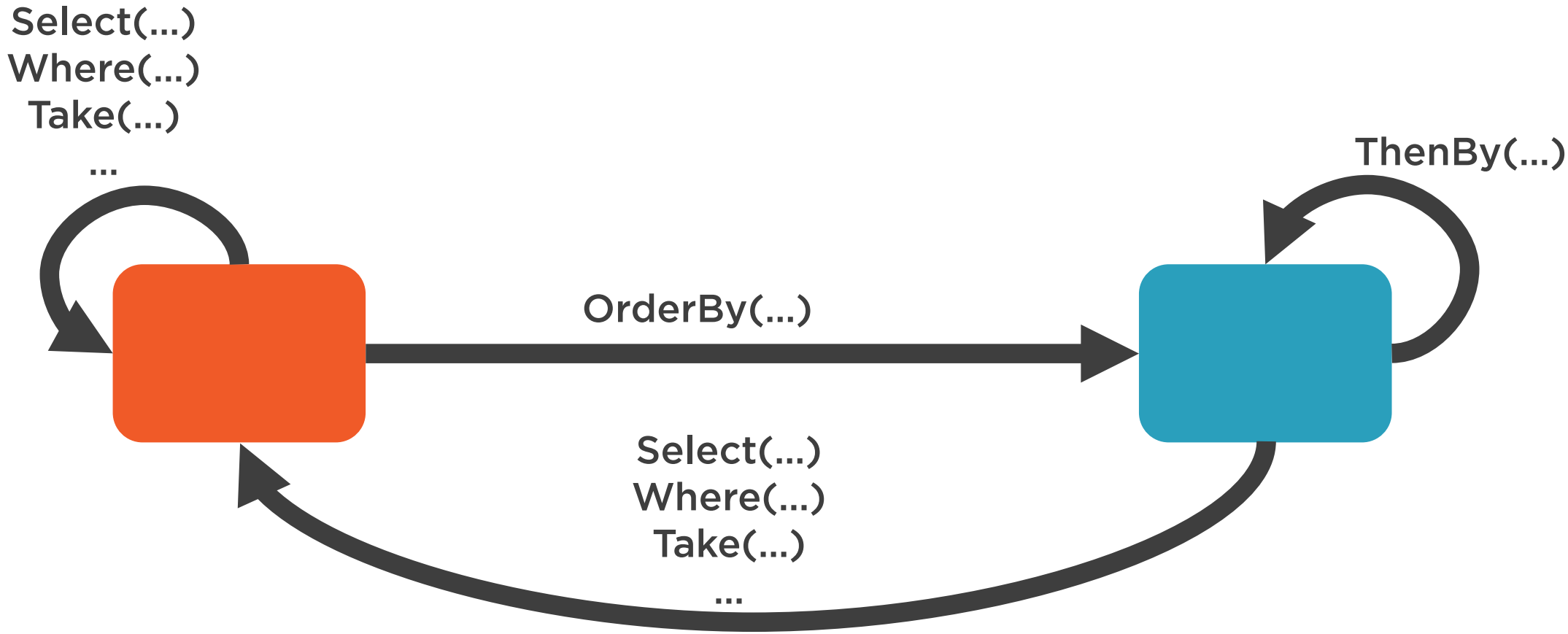
```
public static IOrderedEnumerable<T>
        OrderBy<T, TKey>(
            this IEnumerable<T> source,
            Func<T, TKey> keySelector
        ) { /* implementation */ }
```

OrderBy(...) Isn't a Simple Fluent Call

Augmented Context

A context that exposes a superset of a previous context's public methods.

LINQ's OrderBy(...) Method



IEnumerable<T>

IOrderedEnumerable<T>

Different Patterns for Exception Messages

Positive & Negative Expectation

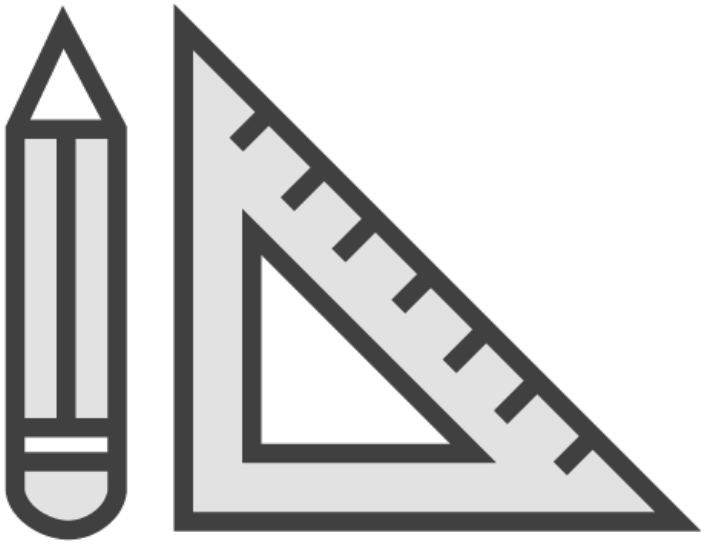
"Received notifications: A, B.
Expected A, B, C, but not D, E."

Exclusive Expectation

"Received notifications: A, B, C.
Expected A, B and nothing else."

Recap

Concepts in action



Void context

- End a “sentence”

Constrained context

- Fewer public methods

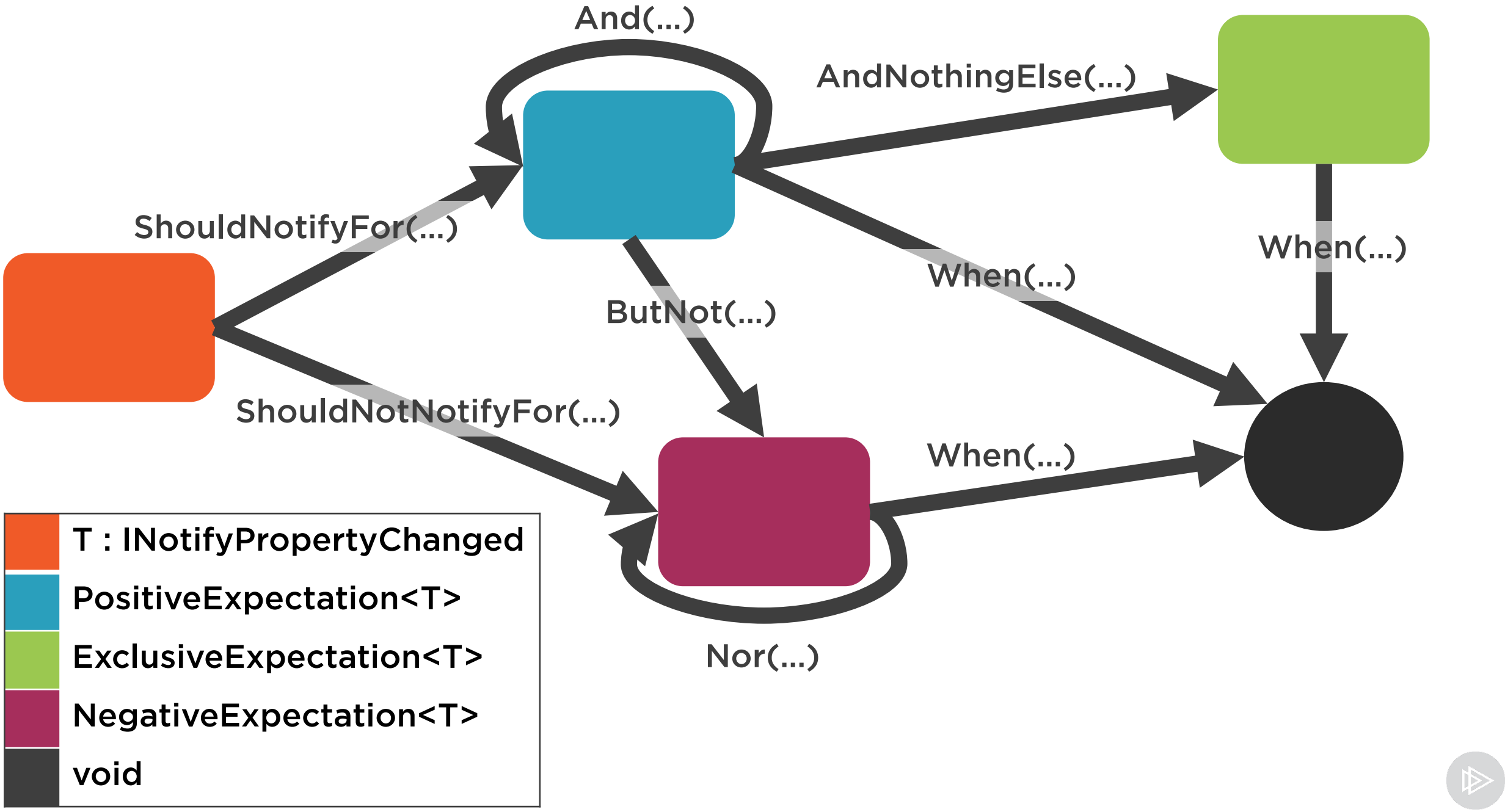
Augmented context

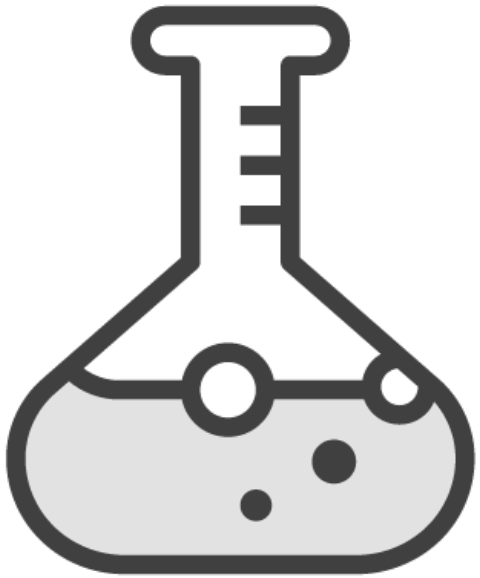
- Additional public methods

Consider superclass / subclass relationship

- Not always helpful

Context graphs


And(...)
AndNothingElse(...)
ShouldNotifyFor(...)
When(...)
When(...)
ButNot(...)
ShouldNotNotifyFor(...)
When(...)
Nor(...)
T : INotifyPropertyChanged
PositiveExpectation<T>
ExclusiveExpectation<T>
NegativeExpectation<T>
void



Watch them fail

Keep them readable

If you wouldn't show a test to a user, it might not be a good test.

Exception messages are important

Test failure messages matter, too

Thanks!