

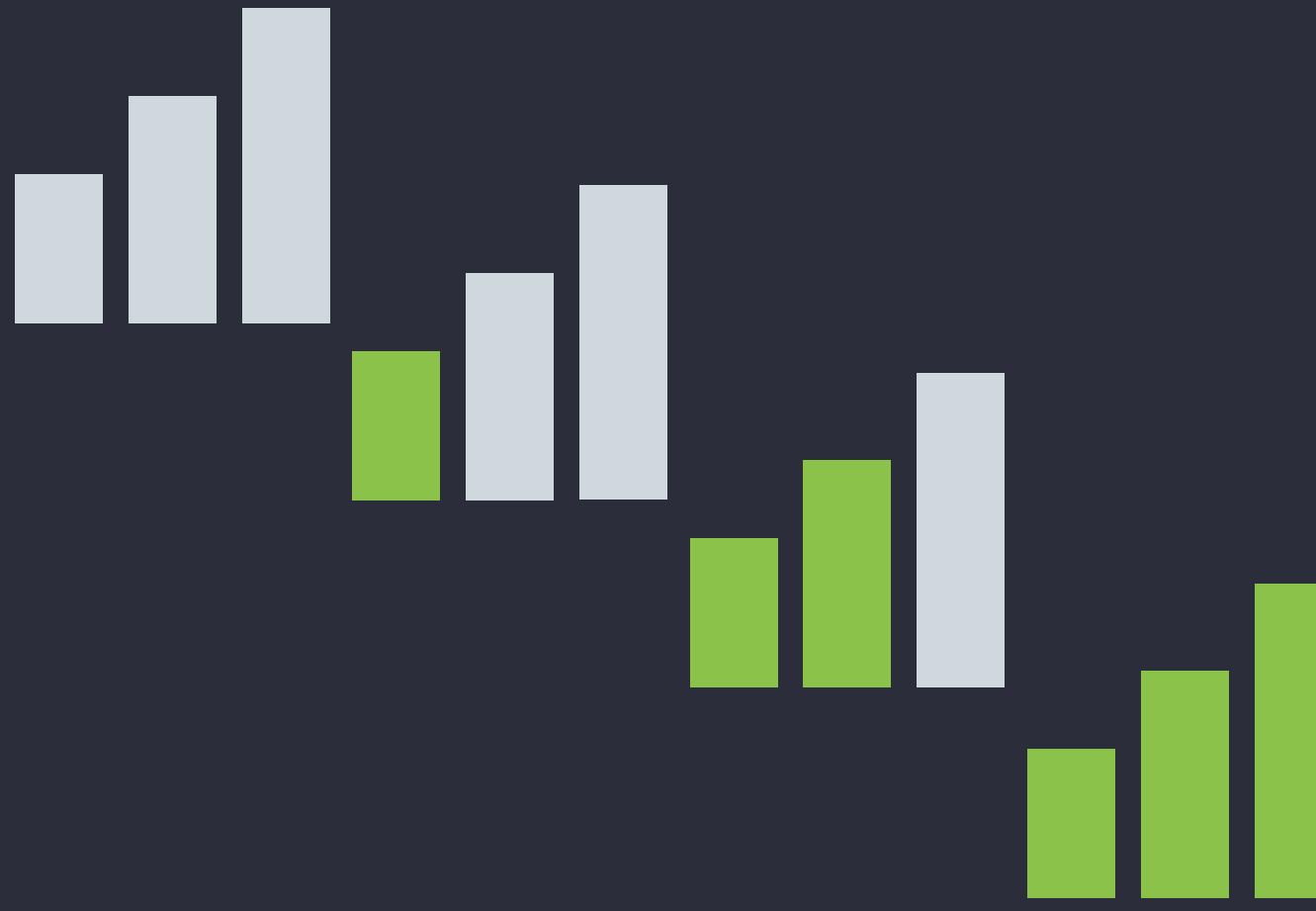


# Why you should be building better Mobile Apps with Reactive Programming

Michael Stonis  
Eight-Bot  
@MichaelStonis



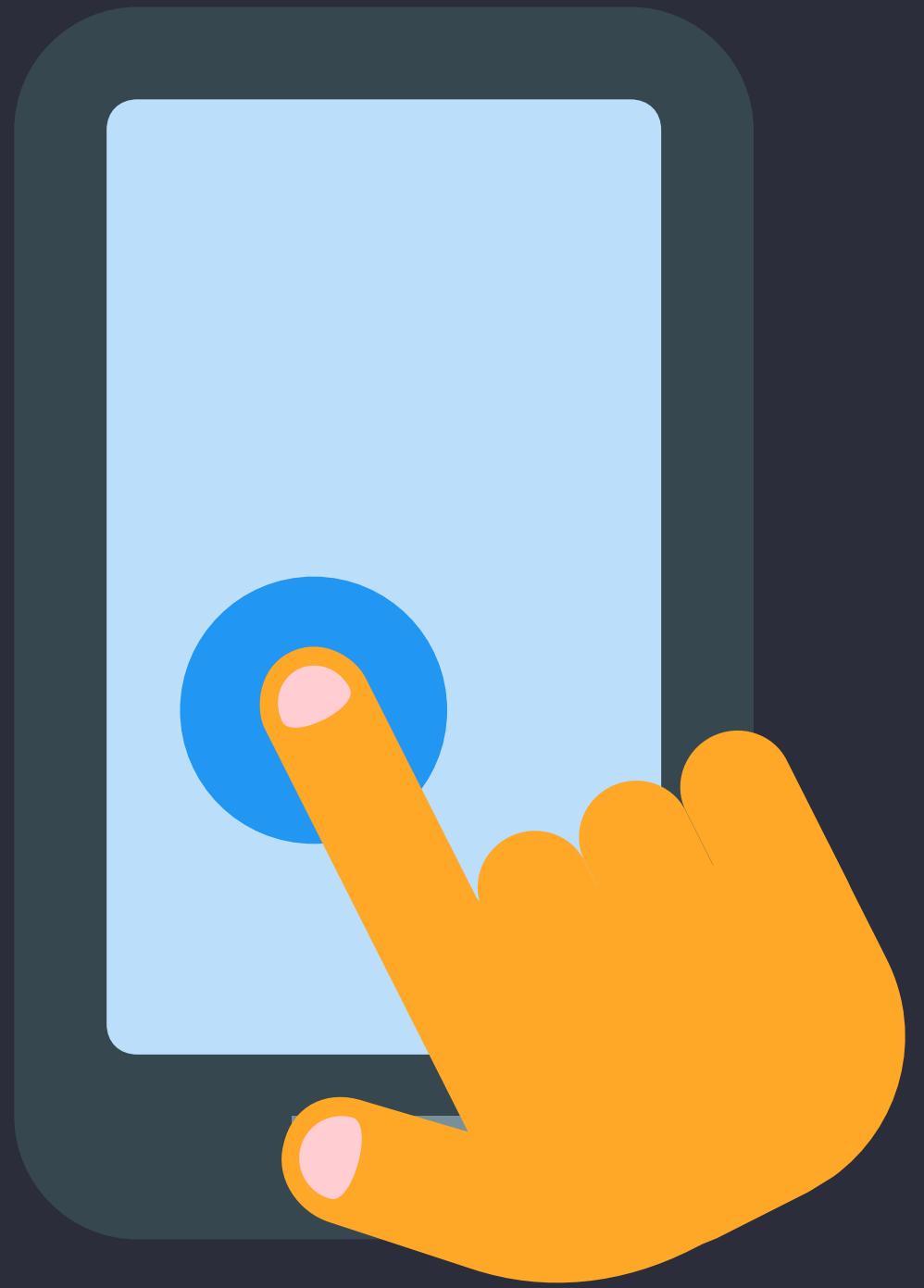
# Developing Mobile Apps is Hard



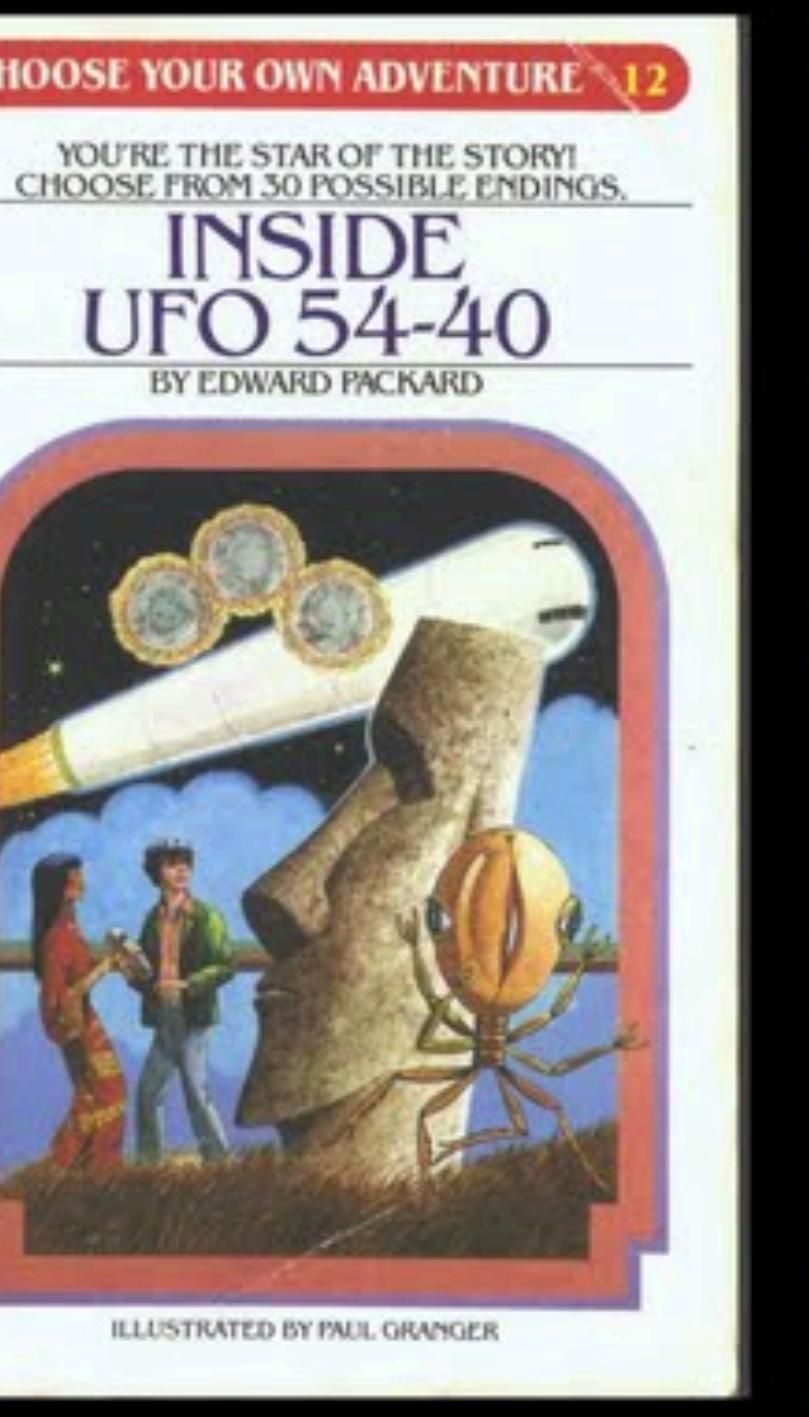
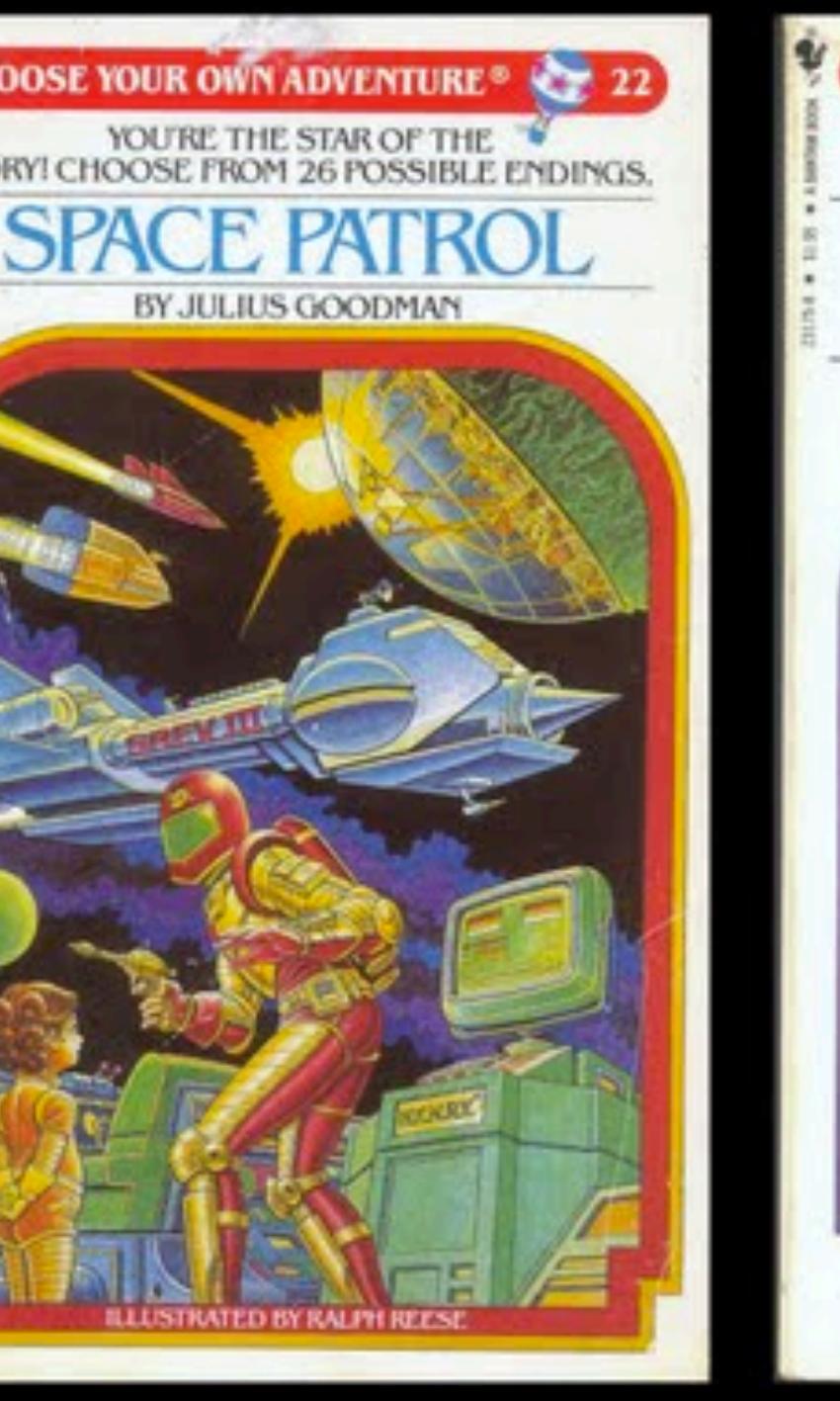
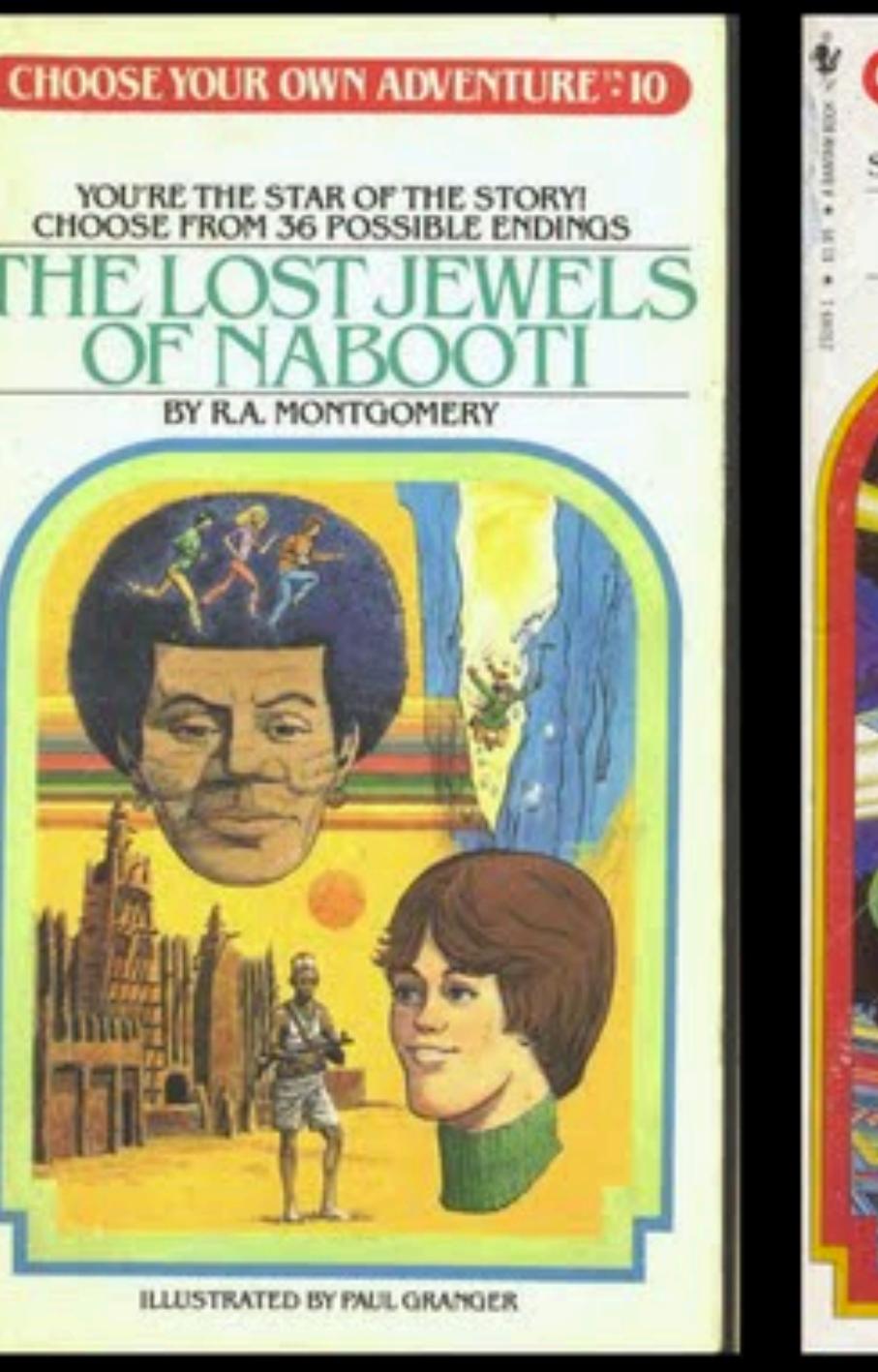
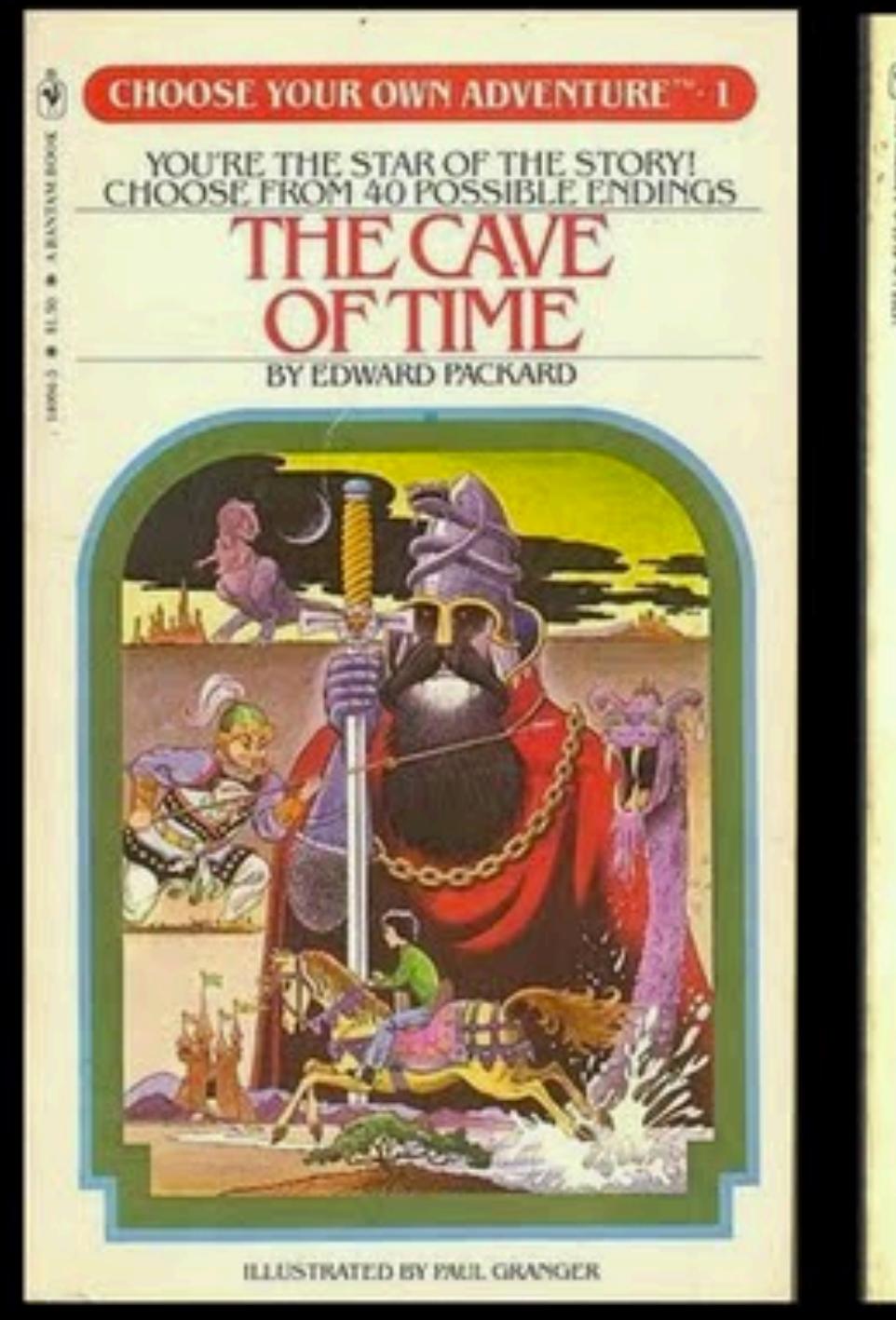
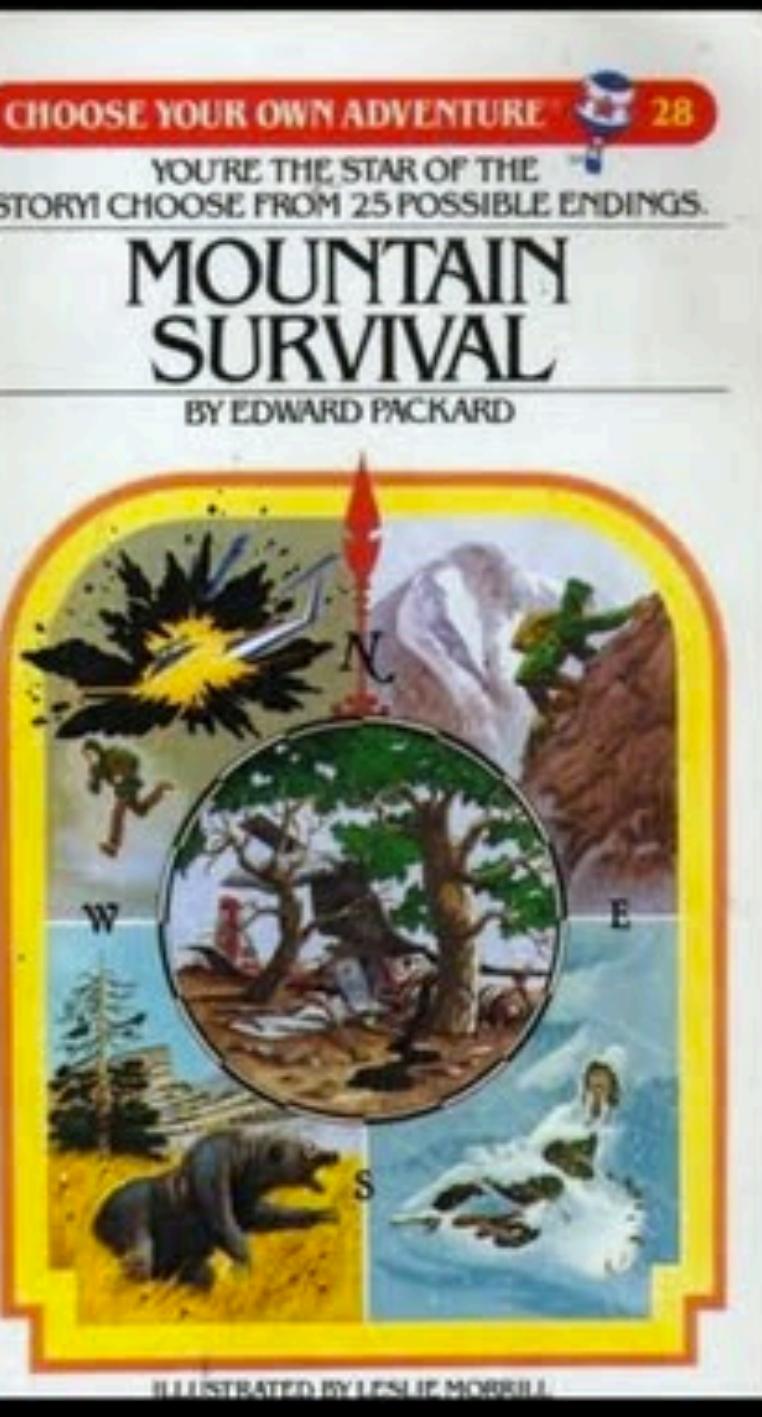
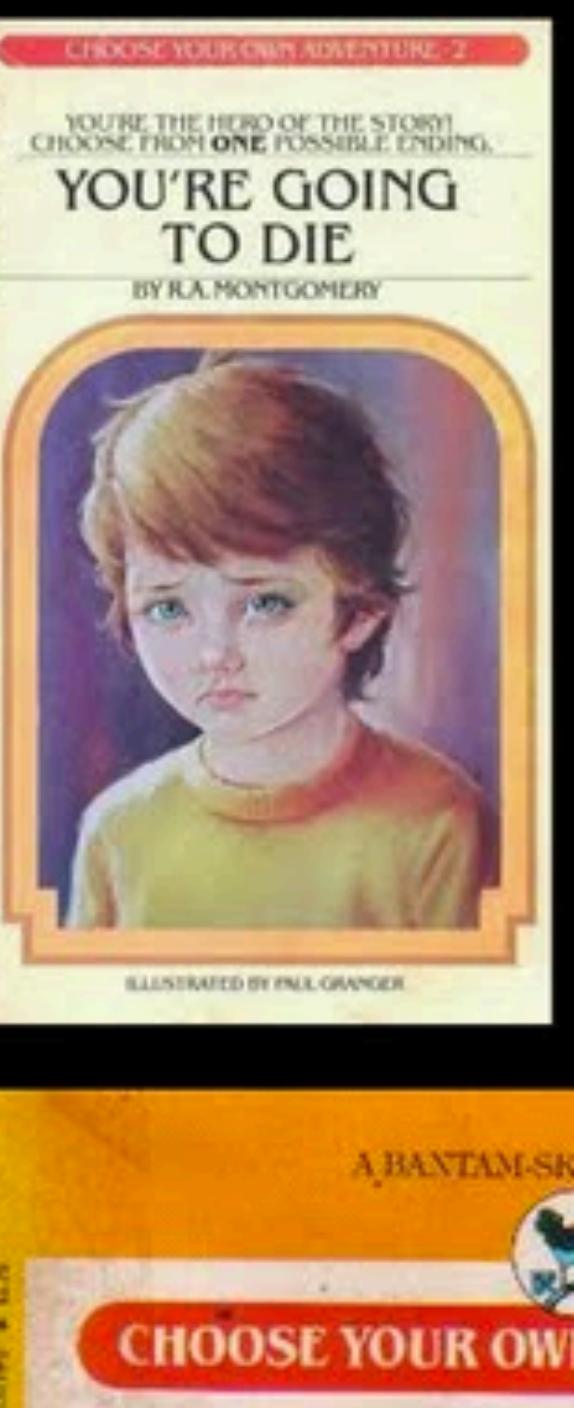
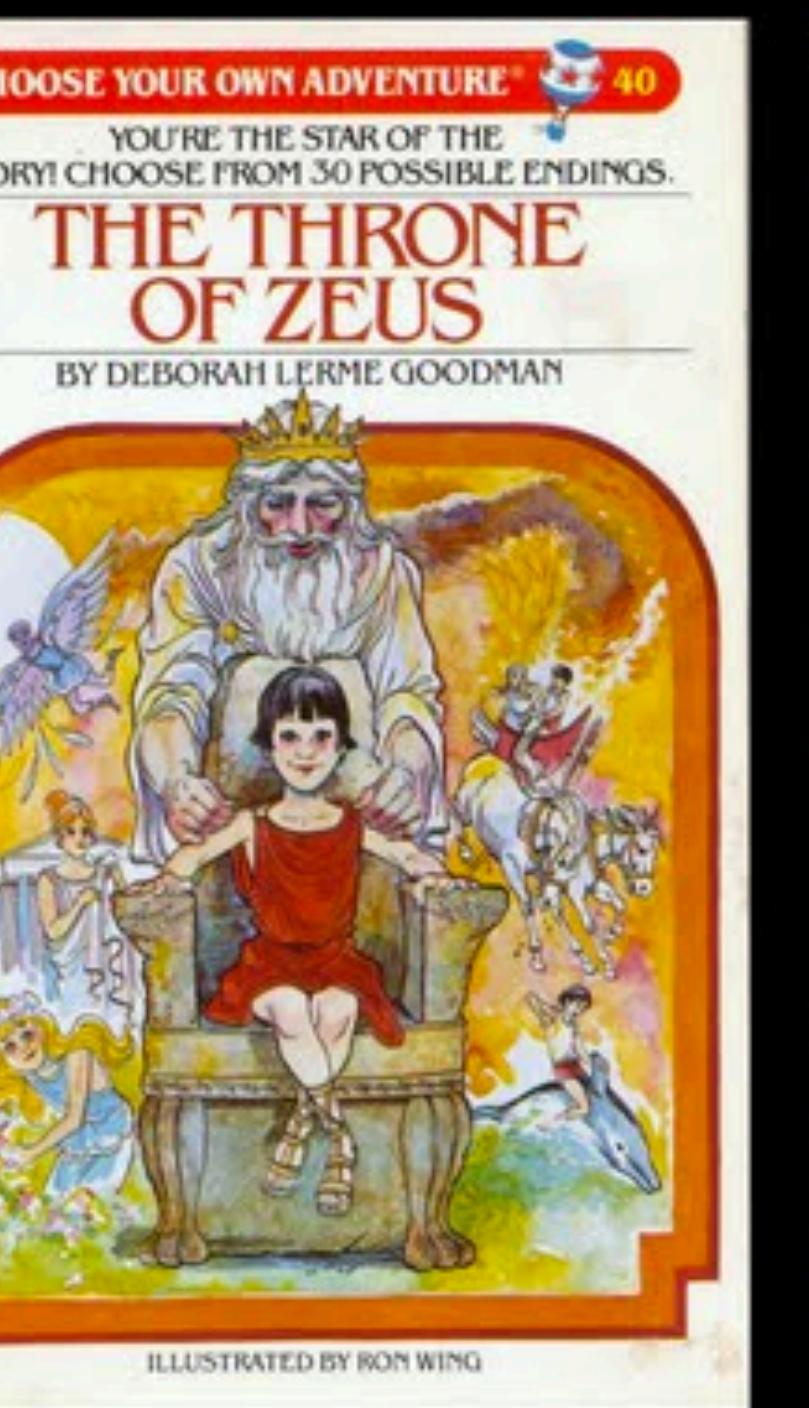
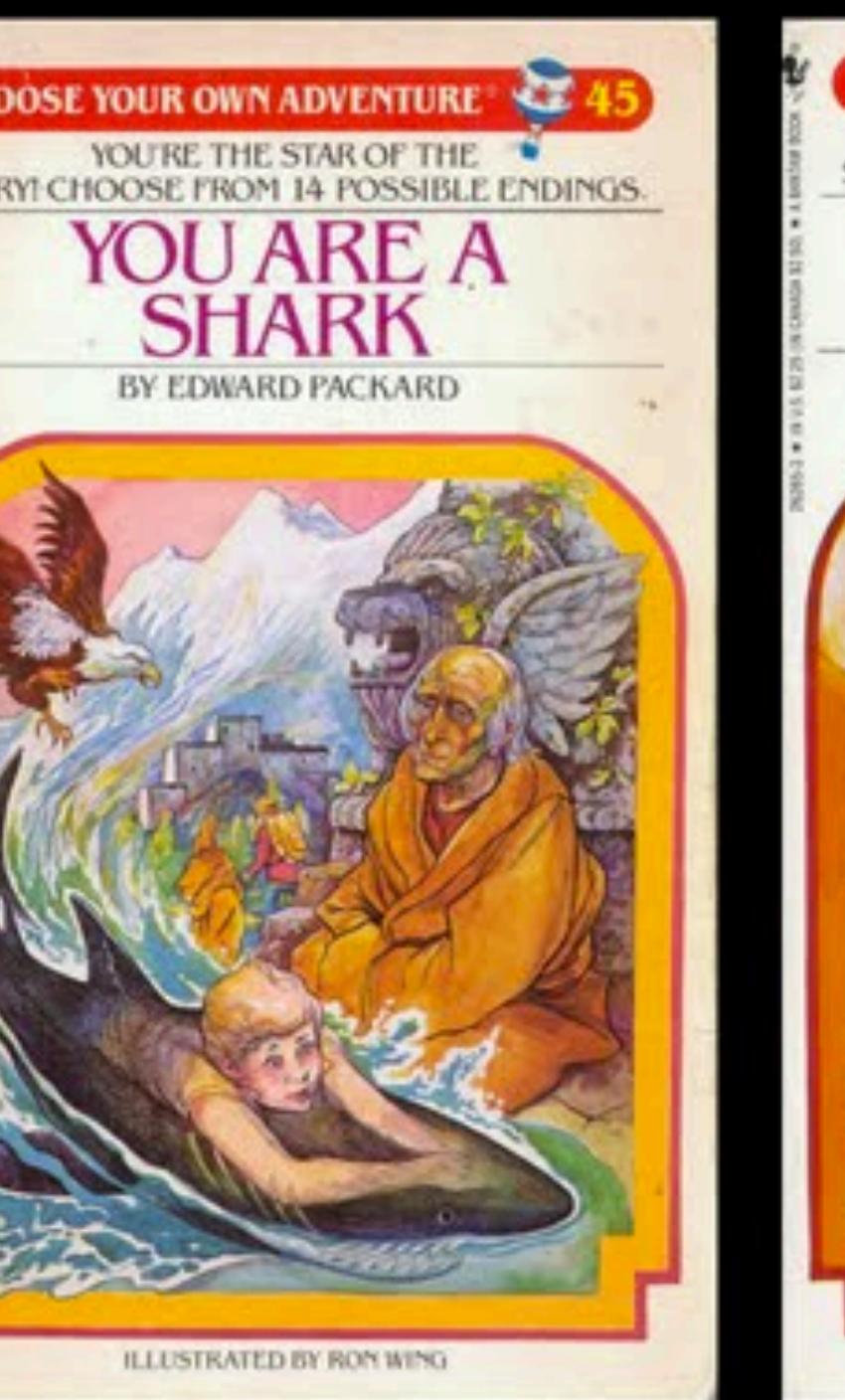
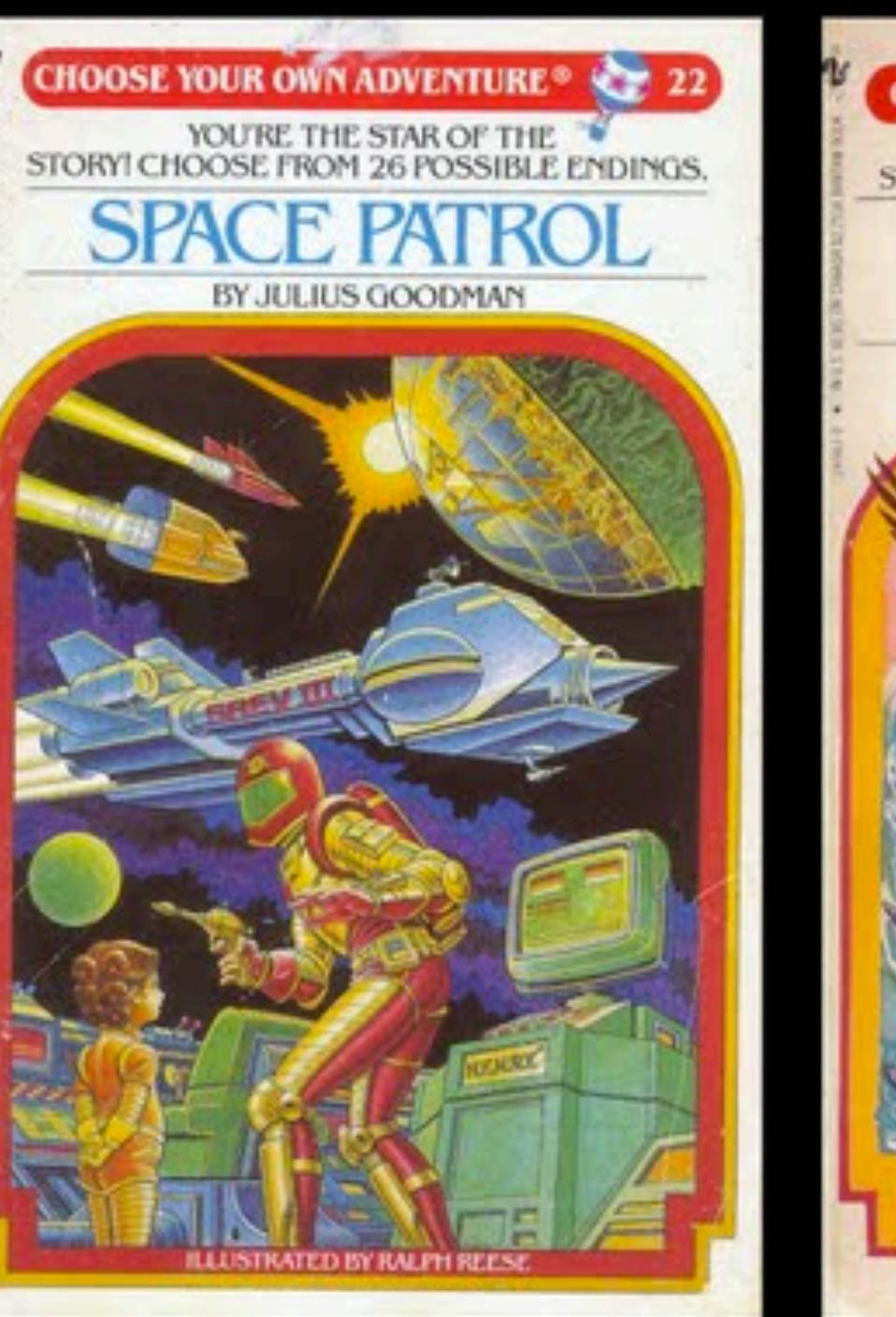
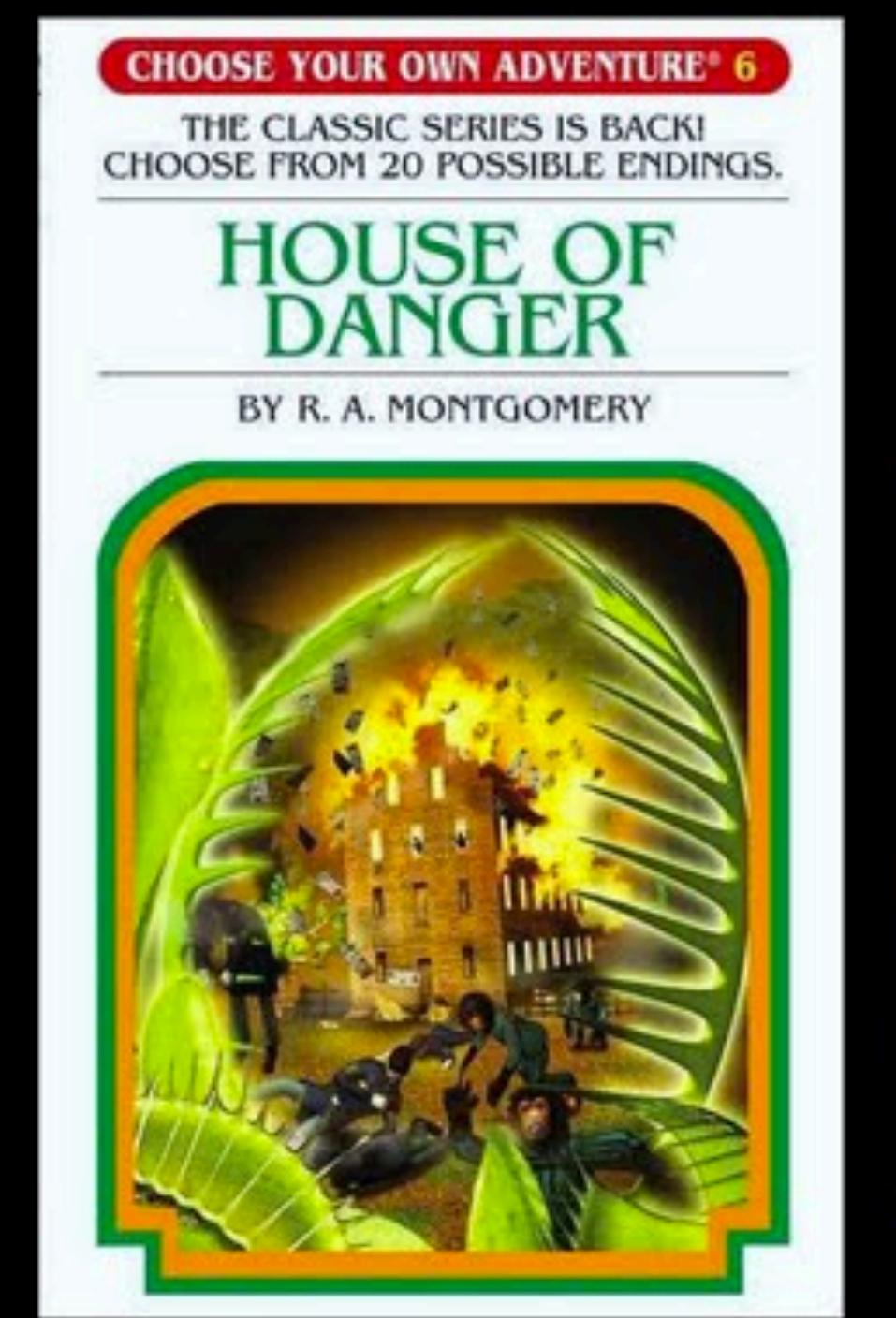
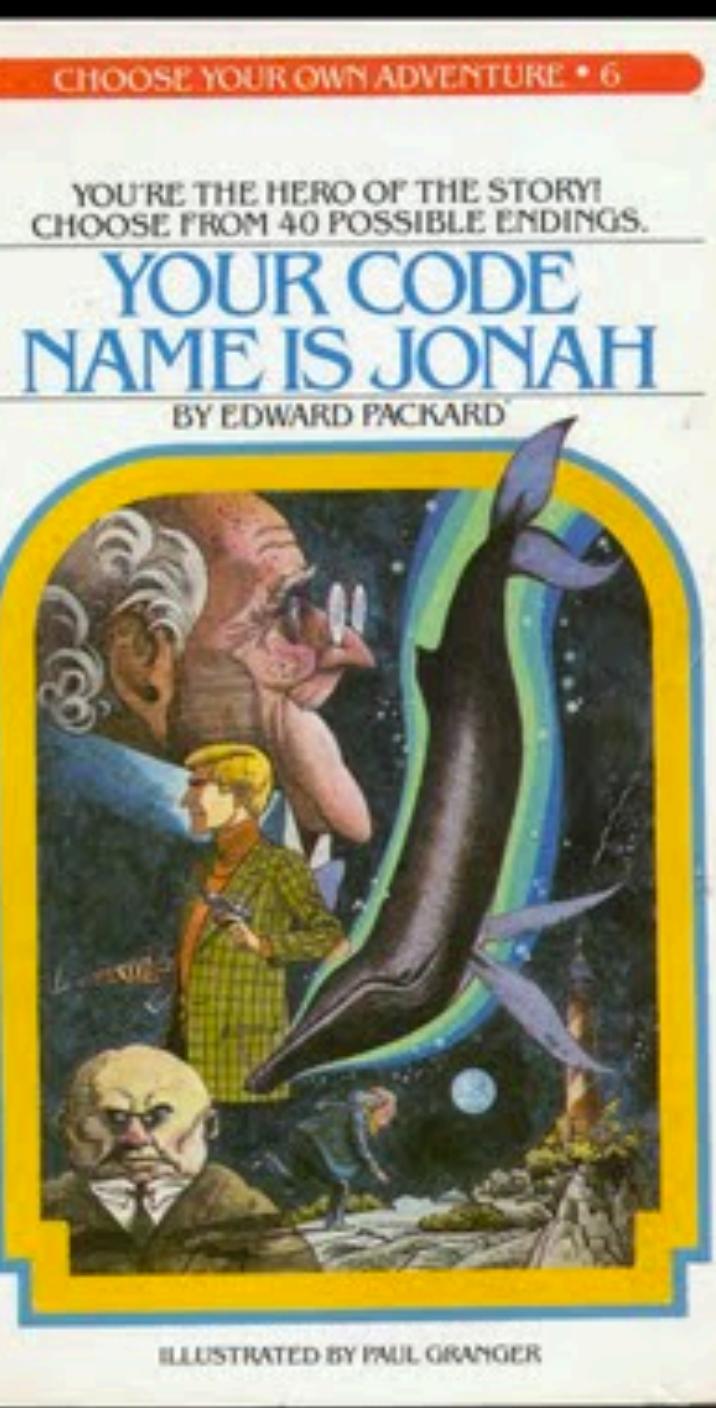
Unreliable Network



Limited Resources



User Interaction





# Reactive Extensions

Created by Microsoft

Released in 2009

As set of libraries designed to aid  
with the creation of *Asynchronous*  
*and Event-Based* applications



# Who uses it?

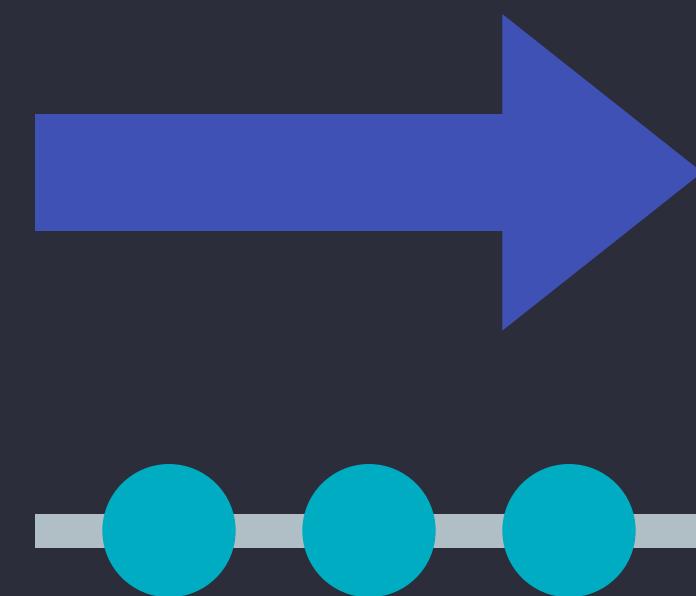


**NETFLIX** **Trello** **airbnb**

# When would I use this?



Real-time Data



Streams



User Interfaces

# Core Concepts

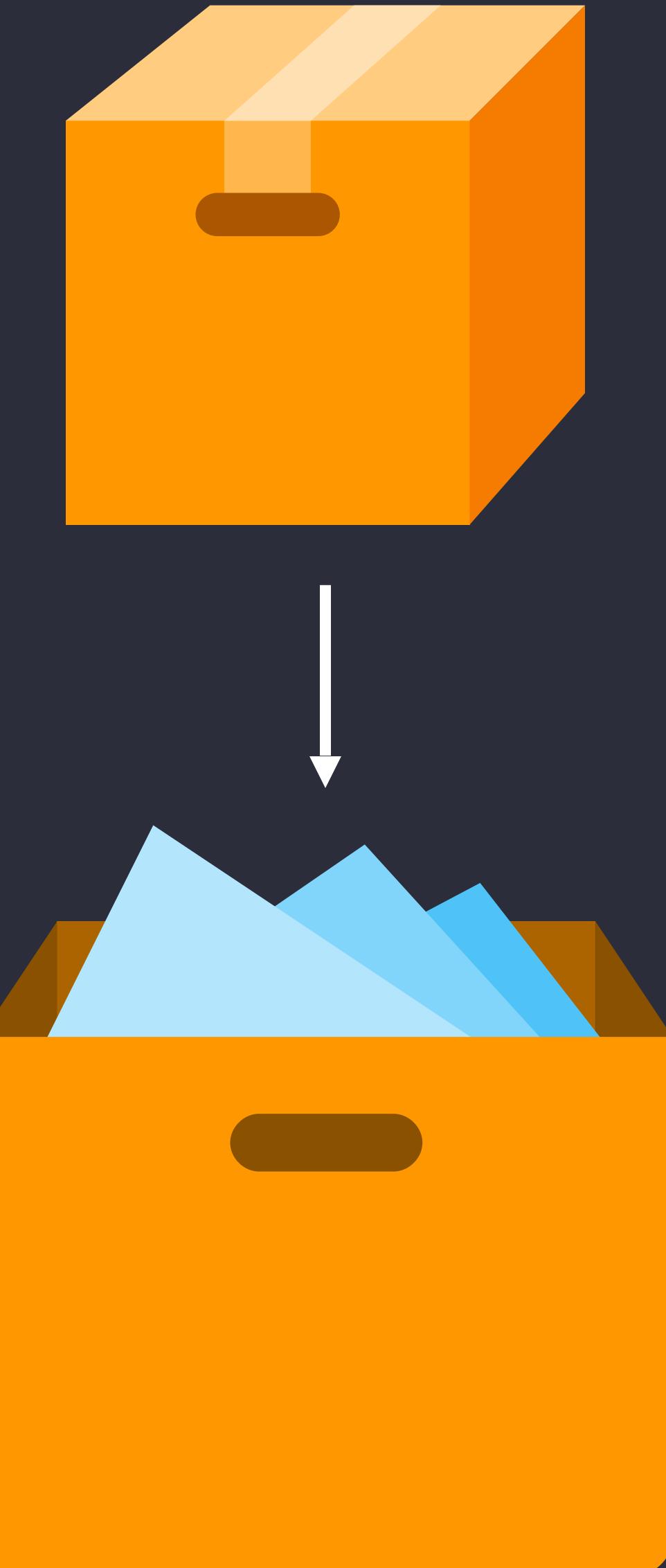
# IEnumerable<T>

Core interface for objects like List<T>

Provides a way to iterate through items

Generally comes as *prepackaged data* that we open up and comb through

We *PULL* data from it



# Typical `IEnumerable<T>` Usage

```
var people = await PersonRepository.GetAllPeopleAsync();  
foreach (var person in people) {  
    Debug.WriteLine($"{person.LastName}, {person.FirstName}");  
}
```

# Typical `IEnumerable<T>` Usage

```
var people = await PersonRepository.GetAllPeopleAsync();

foreach (var person in people) {
    Debug.WriteLine($"{person.LastName}, {person.FirstName}");
}
```

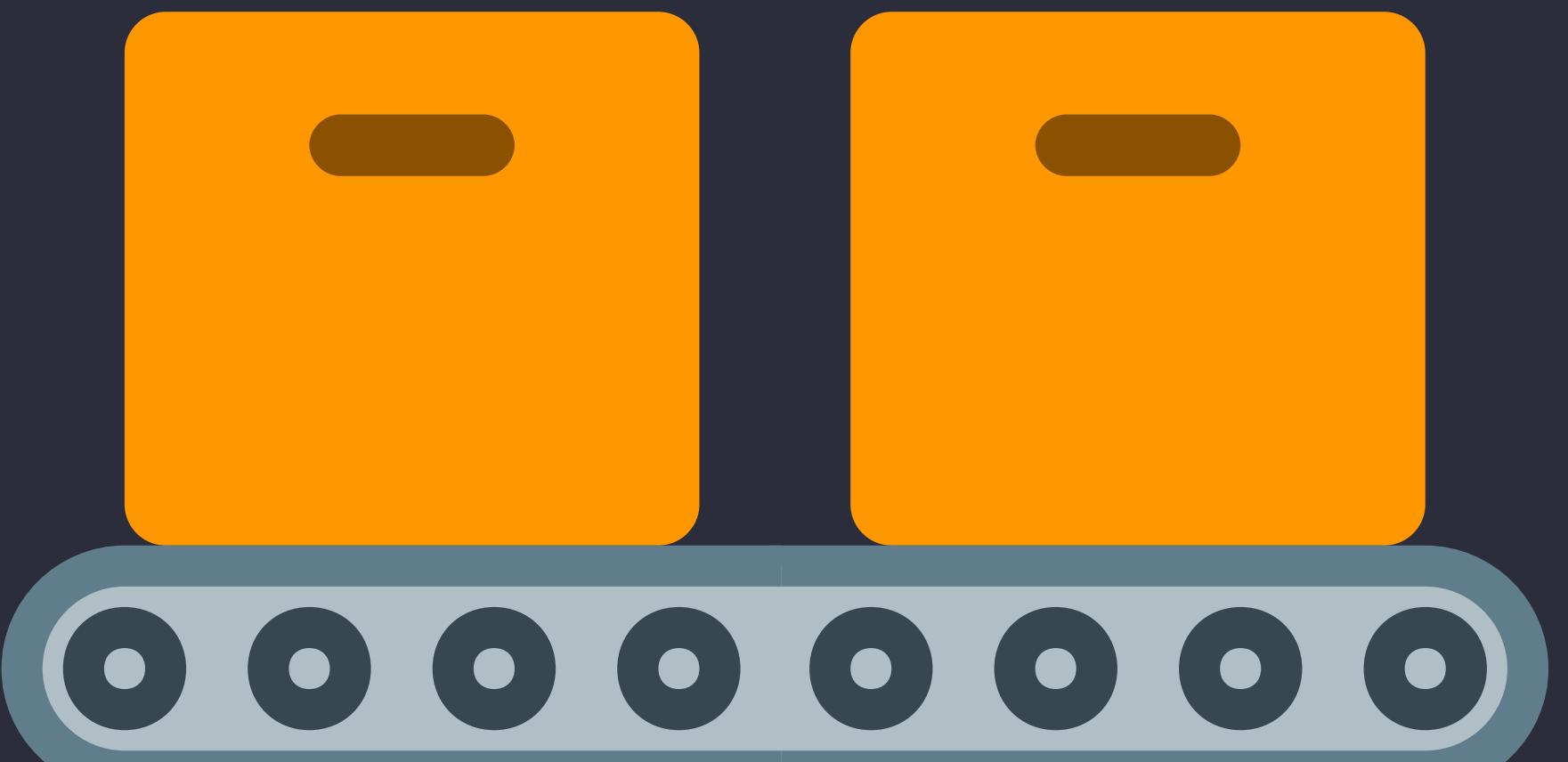
# I`Observable`<T>

*Pushes* data to subscribers

Like a conveyor belt or a stream

May or may not have an end

Values can be processed with LINQ



# Example Observables

```
Observable.Range (1, 100);
```

```
Observable.Timer (TimeSpan.FromSeconds (3));
```

```
Observable
    .FromEventPattern<EventHandler<TextChangedEventArgs>, TextChangedEventArgs> (
        x => textEntry.TextChanged += x,
        x => textEntry.TextChanged -= x
    );
```

# Example Observables

```
Observable.Range (1, 100);
```

```
Observable.Timer (TimeSpan.FromSeconds (3));
```

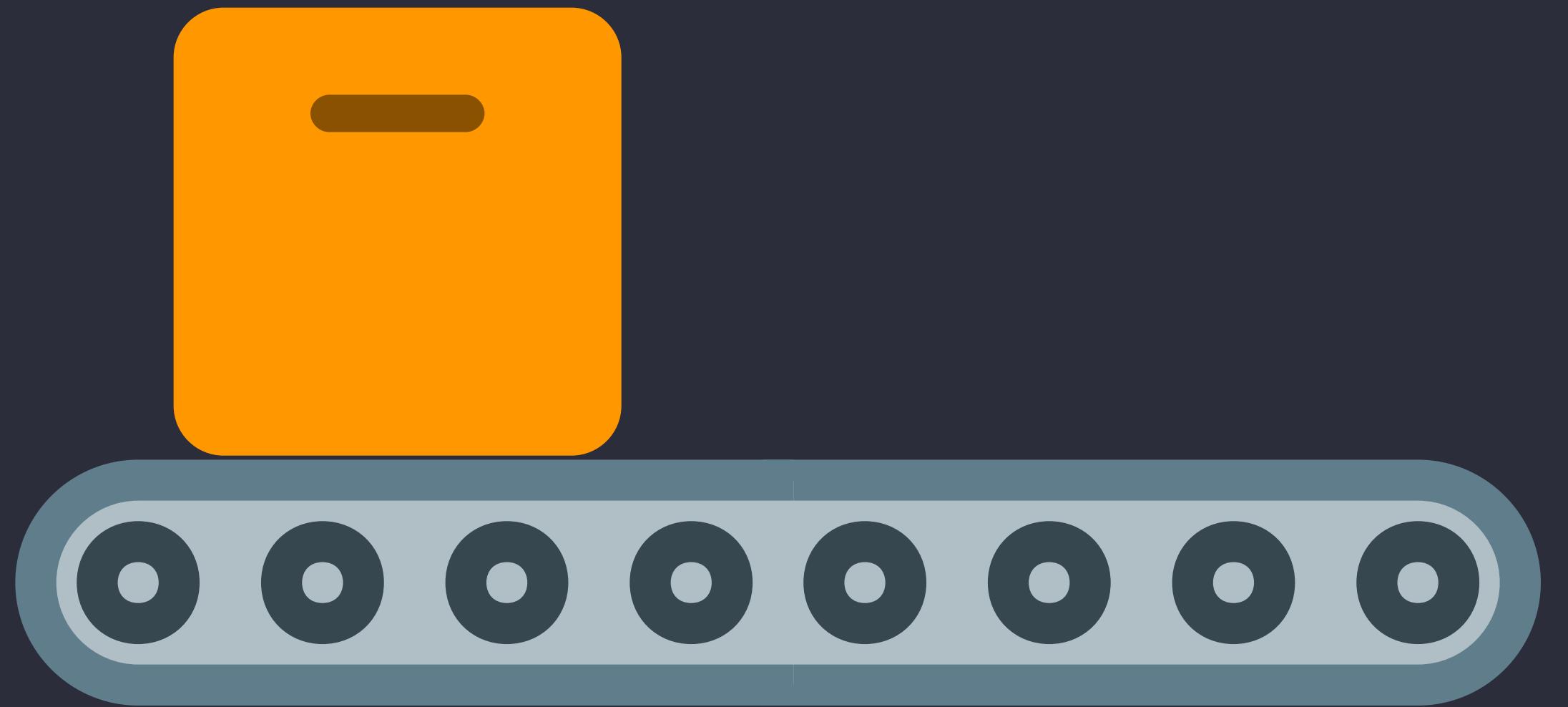
```
Observable
    .FromEventPattern<EventHandler<TextChangedEventArgs>, TextChangedEventArgs> (
        x => textEntry.TextChanged += x,
        x => textEntry.TextChanged -= x
    );
```

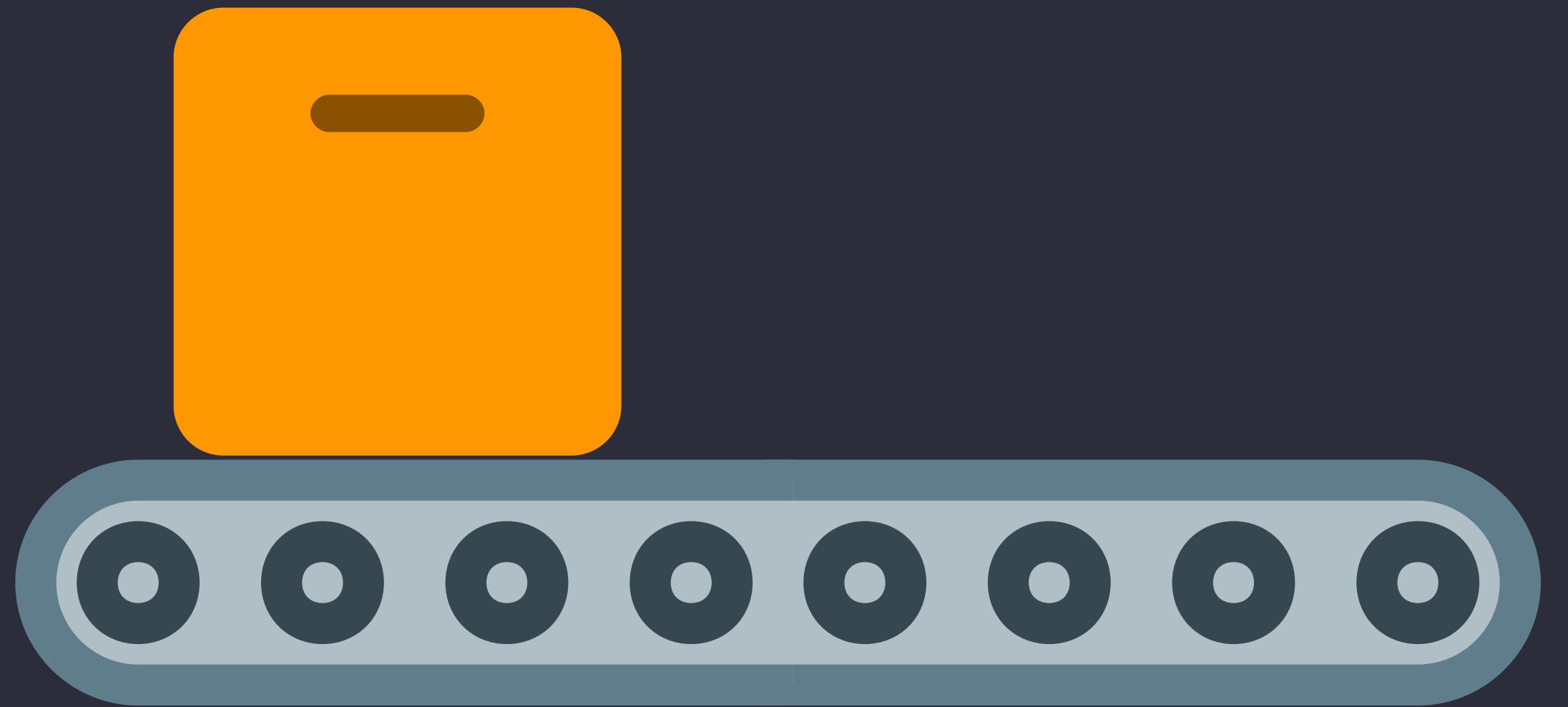
# Example Observables

```
Observable.Range (1, 100);
```

```
Observable.Timer (TimeSpan.FromSeconds (3));
```

```
Observable
    .FromEventPattern<EventHandler<TextChangedEventArgs>, TextChangedEventArgs> (
        x => textEntry.TextChanged += x,
        x => textEntry.TextChanged -= x
    );
```





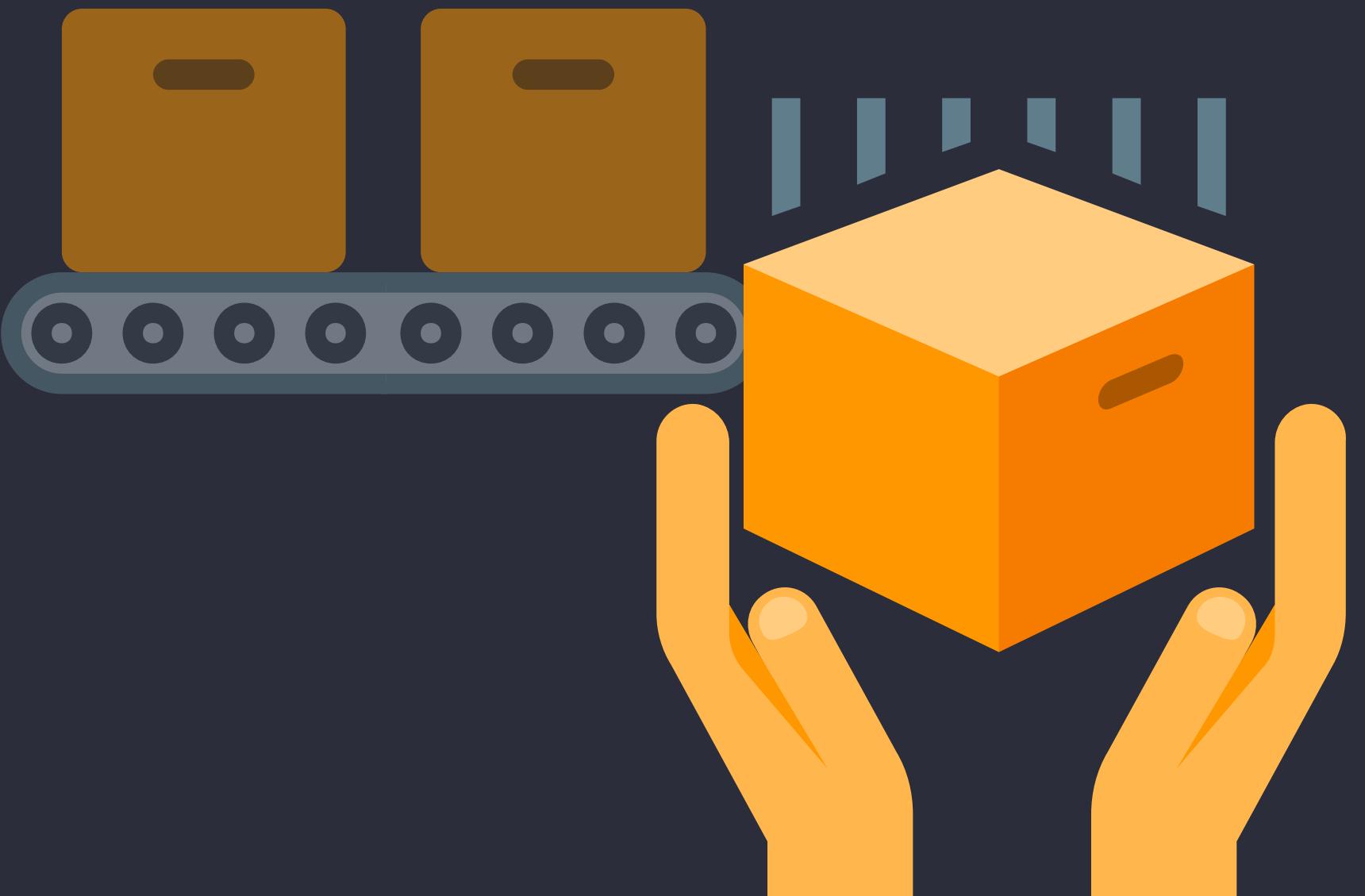
# Subscribing

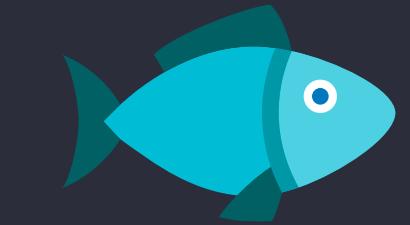
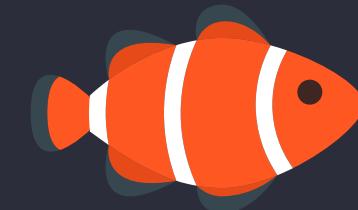
Gets values from `IObservable<T>`

Each value processed using `Action<T>`

Can have multiple subscriptions

Unsubscribe using `IDisposable`





Subscriber<Fish> Observable<Fish>

# Example Subscriptions

```
var fishSubscription =  
    fishObservable  
        .Subscribe (fish => Debug.WriteLine($"{{fish.Name}}"));
```

```
fishSubscription.Dispose();
```

```
var fishSubscription =  
    fishObservable  
        .Where(fish => !fish.IsCrustacean)  
        .Select(fish => fish.Name)  
        .Subscribe (name => Debug.WriteLine($"{{name}}"));
```

```
fishSubscription.Dispose();
```

# Example Subscriptions

```
var fishSubscription =  
fishObservable  
.Subscribe (fish => Debug.WriteLine($"{{fish.Name}}"));
```

```
fishSubscription.Dispose();
```

```
var fishSubscription =  
fishObservable  
.Where(fish => !fish.IsCrustacean)  
.Select(fish => fish.Name)  
.Subscribe (name => Debug.WriteLine($"{{name}}"));
```

```
fishSubscription.Dispose();
```

# Example Subscriptions

```
| var fishSubscription =  
|   fishObservable  
|     .Subscribe (fish => Debug.WriteLine($"{{fish.Name}}"));
```

```
| fishSubscription.Dispose();
```

```
var fishSubscription =  
  fishObservable  
  .Where(fish => !fish.IsCrustacean)  
  .Select(fish => fish.Name)  
  .Subscribe (name => Debug.WriteLine($"{{name}}"));
```

```
fishSubscription.Dispose();
```

# Example Subscriptions

```
var fishSubscription =  
    fishObservable  
        .Subscribe (fish => Debug.WriteLine($"{{fish.Name}}"));
```

```
fishSubscription.Dispose();
```

```
var fishSubscription =  
    fishObservable  
        .Where(fish => !fish.IsCrustacean)  
        .Select(fish => fish.Name)  
        .Subscribe (name => Debug.WriteLine($"{{name}}"));
```

```
fishSubscription.Dispose();
```



# Real-World Examples

# CompositeDisposable

A container for multiple disposable objects

Allows you to dispose of many objects in one call

Awesome for disposing of UI components



# CompositeDisposable

```
var disposables = new CompositeDisposable();

disposables.Add(subscription1);

disposables.Add(subscription2);

disposables.Add(subscription3);

disposables.Clear();
```

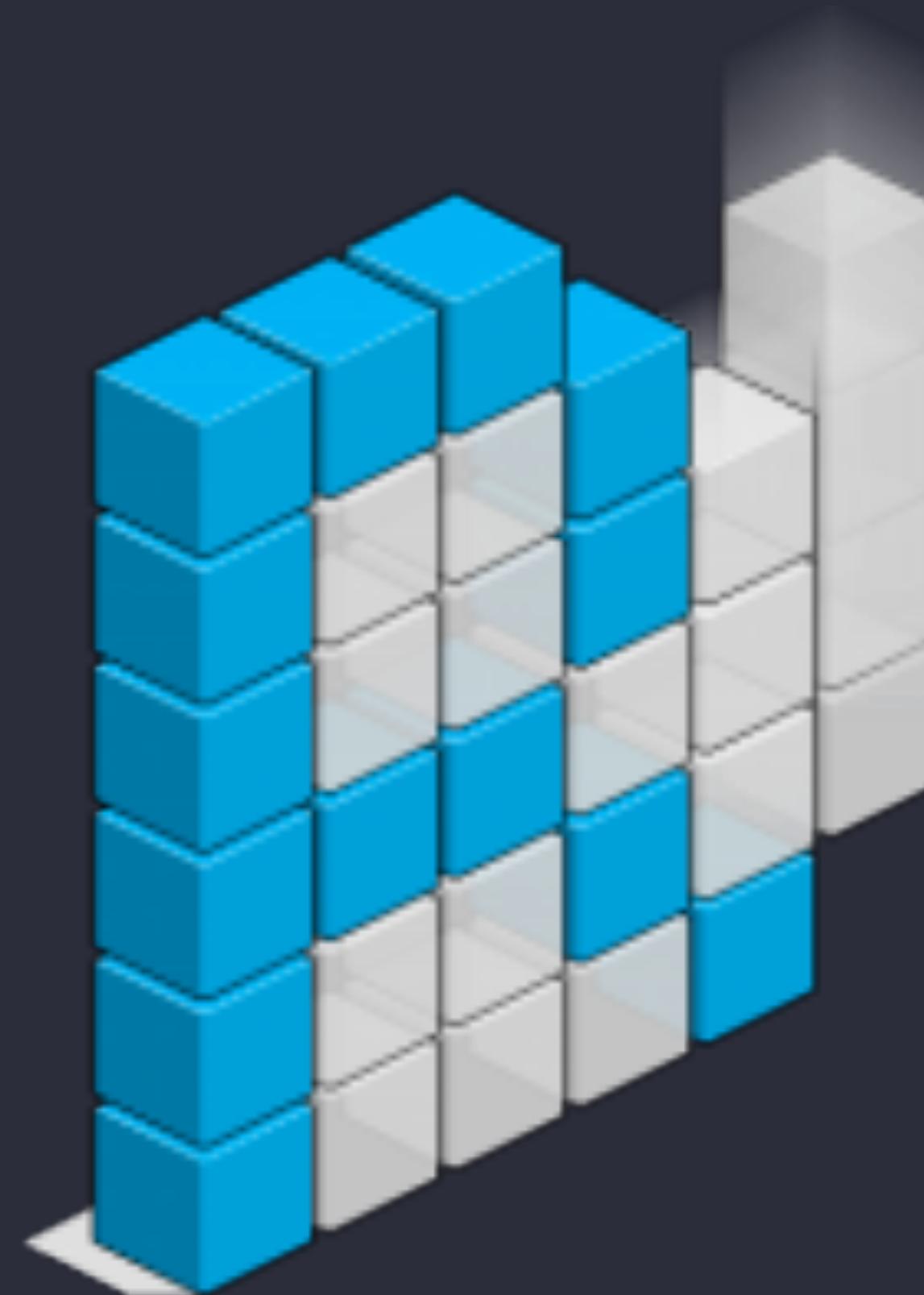
# ReactiveUI

Extends Reactive Extensions for  
making *elegant* and *testable* MVVM  
User Interfaces

Supports all major .Net UI platforms

Works with other MVVM Frameworks  
like MvvmCross or Xamarin.Forms

<http://reactiveui.net>



where Do I Find Out  
More?

# More Resources

- [reactivex.io](http://reactivex.io)
- [introtorx.com](http://introtorx.com)
- [rxmarbles.com](http://rxmarbles.com)
- [github.com/ReactiveX](https://github.com/ReactiveX)
- [reactiveui.net](http://reactiveui.net)
- [github.com/TheEightBot/Reactive-Examples](https://github.com/TheEightBot/Reactive-Examples)



# Questions?



# Why you should be building better Mobile Apps with Reactive Programming

Michael Stonis  
Eight-Bot  
@MichaelStonis