

RD FRAMEWORK

Reactive distributed cross-platform communication

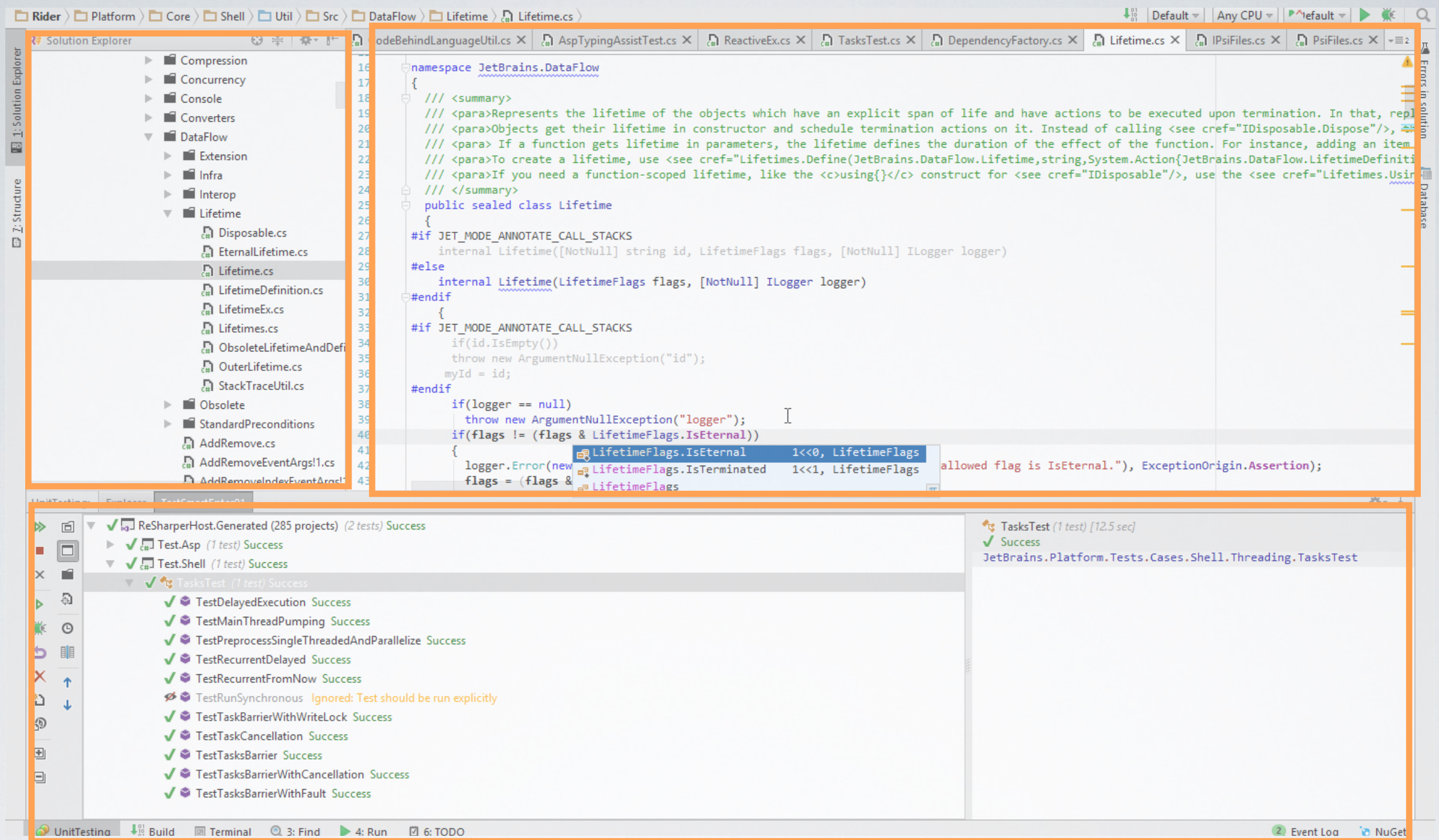
Dmitry Ivanov, JetBrains



JetBrains.RdFramework 2019.3.0 

JetBrains Networking library for reactive distributed communication

NUGET



INCEPTION

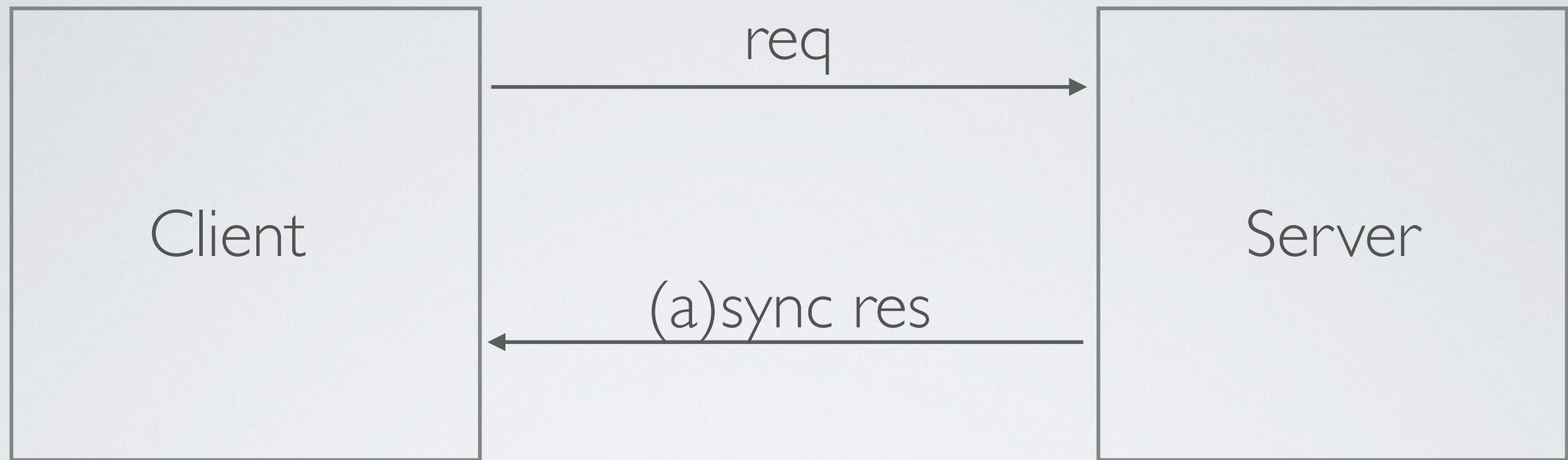
Hierarchical nature of view model


```
16 namespace JetBrains.DataFlow
17 {
18     /// <summary>
19     /// <para>Represents the lifetime of the objects which have an explicit span of life and have actions to be executed upon termination. In that, repl
20     /// <para>Objects get their lifetime in constructor and schedule termination actions on it. Instead of calling <see cref="IDisposable.Dispose"/>,
21     /// <para>If a function gets lifetime in parameters, the lifetime defines the duration of the effect of the function. For instance, adding an item
22     /// <para>To create a lifetime, use <see cref="Lifetimes.Define(JetBrains.DataFlow.Lifetime,string,System.Action{JetBrains.DataFlow.LifetimeDefiniti
23     /// <para>If you need a function-scoped lifetime, like the <code>using{}</code> construct for <see cref="IDisposable"/>, use the <see cref="Lifetimes.Usin
24     /// </summary>
25     public sealed class Lifetime
26     {
27         #if JET_MODE_ANNOTATE_CALL_STACKS
28         internal Lifetime([NotNull] string id, LifetimeFlags flags, [NotNull] ILogger logger)
29         #else
30         internal Lifetime(LifetimeFlags flags, [NotNull] ILogger logger)
31         #endif
32         {
33             #if JET_MODE_ANNOTATE_CALL_STACKS
34             if(id.IsEmpty())
35                 throw new ArgumentNullException("id");
36             myId = id;
37             #endif
38             if(logger == null)
39                 throw new ArgumentNullException("logger");
40             if(flags != (flags & LifetimeFlags.IsEternal))
41             {
42                 logger.Error(new
43                 flags = (flags &
```

Visual Studio interface showing the `Lifetime.cs` file in the `DataFlow` namespace. The file contains XML documentation and a public sealed class `Lifetime`. A red box highlights the XML documentation, an orange box highlights the file name in the tab bar, and a green box highlights a code completion menu for `LifetimeFlags`. A yellow warning icon is visible in the top right corner.

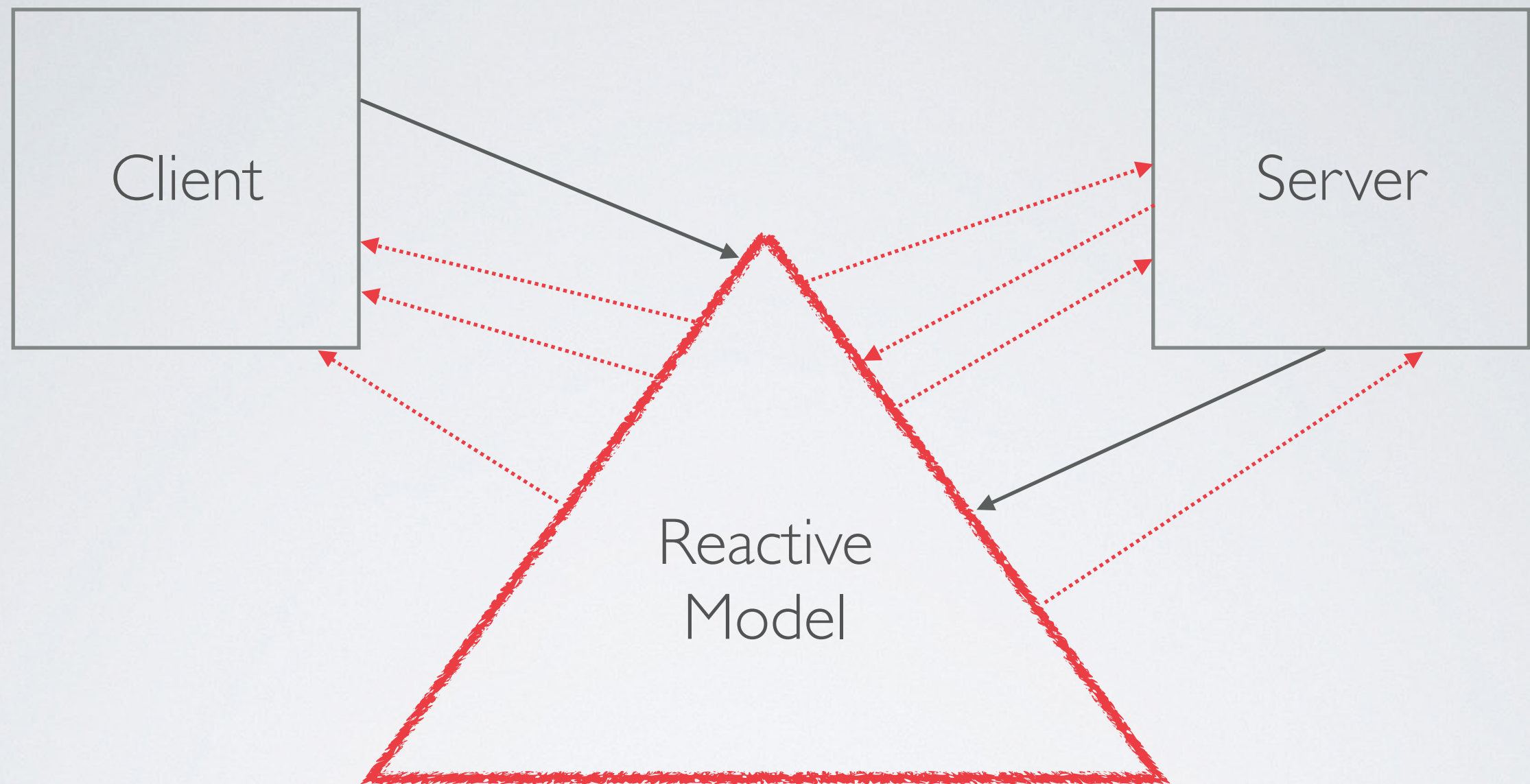
INCEPTION

Hierarchical nature of view model



INCEPTION

Why not RPC

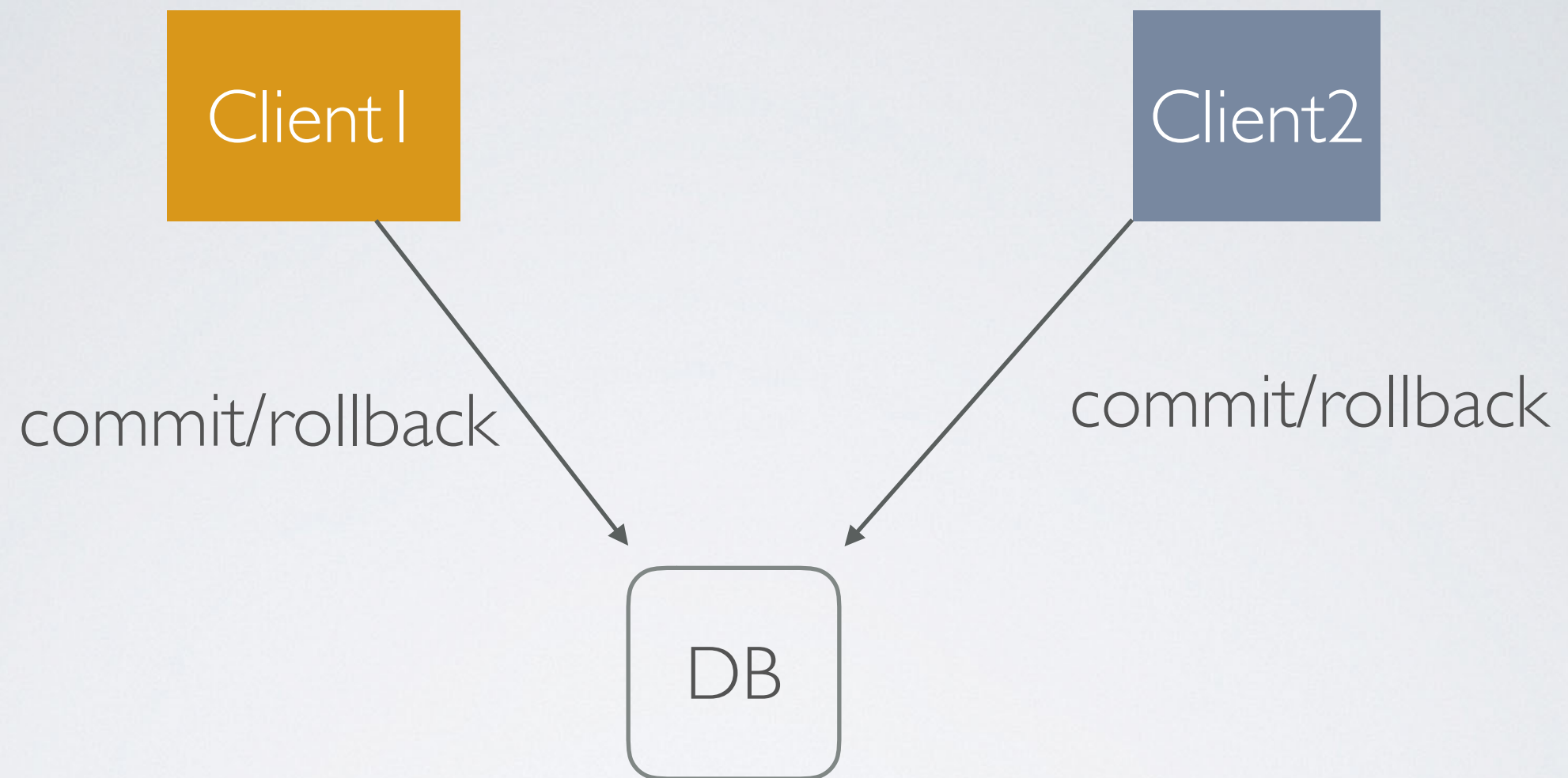


INCEPTION

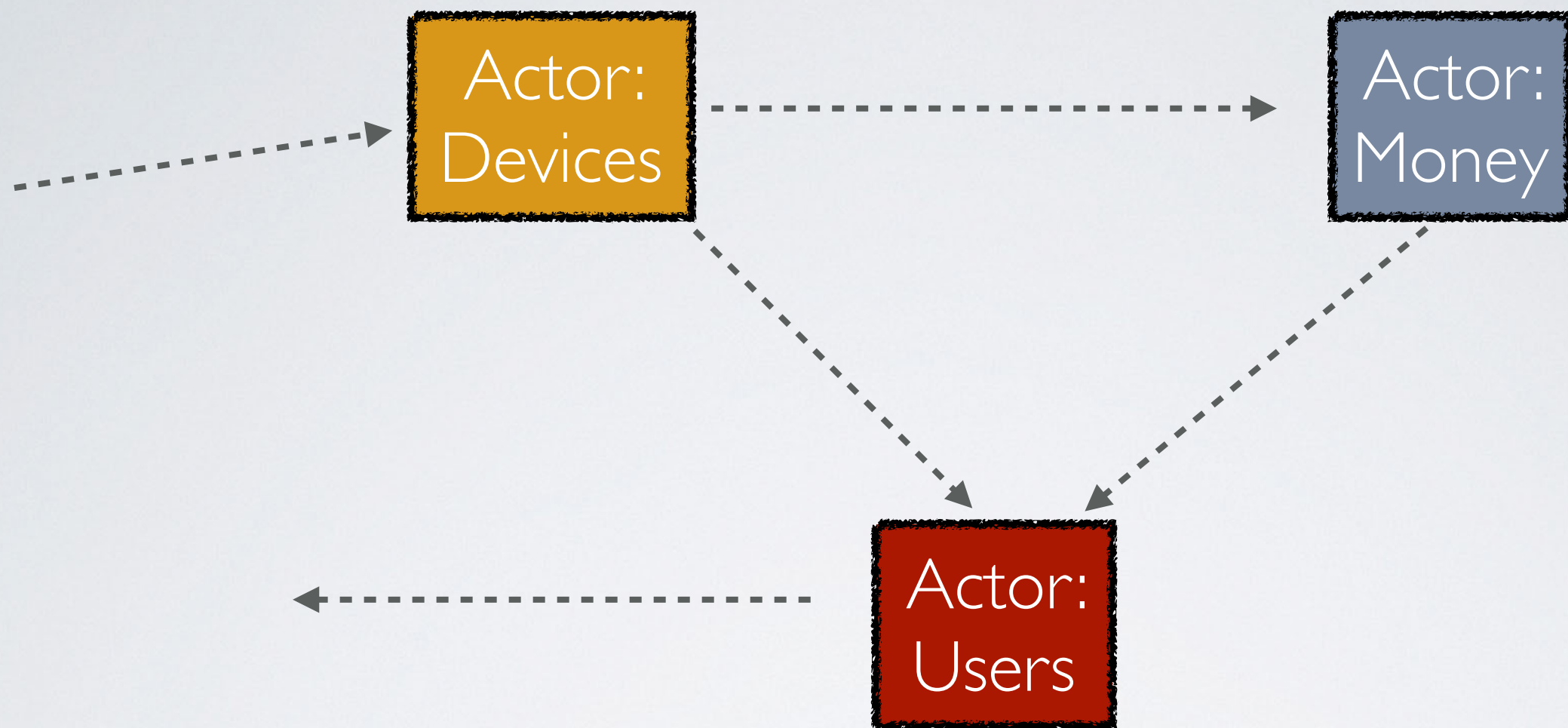
Because it's not **Reactive**!



OTHER MODELS



TRANSACTIONAL DATABASE



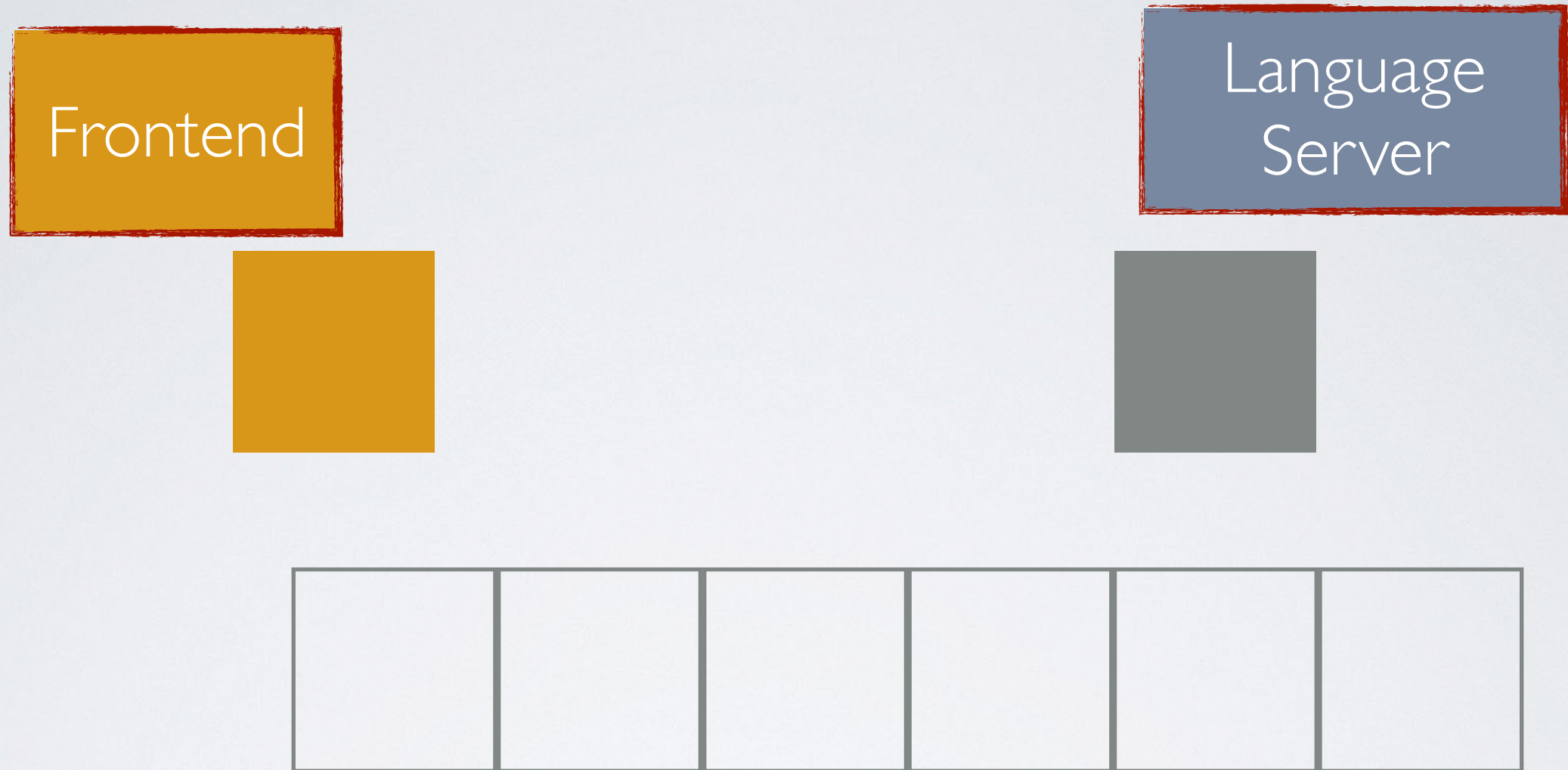
PERSISTENT ACTORS

Frontend

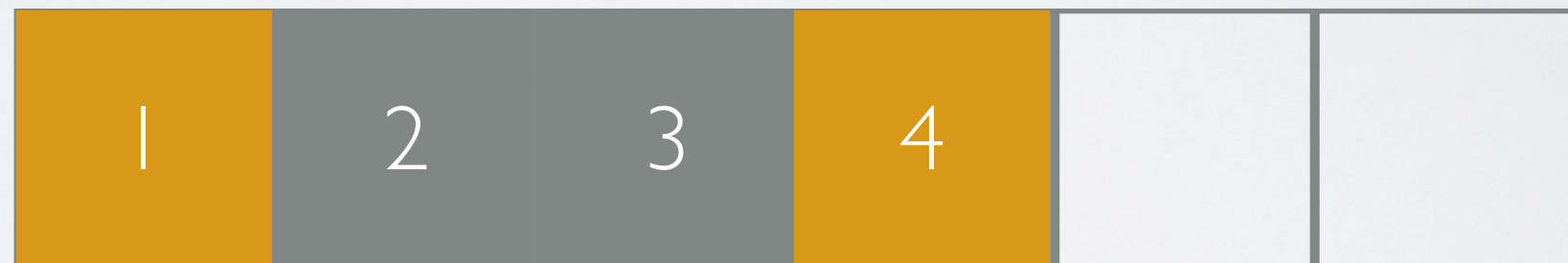
Language
Server



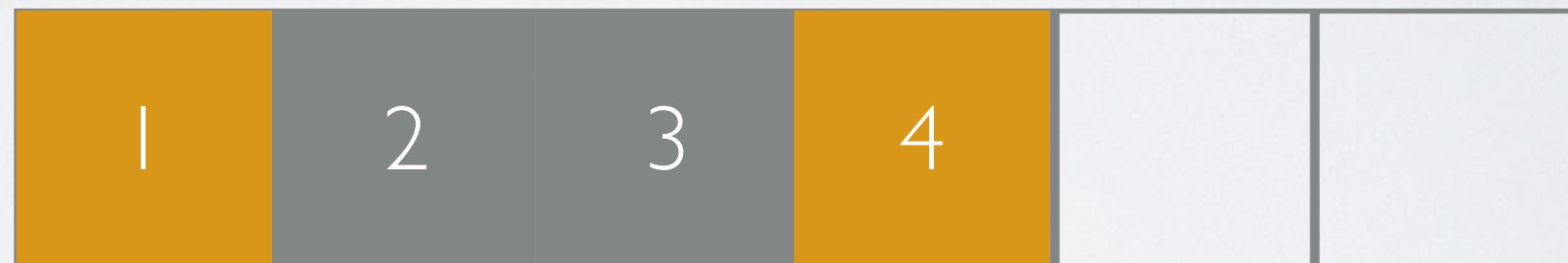
EVENT SOURCING



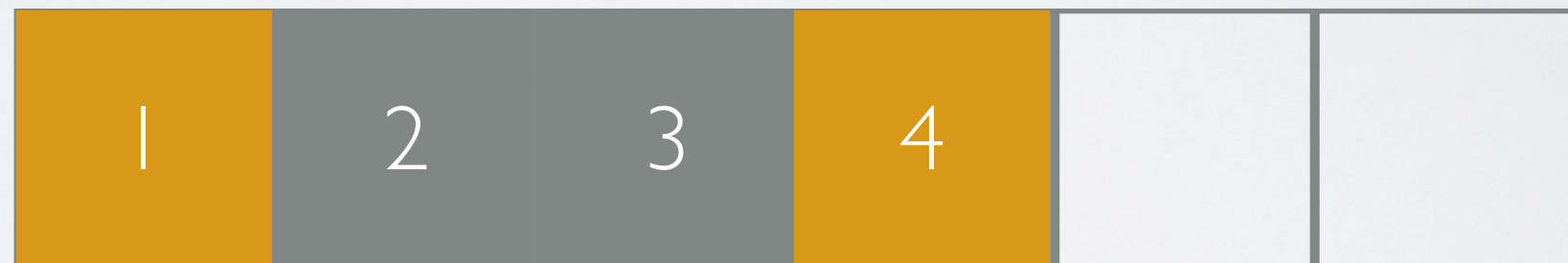
EVENT SOURCING



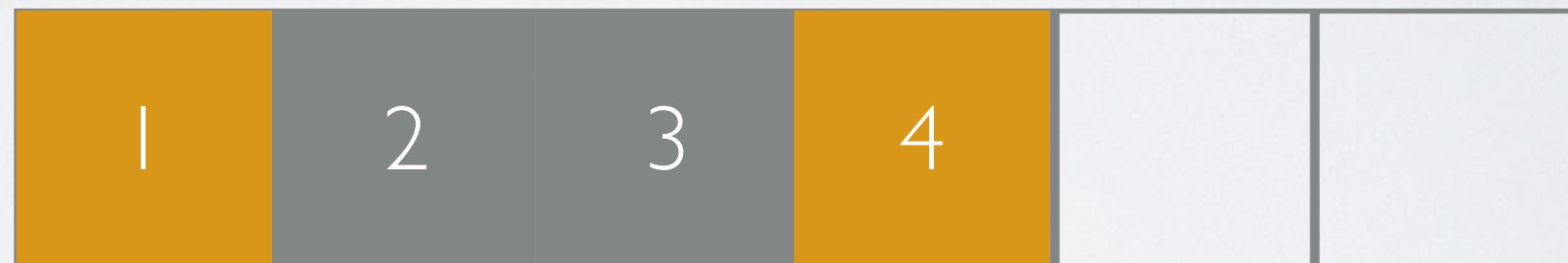
EVENT SOURCING



EVENT SOURCING



EVENT SOURCING



EVENT SOURCING

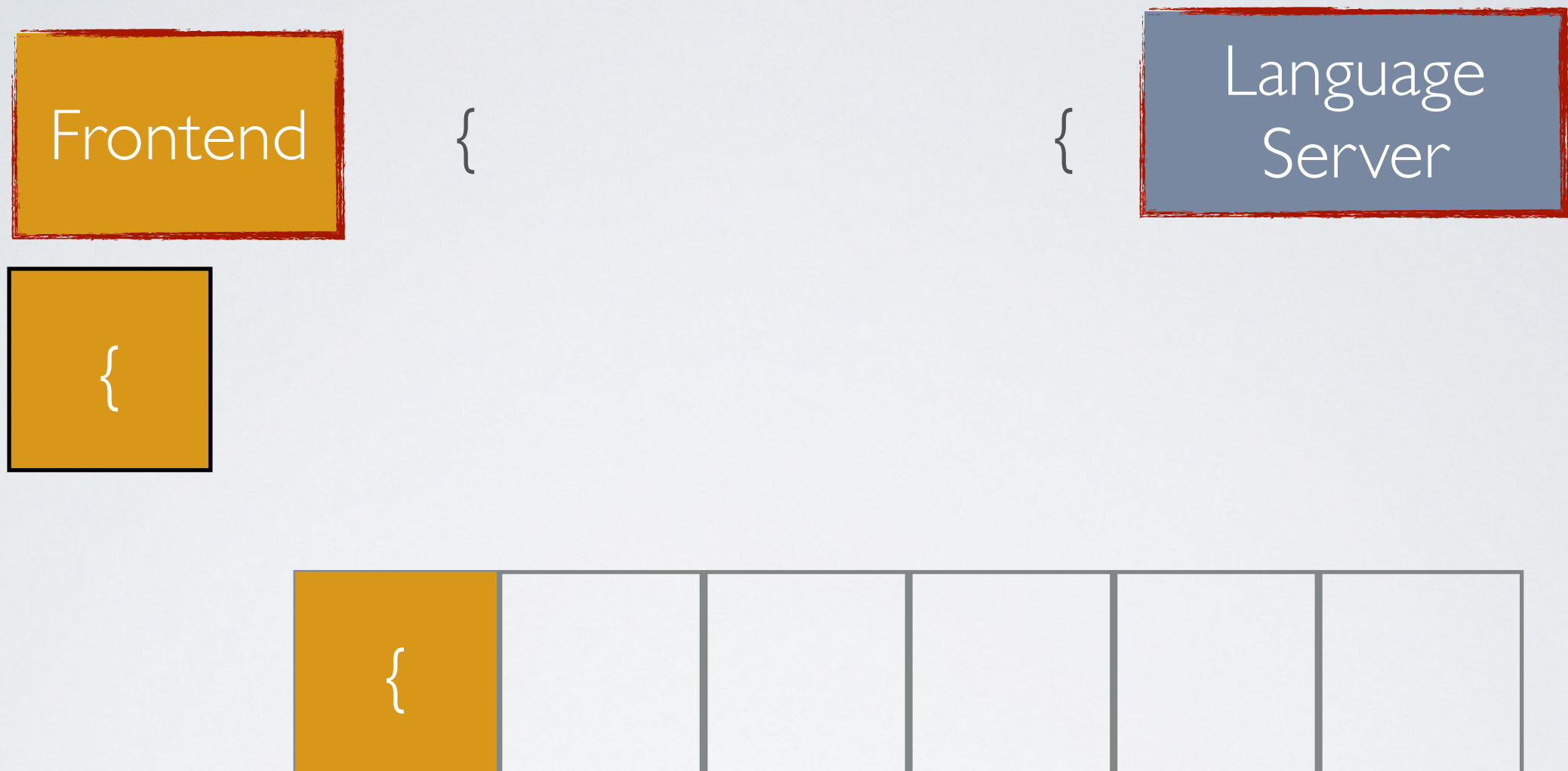
Frontend

Language
Server

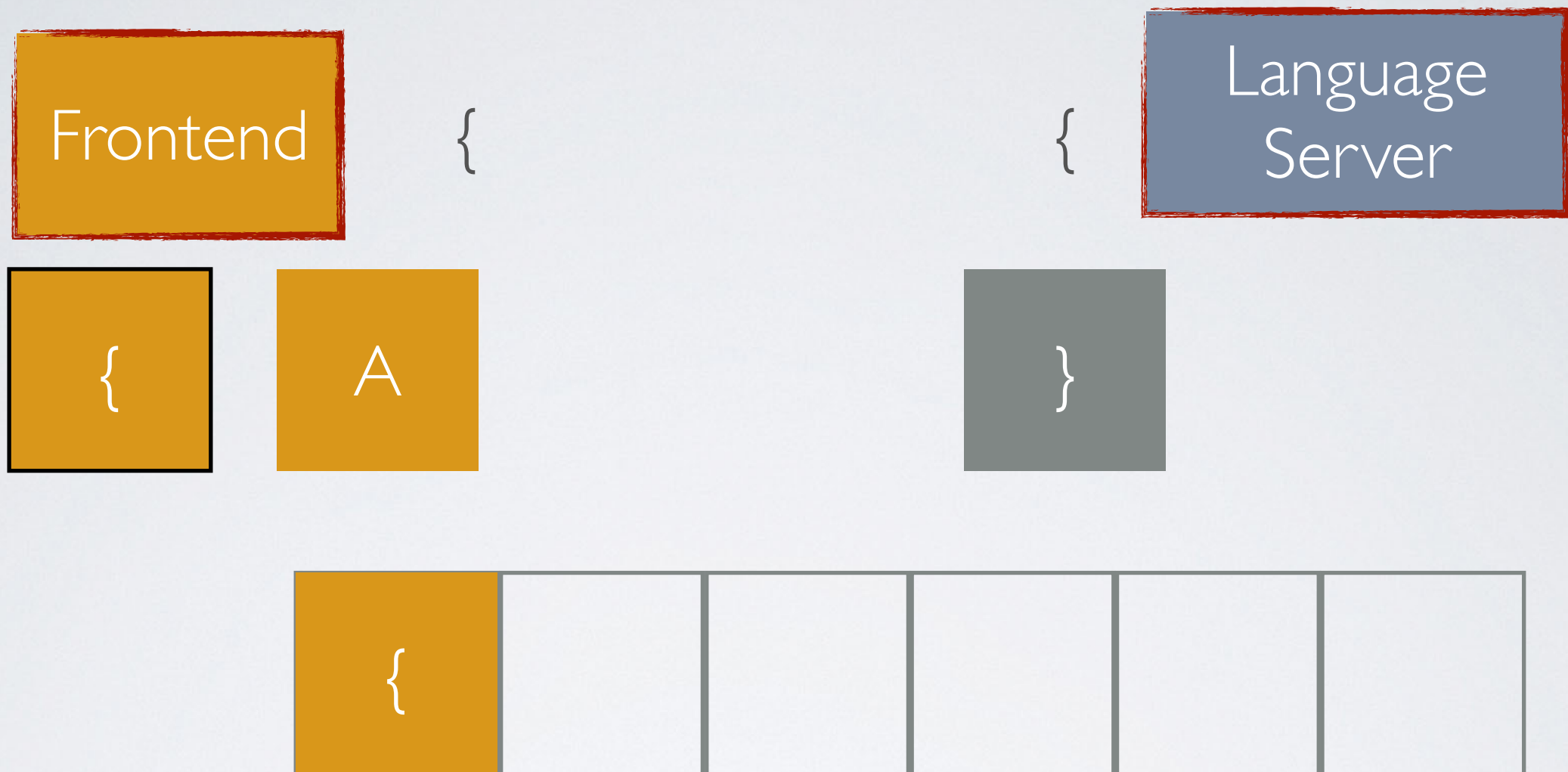
{



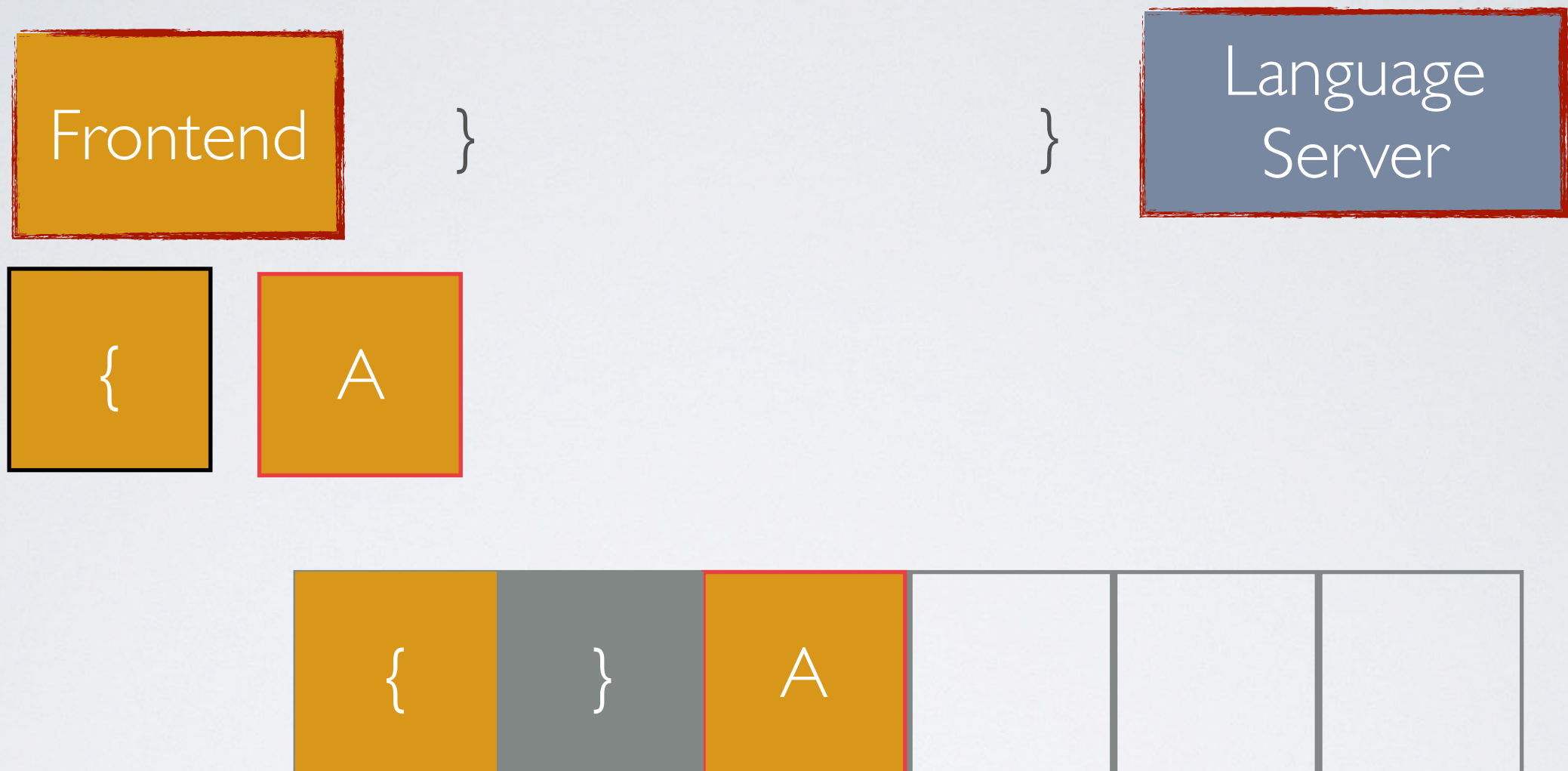
EVENT SOURCING



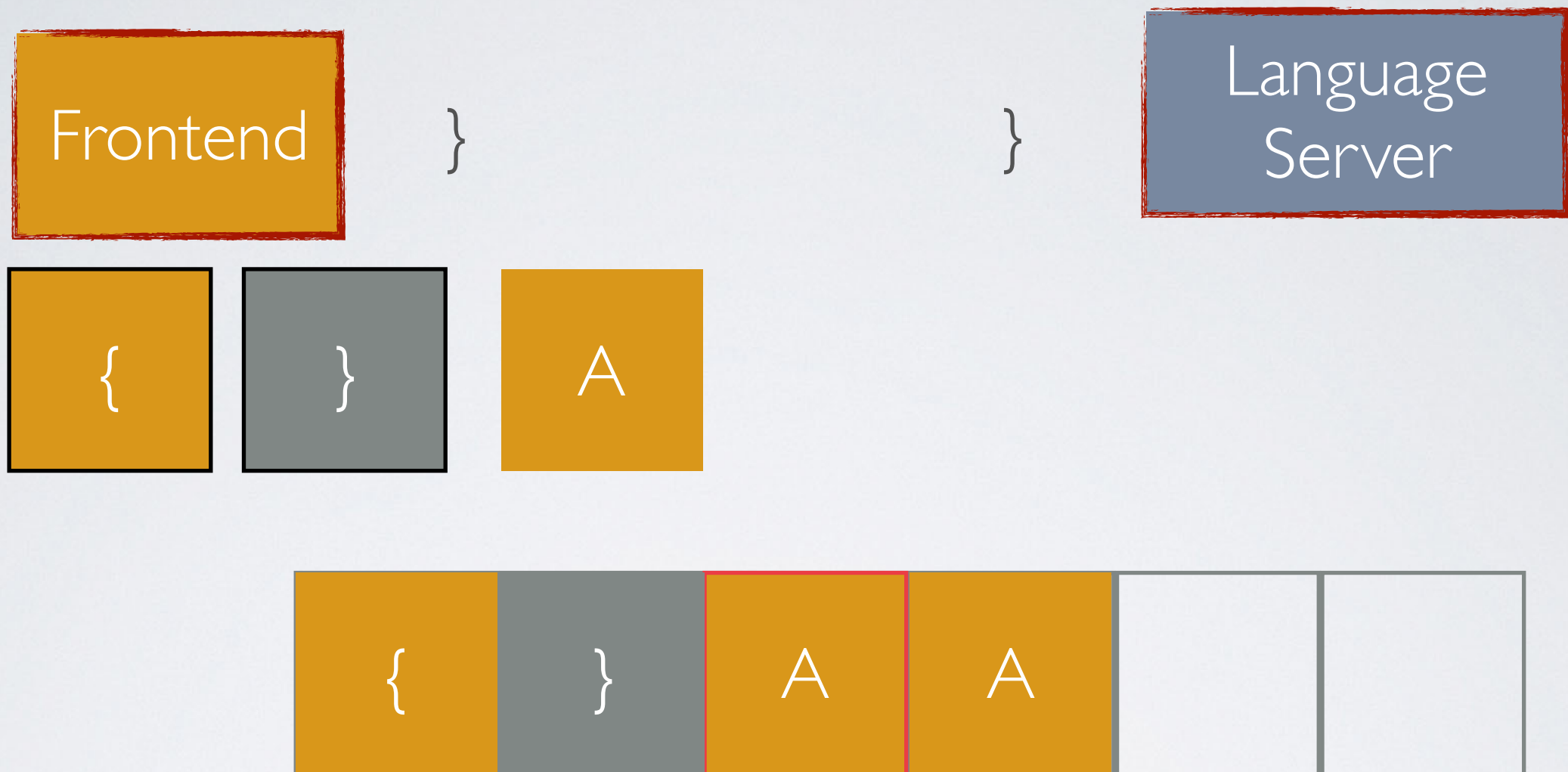
EVENT SOURCING



EVENT SOURCING



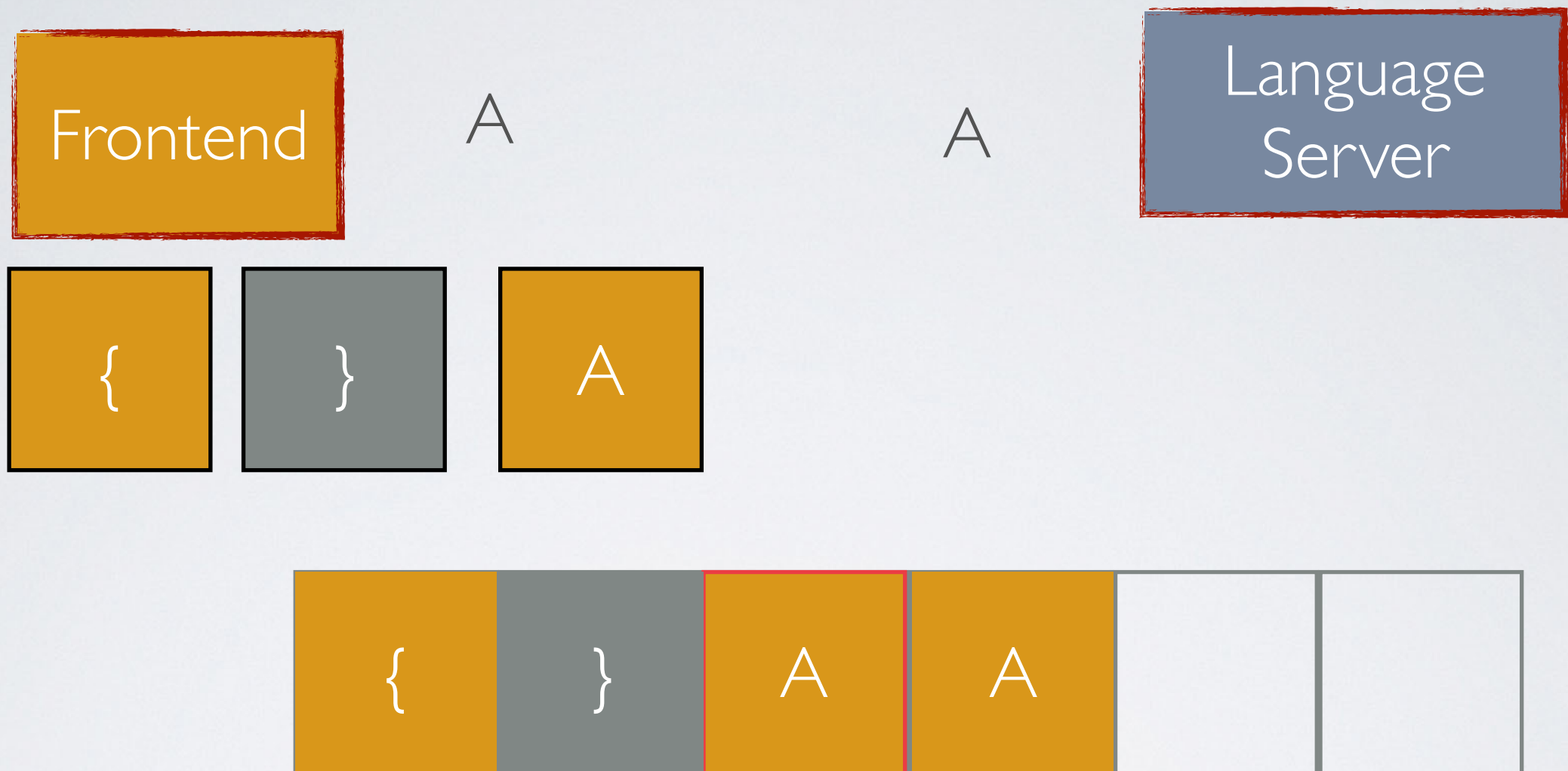
EVENT SOURCING



EVENT SOURCING



EVENT SOURCING



EVENT SOURCING



REACTIVE PRIMITIVES

```
interface ISignal<T> {  
    void Fire(T value)  
    void Advise(Lifetime lf, Action<T> handler)  
}
```

SIGNALS

let's apply lifetime pattern


```
interface ISignalFrontend<T> {  
    void Fire(T value)  
}
```

```
interface ISource<T> {  
    void Advise(Lifetime lf, Action<T> handler)  
}
```

```
interface ISignal<T> : ISource<T>
```

SIGNALS

separation of concerns

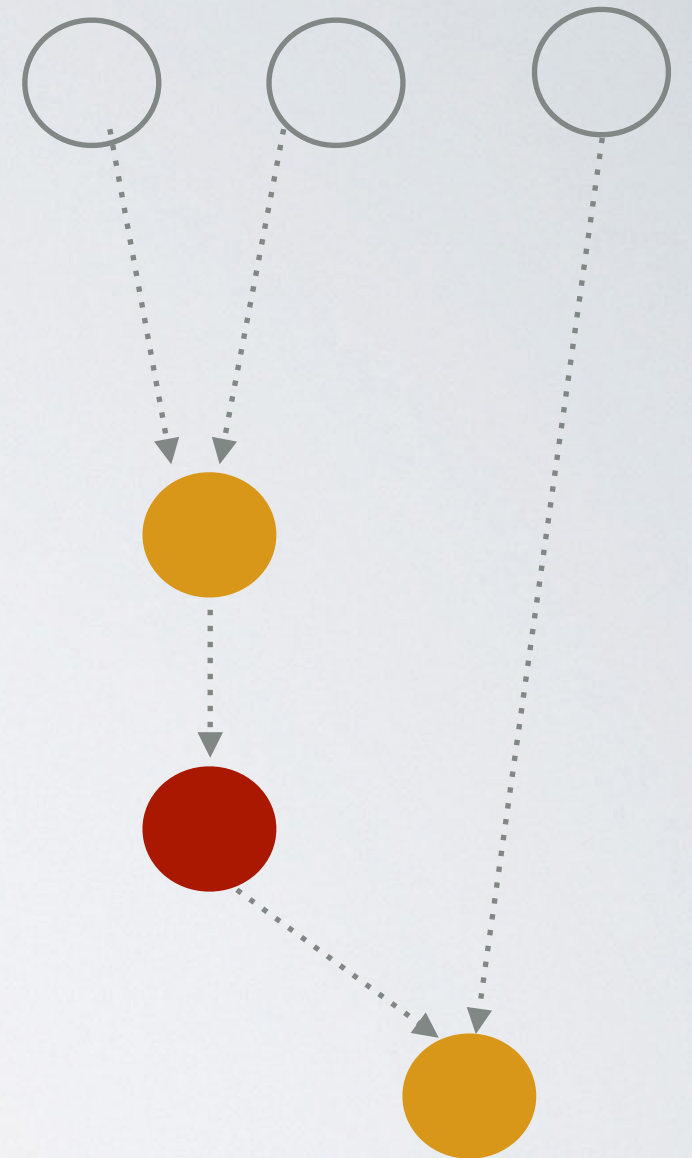

```
interface ISignal<Unit> {  
    void Advise(Lifetime lf, Action handler)  
}
```

SIGNALS

Of void

```
ISource<TRes>  
ISource<TOrig>.Select(  
    Func<TOrig, TRes>  
)
```

```
ISource<TRes>  
ISource<T1>.Compose(  
    ISource<T2>,  
    Func<T1, T2, TRes>  
)
```



SIGNALS

Reactive networks

```
interface ISource<T> {  
    void Advise(Lifetime lf, Action<T> handler)  
}
```

```
interface IViewableProperty<T> : ISource<T> {  
  
    T Value {get; set}  
  
    ISource<T> Change;  
}
```

PROPERTIES

stateful signals


```
interface ISink<T> {  
    void Advise(Lifetime lf, Action<T> handler)  
}  
  
interface IViewableProperty<T> : ISink<T> {  
    T Value {get; set}  
  
    void View(Lifetime lf,  
              Action<Lifetime, T> action)  
}
```

PROPERTIES

presenting **viewable** pattern

```
interface ISink<T> {  
    void Advise(Lifetime lf, Action<T> handler)  
}
```

```
interface IViewableProperty<T> : ISink<T> {  
    T Value {get; set}  
  
    void View(Lifetime lf,  
              Action<Lifetime, T> action)  
}
```

initial value



PROPERTIES

presenting **viewable** pattern

```
interface ISink<T> {  
    void Advise(Lifetime lf, Action<T> handler)  
}
```

```
interface IViewableProperty<T> : ISink<T> {  
    T Value {get; set}  
  
    void View(Lifetime lf,  
              Action<Lifetime, T> action)  
}
```

initial value



PROPERTIES

presenting **viewable** pattern


```
interface ISink<T> {  
    void Advise(Lifetime lf, Action<T> handler)  
}
```

```
interface IViewableProperty<T> : ISink<T> {  
    T Value {get; set}  
  
    void View(Lifetime lf,  
              Action<Lifetime, T> action)  
}
```

initial value



PROPERTIES

presenting **viewable** pattern

```
interface ISink<T> {  
    void Advise(Lifetime lf, Action<T> handler)  
}
```

```
interface IViewableProperty<T> : ISink<T> {  
    T Value {get; set}  
  
    void View(Lifetime lf,  
              Action<Lifetime, T> action)  
}
```

initial value



PROPERTIES

presenting **viewable** pattern

```
enum AddUpdateRemove {Add, Update, Remove}
```

```
class MapEvent<K,V>{  
    AddUpdateRemove kind;  
    K key;  
    V value;  
}
```

```
interface IViewableMap<K,V>  
    : IDictionary<K,V>,  
    ISource<MapEvent<K,V>> {  
  
    ISource<MapEvent<K,V>> Change;  
}
```

MAPS

reactive collections


```
enum AddUpdate {Add, Remove}
```

```
class SetEvent<T>{  
    AddUpdate kind;  
    T item;  
}
```

```
interface IViewableSet<T>  
    : ISet<T>  
    ISource<SetEvent<T>> {  
}
```

SETS

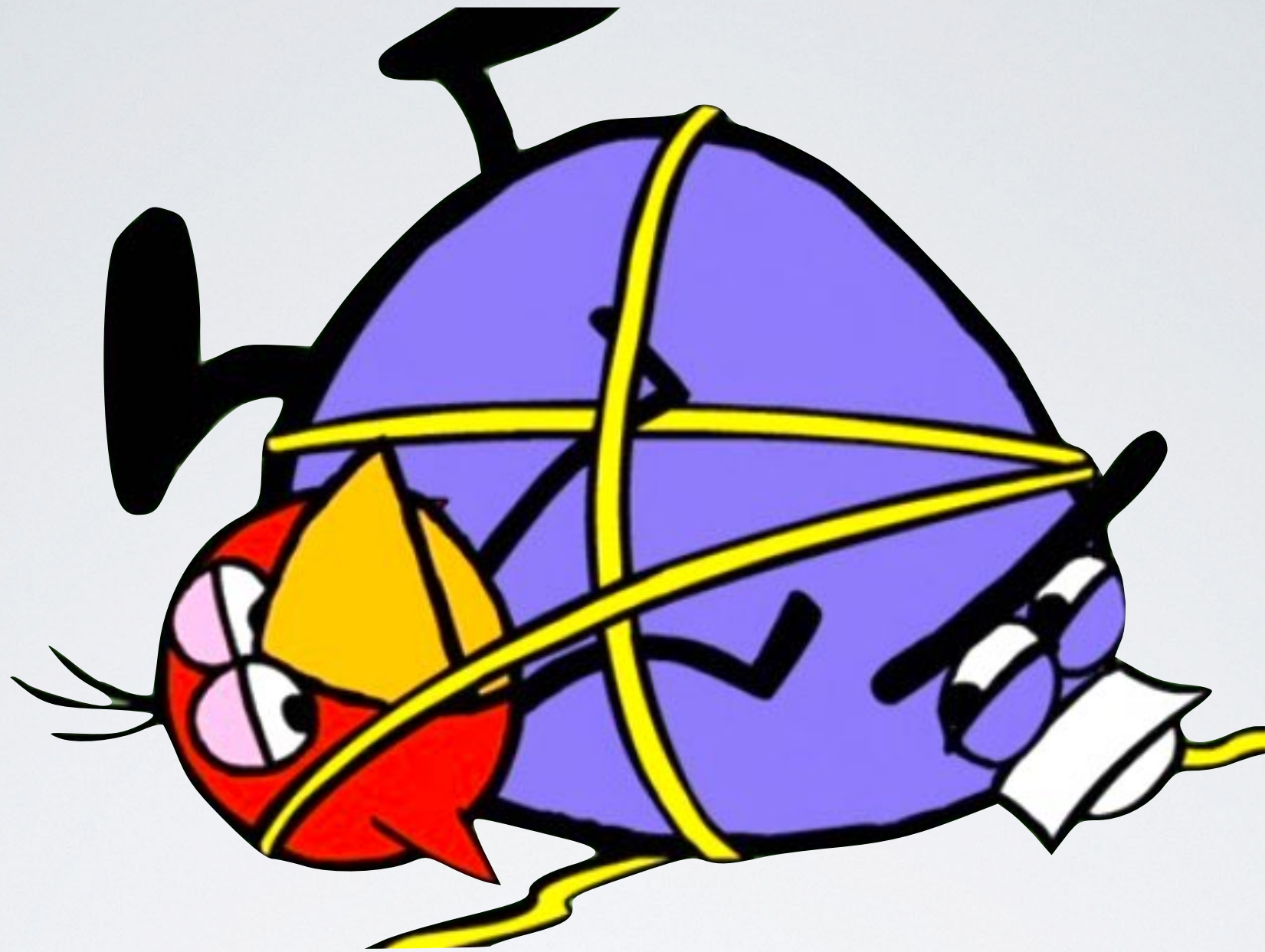
reactive collections

```
class ListEvent<T>{  
    AddUpdateRemove kind;  
    int index;  
    [CanBeNull] V oldValue;  
    [CanBeNull] V newValue;  
}
```

```
interface IViewableList<T>  
    : IList<T>,  
    ISource<ListEvent<T>> {  
}
```

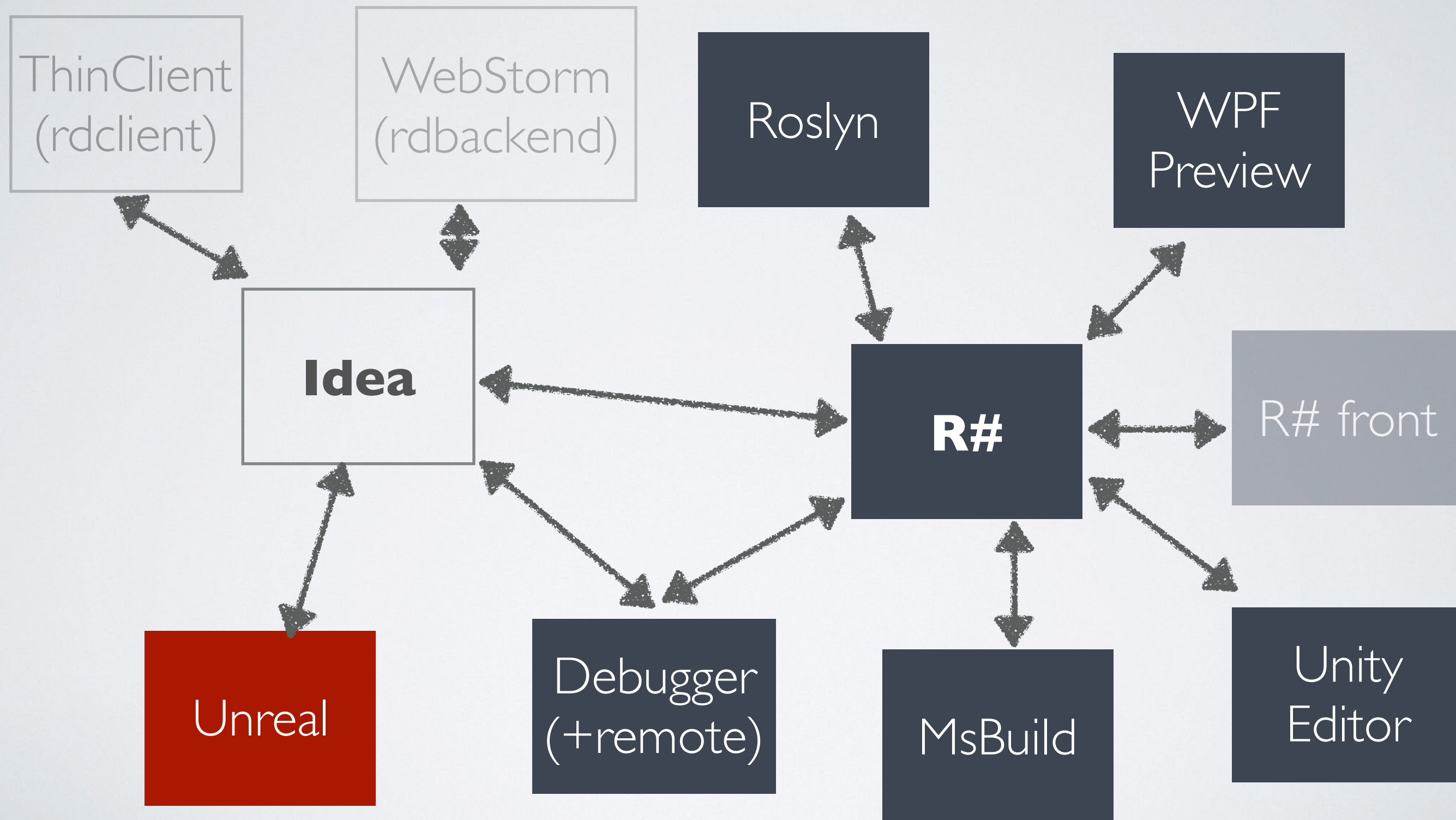
LIST

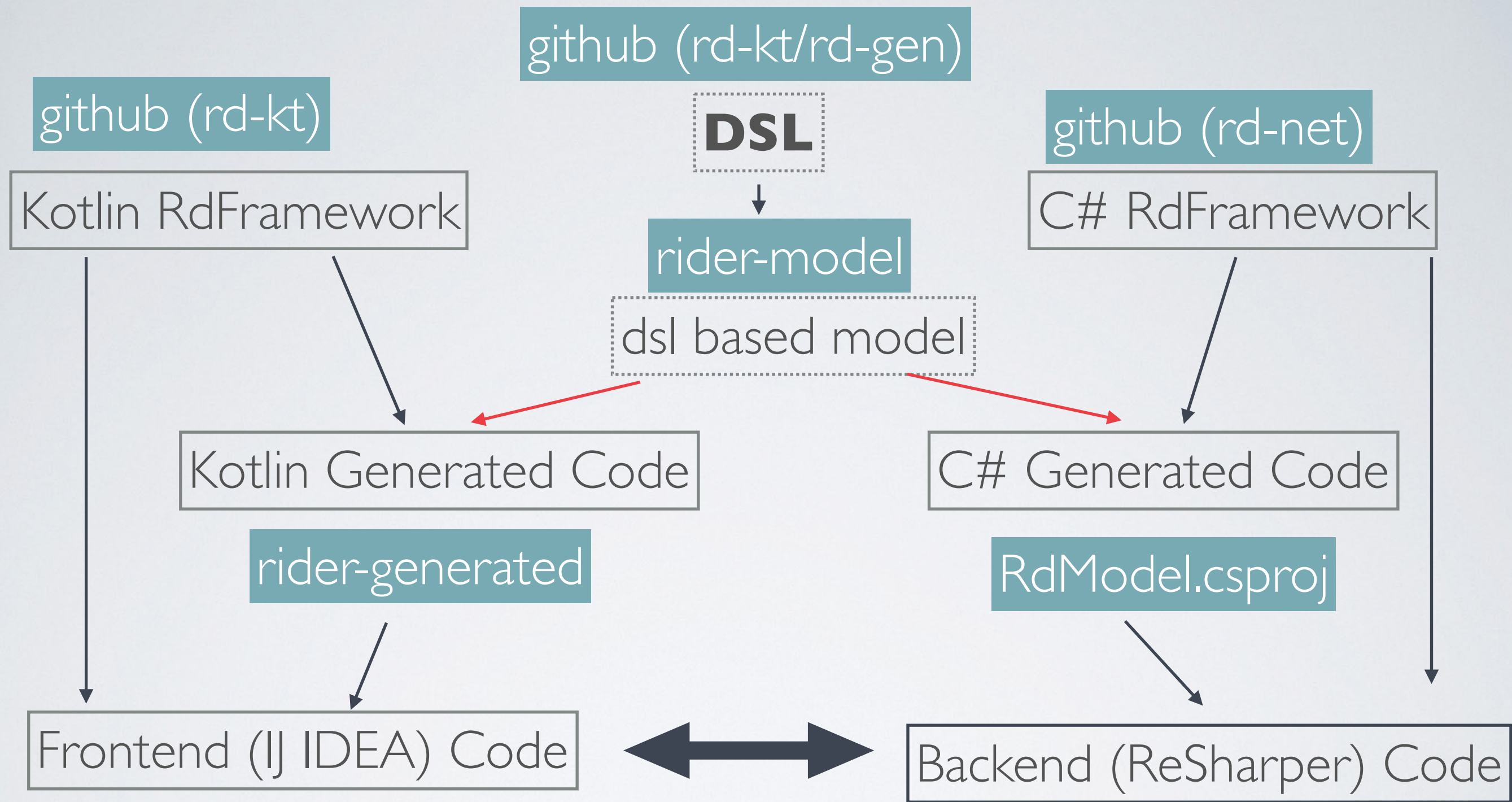
reactive collections



BINDING JVM & CLR

REAL WORLD





RIDER FRAMEWORK

GITHUB

<https://confluence.jetbrains.com/display/ReSharperInt/Make+changes+in+Rd+Framework>

build
build.gradle

docs

gradle

rd-cpp

rd-kt

rd-net

settings.gradle

versions.gradle

rd-core

rd-framework

rd-gen

rd-swing

rd-text

core library

networking

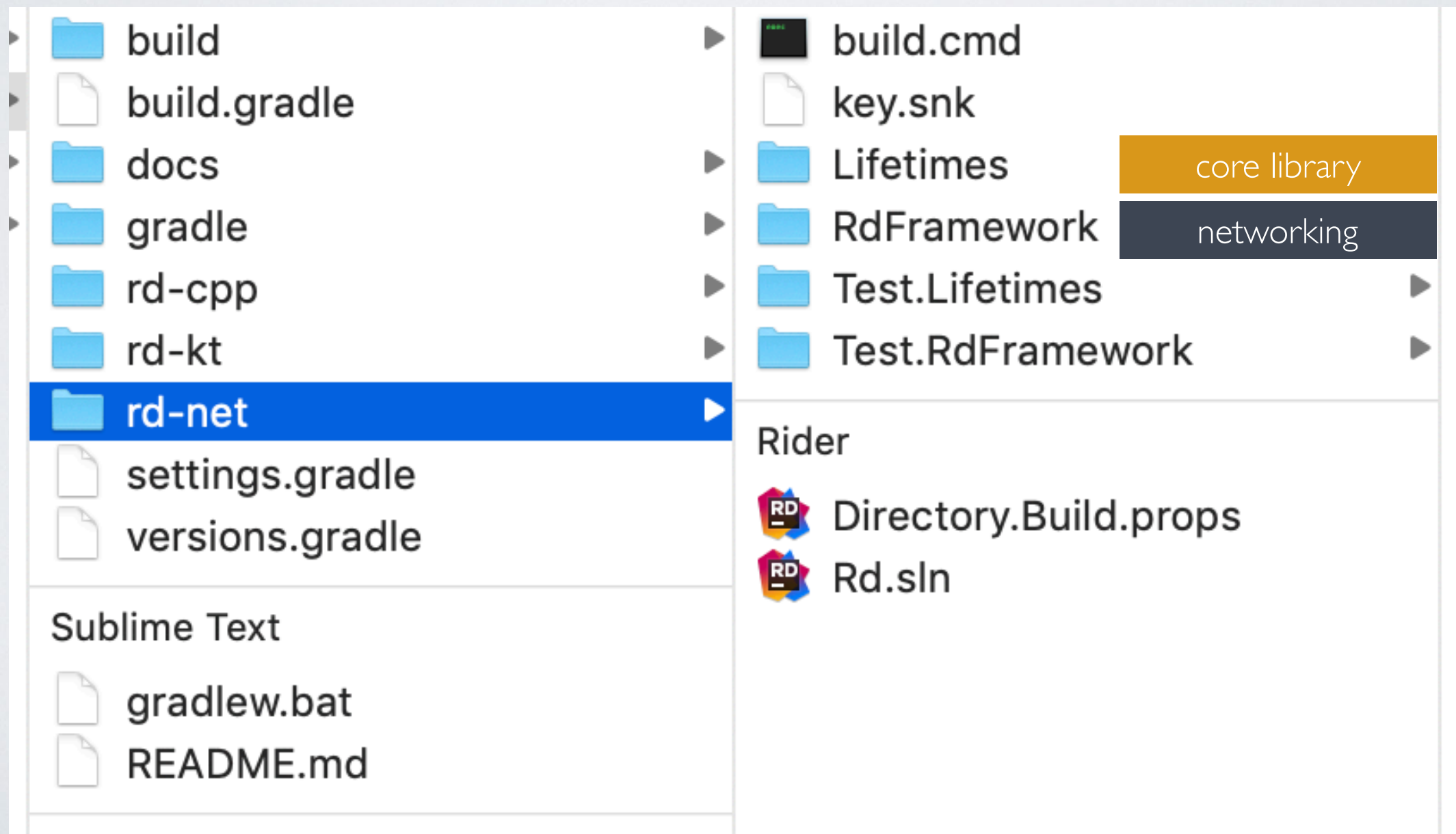
dsl + generator

swing on properties

OT for textbuffer

GITHUB

<https://confluence.jetbrains.com/display/ReSharperInt/Make+changes+in+Rd+Framework>



```
object Solution : Ext(IdeRoot) {  
  init {  
    map("editors", string, classdef("Editor") {  
      })  
    }  
}
```

```
object Solution : Ext(IdeRoot) {  
  init {  
    map("editors", string, classdef("Editor") {  
      list("document", char)  
    })  
  }  
}
```



```
object Solution : Ext(IdeRoot) {  
  init {  
    map("editors", string, classdef("Editor") {  
      list("document", char)  
      property("caret", int)  
    })  
  }  
}
```

```
object Solution : Ext(IdeRoot) {  
  init {  
    map("editors", string, classdef("Editor") {  
      list("document", char)  
      property("caret", int)  
      map("highlighters", Range, Highlighter)  
    })  
  }  
  
  val Range = structdef("Range") {  
    field("start", int)  
    field("length", int)  
  }  
  
  val Highlighter = classdef("Highlighter") {  
    field("attributes", immutableList(string))  
  }  
}
```

```
object Solution : Ext(IdeRoot) {  
  init {  
    map("editors", string, classdef("Editor") {  
      list("document", char)  
      property("caret", int)  
      map("highlighters", Range, Highlighter)  
      property("completion", Completion.nullable)  
    })  
  }  
  
  val Range = structdef("Range") {  
    field("start", int)  
    field("length", int)  
  }  
  
  val Highlighter = classdef("Highlighter") {  
    immutableList(attributes)  
  }  
}
```



```
object Solution : Ext(IdeRoot) {  
  init {  
    map("editors", string, classdef("Editor") {  
      list("document", char)  
      property("caret", int)  
      map("highlighters", Range, Highlighter)  
      property("completion", Completion.nullable)  
    })  
    source("build", void)  
  }  
  
  val Range = structdef("Range") {  
    field("start", int)  
    field("length", int)  
  }  
  
  val Highlighter = classdef("Highlighter") {  
    ...  
  }  
}
```

Primitive types:

- `int8, int16, int32, int64`
- `uint(s)`
- `float, double`
- `char, boolean, void`
- `string`
- `secureString`
- `byteArray, intArray, doubleArray ...`
- `guid, uri, dateTime`

BUILDING BLOCKS

of RdProtocol

Scalar types: (could be used in signals, calls)

- enums (+flags)
- structdef (abstract/sealed) – can contain fields of scalars

BUILDING BLOCKS

of RdProtocol

Bindable types:

- `classdef` (abstract/sealed)
- can contain:
 - signals
 - properties
 - lists, sets, maps
 - calls (*+ lifetime per call*)
 - fields of scalars/bindable

BUILDING BLOCKS

of RdProtocol

Combinators :

- `immutableList(type)`
- `array(type)`
- `type.nullable`

BUILDING BLOCKS

of RdProtocol

Protocol (ctor) :

- Serializers (+ IPolymorphicTypeCatalog)
- ~~Identities~~
- Scheduler
 - RdDispatcher
 - SingleThreadScheduler
 - SynchronousScheduler
- Wire (Local, Socket.Server, Socket.Client)

HOW TO START

QUESTIONS AND ANSWERS

thanks for your attention

dmitry.ivanov@jetbrains.com