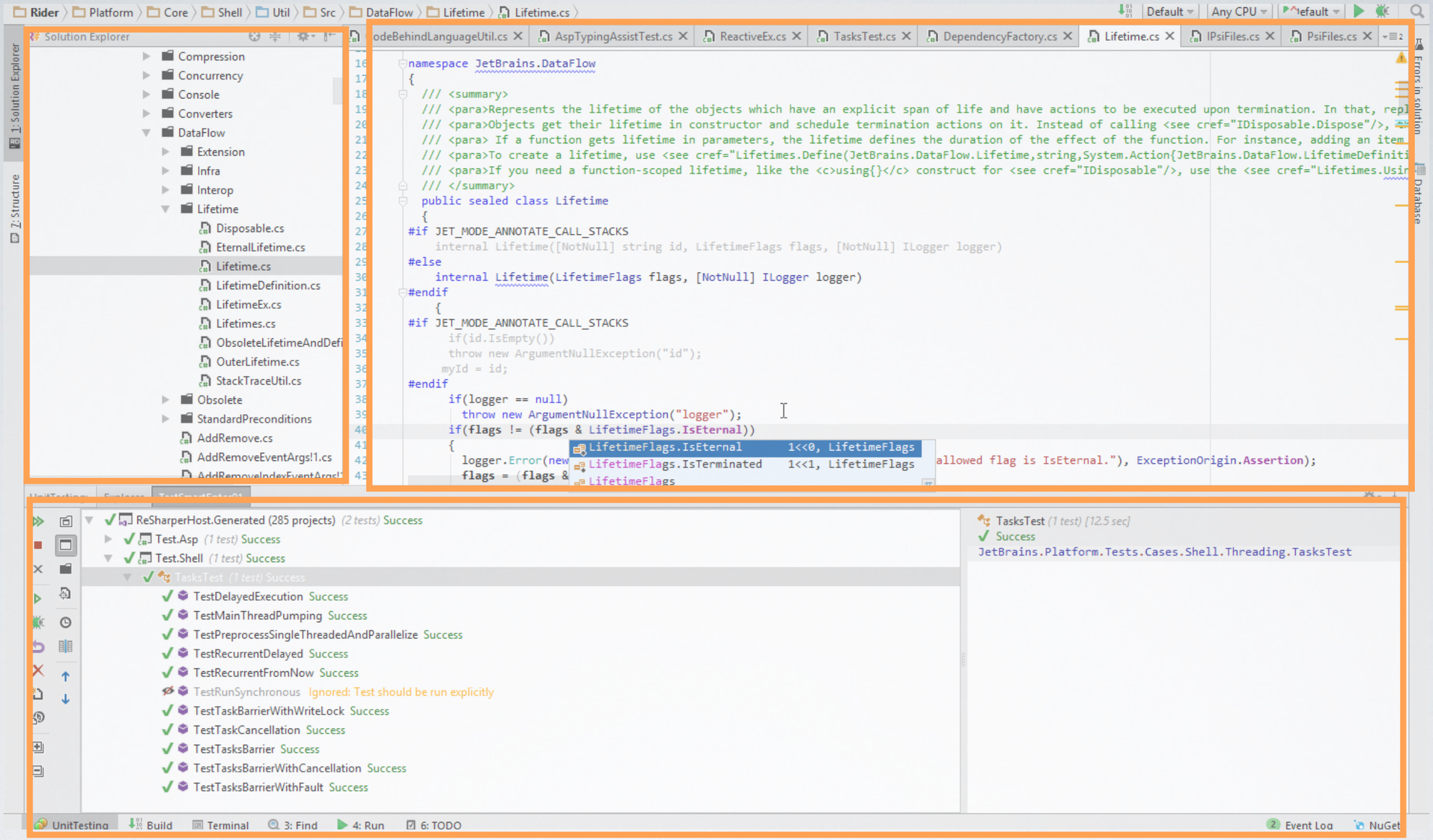
# RD FRAMEWORK

Reactive distributed cross-platform communication

Dmitry Ivanov, JetBrains

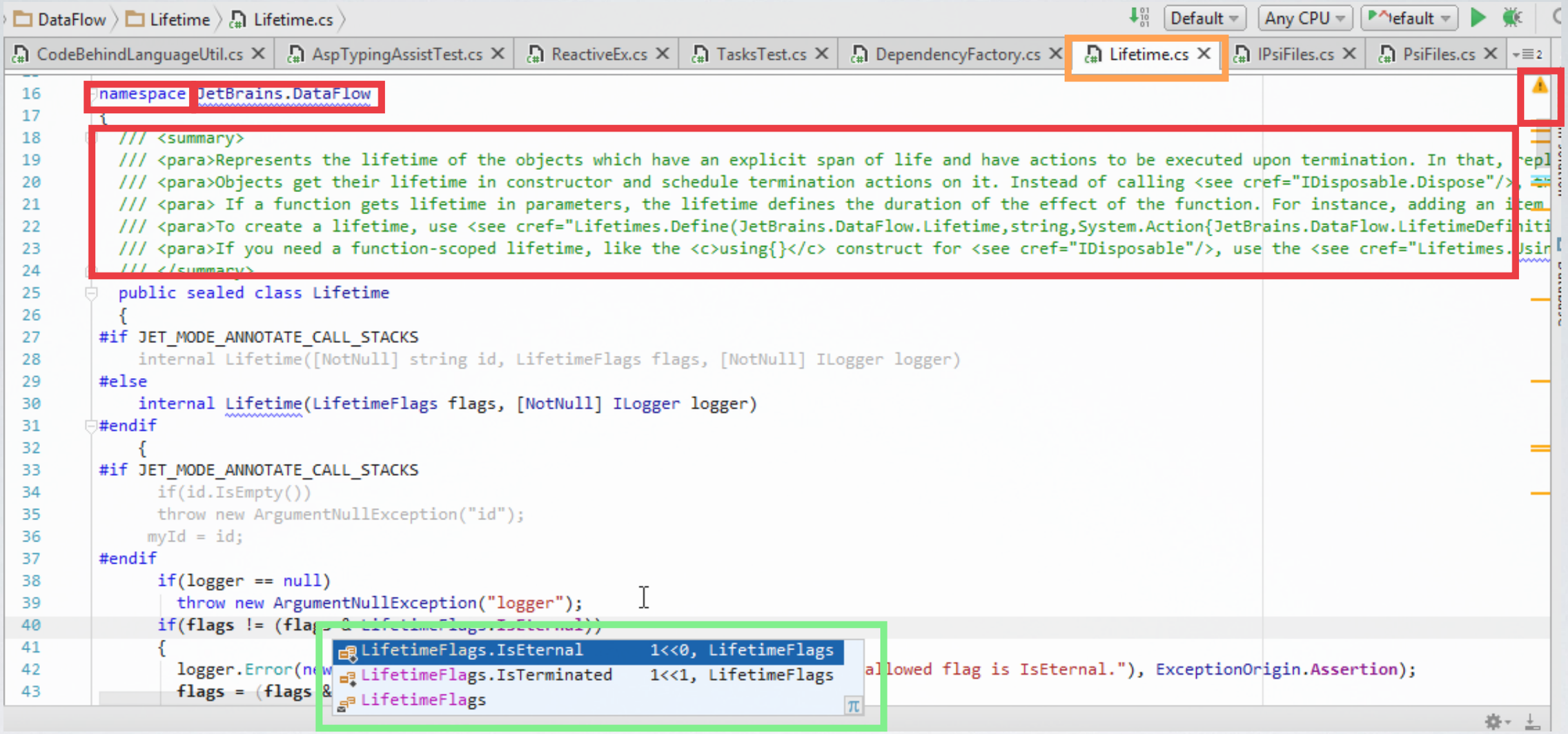

RD
JetBrains.RdFramework 2019.3.0
JetBrains Networking library for reactive distributed communication

# NUGET



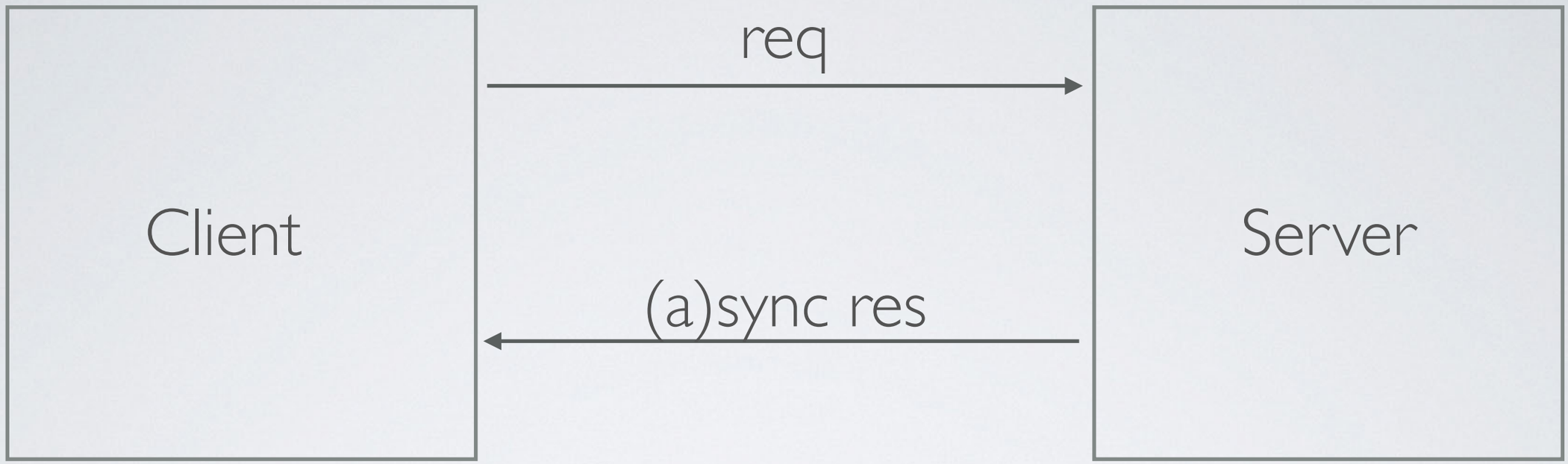

# INCEPTION

## Hierarchical nature of view model



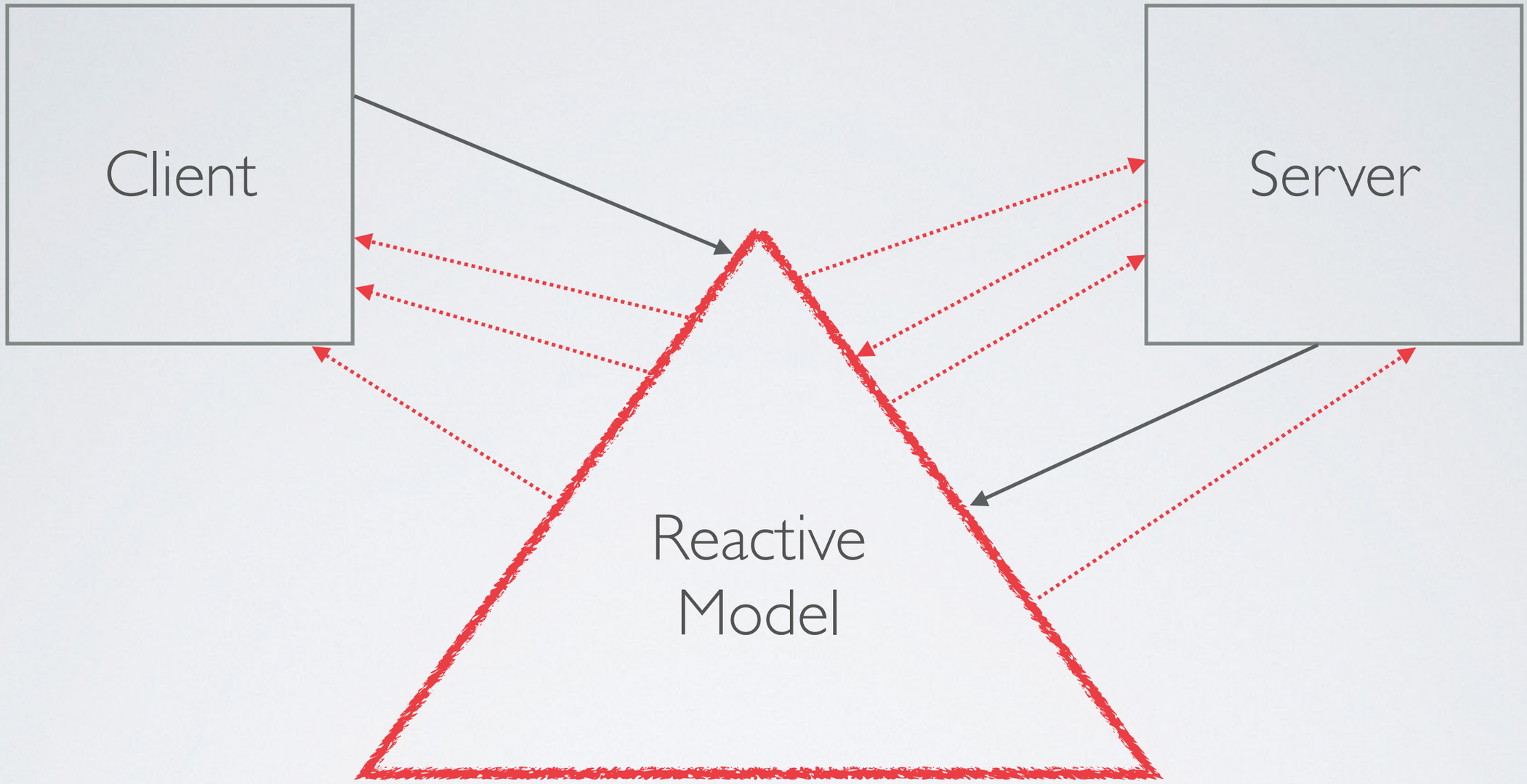

# INCEPTION

## Hierarchical nature of view model




# INCEPTION

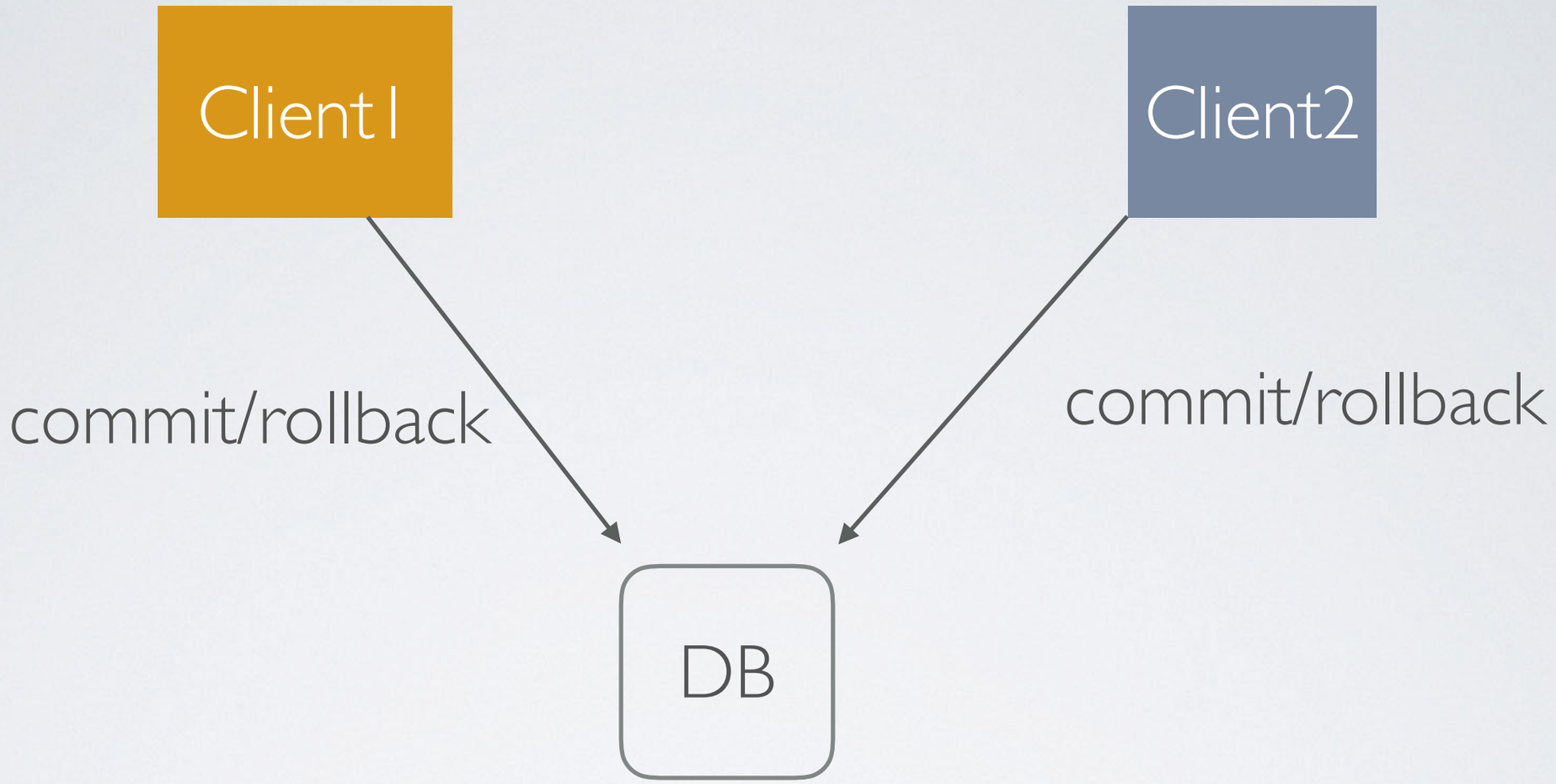
Why not RPC



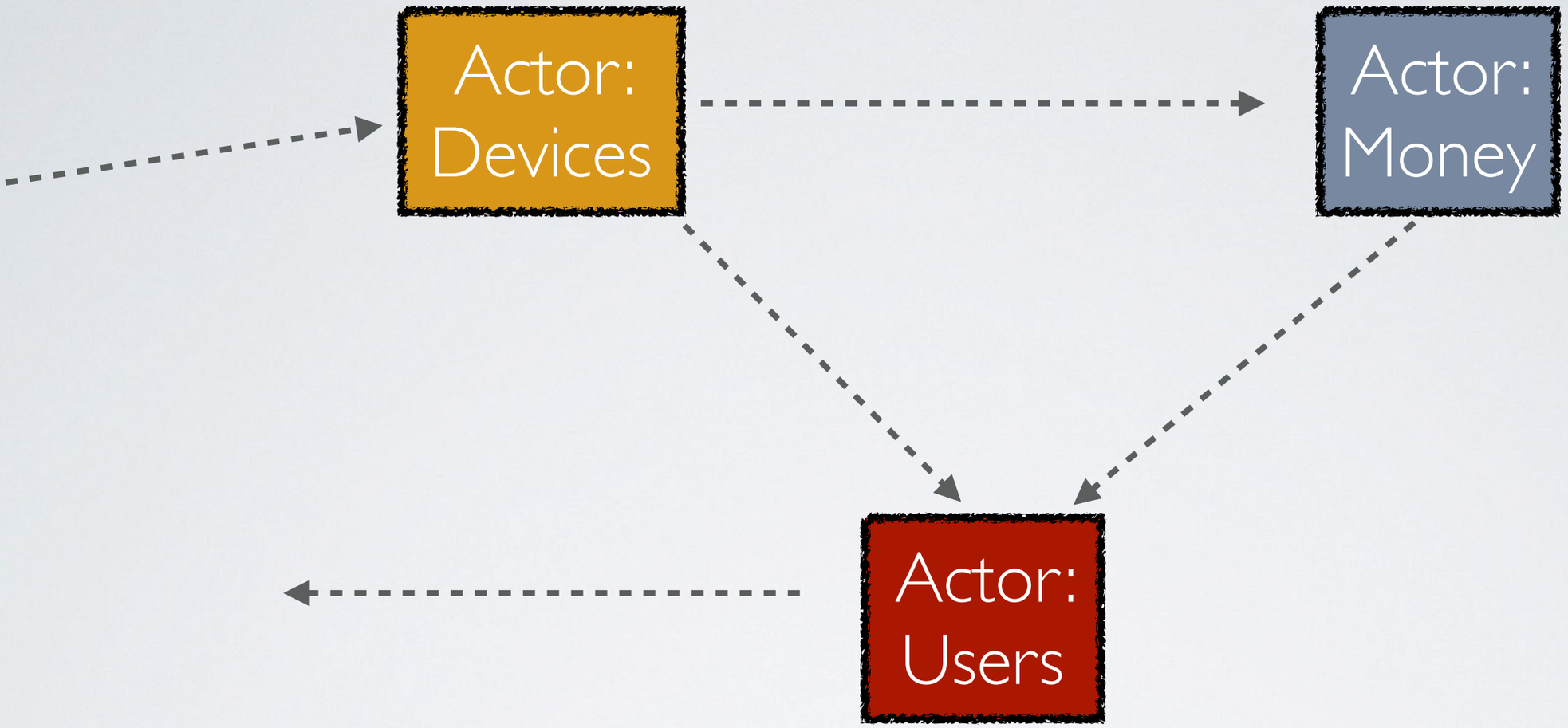

# INCEPTION

Because it's not **Reactive**!

# OTHER MODELS




# TRANSACTIONAL DATABASE




# PERSISTENT ACTORS

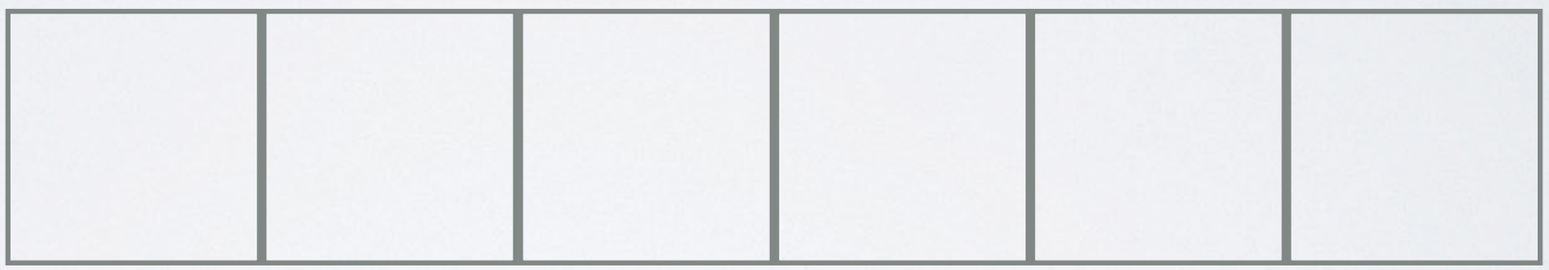
Frontend

Language Server

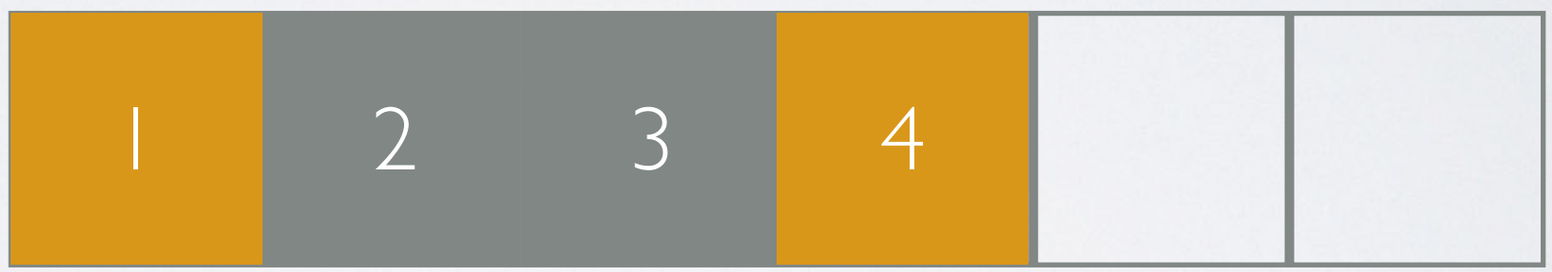
# EVENT SOURCING

Frontend

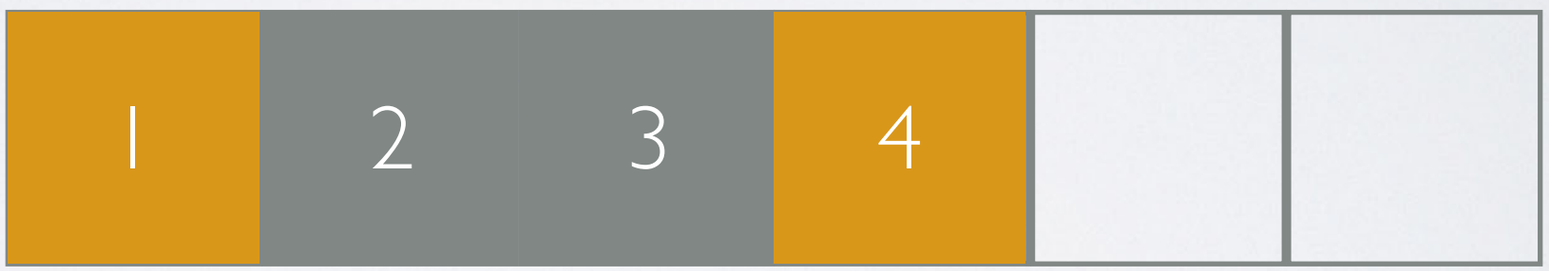
Language Server



# EVENT SOURCING

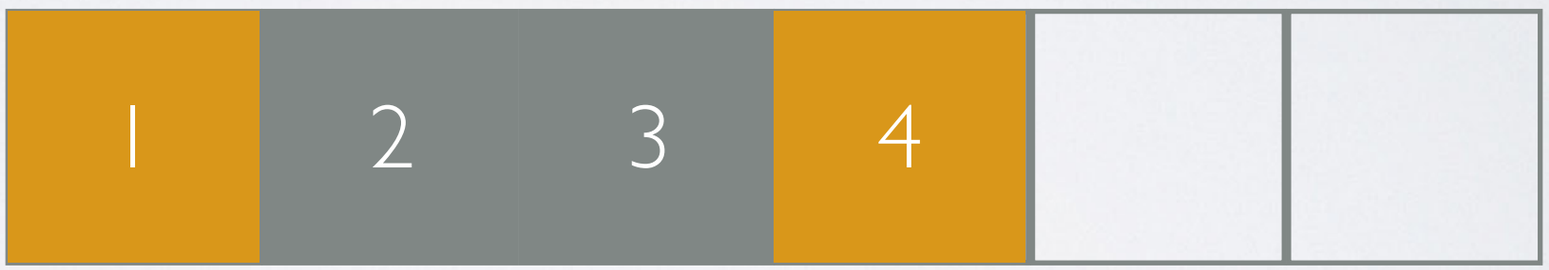




# EVENT SOURCING

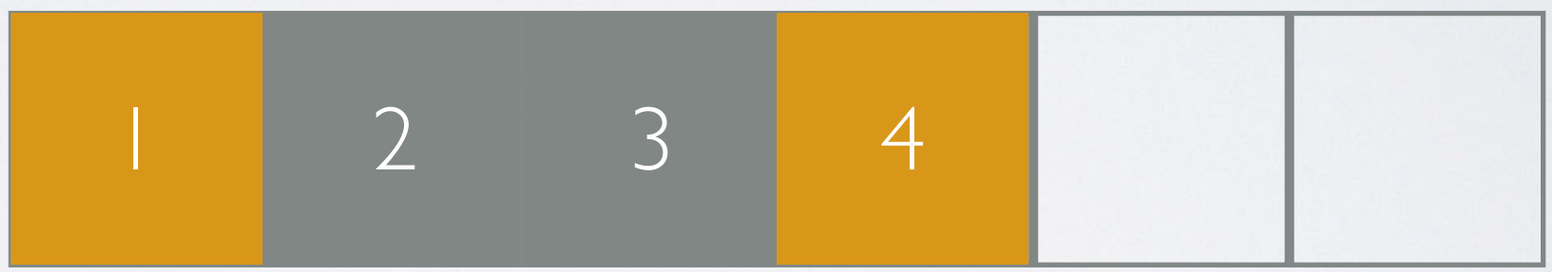




# EVENT SOURCING

Frontend 2 0 Language Server




# EVENT SOURCING

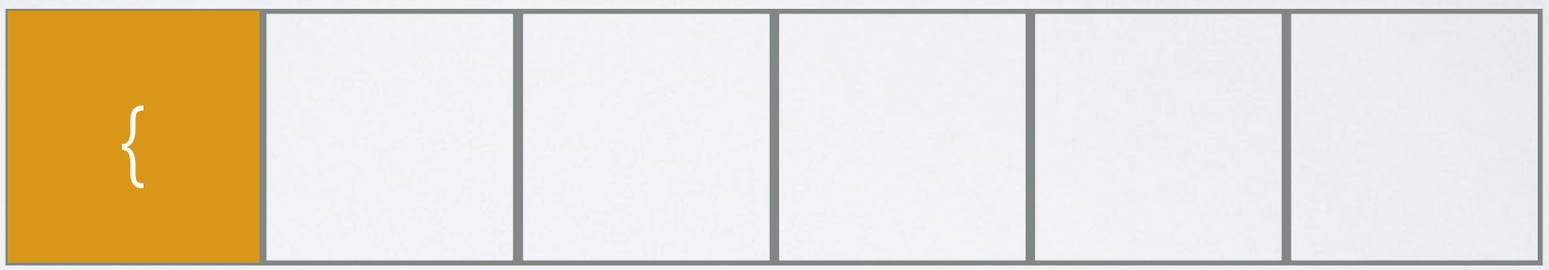




# EVENT SOURCING

Frontend

{

Language Server

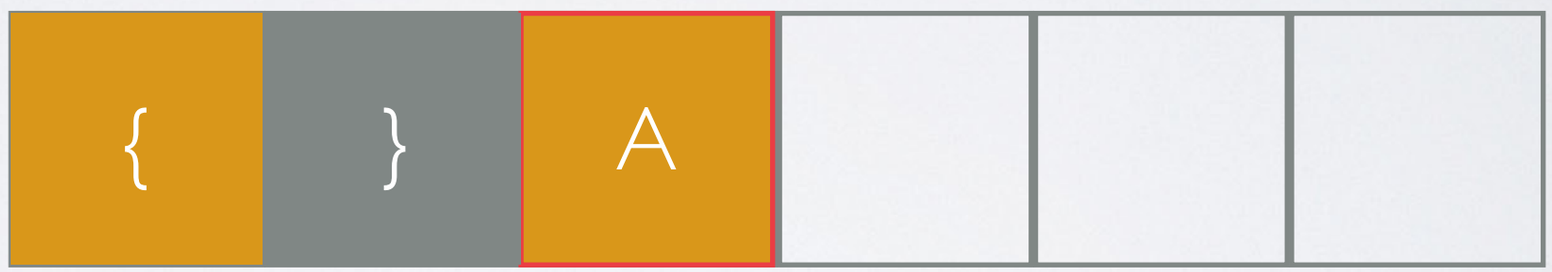



# EVENT SOURCING

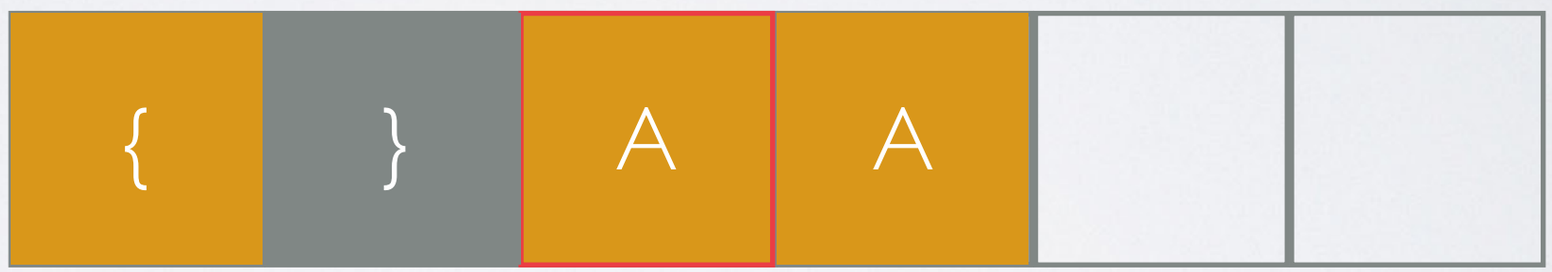

# EVENT SOURCING



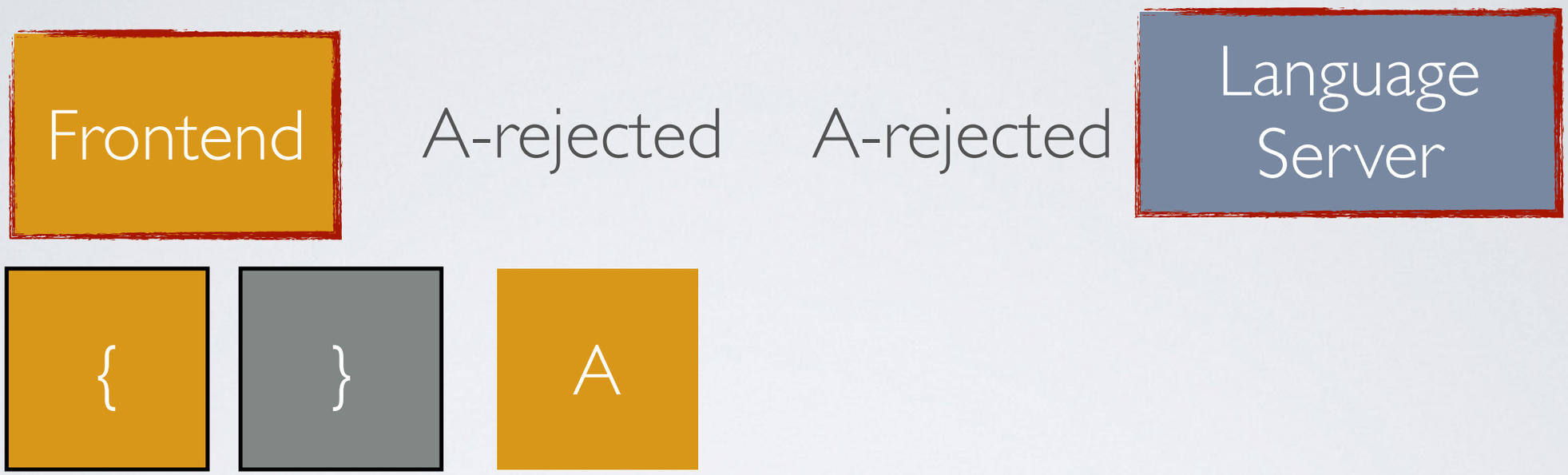
# EVENT SOURCING

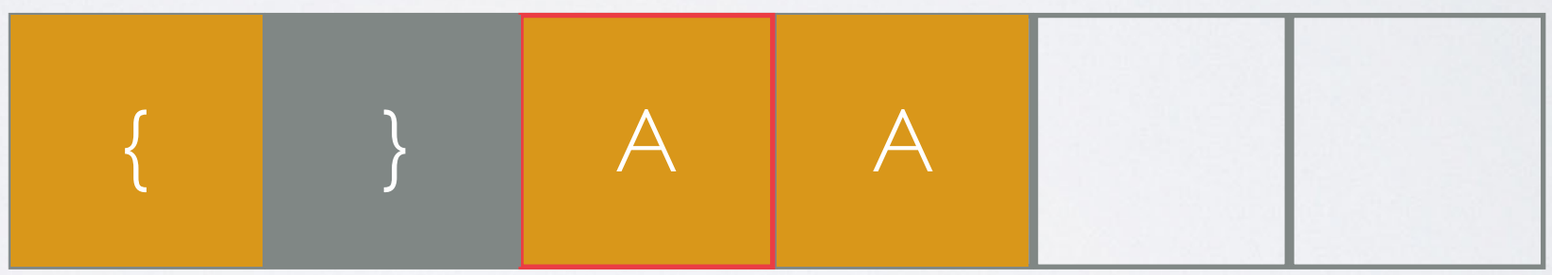




# EVENT SOURCING

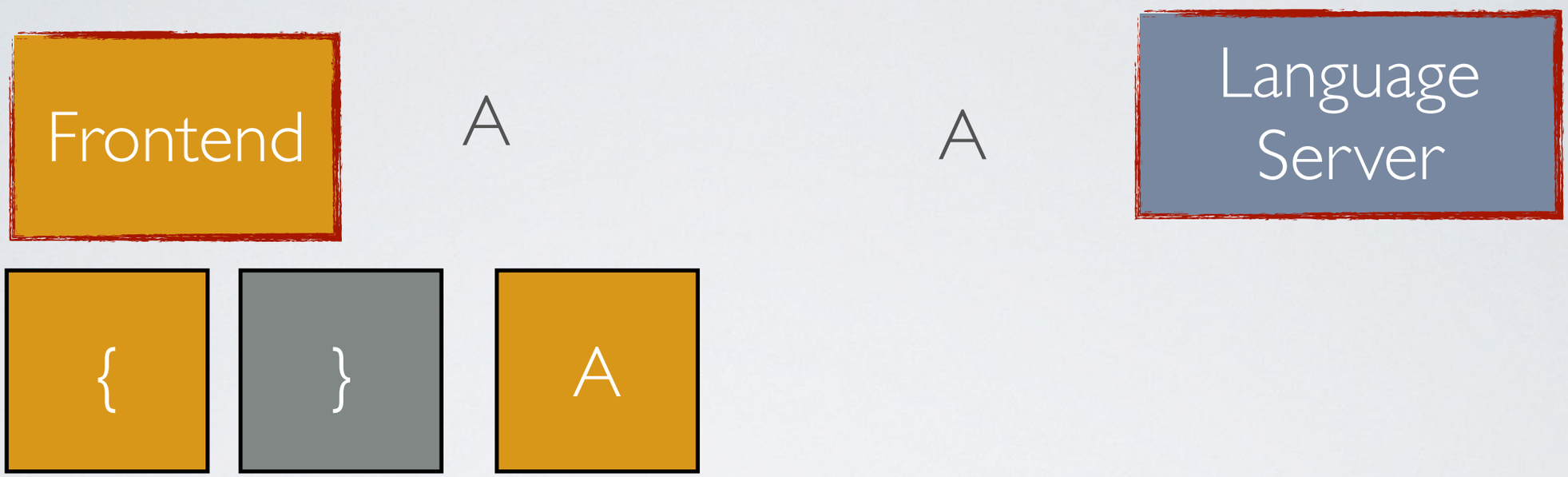




# EVENT SOURCING




# 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

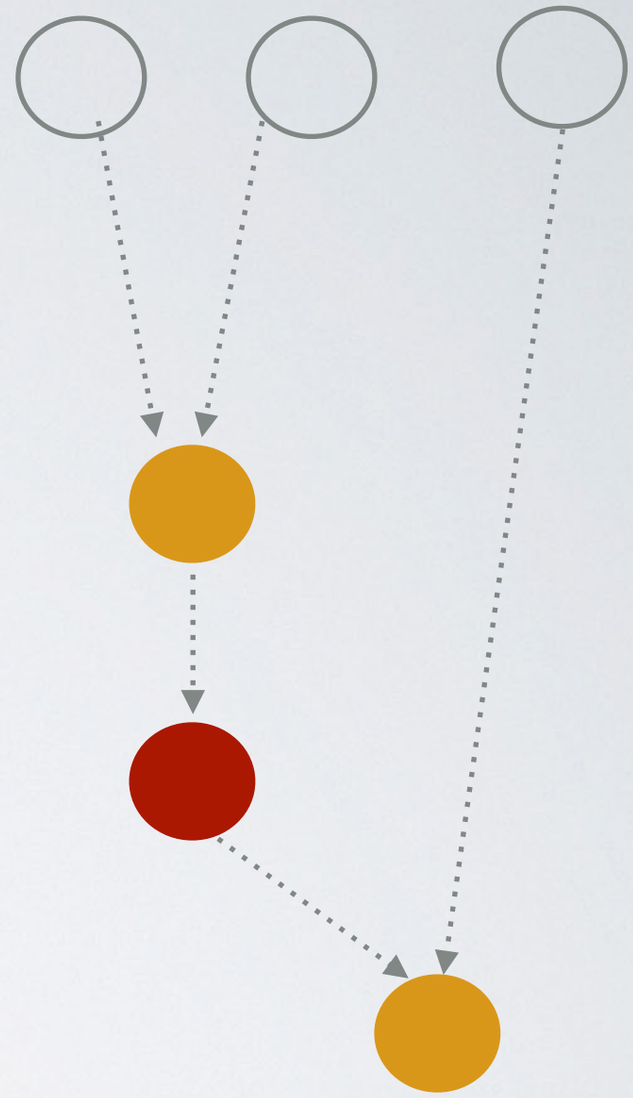
```
interface ISignal<Unit> {
    void Advise(Lifetime lf, Action handler)
}
```

# 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

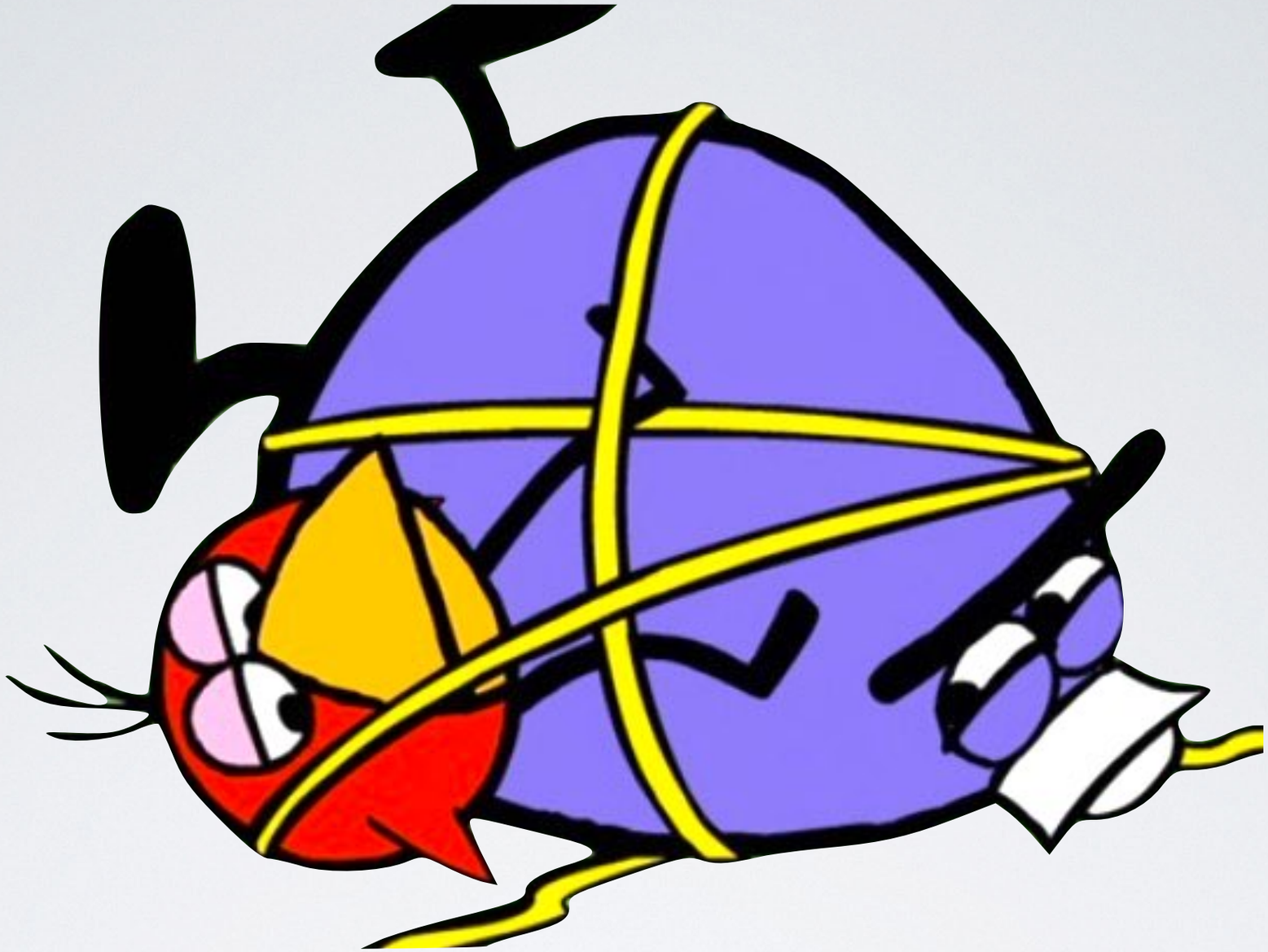
reactive collections

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

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

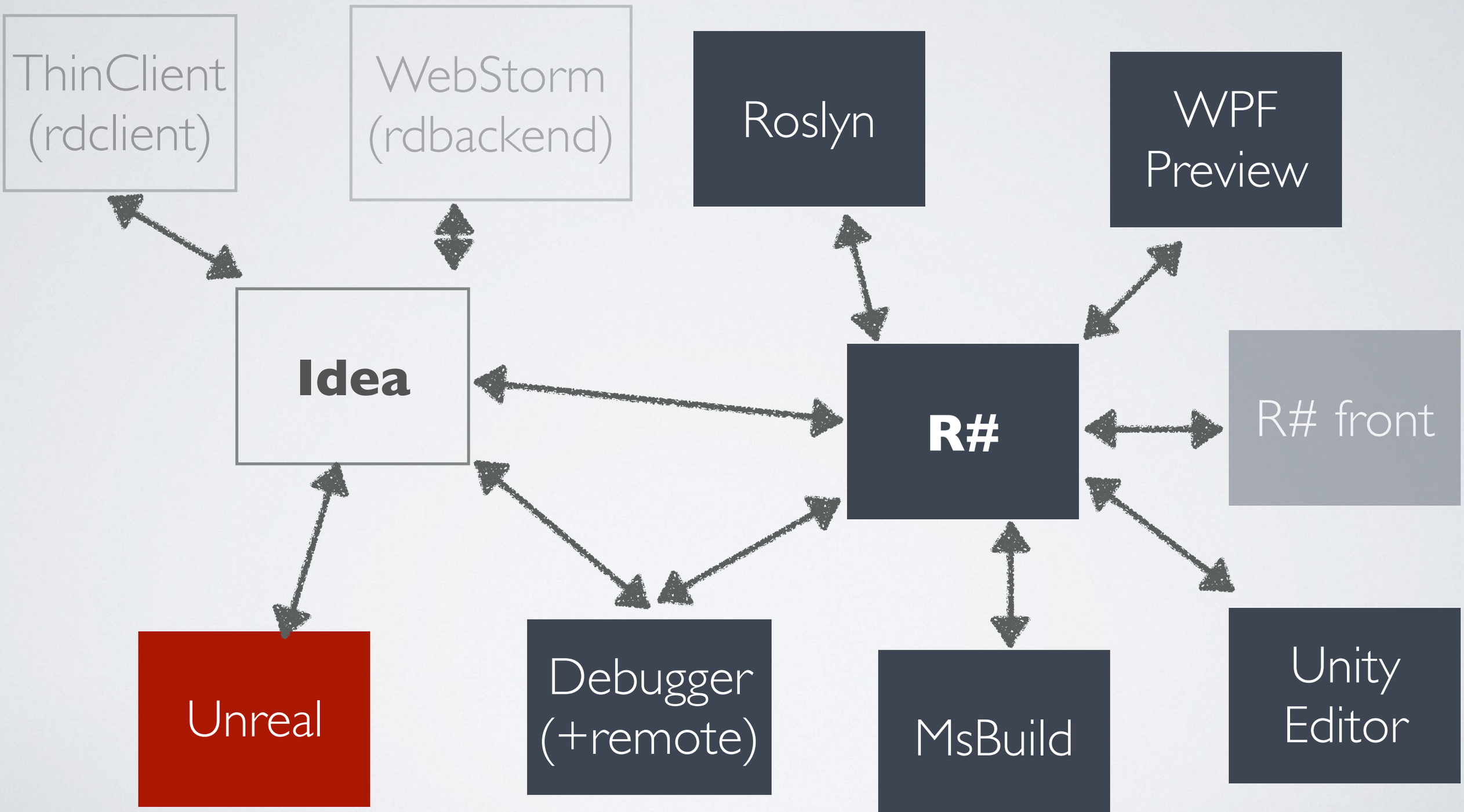
# LIST

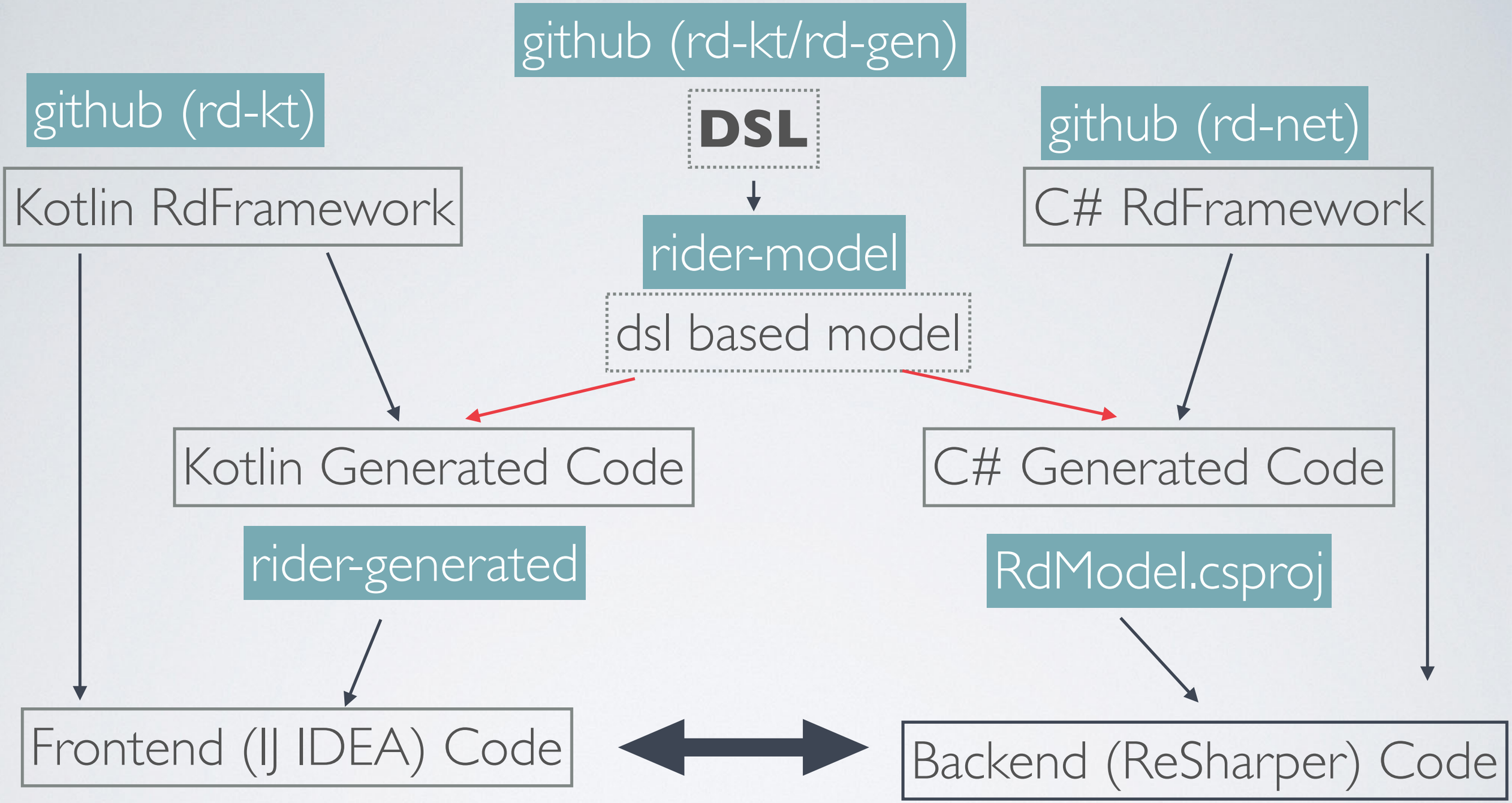
reactive collections



# BINDING JVM & CLR

# REAL WORLD

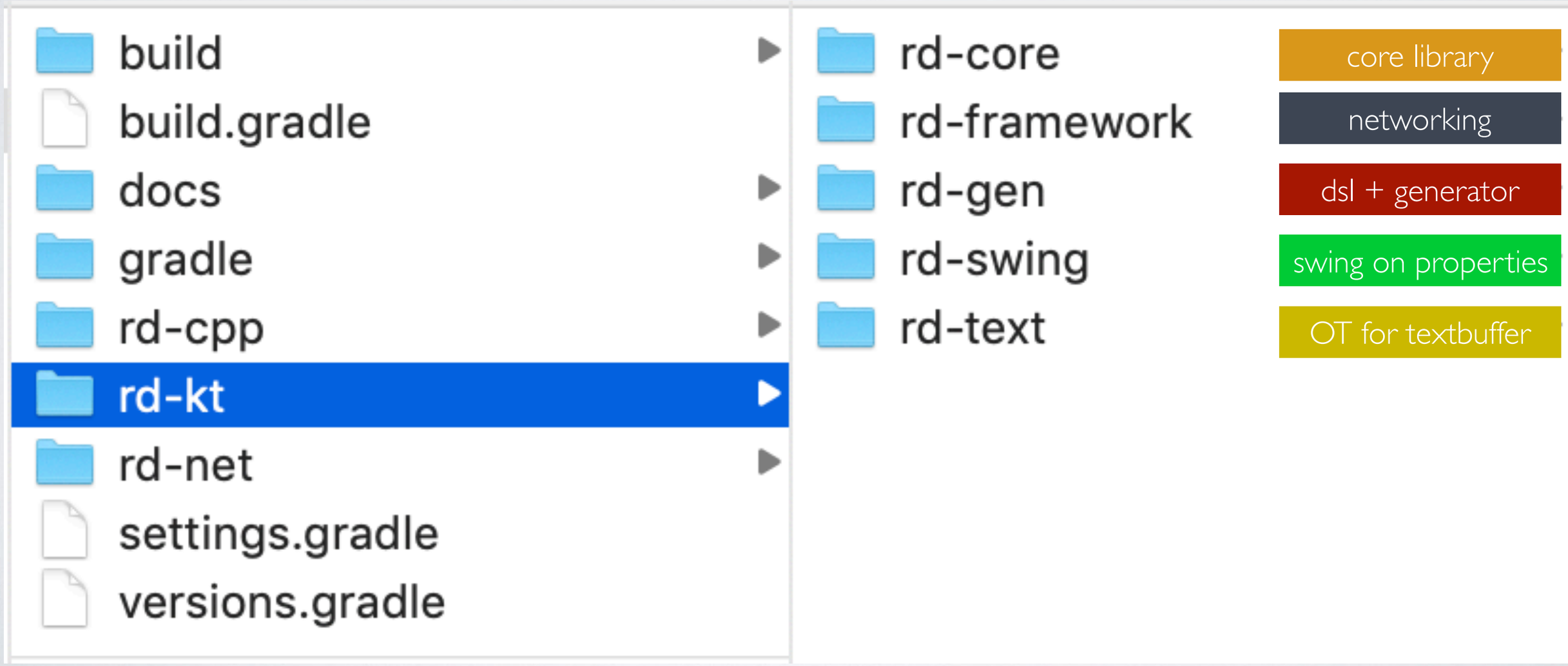

ThinClient
(rdclient)
WebStorm
(rdbackend)
Roslyn
WPF
Preview
Idea
R#
R# front
Unreal
Debugger
(+remote)
MsBuild
Unity
Editor




# RIDER FRAMEWORK

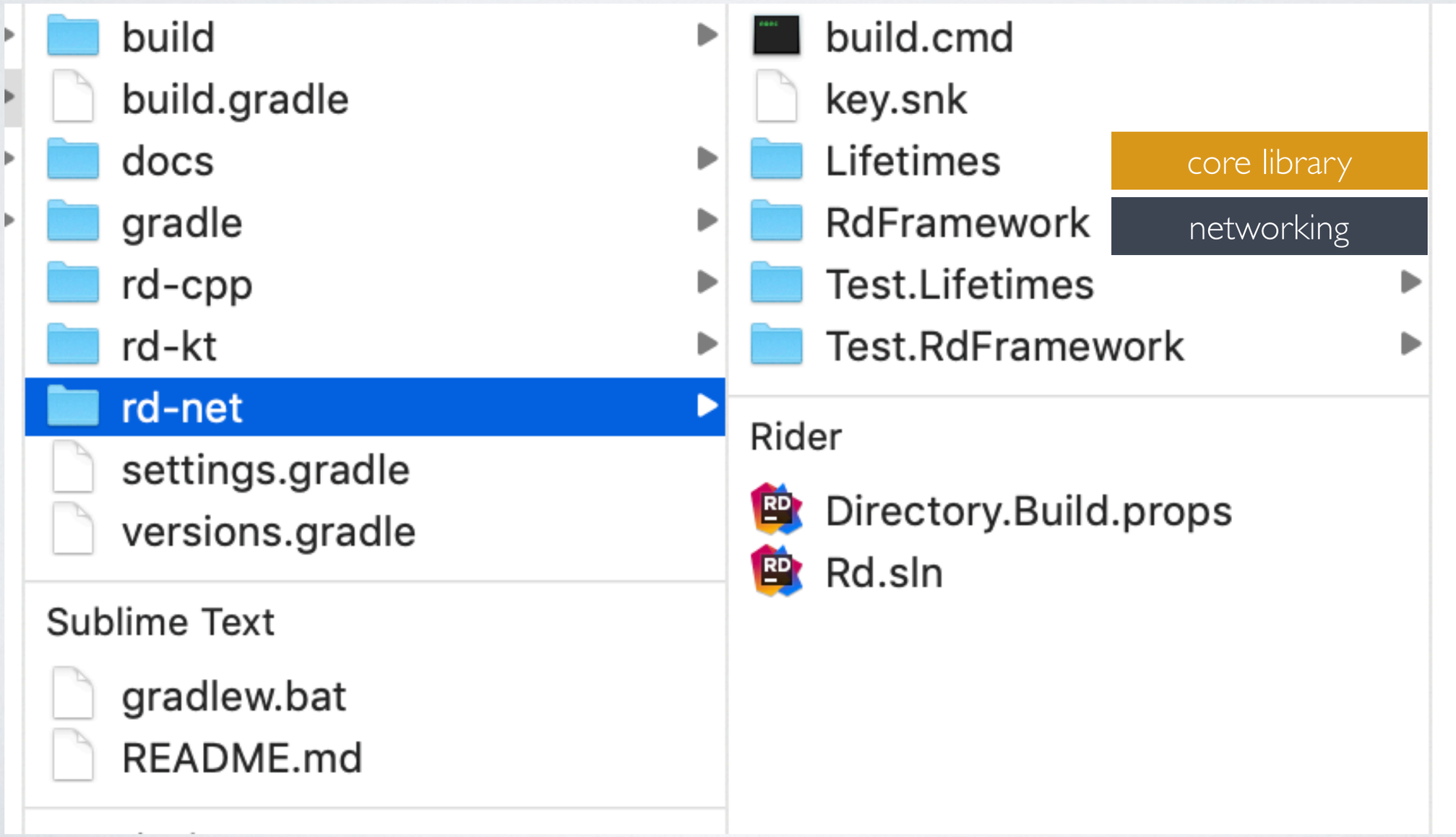
# GITHUB

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



# 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