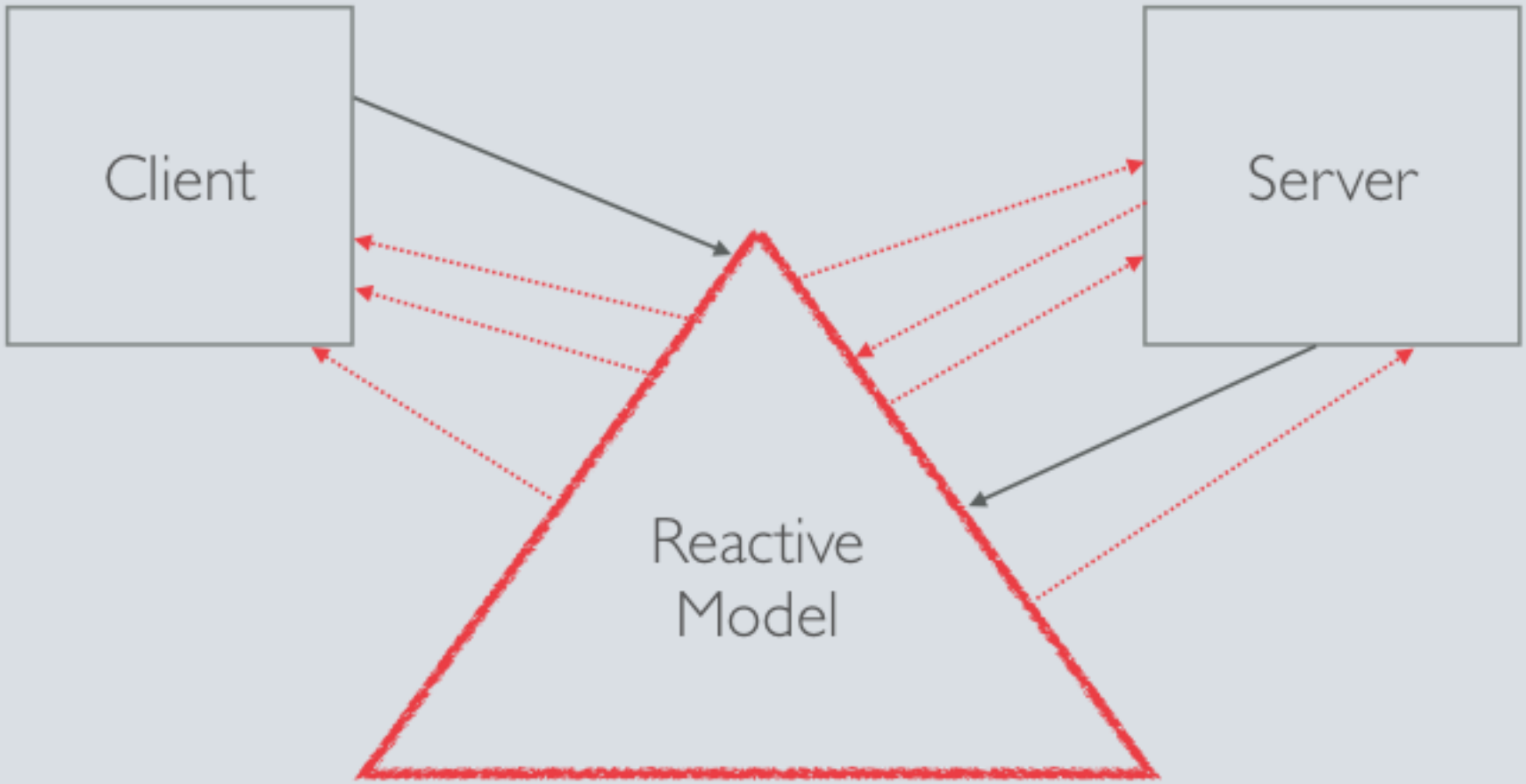
REACTIVE MULTIPROCESSING

JetBrains Rider Framework

Dmitry Ivanov, JetBrains

INCEPTION

Why not RPC



INCEPTION

Because it's not **Reactive**!

FOUNDATIONS

of reactive approach

```
class A: IDisposable {

    Foo foo = new Foo();
    Bar bar = new Bar(foo);

    void Dispose() {
      someAction();
      bar.Dispose();
      foo.Dispose();
    }
}
```

INVERSION OF CONTROL

is not just about component container

```
class Lifetime {
  void Add(Action action)
  void Terminate()
}
```

INVERSION OF CONTROL

presenting **lifetime** pattern

```
class Lifetime {
  static LifetimeDef Define()

  void Add(Action action)
}
```

```
class LifetimeDef {
  Lifetime Lifetime
  void Terminate()
}
```

INVERSION OF CONTROL

presenting **lifetime** pattern

```
class Lifetime {
  static LifetimeDef Define(
    Lifetime parent
  )
  static Lifetime Eternal

  void Add(Action action)
}
```

```
class LifetimeDef {
  Lifetime Lifetime
  void Terminate()
}
```

INVERSION OF CONTROL

presenting **lifetime** pattern

```
class Lifetime {
  static LifetimeDef Define(
    Lifetime parent
  )
  static Lifetime Eternal

  void Add(Action action)
  bool IsTerminated()

  Lifetime intersect(
    Lifetime other
  )
}
```

```
class LifetimeDef {
  Lifetime Lifetime
  void Terminate()
}
```

INVERSION OF CONTROL

presenting **lifetime** pattern

```
class A: IDisposable {

    Foo foo = new Foo();
    Bar bar = new Bar(foo);

    void Dispose() {
      someAction();
      bar.Dispose();
      foo.Dispose();
    }
}
```

INVERSION OF CONTROL

disposable

```
class A {
    LifetimeDef def =
    Lifetime.Define(Lifetime.Eternal);

    Foo foo = new Foo(def.Lifetime);
    Bar bar = new Bar(def.Lifetime, foo);

    A() {
      def.Lifetime.Add(someAction);
    }
}
```

INVERSION OF CONTROL

…and lifetime

```
[Component]
```

```
[Component]
class A {
    A(Lifetime lf, Foo foo, Bar bar) {
      lf.Add(someAction);
    }
}

[Component]
class Bar {
    Bar(Lifetime lf, Foo foo) {
      …
    }
}
```

INVERSION OF CONTROL

alright, this time about component container

REACTIVE PRIMITIVES

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

SIGNALS

transient entities that rule

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

SIGNALS

let's apply lifetime pattern

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


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


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

SIGNALS

separation of concerns

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

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

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

SIGNALS

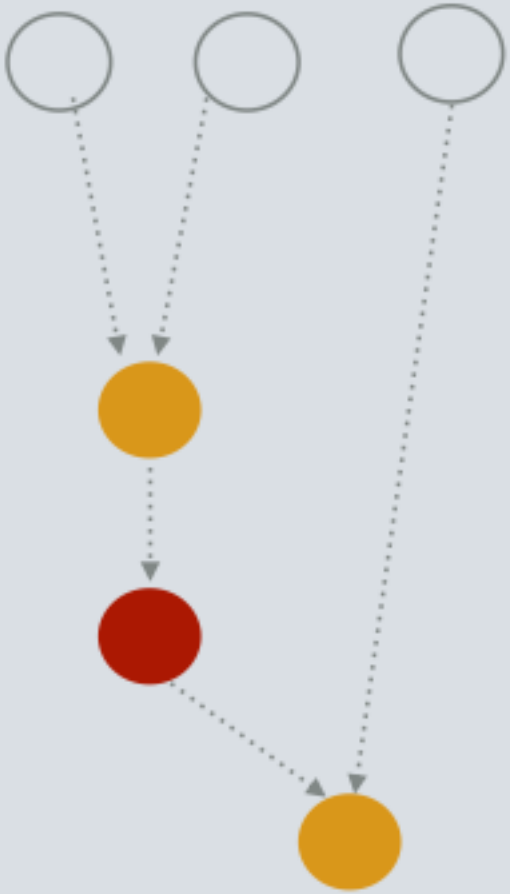
separation of concerns

```
interface IVoidSource {
   void Fire()
}


interface IVoidSink {
   void Advise(Lifetime lf, Action handler)
   void AdviseEternal(Action handler)
}

interface IVoidSignal :
  IVoidSource, IVoidSink
```

SIGNALS

legacy of the void

```
ISink<TRes> compose(
    this ISink<T1> this
    ISink<T2> other,
    Func<T1, T2, TRes> composer
)
```



Full LINQ support

SIGNALS

POWER OVERWHELMING

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

```
    void AdviseEternal(Action<T> handler)
}

interface IProperty<T> : ISink<T> {
    T Value {get; set}
}
```

PROPERTIES

stateful signals

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

```
    void AdviseEternal(Action<T> handler)
}

interface IProperty<T> : ISink<T> {
    T Value {get; set}
    bool HasValue {get;}
}

class Property<T> : IProperty<T> {
    Property() {…}
    Property(T value) {…}
    …
}
```

PROPERTIES

stateful signals

```
class MapEvent<K,V>{
    enum Kind {And, Remove}
```

```
    Kind kind;
    K key;
    V value;
}
interface IViewableMap<K,V>
  : IDictionary<K,V>,
    ISink<MapEvent<K,V>> {

    // advise and go!

   }
```

MAPS

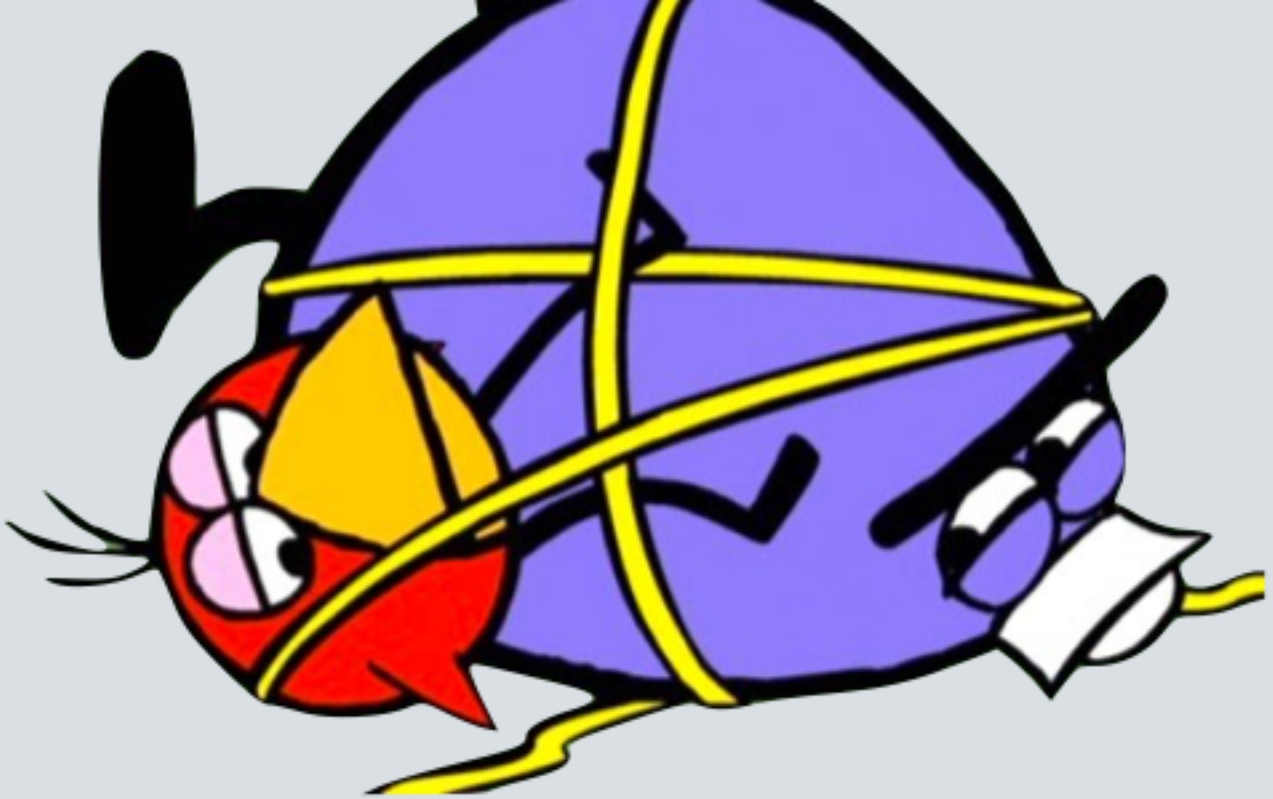
reactive collections

```
class MapEvent<K,V>{
    enum Kind {And, Remove}
```

```
    Kind kind;
    K key;
    V value;
}
interface IViewableMap<K,V>
  : IDictionary<K,V>,
    ISink<MapEvent<K,V>> {

    void View(
            Lifetime lf,
            Action<Lifetime, K, V> action
        )
}
```

MAPS

reactive collections



BINDING JVM & CLR

```
[WebService]
class HelloService {
```

```
    [WebMethod]
    string Hello(Person p) {
      if (p.Money < 10) throw new
        WebException("Access denied");

      if (p.Money < 100)
        return "Hello "+p.Name;

      return "Good day Mr. "+p.Surname;
    }
}
```

Code First

TWO APPROACHES

of building remote applications

Model First

```
   <message name="SayHelloResponse">
      <part name="greeting" type="xsd:string"/>
   </message>

   <portType name="Hello_PortType">
      <operation name="sayHello">
         <input message="tns:SayHelloRequest"/>
         <output message="tns:SayHelloResponse"/>
      </operation>
   </portType>

   <binding name="Hello_Binding" type="tns:Hello_PortType">
      <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="sayHello">
         <soap:operation soapAction="sayHello"/>
         <input>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
               namespace="urn:examples:helloservice"
               use="encoded"/>
         </input>

         <output>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"
               namespace="urn:examples:helloservice"
               use="encoded"/>
         </output>
      </operation>
   </binding>

   <service name="Hello_Service">
      <documentation>WSDL File for HelloService</documentation>
      <port binding="tns:Hello_Binding" name="Hello_Port">
         <soap:address
            location="http://www.examples.com/SayHello/" />
      </port>
   </service>
</definitions>
```

TWO APPROACHES

of building remote applications

```
fun Singleton.classdef (
```

```
    name : String,
    init : ClassdefNode.() -> Unit
  )
```

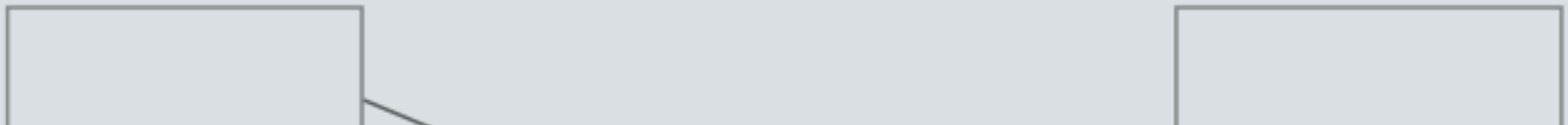
```
classdef (
  “Foo”,
  {body_of_action}
)
```

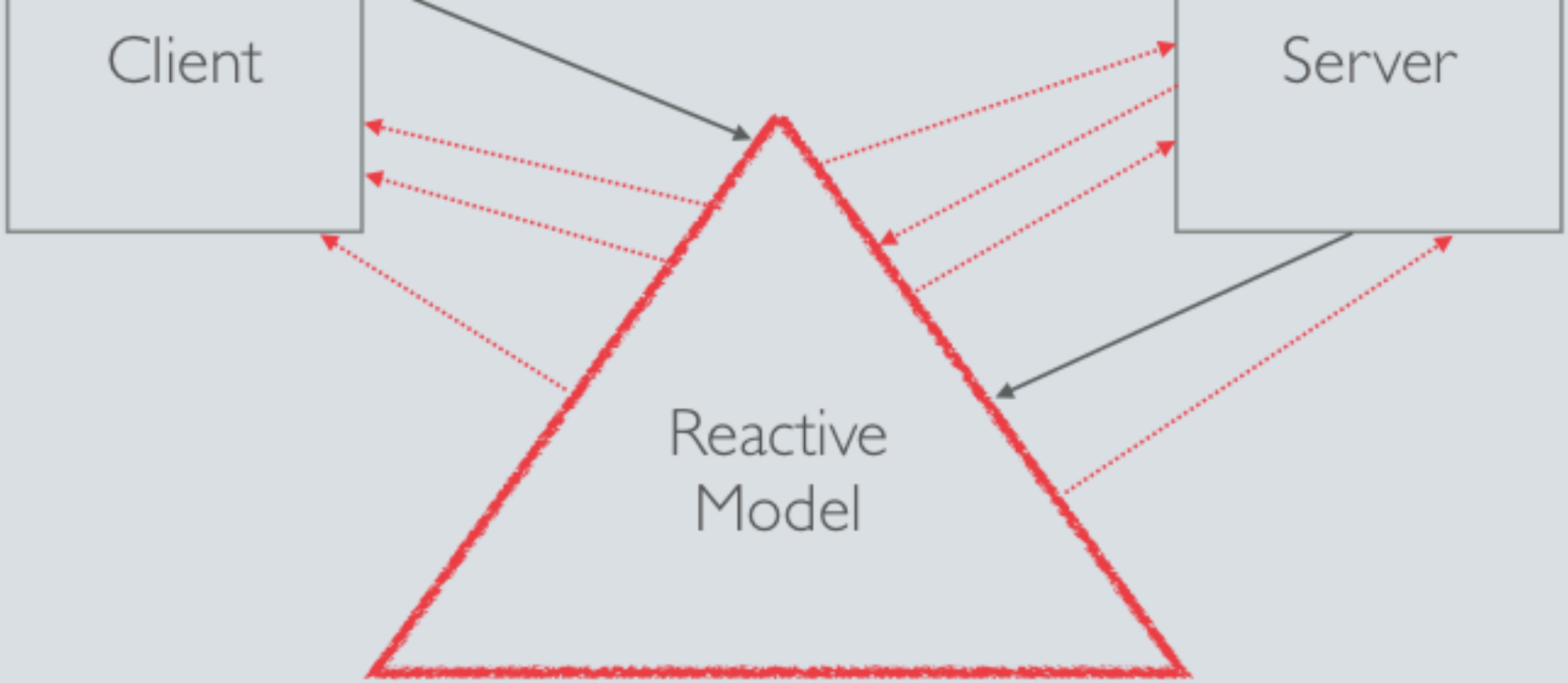
```
classdef (“Foo”) {
  body_of_action
}
```

MODEL FIRST

with right tooling

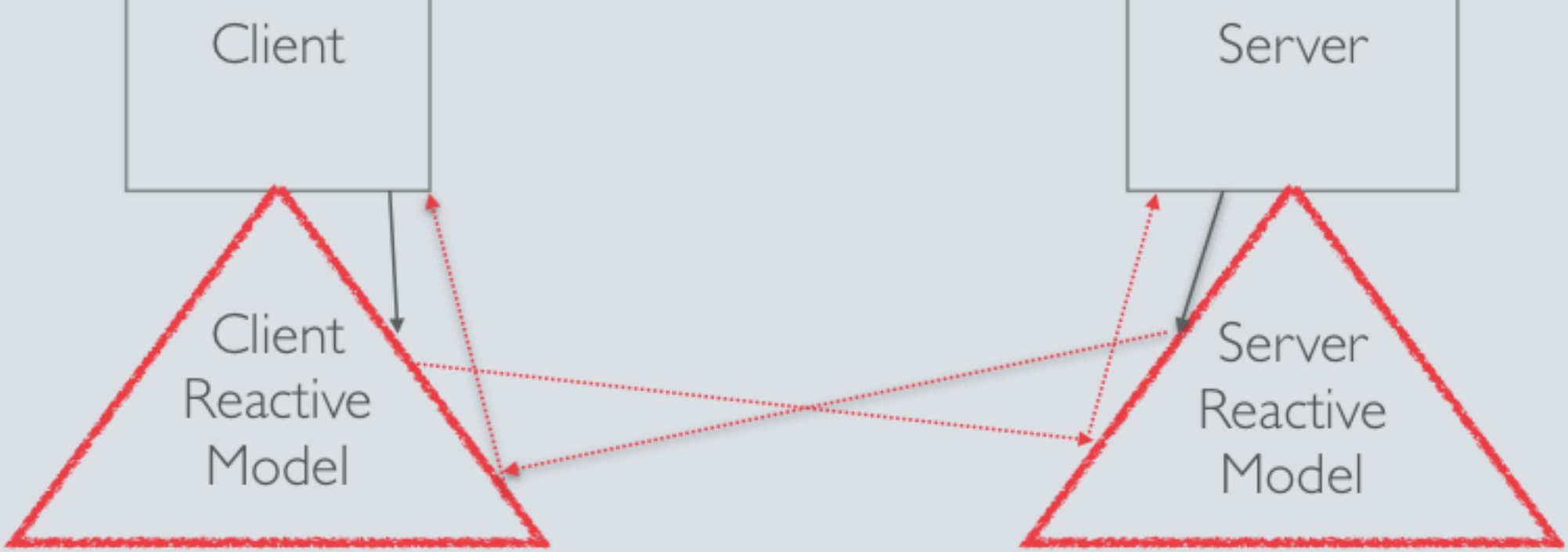
LIVE DEMO





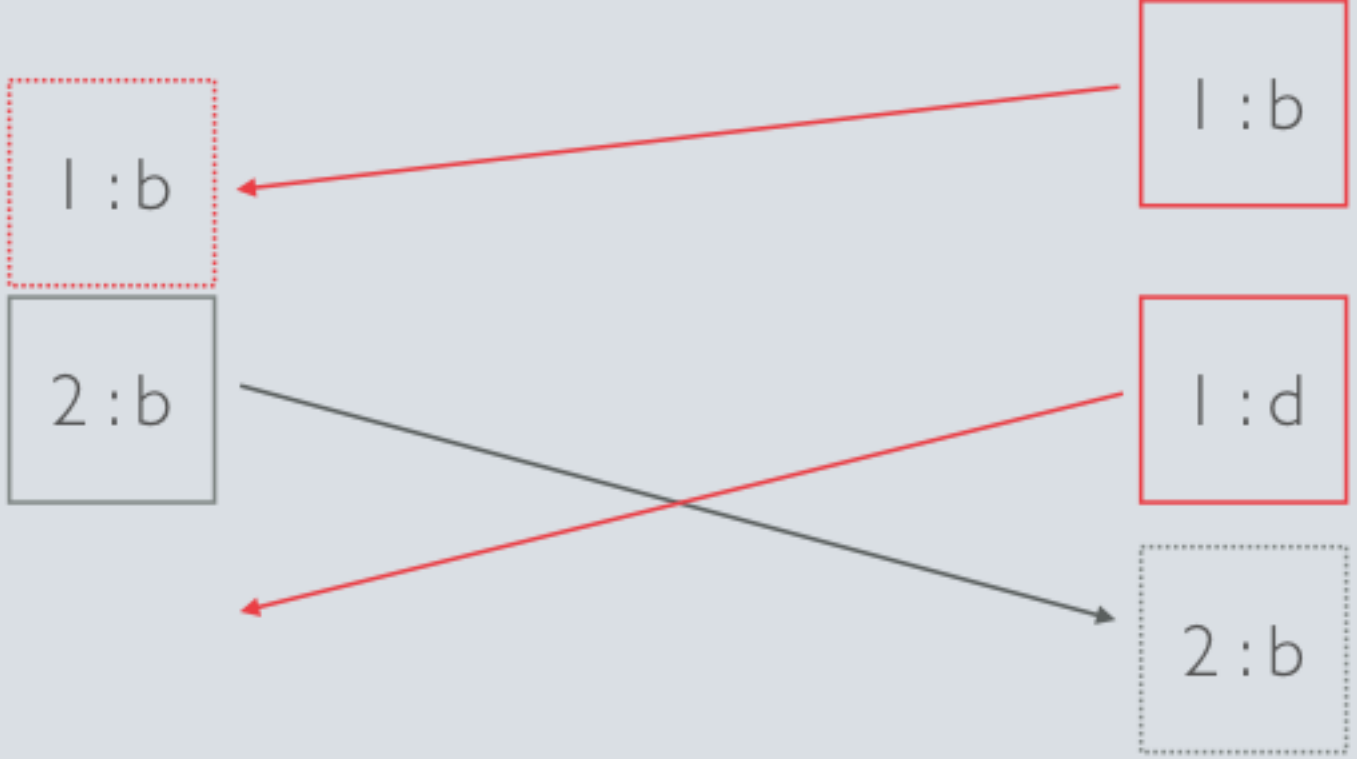
ETHEREAL

to real



ETHEREAL

to real

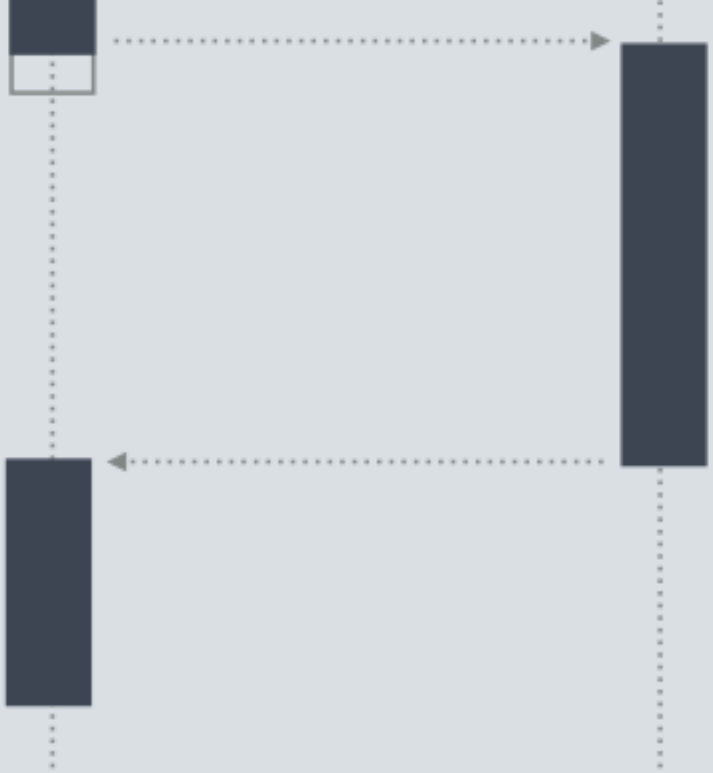


EVENTUAL CONSISTENCY

with versions

Click


Reactive
Handler
MAIN THREAD
synchronous
main thread
thread pool
sink.adviseOn()



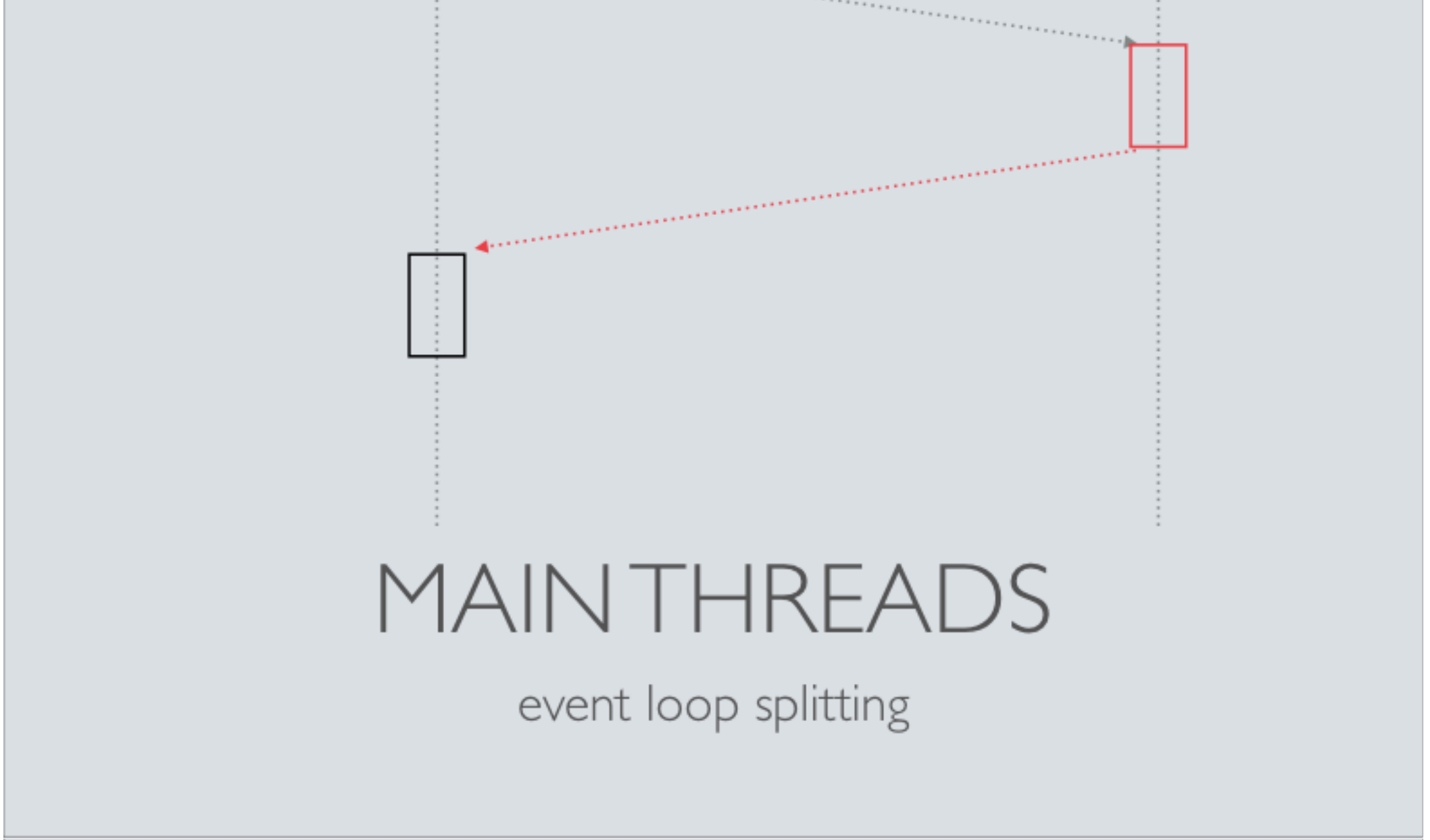
MAIN THREAD

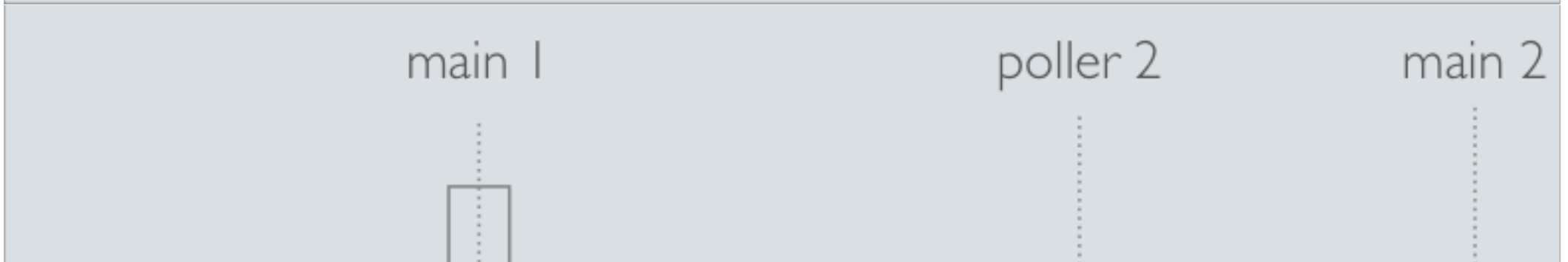
asynchronous

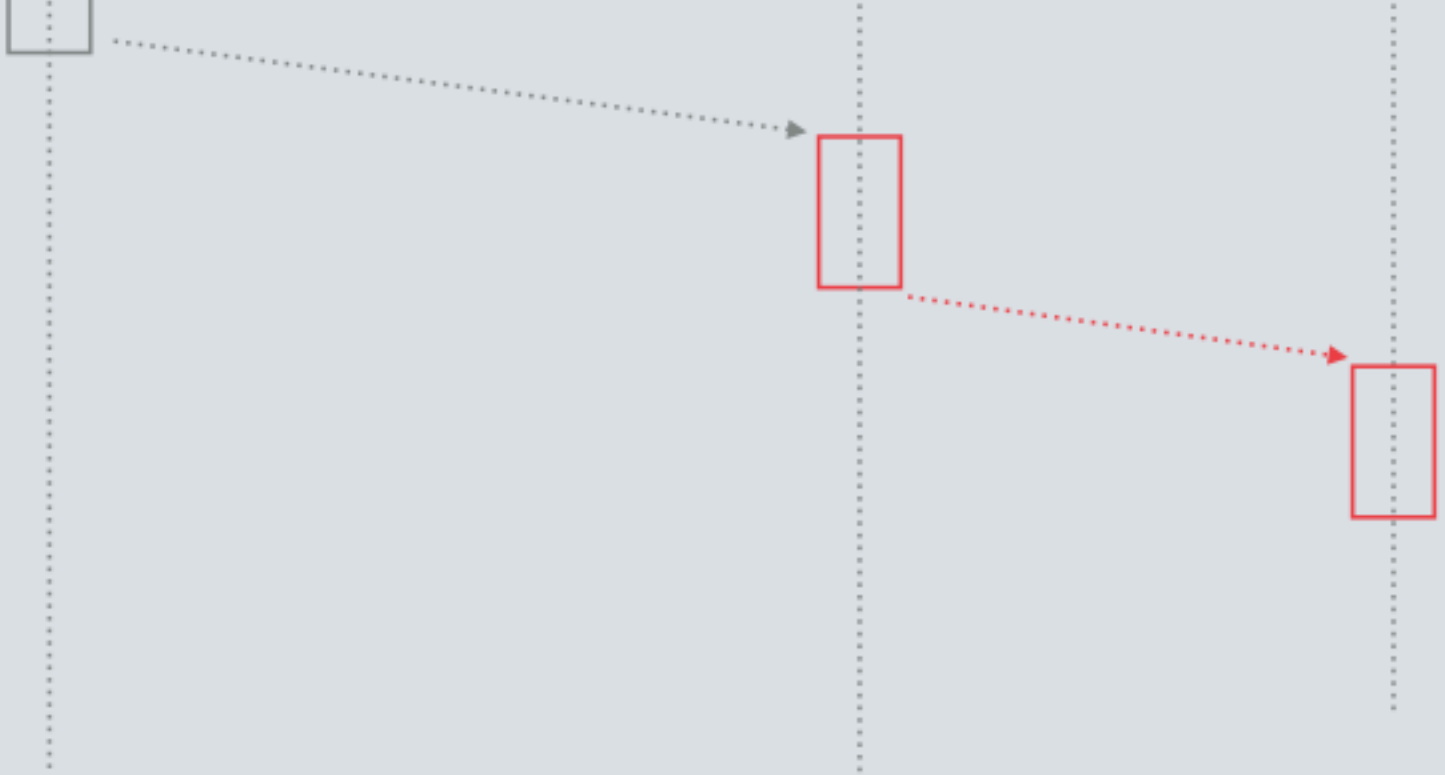
main thread 1

main thread 2

`property`


MAIN THREADS
event loop splitting


main 1
poller 2
main 2



MAIN THREADS

event loop splitting

protocol root (id = 1)

property<property<int>>(id = c/10)

property<property<int>> (id = c/10)

property<int>

INTRINSICS

implementation details

protocol root (id = 1)

property<property<int>> (id =c/10)

property<int> (id = c/11)

`c/10|property(c/11)` `c/11|int(42)`

`c/10|property<int>(c/11, value=42)`

INTRINSICS

implementation details

protocol root (id = 1)

property<property<int>> (id =c/10)

property<int> (id = c/11)

`c/11|int(43)`

INTRINSICS

implementation details

poller

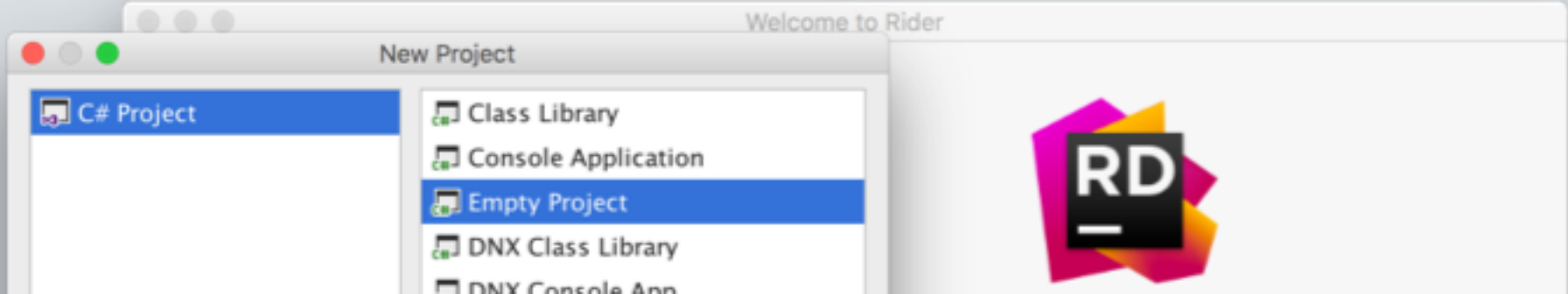
`c/10|property(c/11, v=42)`

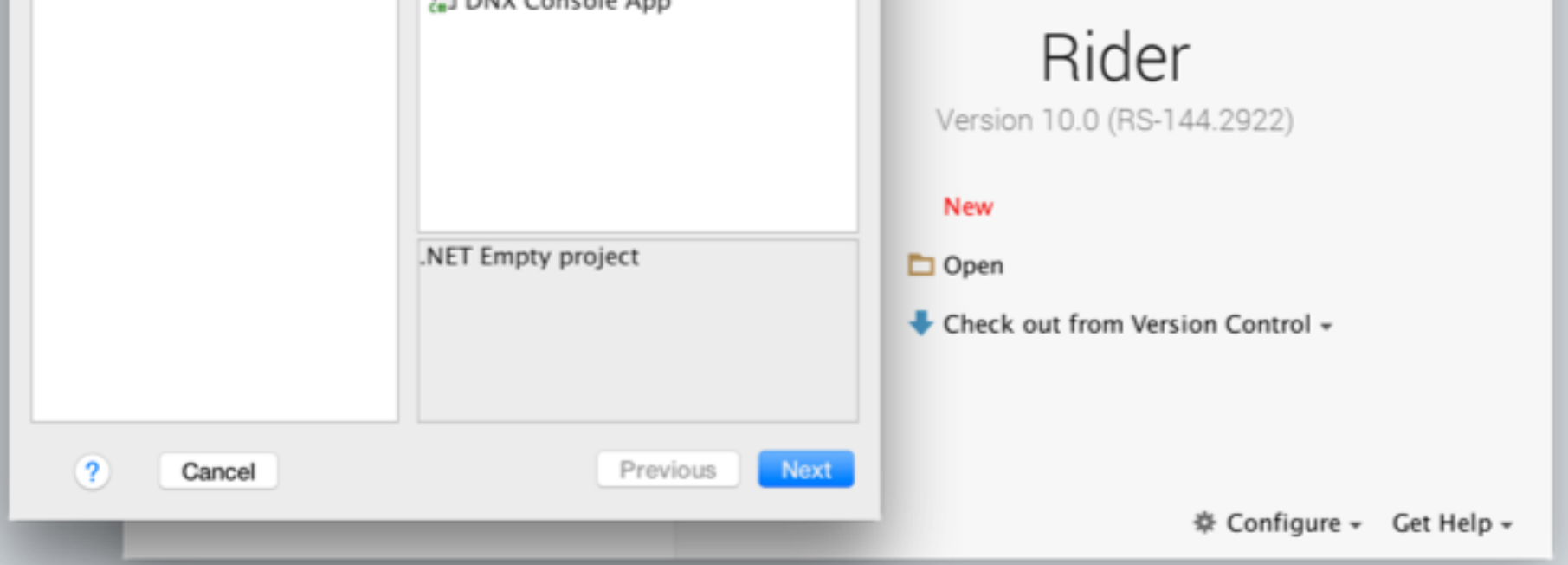
main

c/11|int(43)

INTRINSICS

implementation details





PROJECT RIDER

try it at home

QUESTIONS AND ANSWERS

thanks for your attention

dmitry.ivanov@jetbrains.com