A Session Type Provider

Compile-Time API Generation of Distributed Protocols with Refinements in F#

Rumyana Neykova Raymond Hu **Nobuko Yoshida** Fahd Abdeljallal

Imperial College London

http://mrg.doc.ic.ac.uk

Mobility Research Group

π-calculus, Session Types research at Imperial College

Home | People | Publications | Grants | Talks | Tutorials | Tools | Awards | Kohei Ho...

NEWS

The paper *Multiparty asynchronous session types* by Kohei Honda, Nobuko Yoshida, and Marco Carbone, published in POPL 2008 has been awarded the ACM SIGPLAN Most Influential POPL Paper Award today at POPL 2018.

» more

10 Jan 2018

Estafet has published a page on their usage of the Scribble language developed in our group with RedHat and other Industry partners.

» more

25 Sep 2017

Nick spoke at Golang UK 2017 on applying behavioural types to verify concurrent Go programs.

SELECTED PUBLICATIONS

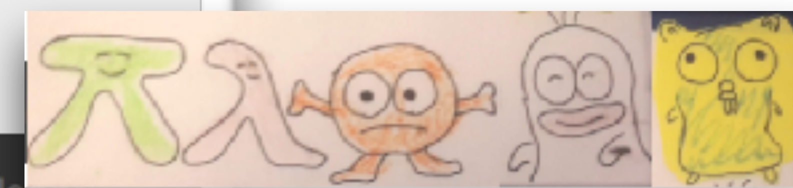
2018

Julien Lange , Nicholas Ng , Bernardo Toninho , Nobuko Yoshida : A Static Verification Framework for Message Passing in Go using Behavioural Types. *To appear in* ICSE 2018 .

Bernardo Toninho , Nobuko Yoshida : Depending On Session Typed Process. *To appear in* FoSSaCS 2018 .

Bernardo Toninho , Nobuko Yoshida : On Polymorphic Sessions And Functions: A Talk of Two (Fully Abstract) Encodings. *To appear in* ESOP 2018 .

Rumyana Neykova , Raymond Hu , Nobuko Yoshida , Fahd Abdeljallal : Session Type Providers: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#. *To appear in* CC 2018 .



Post-docs:

Simon CASTELLAN

David CASTRO

Francisco FERREIRA

Raymond HU

Rumyana NEYKOVA

Nicholas NG

Alceste SCALAS

PhD Students:

Assel ALTAYEVA

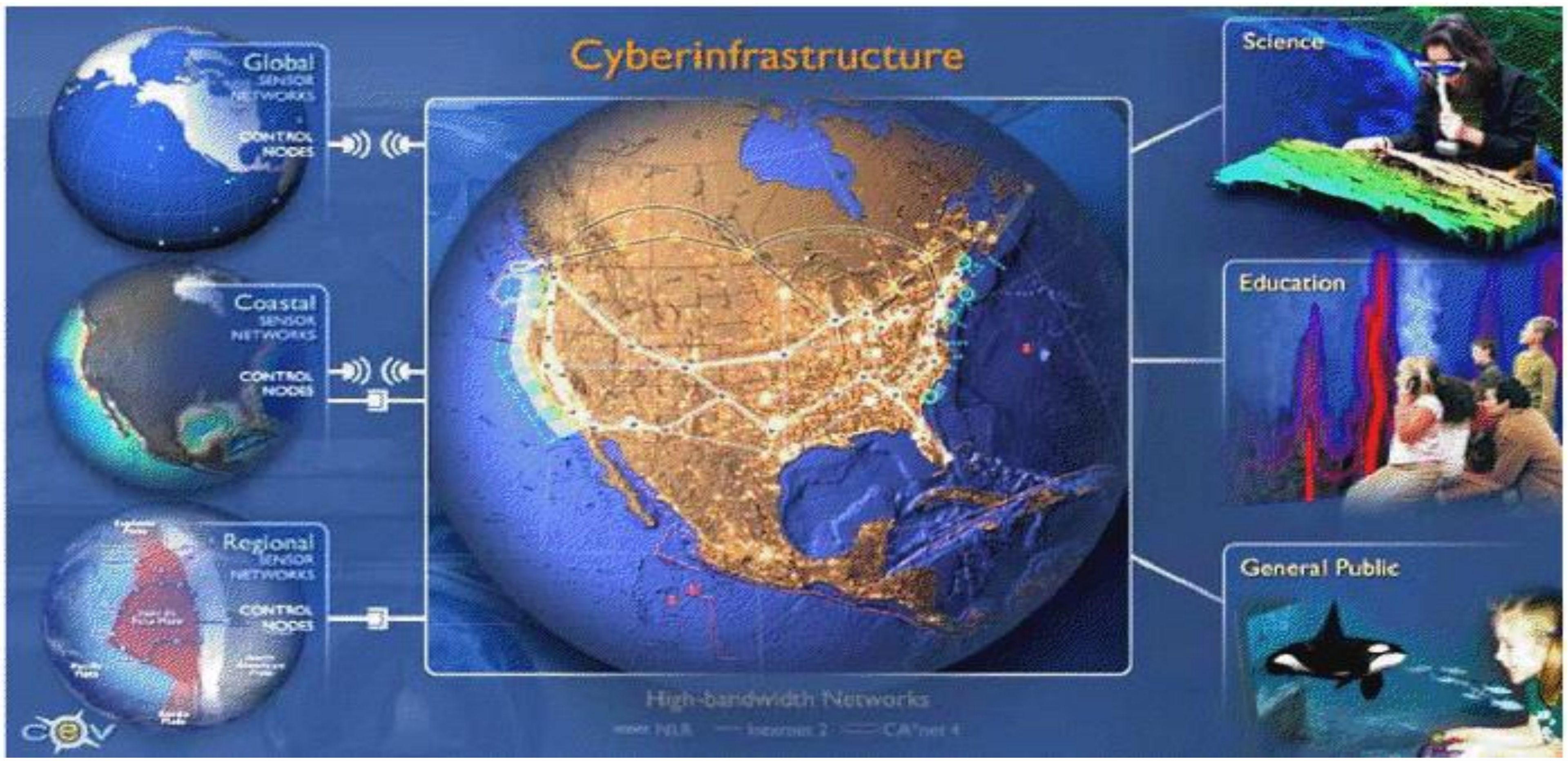
Juliana FRANCO

Eva GRAVERSEN

POPL 2008 MOST INFLUENTIAL PAPER AWARD

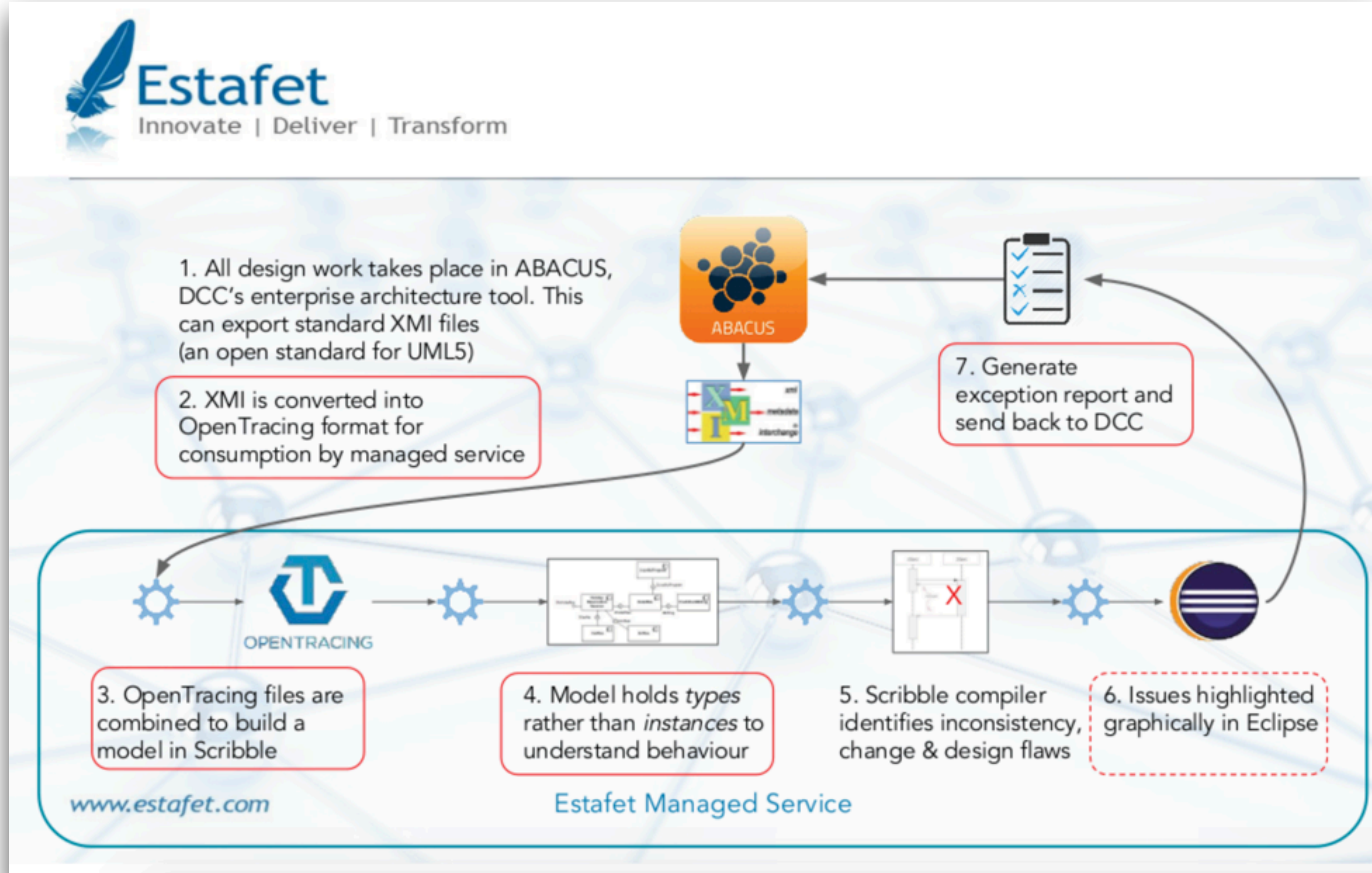
Ocean Observatories Initiative

OOI aims: to deploy an infrastructure (global network) to expand the scientists' ability to remotely study the ocean



Usage: Integrate real-time data acquisition, processing and data storage for ocean research,…

Scribble – Proving a distributed design



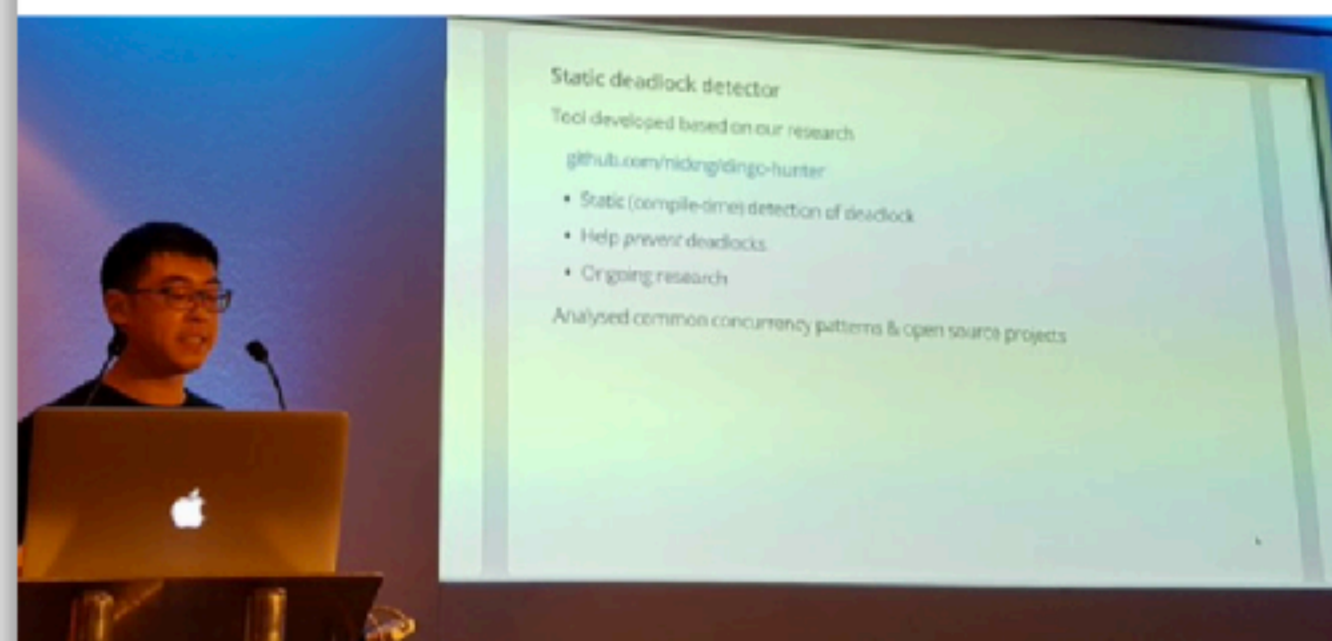
Interactions with Industries

Adam Bowen @adamnbowen · Sep 15
I didn't even know that session types existed an hour ago, but thanks to Nobuko Yoshida's great talk at **#pwlconf**, I want to learn more.

Nobuko Yoshida
Imperial College, London

DoC researcher to speak at Golang UK conference

by Vicky Kapogianni
20 July 2016



DoC researcher to speak at industry-focused Golang UK conference on results of concurrency research

Click here to add content

.@nicholascwng rocking on @GolangUKconf about static deadlock detection in #golang #gouk16

Interactions with Industries

CC'18

F#unctional Londoners Meetup G

6 days ago · 6:30 PM

Session Types with Fahd Abdeljallal

43 Members

Synopsis: Session types are a formalism to codify the structure of a communication, using types to specify the communication protocol used. This formalism provides the... LEARN MORE

ECOOP'17

ibuted Systems
vs.
Compositionality

Dr. Roland Kuhn
@rolandkuhn — *CTO of Actyx*

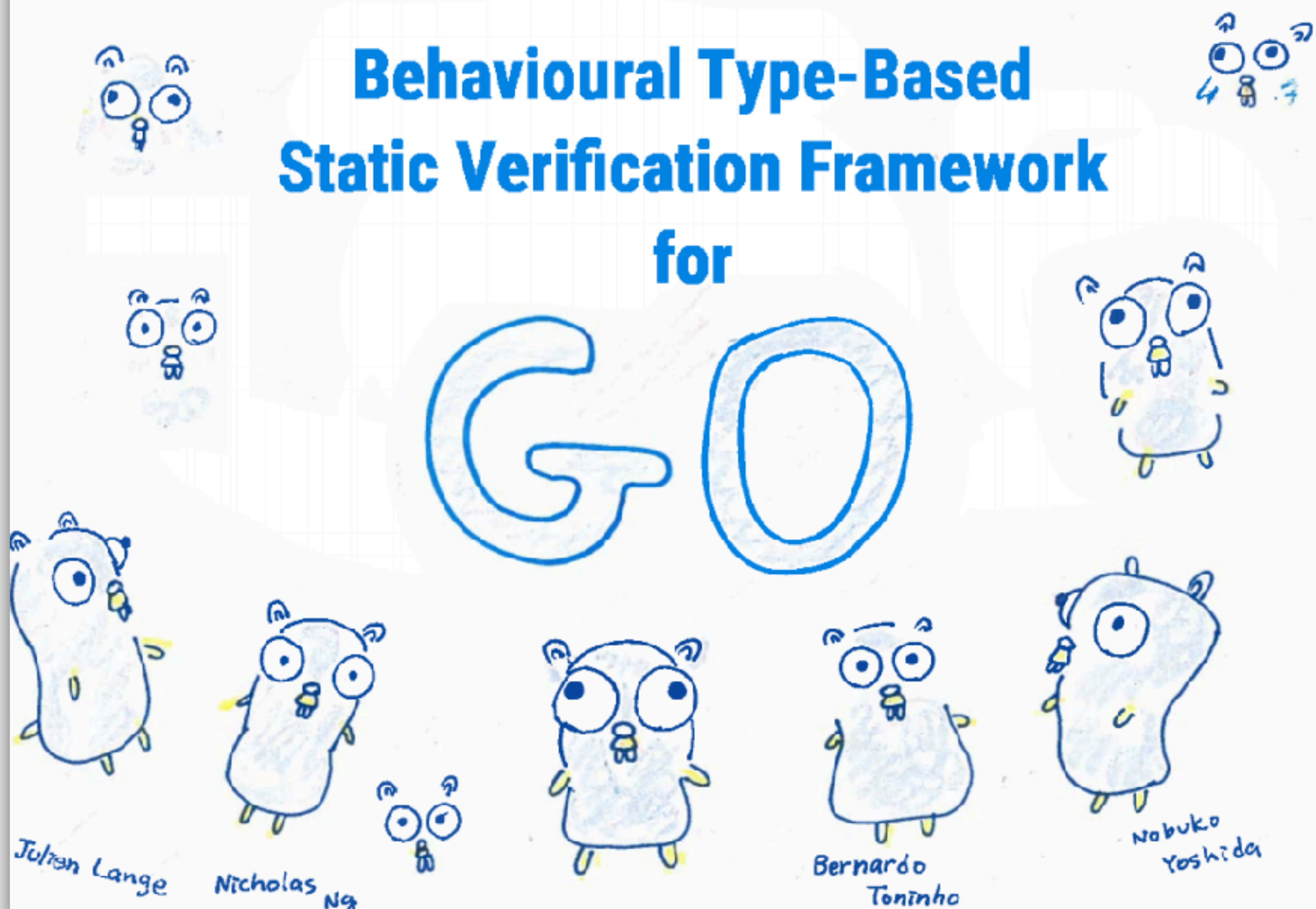
actyx

ECOOP'16

Current State

- behaviors can be composed both sequentially and concurrently
- effects are not yet tracked
- Scribble generator for Scala not yet there
- theoretical work at Imperial College, London (Prof. Nobuko Yoshida & Alceste Scalas)

Behavioural Type-Based Static Verification Framework for GO



Imperial College London

News

Home | College and Campus | Science | Engineering | Health | Business | Search here... | Go

Go concurrency verification research at DoC grabs headline

POPL'17

A paper by DoC researchers at POPL on Go concurrency verification was featured in a tech blog and generates a buzz outside of the research community.

A paper by researchers at the department was recently featured in the morning paper, a blog by venture capitalist Adrian Colye, which summarises an important, influential, topical or otherwise interesting paper in the field of computer science every weekday in an easily digestible way by non-researchers. On the 2 Feb 2017 issue of the morning paper, It was highlighted as "the true spirit of POPL (Principles of Programming Languages)".

ICSE'18

the morning paper

an interesting/influential/important paper from the world of CS every weekday morning, as selected by Adrian Colyer

A static verification framework for message passing in Go using behavioural types

JANUARY 25, 2018

tags: Concurrency, Programming Languages

A static verification framework for message passing in Go using behavioural types Lange et al., *ICSE 18*

With thanks to Alexis Richardson who first forwarded this paper to me.

We're jumping ahead to ICSE 18 now, and a paper that has been accepted for publication there later this year. It fits with the theme we've been exploring this week though, so I thought I'd cover it now. We've seen verification techniques applied in the context of **Rust** and **JavaScript**, looked at the integration of **linear types in Haskell**, and today it is the turn of Go!

SUBSCRIBE

never miss an issue! The Morning Paper delivered straight to your inbox.

SEARCH

type and press enter

ARCHIVES

Select Month

MOST READ IN THE LAST FEW DAYS

Selected Publications 2017/2018

- **[LICS'18]** Romain Demangeon, NY: Casual Computational Complexity of Distributed Processes.
- **[CC'18]** Rumyana Neykova , Raymond Hu, NY, Fahd Abdeljallal: Session Type Providers: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#.
- **[FoSSaCS'18]** Bernardo Toninho, NY: Depending On Session Typed Process.
- **[ESOP'18]** Bernardo Toninho, NY: On Polymorphic Sessions And Functions: A Talk of Two (Fully Abstract) Encodings.
- **[ESOP'18]** Malte Viering, Tzu-Chun Chen, Patrick Eugster, Raymond Hu , Lukasz Ziarek: A Typing Discipline for Statically Verified Crash Failure Handling in Distributed Systems.
- **[ICSE'18]** Julien Lange, Nicholas Ng, Bernardo Toninho, NY : A Static Verification Framework for Message Passing in Go using Behavioural Types
- **[ECOOP'17]** Alceste Scala, Raymond Hu, Ornela Darda, NY: A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming..
- **[COORDINATION'17]** Keigo Imai, NY, Shoji Yuen: Session-ocaml: a session-based library with polarities and lenses.
- **[FoSSaCS'17]** Julien Lange, NY: On the Undecidability of Asynchronous Session Subtyping.
- **[FASE'17]** Raymond Hu, NY: Explicit Connection Actions in Multiparty Session Types.
- **[CC'17]** Rumyana Neykova, NY: Let It Recover: Multiparty Protocol-Induced Recovery.
- **[POPL'17]** Julien Lange, Nicholas Ng, Bernardo Toninho, NY: Fencing off Go: Liveness and Safety for Channel-based Programming.

Selected Publications 2017/2018

- **[LICS'18]** Romain Demangeon, NY: Casual Computational Complexity of Distributed Processes.
- **[CC'18]** Rumyana Neykova , Raymond Hu, NY, Fahd Abdeljallal: Session Type Providers: Compile-time API Generation for Distributed Protocols with Interaction Refinements in F#.
- **[FoSSaCS'18]** Bernardo Toninho, NY: Depending On Session Typed Process.
- **[ESOP'18]** Bernardo Toninho, NY: On Polymorphic Sessions And Functions: A Talk of Two (Fully Abstract) Encodings.
- **[ESOP'18]** Malte Viering, Tzu-Chun Chen, Patrick Eugster, Raymond Hu , Lukasz Ziarek: A Typing Discipline for Statically Verified Crash Failure Handling in Distributed Systems.
- **[ICSE'18]** Julien Lange, Nicholas Ng, Bernardo Toninho, NY : A Static Verification Framework for Message Passing in Go using Behavioural Types.
- **[ECOOP'17]** Alceste Scala, Raymond Hu, Ornela Darda, NY: A Linear Decomposition of Multiparty Sessions for Safe Distributed Programming.
- **[COORDINATION'17]** Keigo Imai, NY, Shoji Yuen: Session-ocaml: a session-based library with polarities and lenses.
- **[FoSSaCS'17]** Julien Lange, NY: On the Undecidability of Asynchronous Session Subtyping.
- **[FASE'17]** Raymond Hu, NY: Explicit Connection Actions in Multiparty Session Types.
- **[CC'17]** Rumyana Neykova, NY: Let It Recover: Multiparty Protocol-Induced Recovery.
- **[POPL'17]** Julien Lange, Nicholas Ng, Bernardo Toninho, NY: Fencing off Go: Liveness and Safety for Channel-based Programming.

CC'18

A Session Type Provider

Compile-Time API Generation of Distributed Protocols with Refinements in F#

Rumyana Neykova, Imperial College London, United Kingdom
Raymond Hu, Imperial College London, United Kingdom
Nobuko Yoshida, Imperial College London, United Kingdom
Fahd Abdeljallal, Imperial College London, United Kingdom

Abstract

We present a library for the specification and implementation of distributed protocols in native F# (and other .NET languages) based on multiparty session types (MPST). There are two main contributions. Our library is the first practical development of MPST to support what we refer to as *interaction refinements*: a collection of features related to the refinement of *protocols*, such as message-type refinements (value constraints) and message-value dependent control flow. A well-typed endpoint program using our library is …anteed to perform only compliant session I/O actions … the refined protocol, up to premature termination. … our library is developed as a session *type provider*,

1 Introduction

Type providers [20, 27] are a .NET feature for a form of compile-time meta programming, designed to bridge between programming in statically typed languages such as F# and C#, and working with so-called *information spaces*—structured data sources such as SQL databases or XML data.

A type provider works as a compiler plugin that performs on-demand generation of *types*: it takes a schema for an external information space, and generates types that allow the data to be manipulated via a strongly-typed interface, with benefits such as static error detection and IDE auto-completion. For example, an instantiation of the in-built type provider for WSDL Web services [6] may look like

Graydon Hoare @graydon_pub

(This stuff is _fantastic_)

11:31 PM - 11 Mar 2018

32 Retweets 83 Likes

shots fired @zeeshanlakhani · Mar 12
Replying to @graydon_pub @dsyme
Awesome!

Brendan Zabarauskas @brendanzab ·
Replying to @graydon_pub
This stuff fills me with hope!

Ryan Riley @panesofglass · Mar 12
Replying to @graydon_pub
This is amazing! I guess I need to switch

A Session Type Provider

Compile-Time API Generation of Distributed Protocols with Refinements in F#

Rumyana Neykova Raymond Hu **Nobuko Yoshida** Fahd Abdeljallal

Imperial College London

Part One
Type Providers

Type Providers

Problem: Languages do not integrate information

- We need to bring information into the language

PLDI'16

Types from data: Making structured data first-class citizens in F#

Tomas Petricek
University of Cambridge
tomas@tomasp.net

Gustavo Guerra
Microsoft Corporation, London
gustavo@codebeside.org

Don Syme
Microsoft Research, Cambridge
dsyme@microsoft.com

Abstract

Most modern applications interact with external services and access data in structured formats such as XML, JSON and CSV. Static type systems do not understand such formats, often making data access more cumbersome. Should we give up and leave the messy world of external data to dynamic typing and runtime checks? Of course, not!

We present F# Data, a library that integrates external structured data into F#. As most real-world data does not come with an explicit schema, we develop a shape inference

```
let doc = Http.Request("http://api.owm.org/?q=NYC")
match JsonValue.Parse(doc) with
| Record(root) →
  match Map.find "main" root with
  | Record(main) →
    match Map.find "temp" main with
    | Number(num) → printfn "Lovely %f!" num
    | _ → failwith "Incorrect format"
  | _ → failwith "Incorrect format"
| _ → failwith "Incorrect format"
```

Before Type Providers

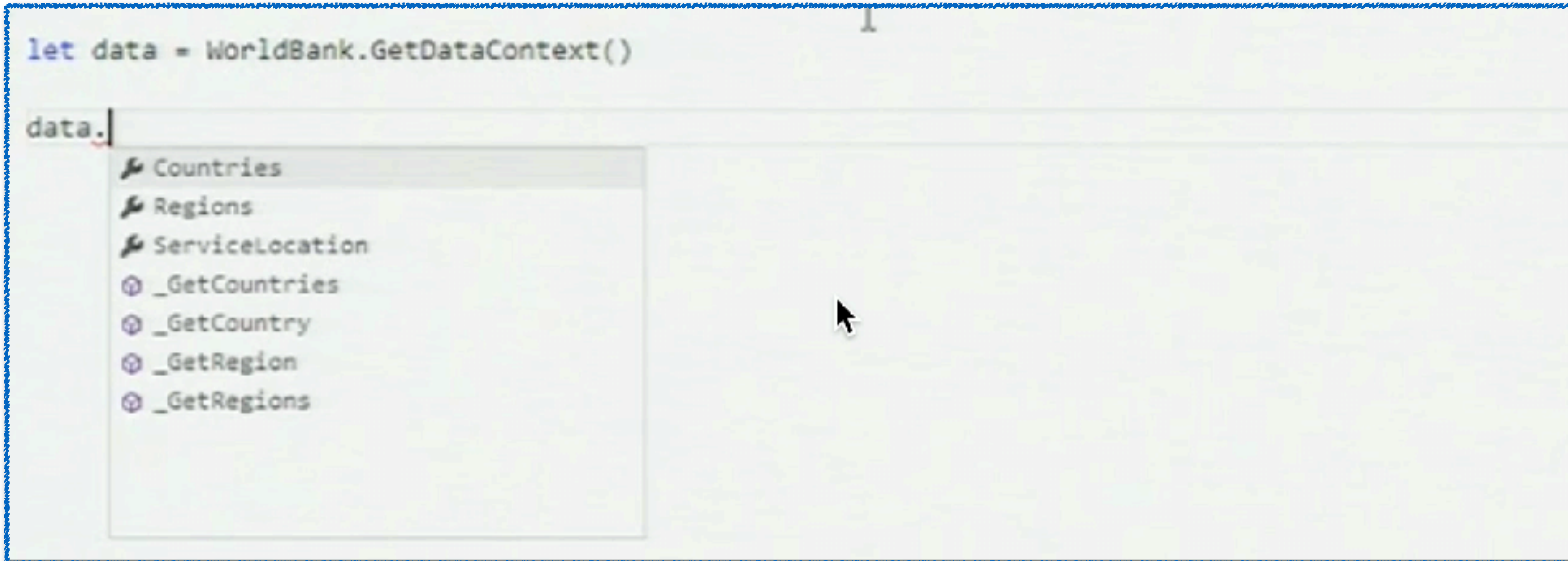
With Type Providers

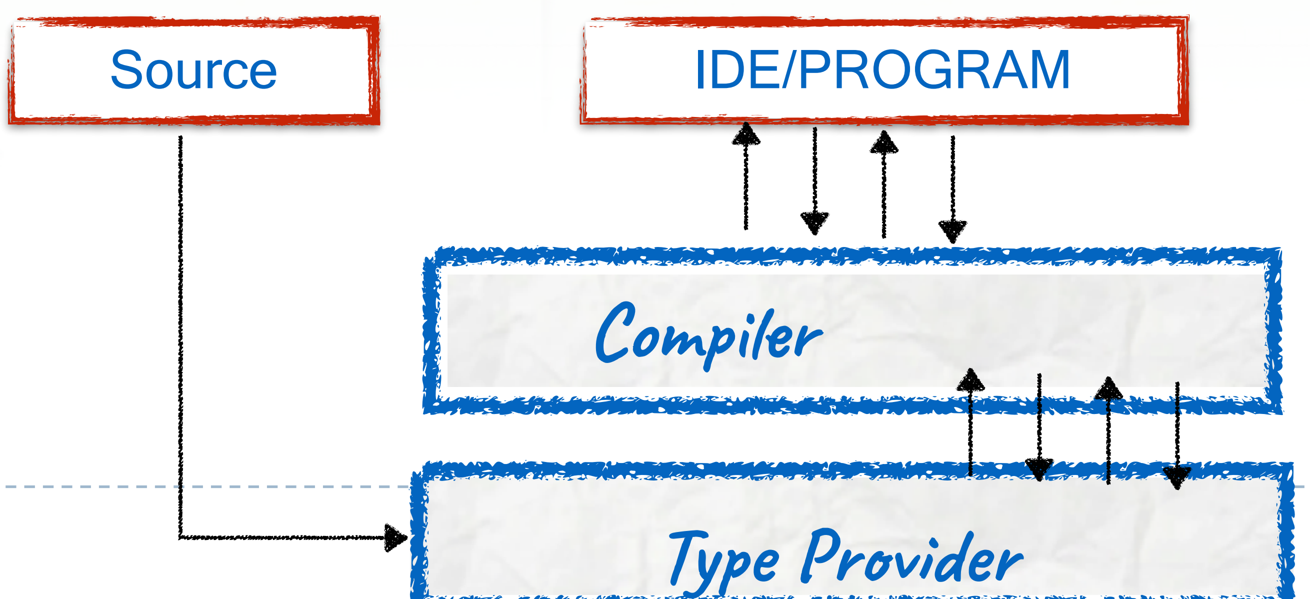
```
let doc = Http.Request("http://api.owm.org/?q=NYC")
match JsonValue.Parse(doc) with
| Record(root) →
  match Map.find "main" root with
  | Record(main) →
    match Map.find "temp" main with
    | Number(num) → printfn "Lovely %f!" num
    | _ → failwith "Incorrect format"
  | _ → failwith "Incorrect format"
| _ → failwith "Incorrect format"
```

```
type W = JsonProvider⟨"http://api.owm.org/?q=NYC"⟩
printfn "Lovely %f!" (W.GetSample().Main.Temp)
```

- ☑ all data is typed
- ☑ on-demand generation
- ☑ autocompletion
- ☑ background type-checking

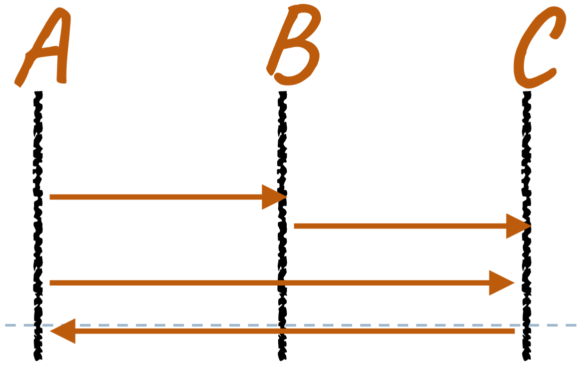
WorldBank Type Providers





Useful for structured data?

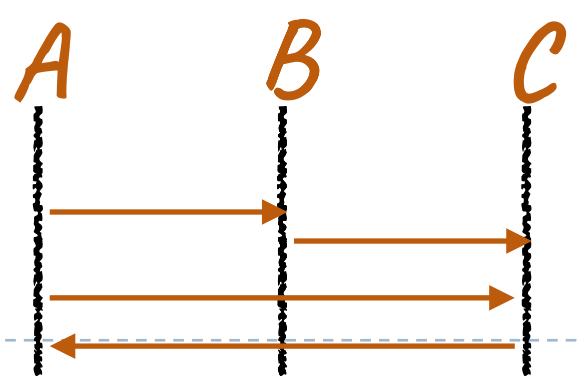
How about structured communication?



A generalisation to distributed protocols requires

- a notion of **schema for structured interactions** between services
- an understanding of how to extract the **localised behaviour** for each services

How about structured communication?



Part Two
Session Types

POPL'08

Multiparty Asynchronous Session Types

Kohei Honda
Queen Mary, University of London
kohei@dcs.qmul.ac.uk

Nobuko Yoshida
Imperial College London
yoshida@doc.ic.ac.uk

Marco Carbone
Queen Mary, University of London
carbonem@dcs.qmul.ac.uk

Abstract

Communication is becoming one of the central elements in software development. As a potential typed foundation for structured communication-centred programming, session types have been studied over the last decade for a wide range of process calculi and programming languages, focussing on binary (two-party) sessions. This work extends the foregoing theories of binary session types to multiparty, asynchronous sessions, which often arise in practical communication-centred applications. Presented as a typed calculus for mobile processes, the theory introduces a new notion of types in which interactions involving multiple peers are directly abstracted as a global scenario. Global types retain a friendly type syntax of binary session types while capturing complex causal chains of multiparty asynchronous interactions. A global type plays the role of a shared agreement among communication peers, and is used as a ba-

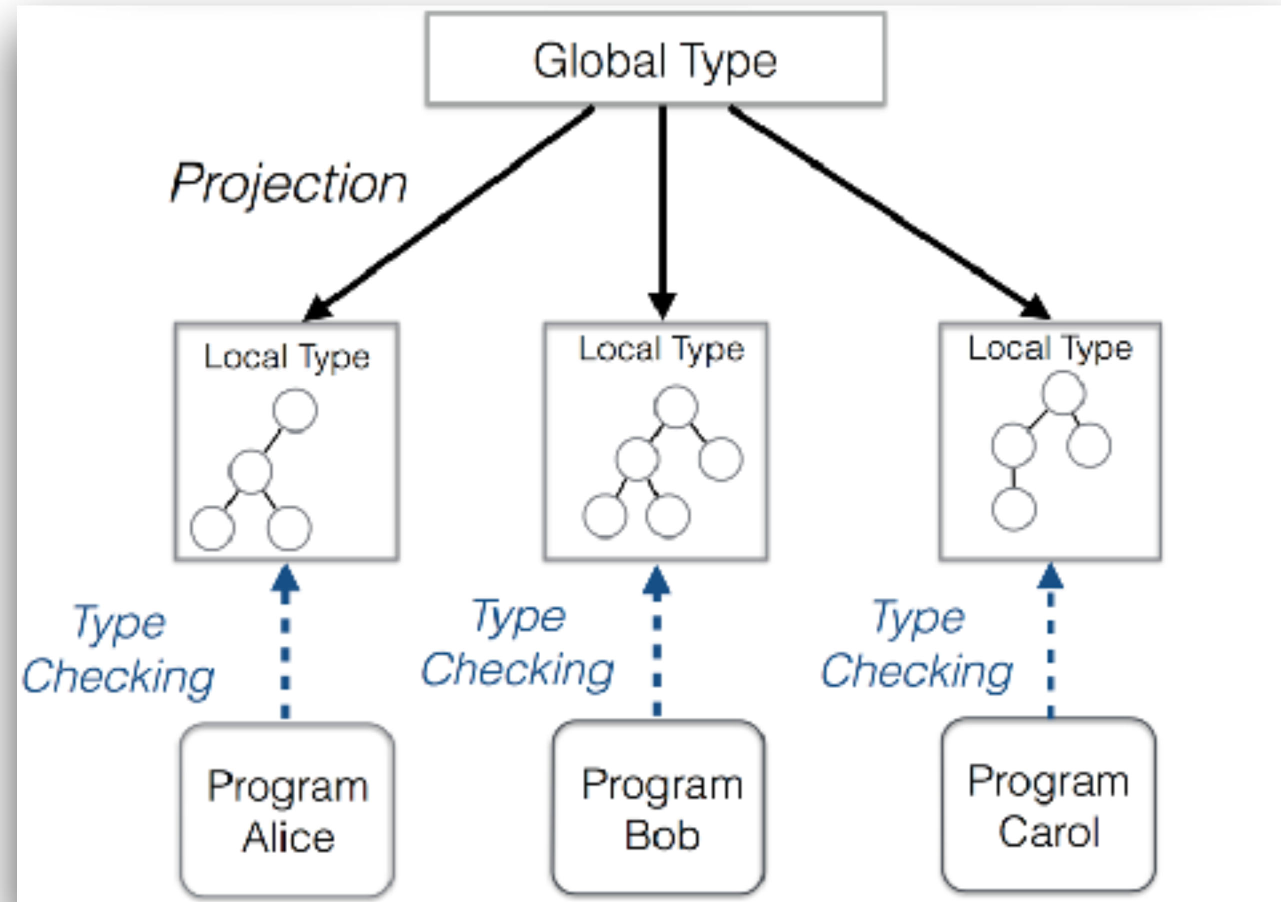
vices (Carbone et al. 2006, 2007; WS-CDL; Sparkes 2006; Honda et al. 2007a). A basic observation underlying session types is that a communication-centred application often exhibits a highly structured sequence of interactions involving, for example, branching and recursion, which as a whole form a natural unit of conversation, or *session*. The structure of a conversation is abstracted as a type through an intuitive syntax, which is then used as a basis of validating programs through an associated type discipline.

As an example, the following session type describes a simple business protocol between Buyer and Seller from Buyer's viewpoint: Buyer sends the title of a book (a string), Seller sends a quote (an integer). If Buyer is satisfied by the quote, then sends his address (a string) and Seller sends back the delivery date (a date); otherwise it quits the conversation.

$$!\mathsf{string}; ?\mathsf{int}; \oplus\{\mathsf{ok} : !\mathsf{string}; ?\mathsf{date}; \mathsf{end}, \quad \mathsf{quit} : \mathsf{end}\} \qquad (1)$$

Session Types

Scribble



- Protocol Validation

```
(int) from C to S;
(bool) from S to C;
```

✅

- Program Verification

```
runB c = let (x, c’) =
    receive c in send true c’
```

✅

A system of *well-behaved processes* is free from deadlocks, orphan messages and reception errors

Useful for structured data? 👍

Data Type providers bring information into the language as strongly tooled, strongly typed

How about structured communication? 👍

Session Type providers *bring* ***communication*** *into the language as strongly tooled, strongly typed*

Our Solution: Session Type Providers

```
Div(x:int, y:int) from C to S;
Res(z:float) from S to C;
```

```
type Prot = STP<“Prot.scr”, C>
 let s = new Prot().Init()
   s.
```

Session Type Provider

Our Solution: Session Type Providers

```
Div(x:int, y:int) from C to S;
Res(z:float) from S to C;
```

```
type Prot = STP<"Prot.scr", C>
 let s = new Prot().Init()
   s.
```

send

State2 State1.send(S Role, Div label, int x, int y)
Constraints: y!=0

Session Type Provider

Our Solution: Session Type Providers

```
Div(x:int, y:int) from C to S;
Res(z:float) from S to C;
```

```
type Prot = STP<"Prot.scr", C>
 let s = new Prot().Init()
   s.send(S, Div, 6, 3)
```

Session Type Provider

Our Solution: Session Type Providers

```
Div(x:int, y:int) from C to S;
Res(z:float) from S to C;
```

```
type Prot = STP<"Prot.scr", C>
 let s = new Prot().Init()
   s.send(S, Div, 6, 3)
    .
```

receive State3 State1.receive(S Role, Res label, Buf<float> f)

Session Type Provider

Our Solution: Session Type Providers

```
Div(x:int, y:int) from C to S;
Res(z:float) from S to C;
```

```
type Prot = STP<"Prot.scr", C>
 let s = new Prot().Init()
   s.send(S, Div, 6, 3)
    .receive(S, Res, y)
```

Session Type Provider

Our Solution: Session Type Providers

```
Div(x:int, y:int) from S to C;
Res(z:float) from S to C;
```

```
type Prot = STP<“Prot.scr”, C>
 let s = new Prot().Init()
   s.
```

Session Type Provider

Our Solution: Session Type Providers

```
Div(x:int, y:int) from C to S;
Res(z:float) from S to C;
```

```
type Prot = STP<"Prot.scr", C>
 let s = new Prot().Init()
   s.send(S, Div, 6, "hello")
```

Wrong **payload**

Session Type Provider

Our Solution: Session Type Providers

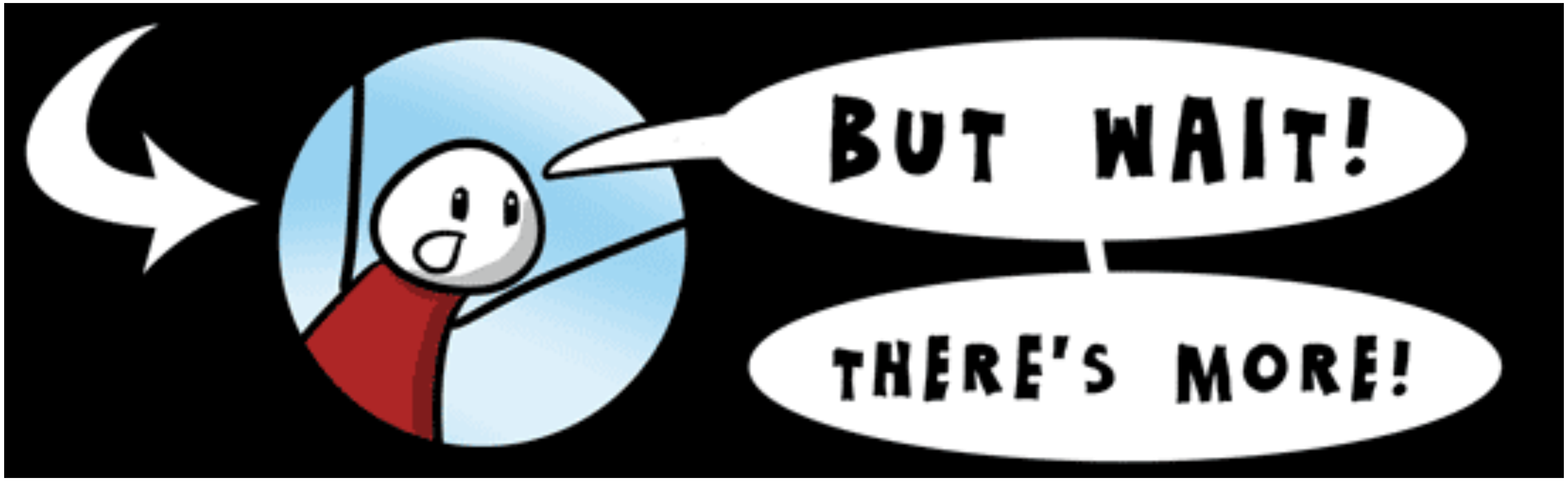
```
Div(x:int, y:int) from C to S;
Res(z:float) from S to C;
```

```
type Prot = STP<"Prot.scr", A>
```

Wrong **protocol**

Session Type Provider

Session Type providers *bring* **communication** *into the language as strongly tooled, strongly typed*



Calculator Revisited!

y!=0

```
global protocol Ca      role S, role C){
 choice at C {
  Div(x:int, y:int) from C to S;
  Res(z:float) from C to S;
  do Calc(C, S);
 } or {
  Add(x:int, y:int) from C to S;
  Res(z:int) from S to C;
  do Calc(C, S);
 } or {
  Sqrt(x:float) from C to S;
  Res(y:floa ) from S to C;
  do Calc(C, S
 } or {
  Bye() fro
  Bye() from S
 }
}
```

x>0

Scribble with refinements

```
global protocol Calc(role S, role C){
 choice at C {
  Div(x:int, y:int) from C to S;@y!=0
  Res(z:float) from S to C;
  do Calc(C, S);
 } or {
  Add(x:int, y:int) from C to S;
  Res(z:int) from S to C;
  do Calc(C, S);
 } or {
  Sqrt(x:float) from C to S;@x>0
  Res(y:float) from S to C;
  do Calc(C, S);
 } or {
  Bye() from C to S;
  Bye() from S to C;
 }
}
```

interaction refinement

New

Scribble with refinements

```
global protocol Calc(role S, role C){
 choice at C {
  Div(x:int, y:int) from C to S;@y!=0
  Res(z:float) from S to C;
  do Calc(C, S);
 } or {
  Add(x:int, y:int) from C to S;
  Res(z:int) from S to C;
  do Calc(C, S);
 } or {
  Sqrt(x:float) from C to S;@x>0
  Res(y:float) from S to C;
  do Calc(C, S);
 } or {
  Bye() from C to S;
  Bye() from S to C;
 }
}
```

interaction refinement E

$$E \quad ::= \quad x \mid n \mid \mathsf{true} \mid \mathsf{false} \mid E \oplus E \mid \ominus E \mid f(E_1, \ldots, E_n)$$

$$\oplus \quad ::= \quad \mathsf{and} \mid \mathsf{or} \mid = \mid < \mid > \mid + \mid * \qquad \ominus \ ::= \ \mathsf{not} \mid -$$

Part Three

A Session Type Provider

What do you get from a session type provider?

Session Types — **Safety**

- A statically well-typed endpoint program will never perform a non-compliant I/O action w.r.t. the source protocol.

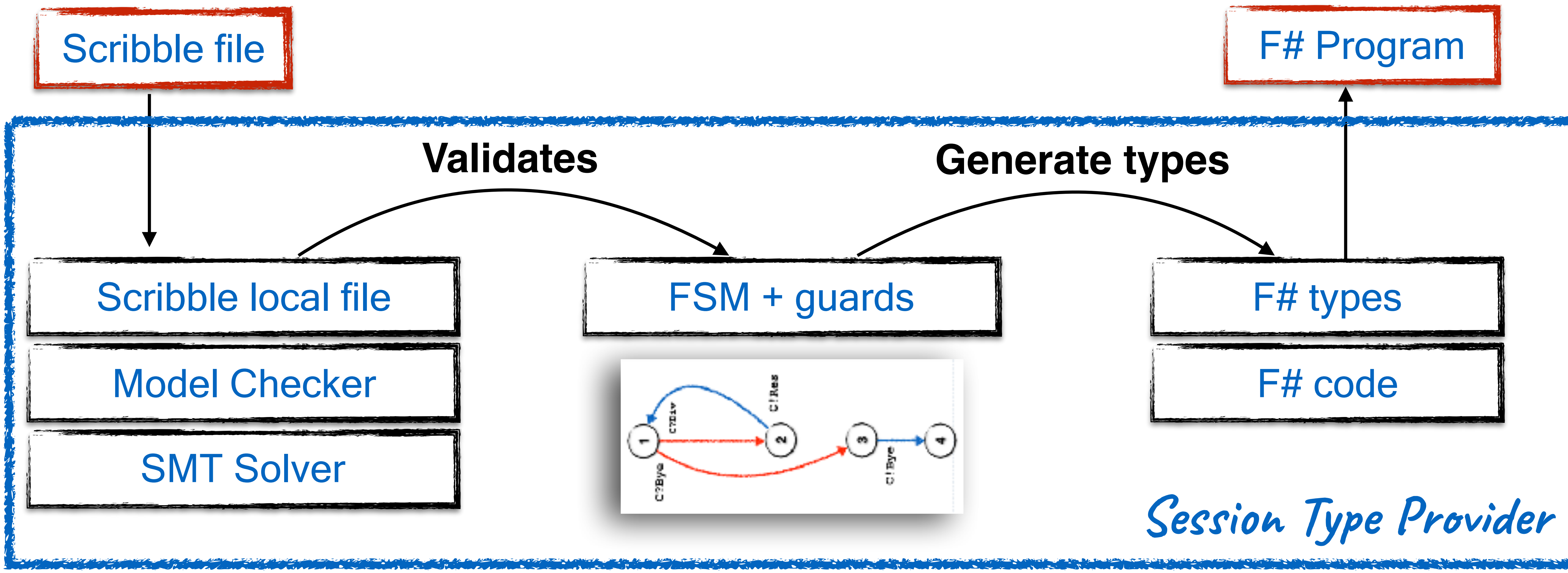
Type Providers — **Usability**

- compile-time generation
- background type checking & auto-completion
- a platform for tool integration (e.g. protocol validation)

Interaction refinements — **Reliability**

- runtime enforcement of constraint
- implicitly send values that can be inferred (safe by construction)
- do not send values that can be locally inferred

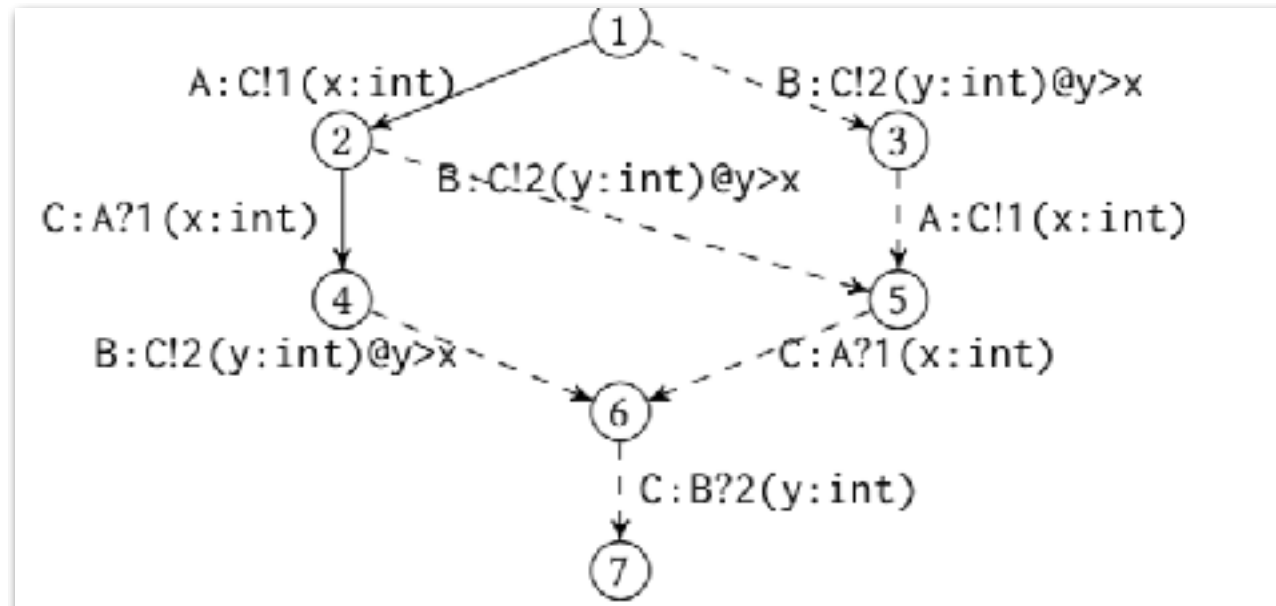
A Session Type Provider (Architecture)



The type provider framework is used for tool integration

Model Properties CFSM F# Type Code

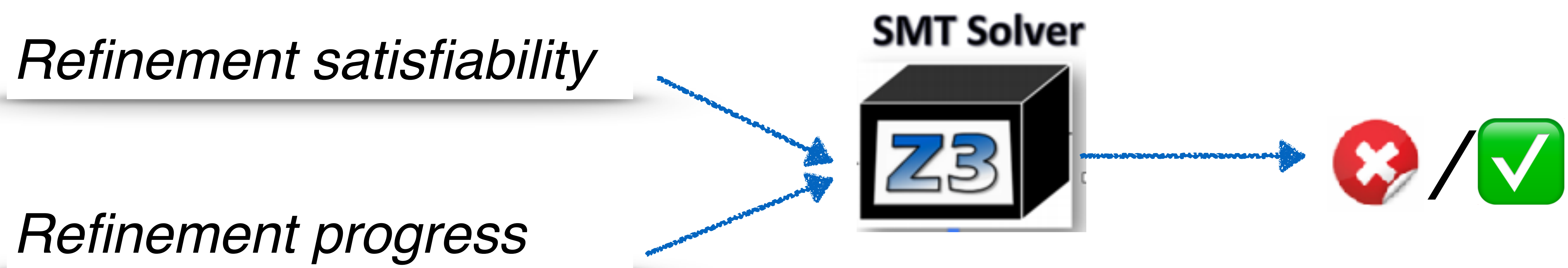
```
1(x:int) from A to C;
2(y:int) from B to C; @y>x
```



Bounded model checking as a validation methodology [FASE'17]

Safety Properties:

- ☑ reception-error freedom
- ☑ orphan-message freedom
- ☑ deadlock freedom


Refinement satisfiability
Refinement progress
SMT Solver
Z3
/

Refinement satisfiability

- check if the conjunction of all formulas is satisfiable
  e.g. (and (> y (+ x 1))(< y 4)(> x 3))

```
1(x:int) from A to B; @x>3
choice at B {2() from B to A;}
        or {3(y:int) from B to A; @y>x+1 and y<4}
```

Checks if all execution paths are reachable

```
1(x:int) from A to B; @x>3
choice at B {2() from B to A;}
        or {3(y:int) from B to A; @y>x+1 and y>4}
```

Refinement satisfiability

- check if the conjunction of all formulas is satisfiable
  e.g. (and (> y (+ x 1))(< y 4)(> x 3))

```
1(x:int) from A to B; @x>3
choice at B {2() from B to A;}
         or {3(y:int) from B to A; @y>x+1 and y<4}
```

```
1(x:int) from A to B; @x>3
choice at B {2() from B to A;}
         or {3(y:int) from B to A; @y>x+1 and y>4}
```

Model | **Properties** | CFSM | F# Type | Code

Refinement progress

- check if formula is satisfiable for all preceding solutions
  e.g.(forall ((x Int)(y Int))(=> (> x 3)(or (< x y)(> x y))))

```
1(x:int) from A to B; @x>3
2(y:int) from A to B;
choice at B {3() from B to A; @x>y}
```

```
1(x
2(y
cho
        or {4(y:int) from B to A; @x>y}
```

Ensures that at any output point in the protocol implementations there will be **always** some values for which the formula holds

```
1(x:int) from A to B; @x>3
2(y:int) from A to B; @y<=3
choice at B {3() from B to A; @x>=y}
        or {4(y:int) from B to A; @x<y}
```

Refinement progress

- check if formula is satisfiable for all preceding solutions
  e.g.(forall ((x Int)(y Int))(=> (> x 3)(or (< x y)(> x y))))

```
1(x:int) from A to B; @x>3
2(y:int) from A to B;
choice at B {3() from B to A; @x>y}
         or {4() from B to A; @x<y}
```

✗

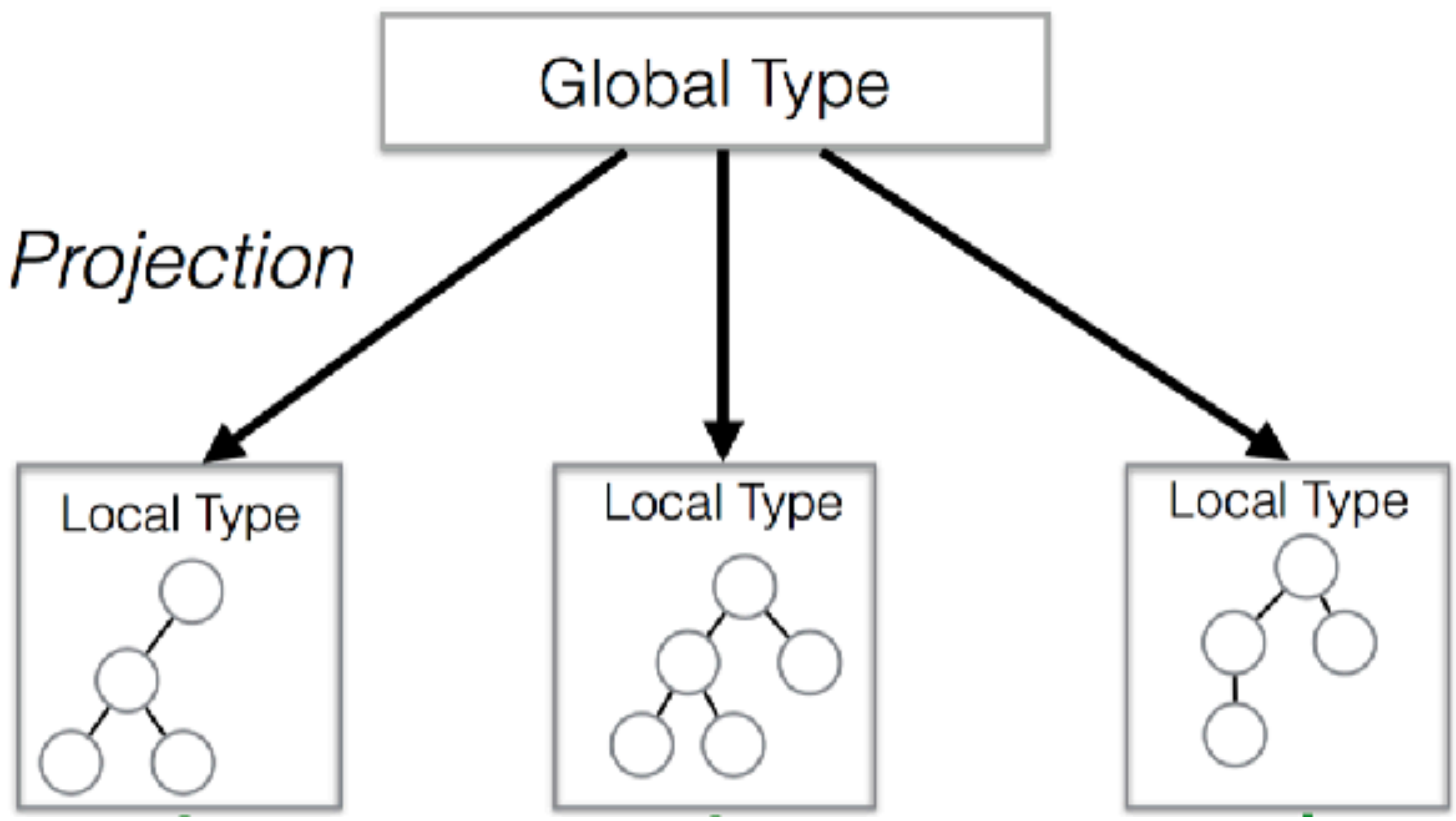
```
1(x:int) from A to B; @x>3
2(y:int) from A to B;
choice at B {3() from B to A; @x>=y}
         or {4() from B to A; @x<y}
```

✓

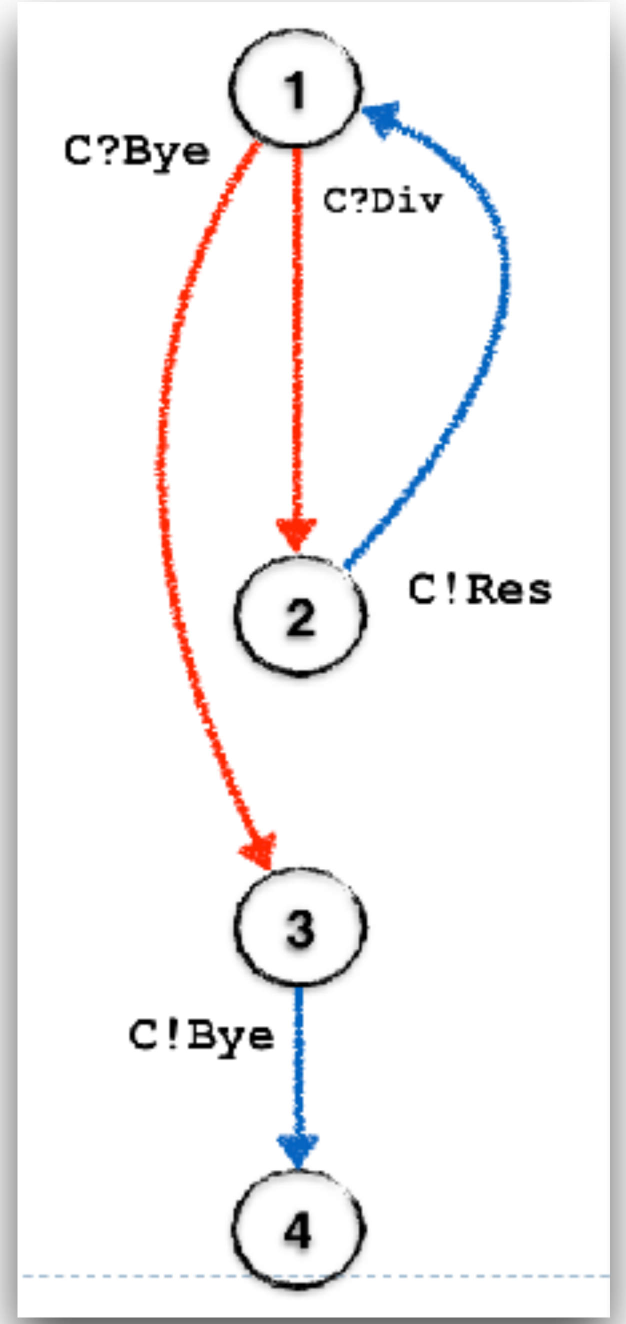
```
1(x:int) from A to B; @x>3
2(y:int) from A to B; @y<=3
choice at B {3() from B to A; @x>y}
         or {4() from B to A; @x<y}
```

✓/✗

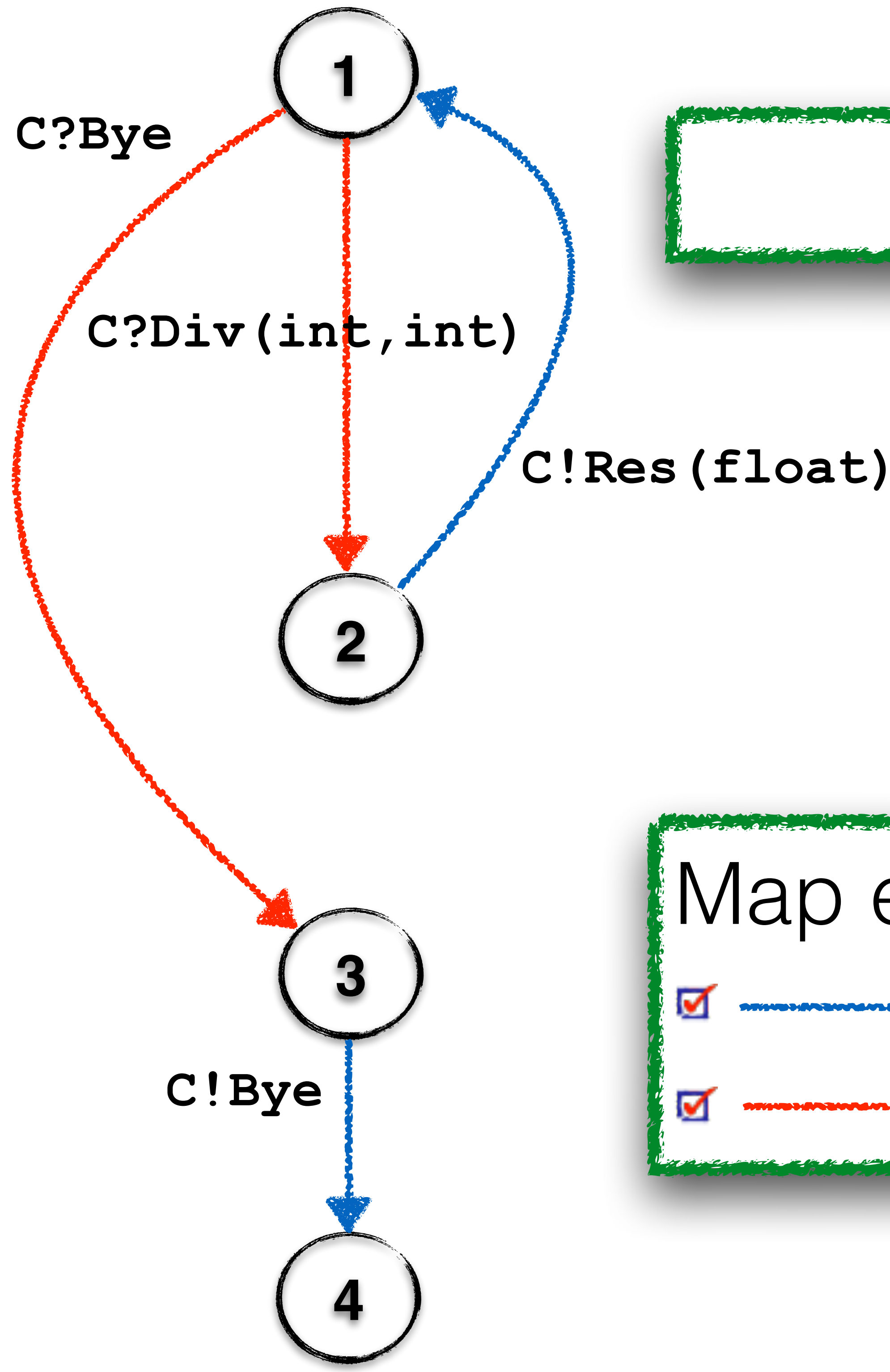
```
(x:T1) from A to B;   (y:T2) from B to C;  (z:T3) from C to A;
```



```
global protocol Calc(role S, role C){
choice at C {
 Div(x:int, y:int) from C to S; @y!=0
 Res(z:float) from S to C;
 do Calc(C, S);
 } or {
 Bye() from C to S;
 Bye() from S to C;
 }
}
```



Model | Properties | CFSM | **F# Type** | Code

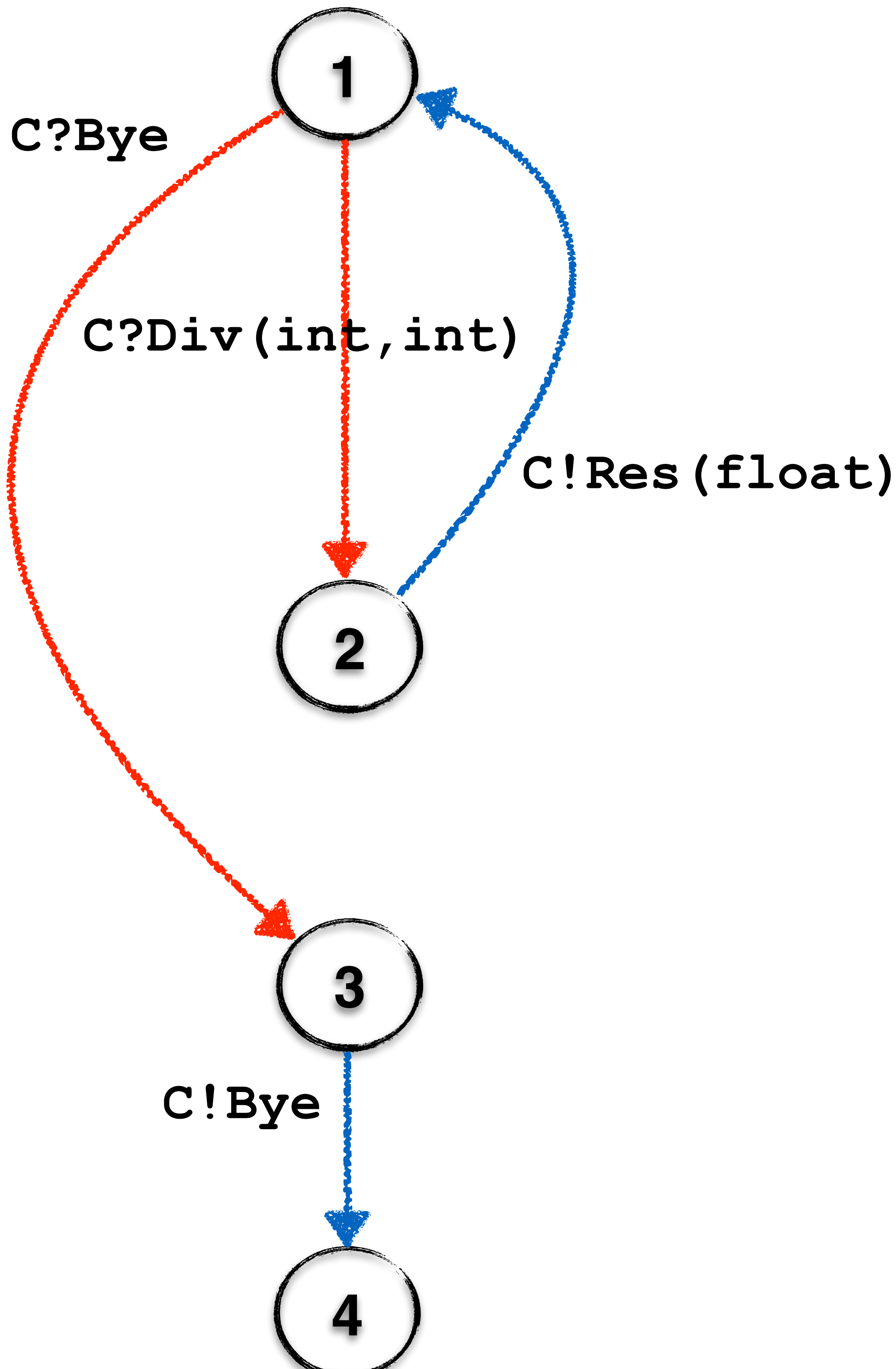


Map each state to a class

Map each transition to a method, e.g:

- ☑ → send method
- ☑ → receive method

Model | Properties | CFSM | **F# Type** | Code

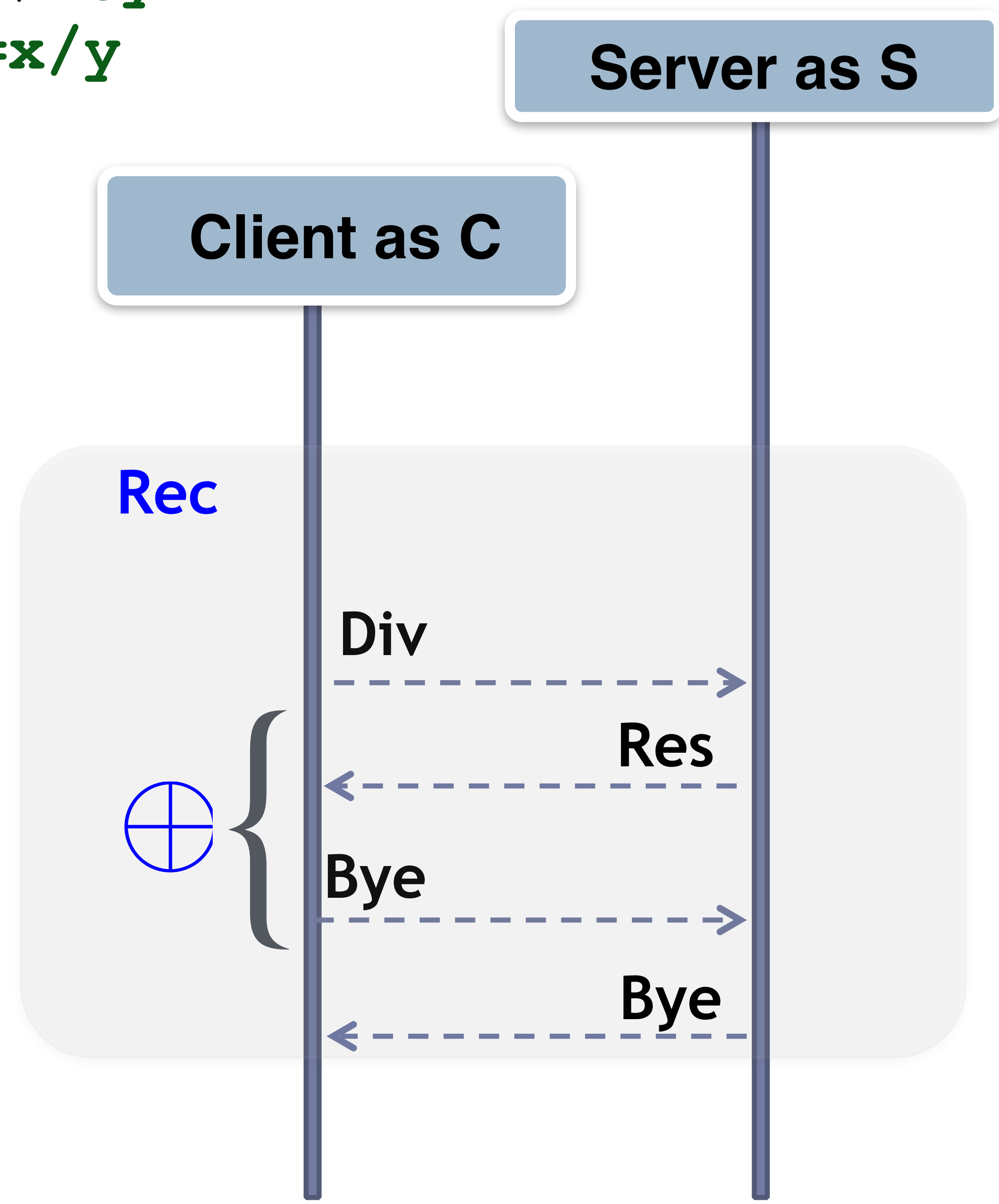


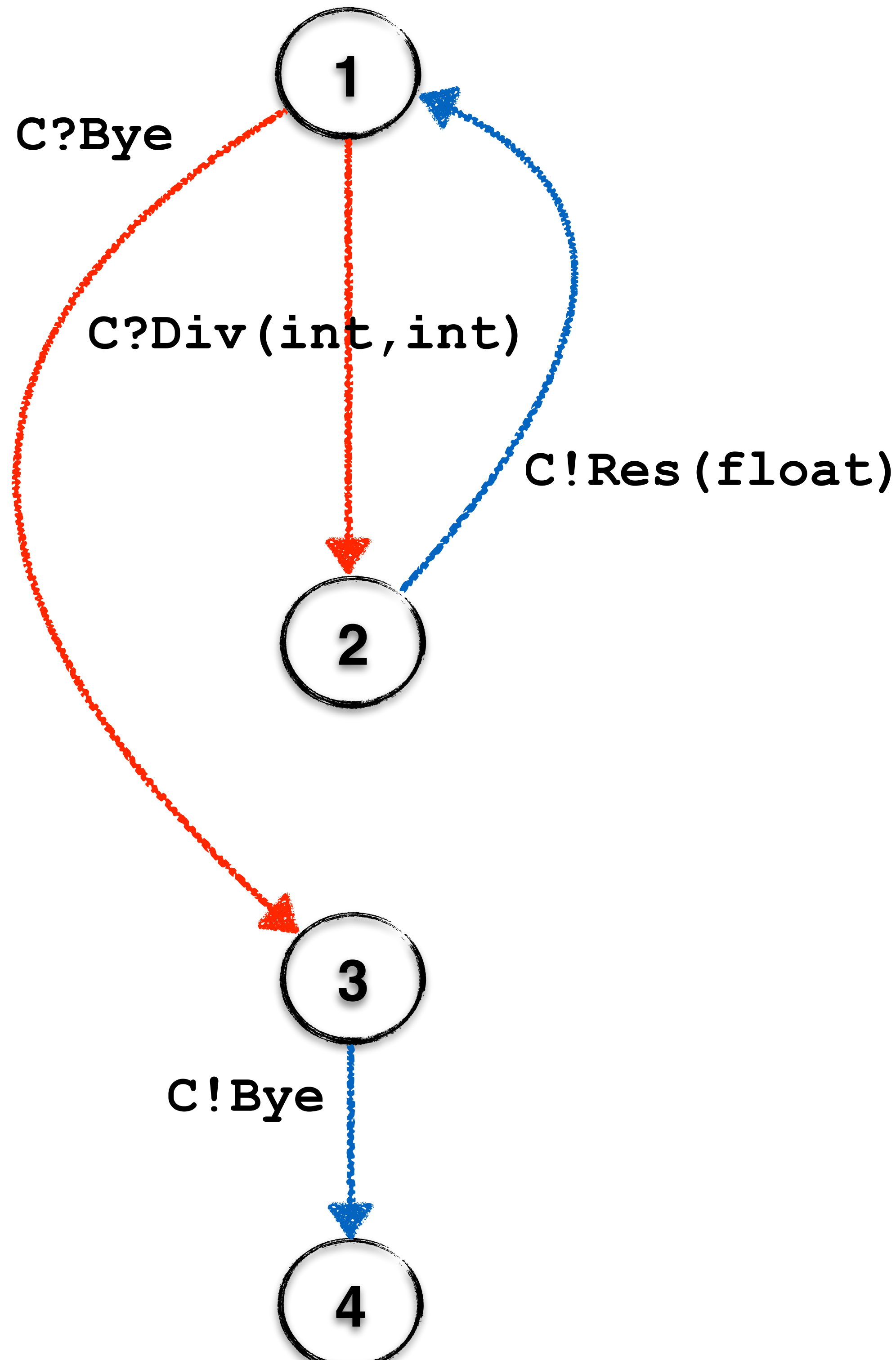

```
type State2 =
    member send: C*Res*float→ State1
```

```
type State3 =
    member send: C*Bye→ State4
```

```
type State4 =
    member finish: unit→ End
```

```
global protocol Calc(role S, role C){
 choice at C {
   Div(x:int, y:int) from C to S; @y!=0
   Res(z:float) from S to C; @z=x/y
   do Addeer(C, S);
 } or {
   Bye() from C to S;
   Bye() from S to C;
 }
}
```





type State1 =
member branch: unit→ ChoiceS1

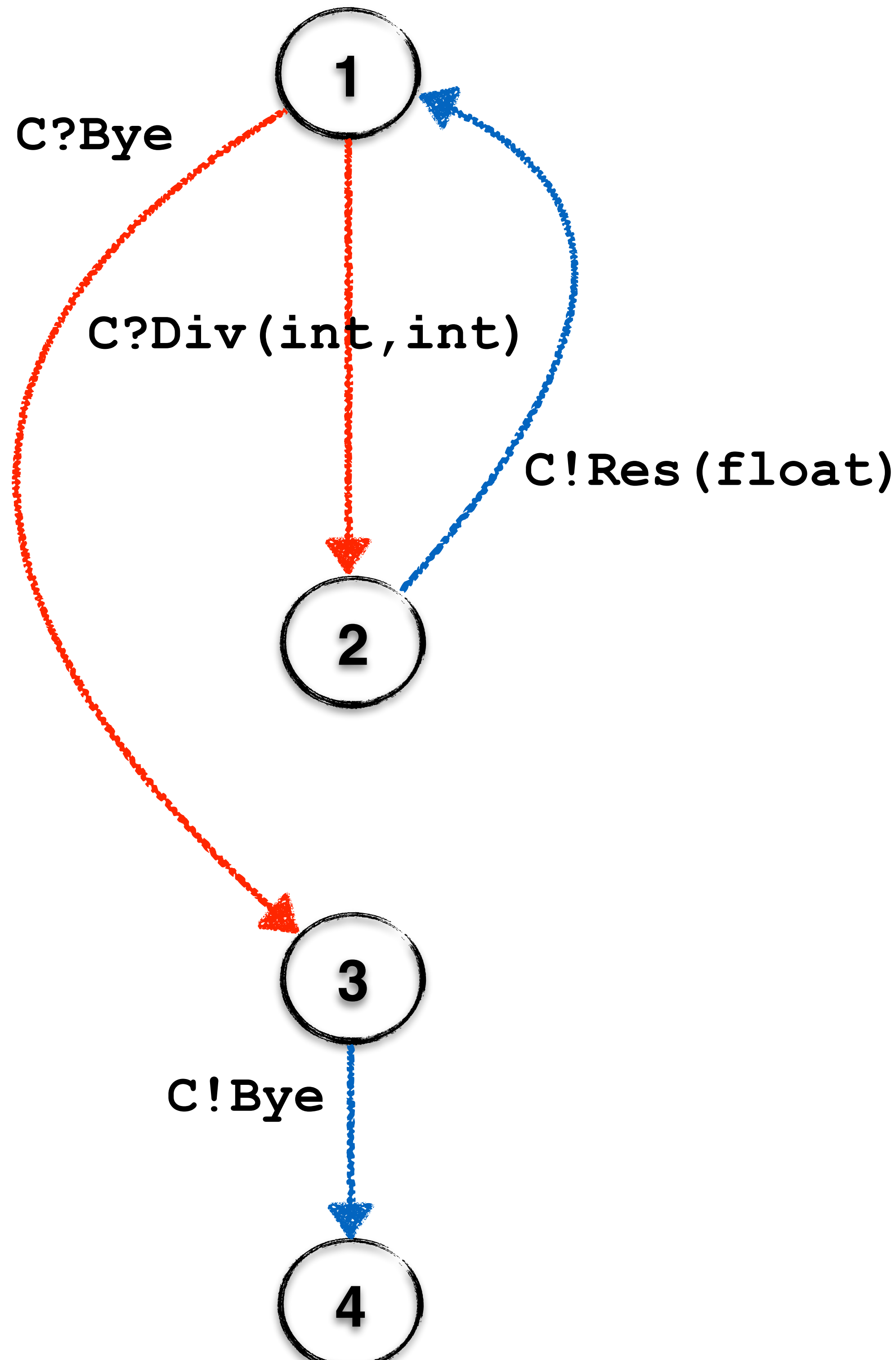
type Div = interface ChoiceS1
    member receive: int*int→ State2

type Bye = interface ChoiceS1
    member receive: → State3

type State2 =
    member send: C*Res*~~float~~→ State1

type State3 =
    member send: C*Bye→ State4

type State4 =
    member finish: unit→ End



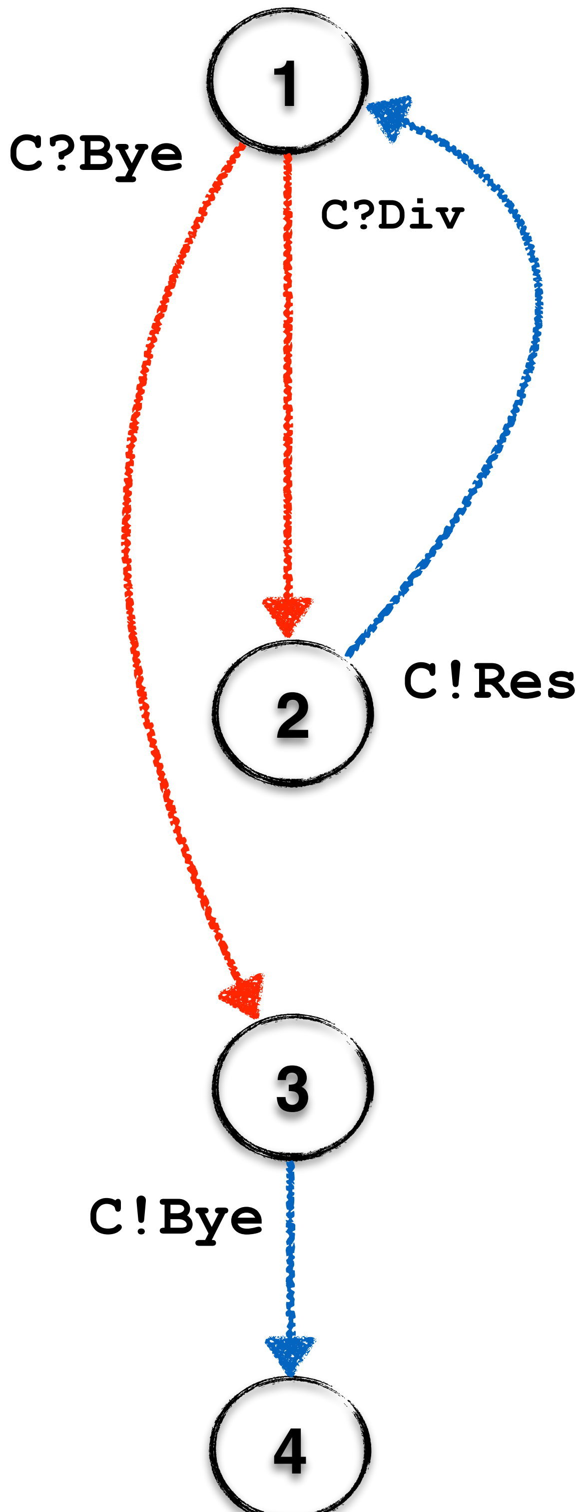

```
type State1 =
member branch: unit→ ChoiceS1


type Div = interface  ChoiceS1
    member receive: int*int→ State2
type Bye = interface ChoiceS1
    member receive: → State3
```

```
type State2 =
    member send: C*Res*float→ State1
```

```
type State3 =
    member send: C*Bye→ State4
```

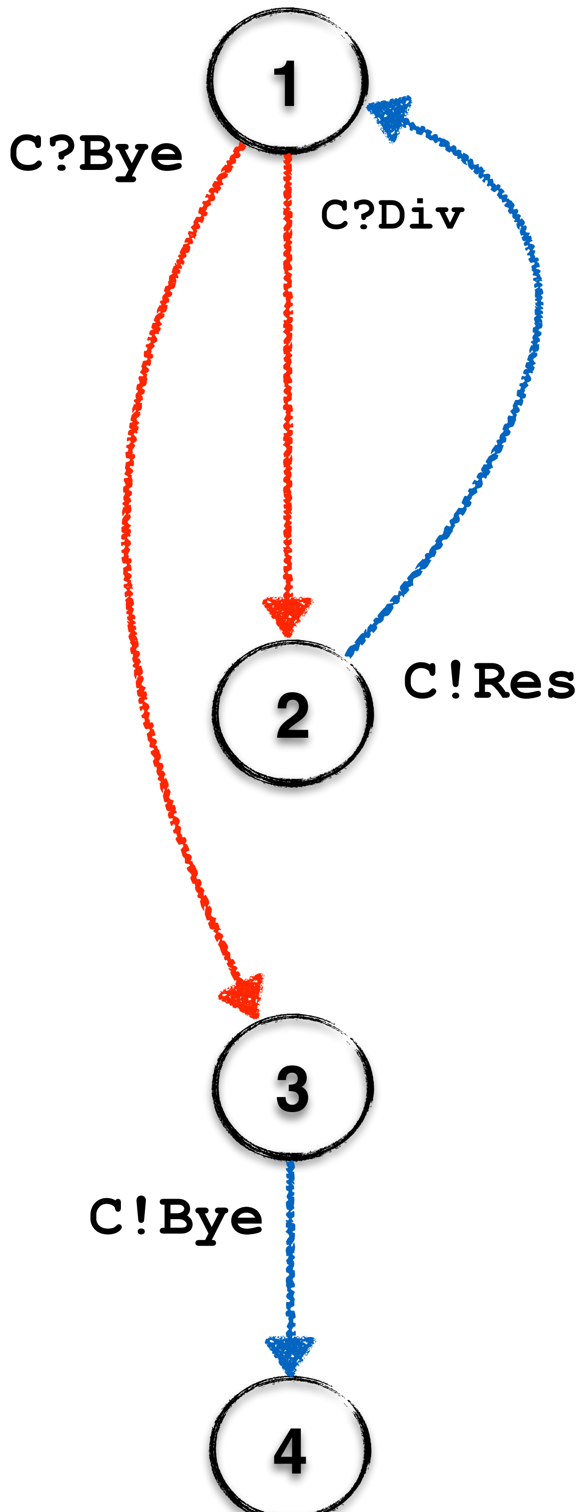
```
type State4 =
    member finish: unit→ End
```

Model Properties CFSM F# Type **Code**




```
let rec calcServer (c:Calc.State1) =

  match c.branch() with
  |:? Calc.Bye as bye->



  |:? Calc.Div as div ->



calcServer c1
```

Model | Properties | CFSM | F# Type | **Code**



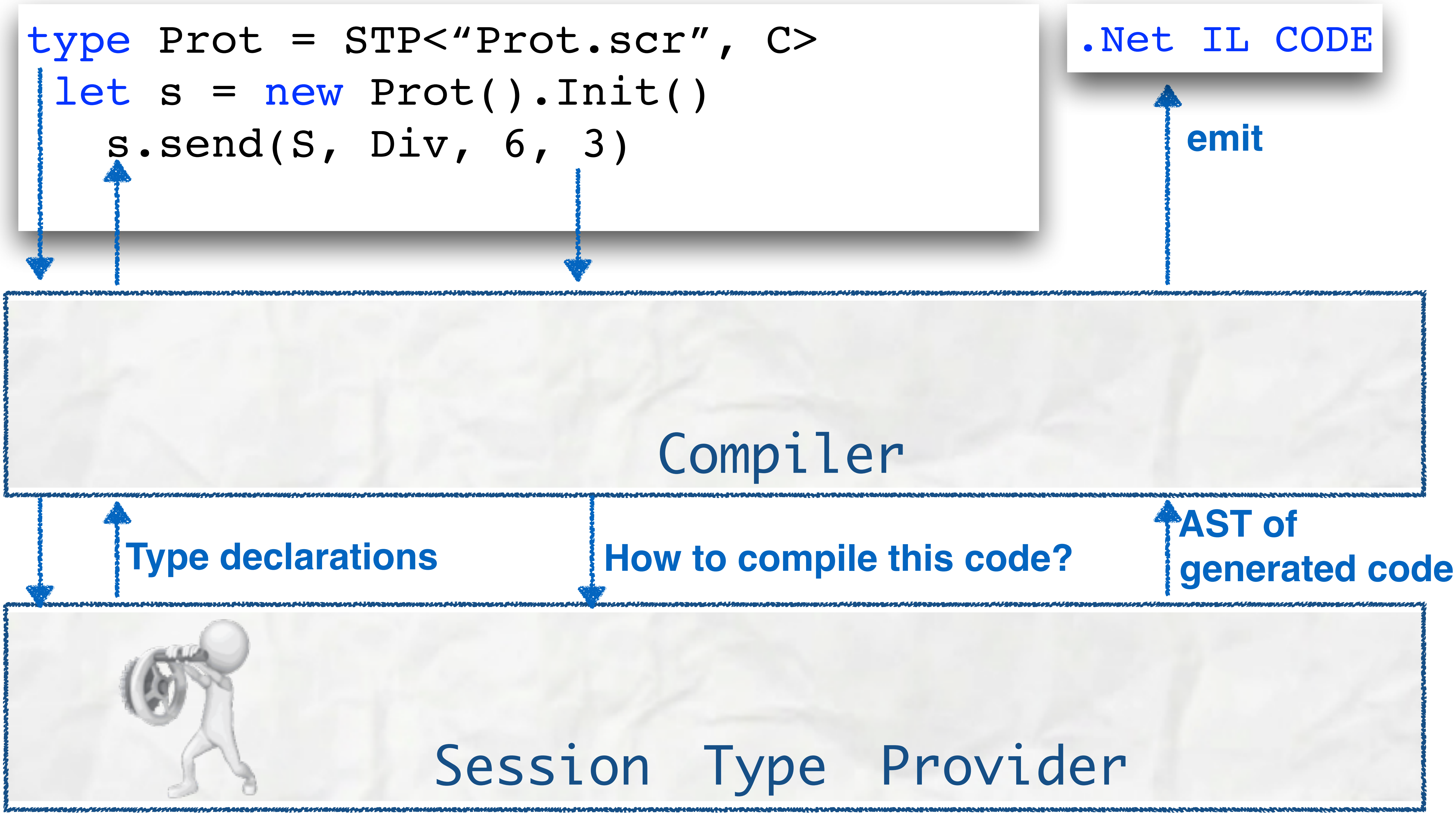

```
let rec calcServer (c:Calc.State1) =
 let x, y = new Buf<int>(),new Buf<int>()
  match c.branch() with
   |:? Calc.Bye as bye->
    bye.receive(C)
        .send(C, Bye).finish()

   |:? Calc.Div as div ->
    let c1 = div.receive(C, x, y)
                  .send(C, Res, x.Val/y.Val)

    calcServer c1
```

send

- constraints as lambda functions
- serialise payload
- manage and use TCP sockets

- ☑ quotations
- ☑ splicing

Model | Properties | CFSM | F# Type | **Code**

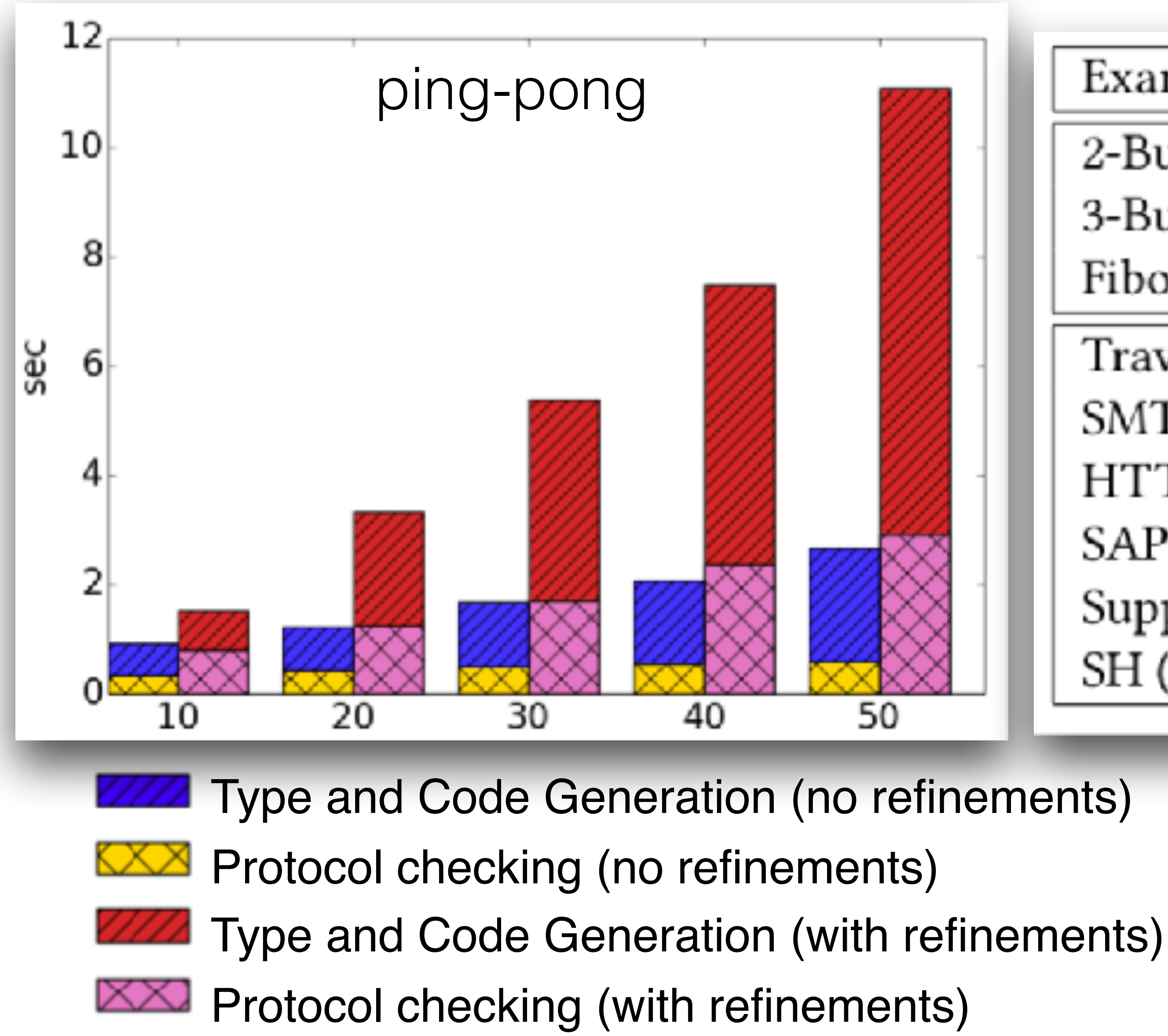


Model Properties CFSM F# Type Code

Safety guarantees

A statically well-typed STP-endpoint program **will never** perform a non-compliant I/O action w.r.t. the source protocol.

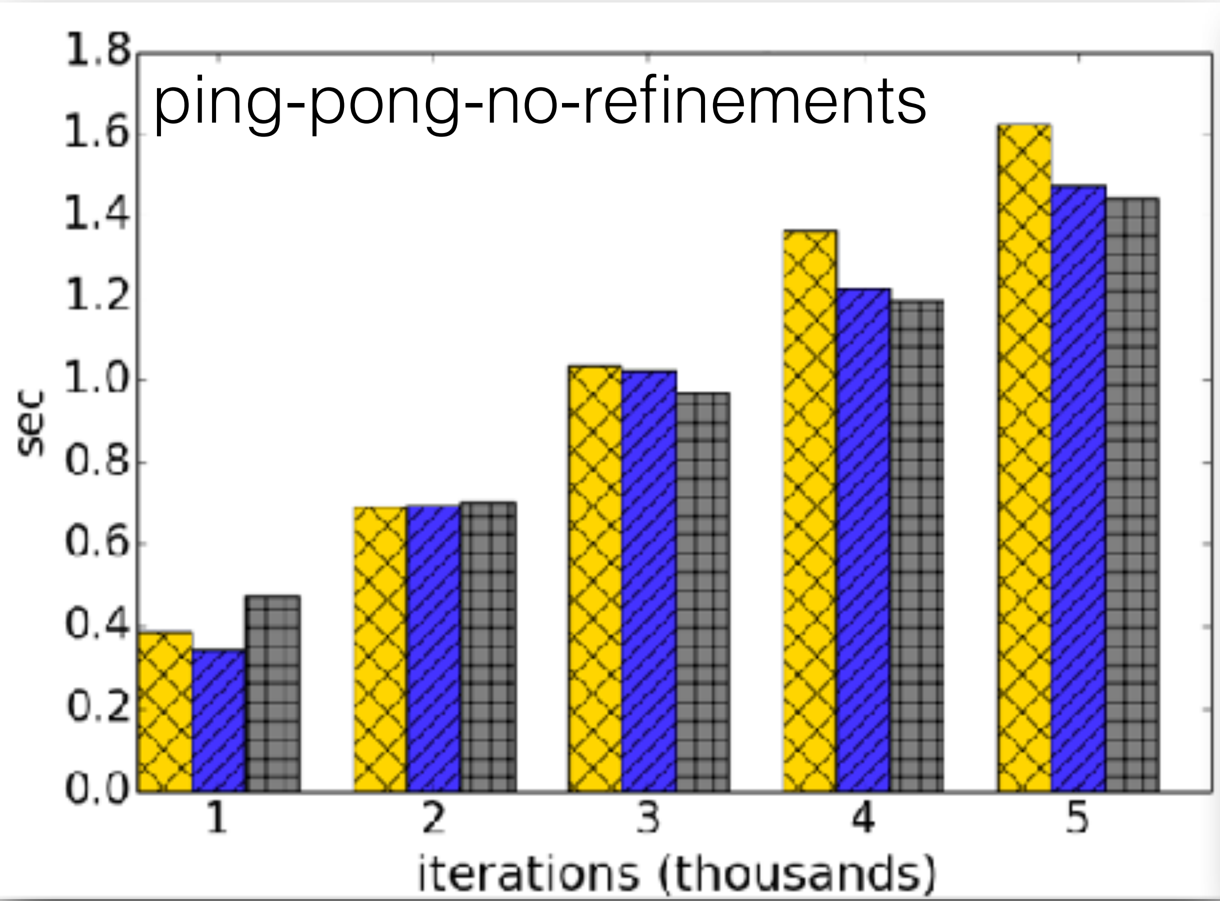
Compile-time performance

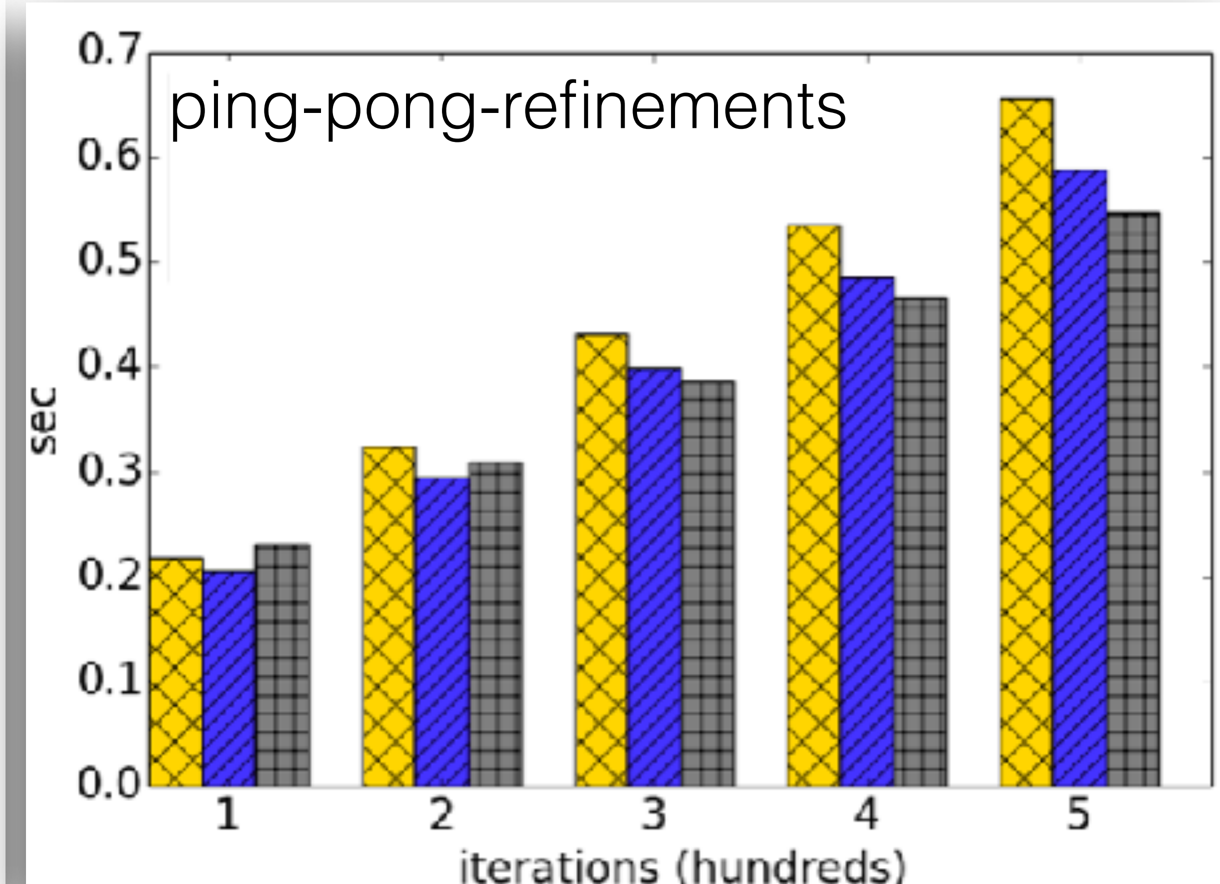


Example (role)	#LoC	#States	#Types	Gen (ms)
2-Buyer (B_1) [13]	16	7	7	280
3-Buyer (B_1) [5]	16	7	7	310
Fibonacci (S) [14]	17	5	7	300
Travel Agency (A) [24]	26	6	10	278
SMTP (C) [14]	165	18	29	902
HTTP (S) [3]	140	6	21	750
SAP-Negotiation (C) [18]	40	5	9	347
Supplier Info (Q) [24]	86	5	25	1582
SH (P)	30	12	15	440

API Generation does not impact the development time

Run-time performance





- Runtime overhead due to:
  - branching, runtime checks, serialisation
- The performance overhead of the library stays in 5%-7% range
- The performance overhead of run-time checks is up to 10%-12%

Future work and Resources

Framework Summary

- Type-driven development of distributed protocols
- Support for refinements on message interactions
- …ask me for more supported features

Future Work

- Static verification of refinements
- Partial model checking
- Support for erased type providers (event-driven branching)

Resources:

- Session type provider: https://session-type-provider.github.io
- Scribble: http://scribble.doc.ic.ac.uk/
- MRG: mrg.doc.ic.ac.uk

Thank you!

Q & A

Questions

Answers

parse-> analyse -> pretty print

Q & A

Questions

parse-> analyse -> pretty print

Answers

Check the tool for more features:

- documentation on the fly
- non-blocking receive
- explicit connections
- recompilation on protocol change
- online vs offline mode
- support by any .Net language

Related work

- Related works on Interaction Refinements
  - A theory of design-by- contract for distributed multiparty interactions [CONCUR’12]
  - Linearly refined session types [LINEARITY’12]
  - A concurrent programming language with refined session types. [BEAT’13]
  - Certifying data in multiparty session types [JLAMP’17]

- no implementation
- based on syntactic checks
- developed for pi-calculus