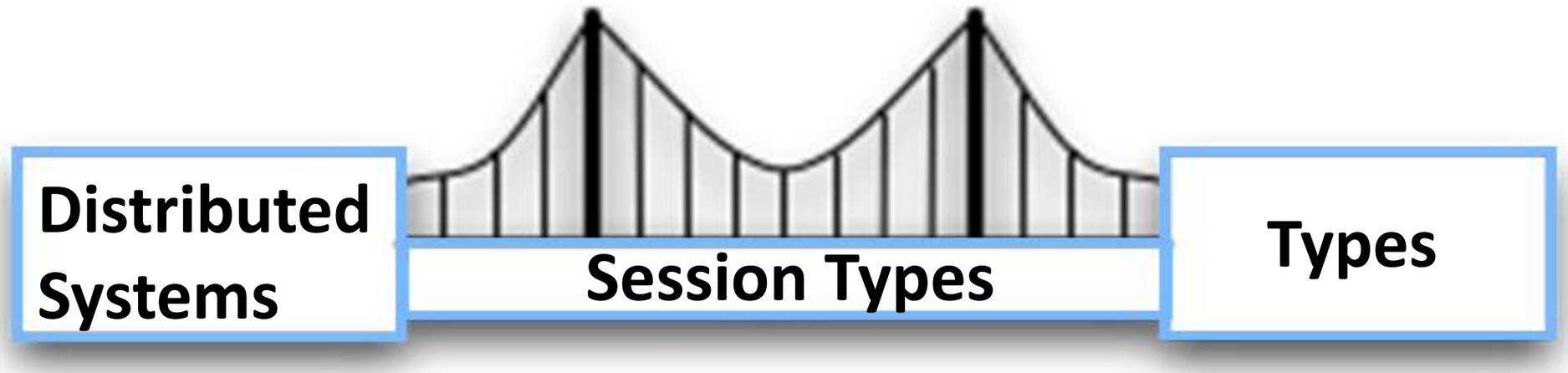


A flavor of Session Types

Rumyana Neykova, Nobuko Yoshida

Observation 1: Types

- One of the computing most successful concepts
- Codify the structure of the data
- Serve as a fundamental unit of compositionality
- Allow easy error prevention
- Appears from the oldest to the newest programming languages

> Robin Milner: Types are the yeast of computer programming: they make it digestible !!!

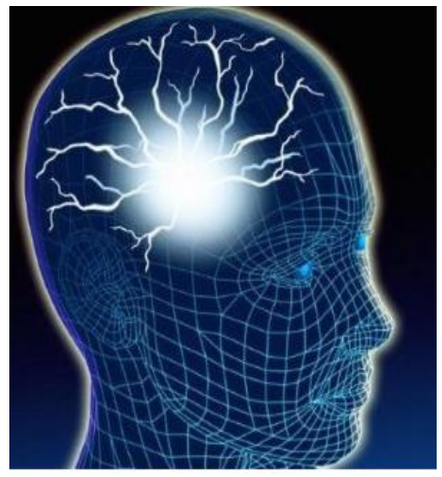
Observation 2: But distributed systems ...

focus on the

communication

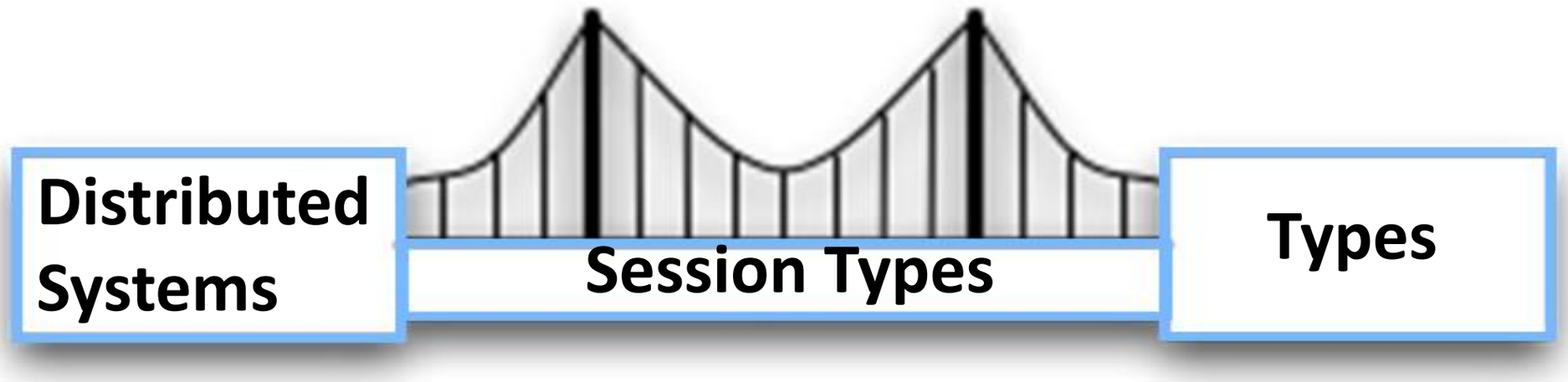
not on

computation



Then…

- Can we have **types that describe the communication**, not the computation ?



- How to **formally** abstract/specify and **practically** implement/control communications?

Session types to the rescue

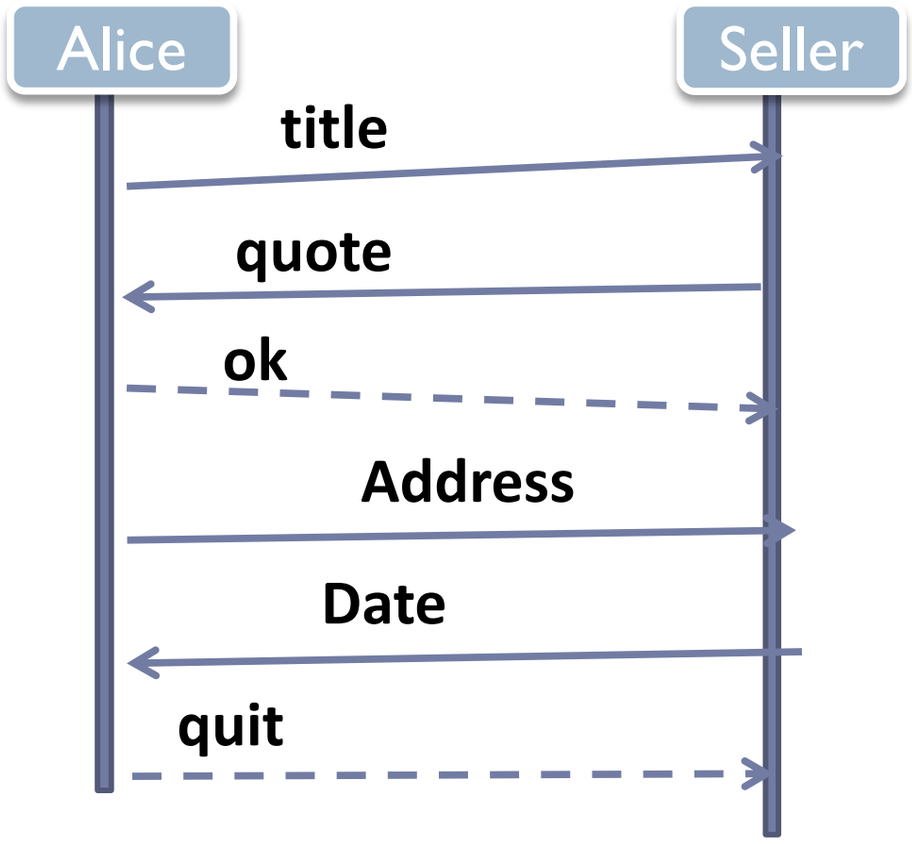
- Primitives – to build the types
  - send, receive (well , there are few more, but it boils down to these two ☺)

send(int).send(int).receive(bool)

- Context – to be checked by the type system
  - protocols – describe the communication between processes

SESSION= STRUCTURED SEQUENCE OF INTERACTIONS

A Protocol



- Protocol: Buyer-Seller
- Description: Alice buying a book

send(string).receive(int).⊕{ok: send(string).receive(date), quit:end}

receive(string).send(int).&{ok: receive(string).send(date), quit: end}

Are we compatible?

send(int).send(int).receive(bool)

receive(int).receive(int).send(bool)

It is all about duality!

Are we compatible?

receive(int).send(int).receive(bool)

receive(int).receive(int).send(bool)

Session Types in a Nutshell

SESSION = STRUCTURED SEQUENCE OF COMMUNICATION

send(int).send(int).receive(bool)

"…Session Types ***structure*** a ***series of interactions*** in a simple and concise syntax and ensure ***type safe communication***."

What is type safe communication?

Communication Safety

- No communication mismatch

Session Fidelity

- Communication follow the described protocol

Progress

- No deadlock/ stuck in a session

Applications

Scribble

Session C

Java

ERLANG

RabbitMQ
Messaging that just works

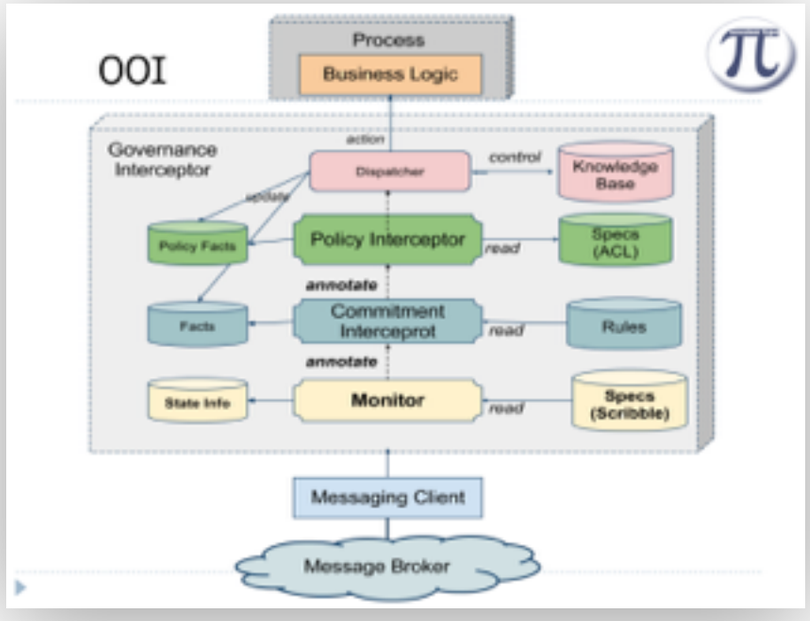
python

AMQP
Advanced Message Queuing Protocol

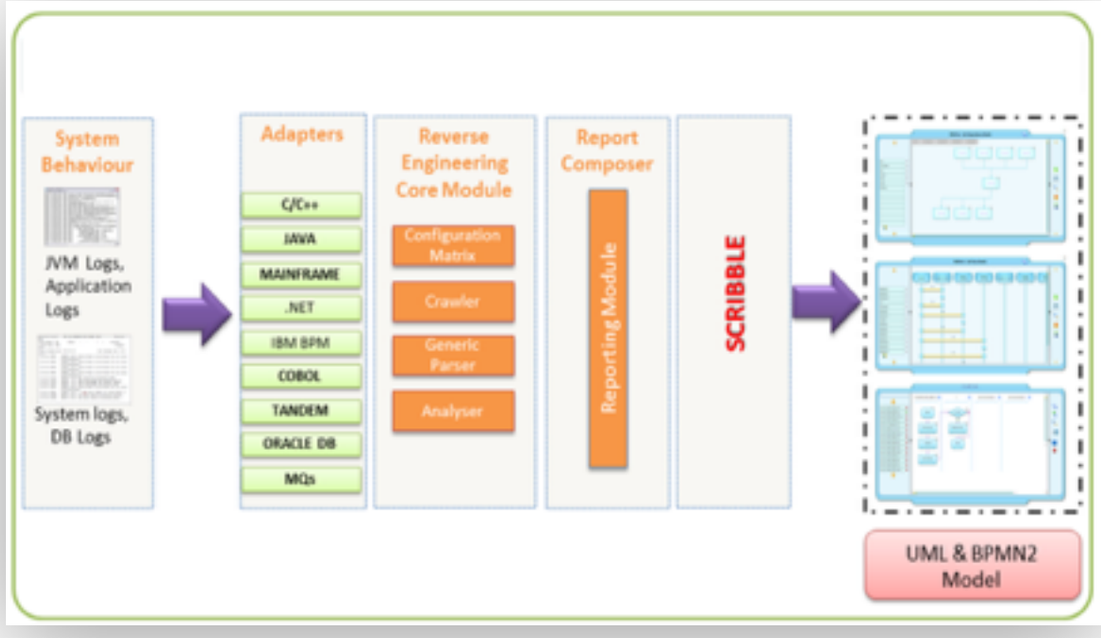
OOI
OCEAN OBSERVATORIES INITIATIVE

Session Type based Tools

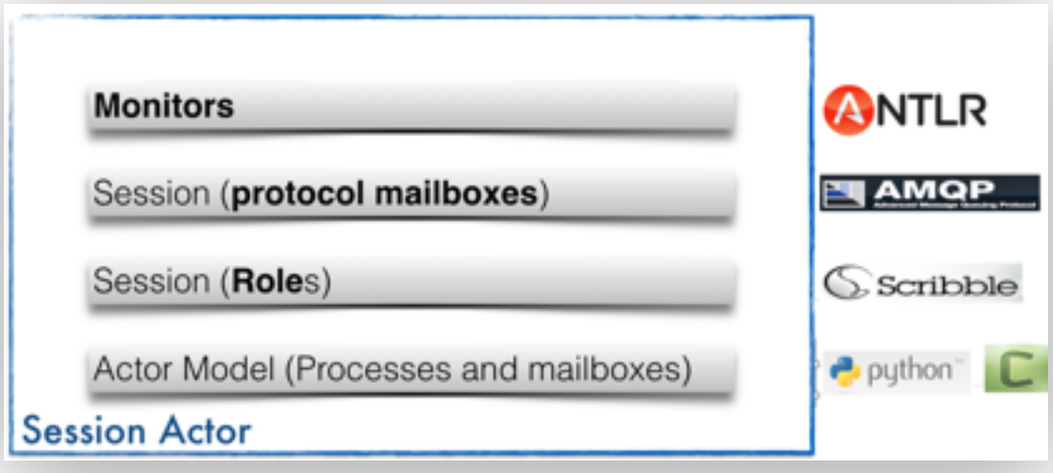
OOI Governance



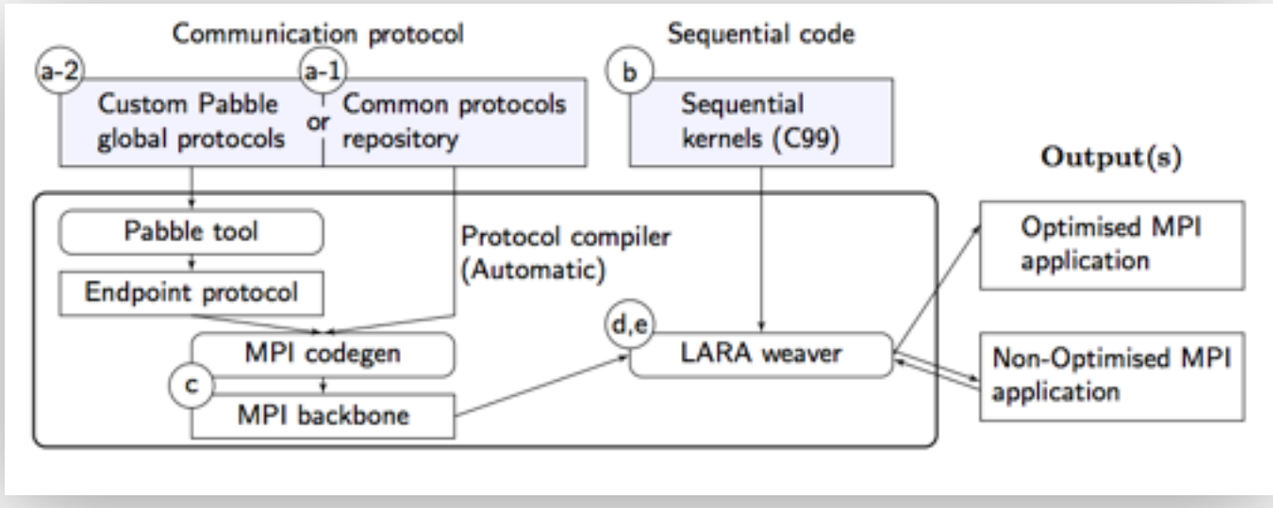
ZDLC: Process Modeling



Actor Verification



MPI code generations



www.scribble.org

Scribble

Fork me on GitHub

What is Scribble?

Scribble is a language to describe application-level protocols among communicating systems. A protocol represents an agreement on how participating systems interact with each other. Without a protocol, it is hard to do meaningful interaction: participants simply cannot communicate effectively, since they do not know when to expect the other parties to send data, or whether the other party is ready to receive data.

However, having a description of a protocol has further benefits. It enables verification to ensure that the protocol can be implemented without resulting in unintended consequences, such as deadlocks.

An example

```
module examples;

global protocol HelloWorld(role Me, role World) {
        hello(Greetings) from Me to World;
        choice at World {
                goodMorning(Compliments) from World to Me;
        } or {
                goodAfternoon(Salutations) from World to Me;
        }
}
```

A very simply example, but this illustrates the basic syntax for a hello world interaction, where a party performing the role Me sends a message of type *Greetings* to another party performing the role 'World', who subsequently makes a decision which determines which path of the choice will be followed, resulting in a *GoodMorning* or *GoodAfternoon* message being exchanged.

Find out more ...

Language Guide | Tools | Specification | Forum

Describe

Scribble is a language for describing multiparty protocols from a global, or endpoint neutral,

Verify

Scribble has a theoretical foundation, based on the Pi Calculus and Session Types, to ensure that protocols described using the language are sound, and

Project

Endpoint projection is the term used for identifying the responsibility of a particular role (or

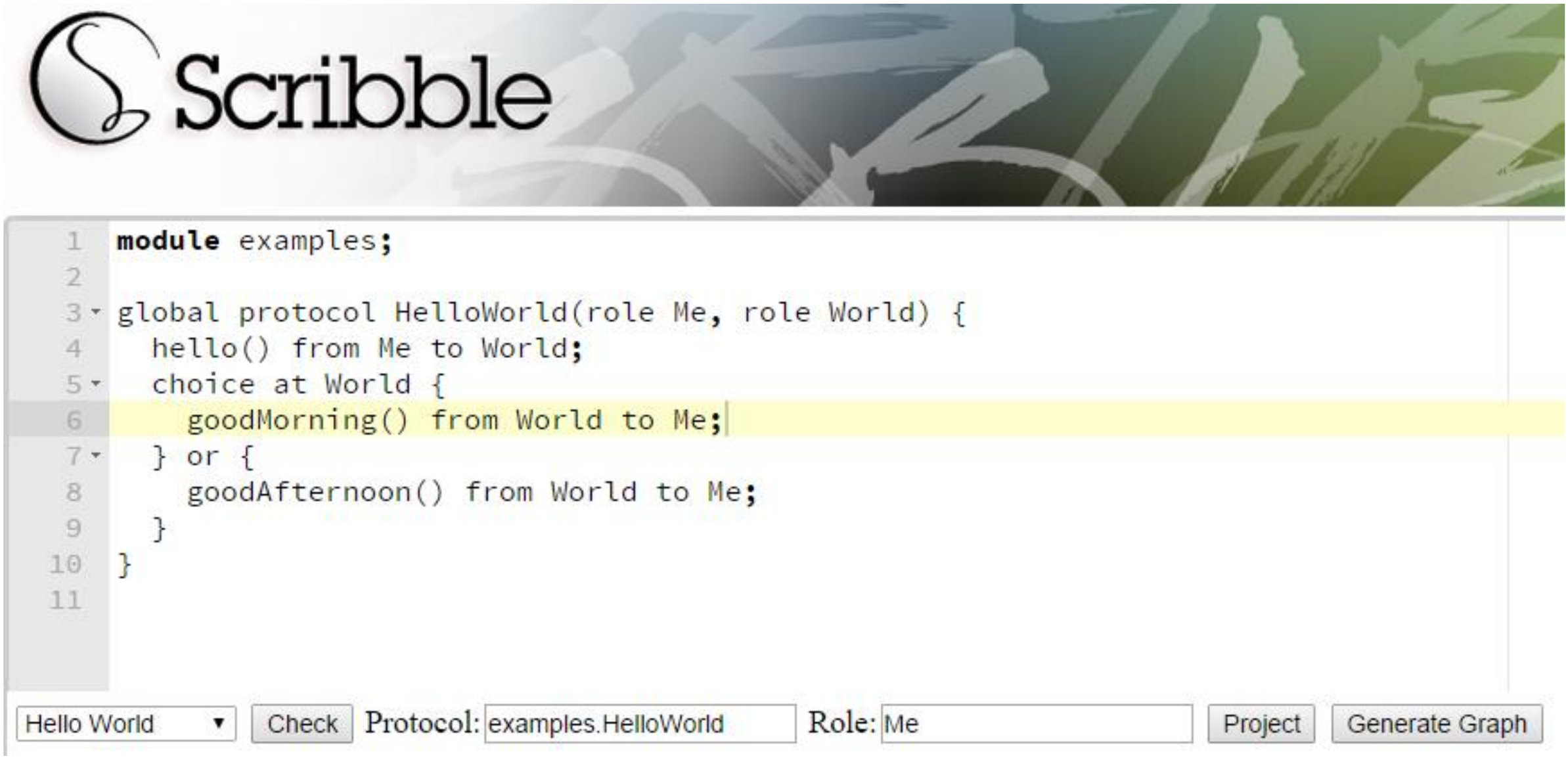
Implement

Various options exist, including (a) using the endpoint projection for a role to generate a skeleton code, (b) using session type APIs to clearly describe the behaviour, and (c)

Monitor

Use the endpoint projection for roles defined within a Scribble protocol, to monitor the activity of

http://scribble.doc.ic.ac.uk:55000/



Scribble: www.scribble.org

protocol def
recursion
send-receive
choice

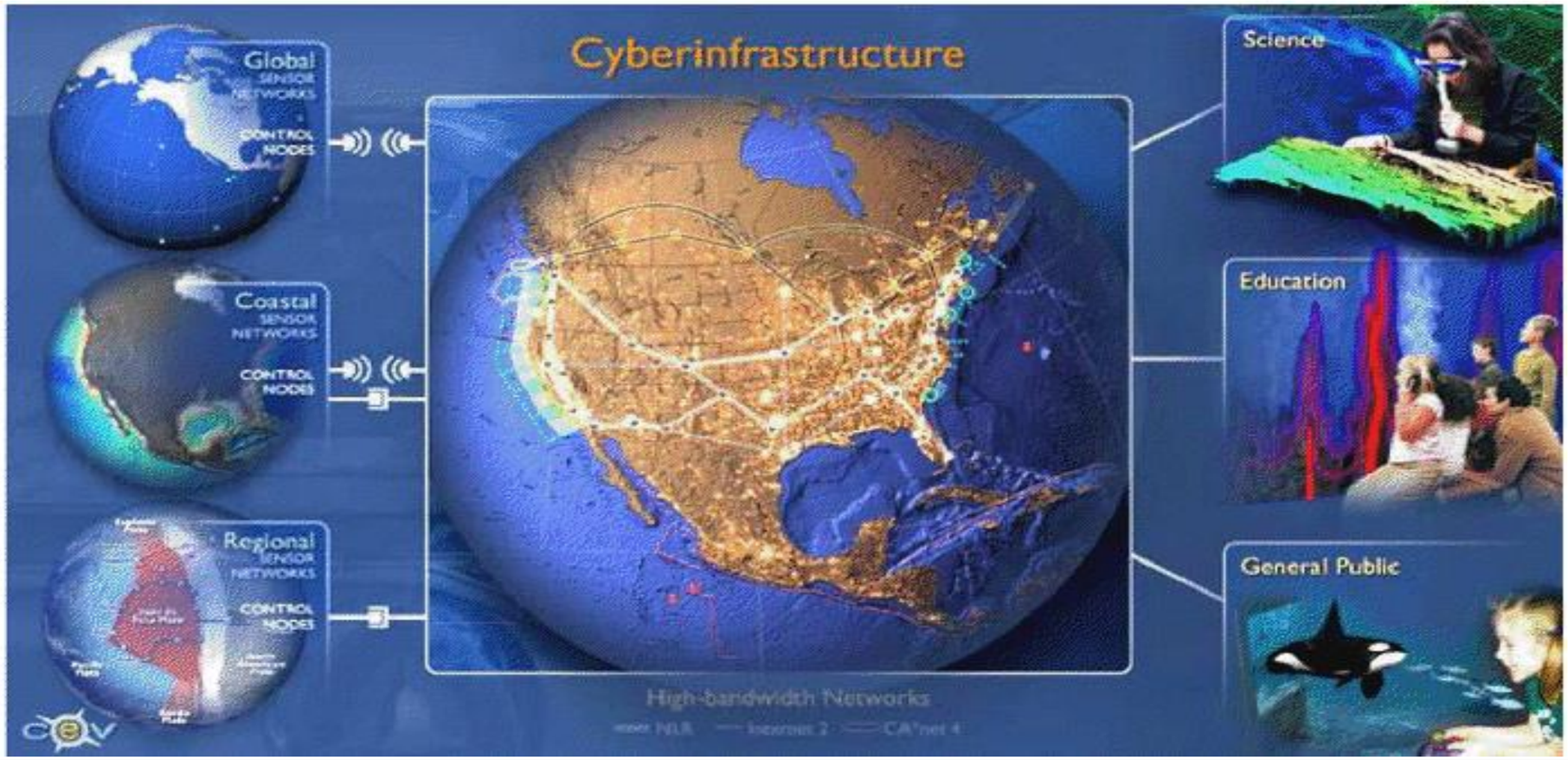
```
protocol Q&A(me,  you){
rec loop {
      ask(question: string) from you to me;
      choice at me
            { response (string) from me to you;
              loop; }
   or { enough() from me to you; }}
```

Scribble SMTP

TABLE OF CONTENTS

Case Study: OOI

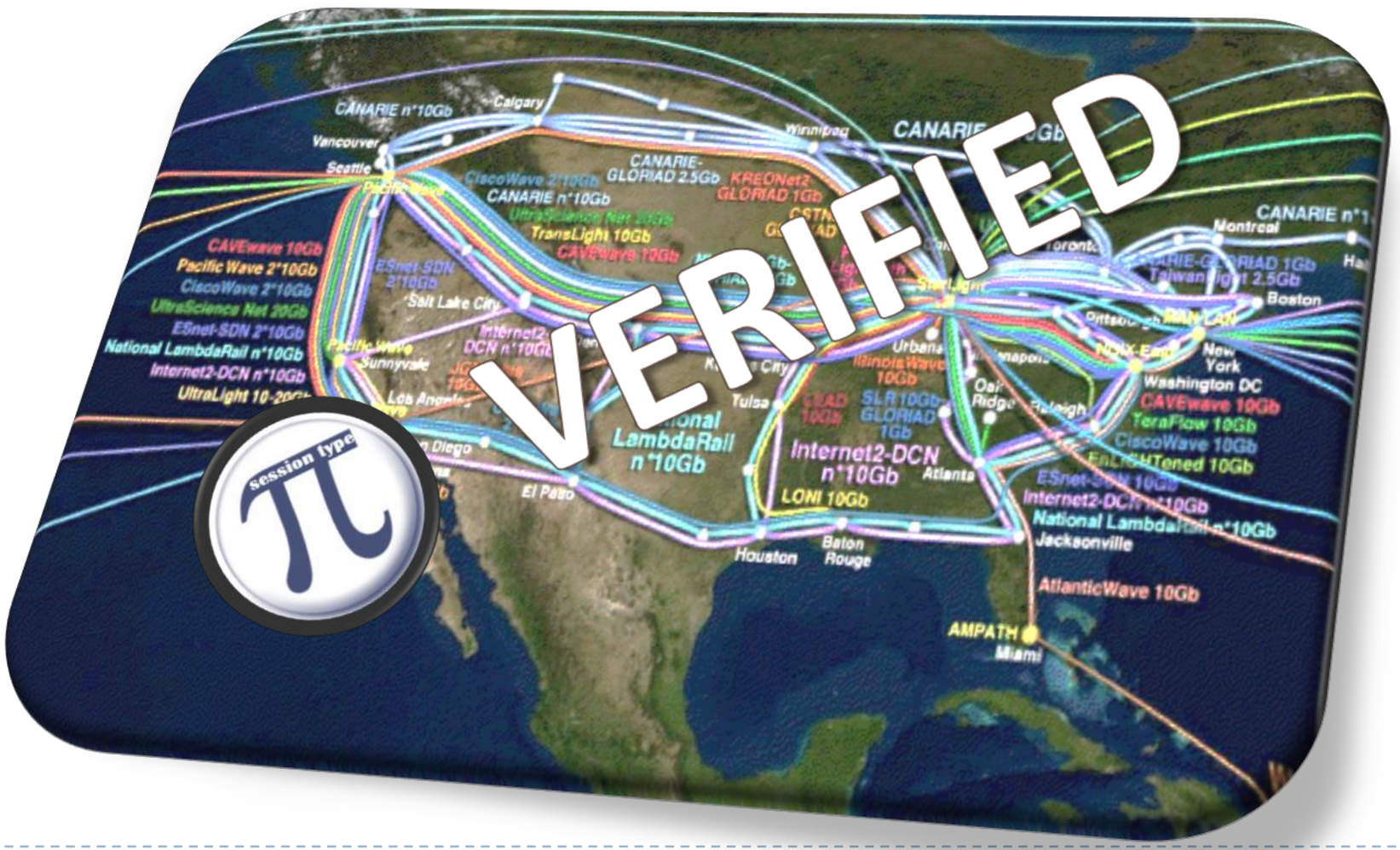
OOI aims: to deploy an infrastructure (global network) to expand the scientists' ability to remotely study the ocean



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

Checking the ocean

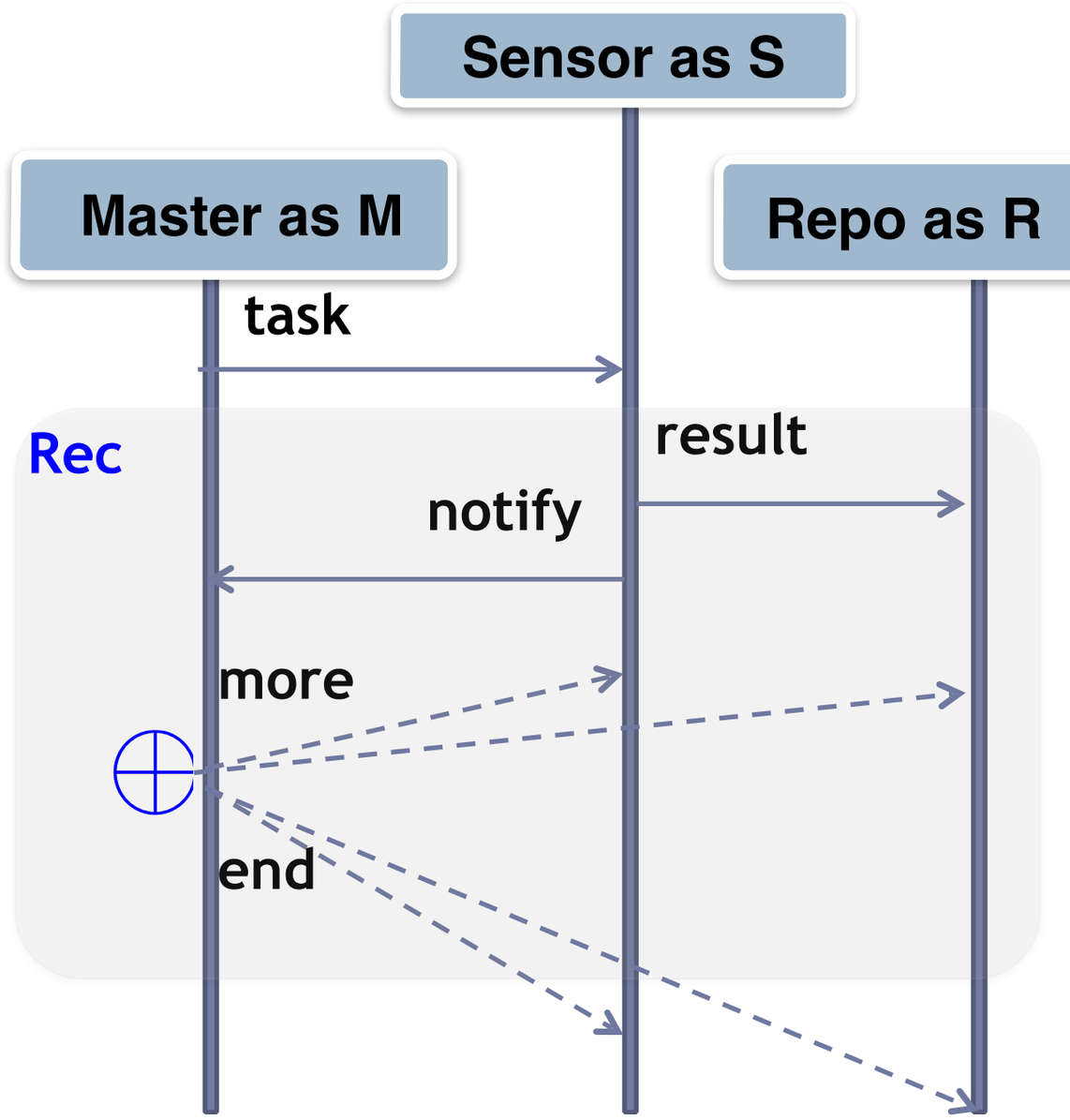
Collaboration: build session types layer on top of OOI network to guarantee global safety



A protocol in Scribble

```
global protocol TempMeasurement(
  role M, role S, role R) {

  task from M to S;
  rec Loop {
    result from S to R;
    notify from S to M;
    choice at M{
      more from M to S;
      more from M to R;
      continue Loop;
    } or {
      end from M to S;
      end from M to R;
    }
}
```



Applications…food for though

Type are used for early error detection

Types are used to optimise the memory layout

Types are used as a guidance when we design programs

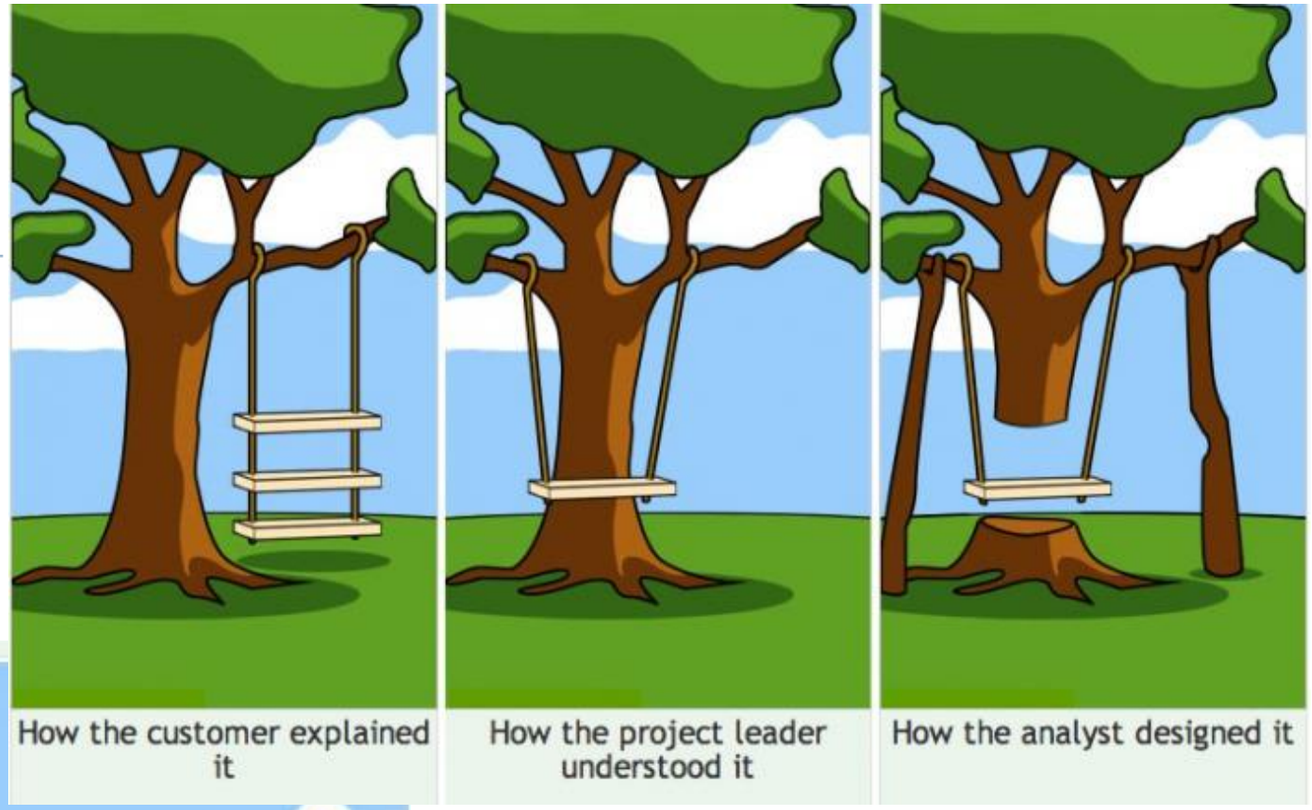
There is so much garbage here ☹

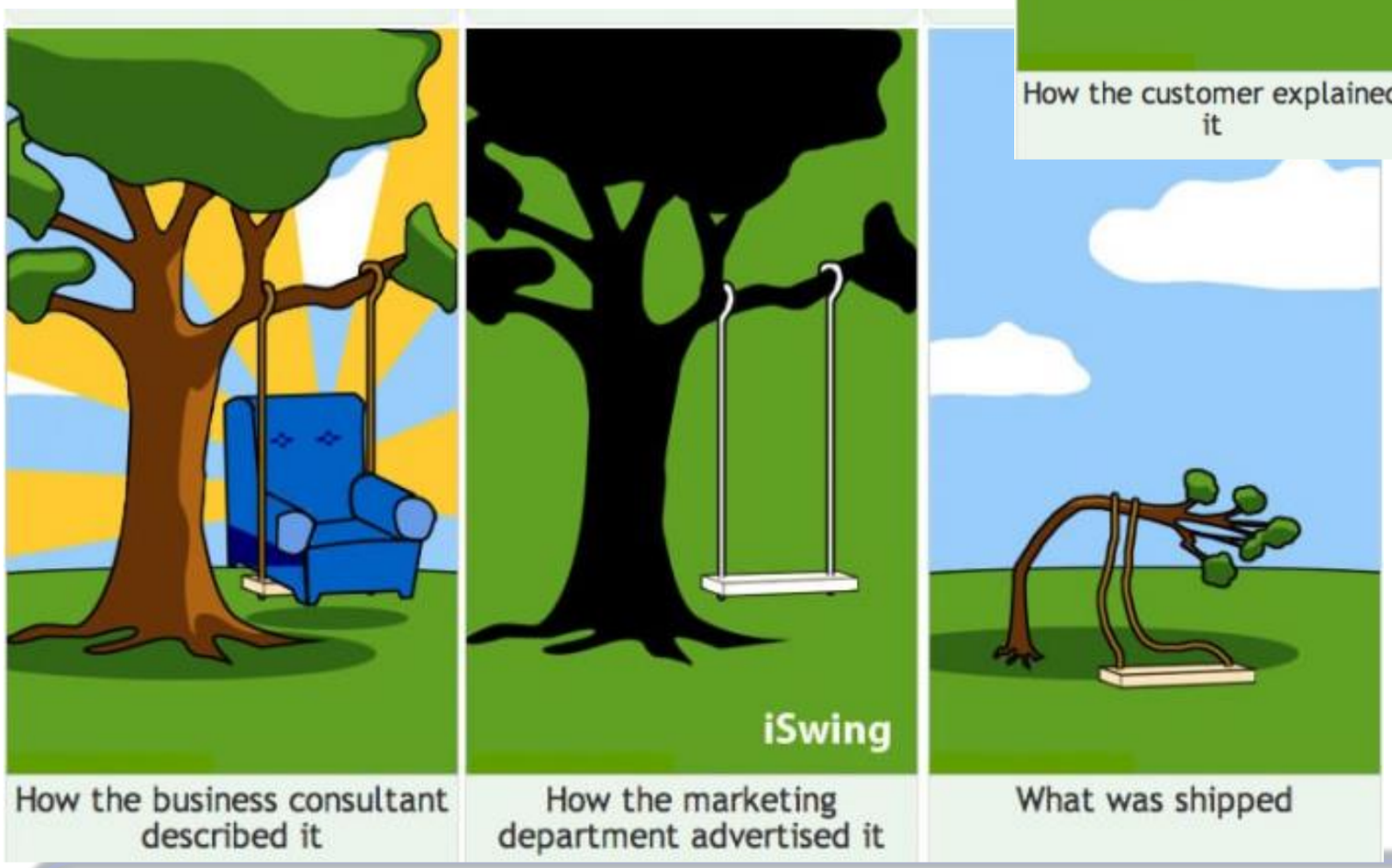
Session types prevent cluttering the network with nonsense messages.

Hey, same direction ..will give you a ride.

Session types can be used to optimise the communication flow, grouping messages together

Lost in Translation





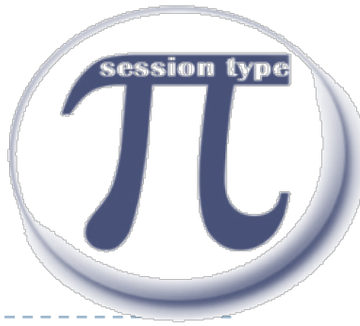
Session types can be used for Testable architectures -

where the specification match the implementation

It is your turn …

```
protocol Q&A(you, me)
{
    rec Loop
     {
          Questions from you to me;
          Answers from me to you;
          Loop;
     }
}
```

Ok … why we need types again?



- Having a context allows to control the communication
- Having granularity allows to put constraints on the right place
- Early error detection is much cheaper
- More documented code
- Nice abstraction means easy programming – you program with send and receive (no threads, sockets, channels)