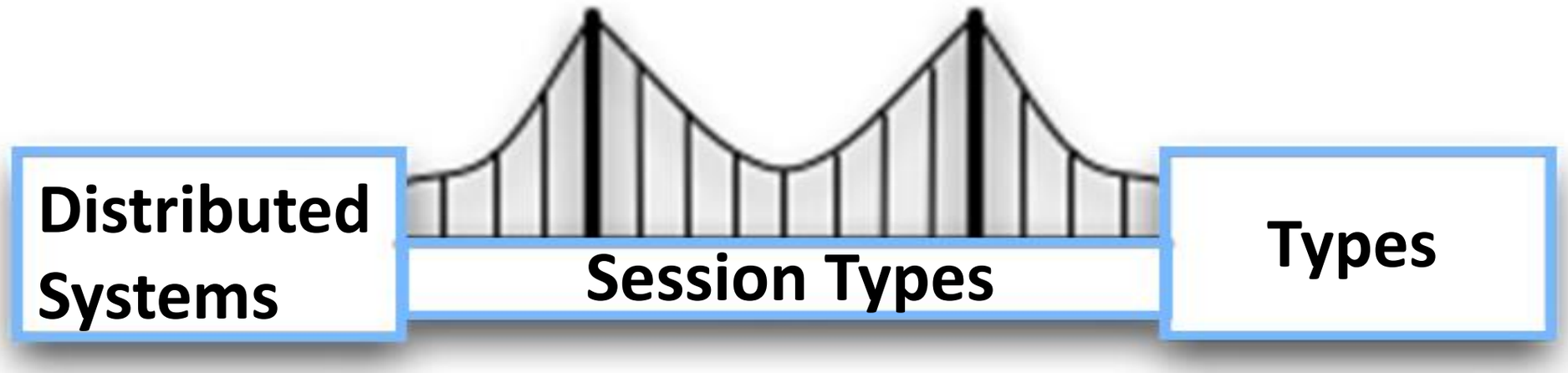


A flavor of Session Types

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 !!!

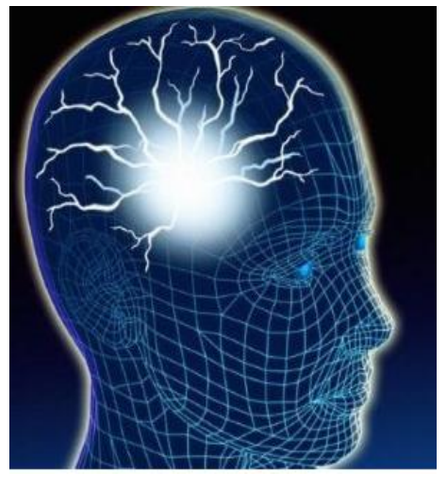
Observation 2: But distributed systems ...

focus on the

communication

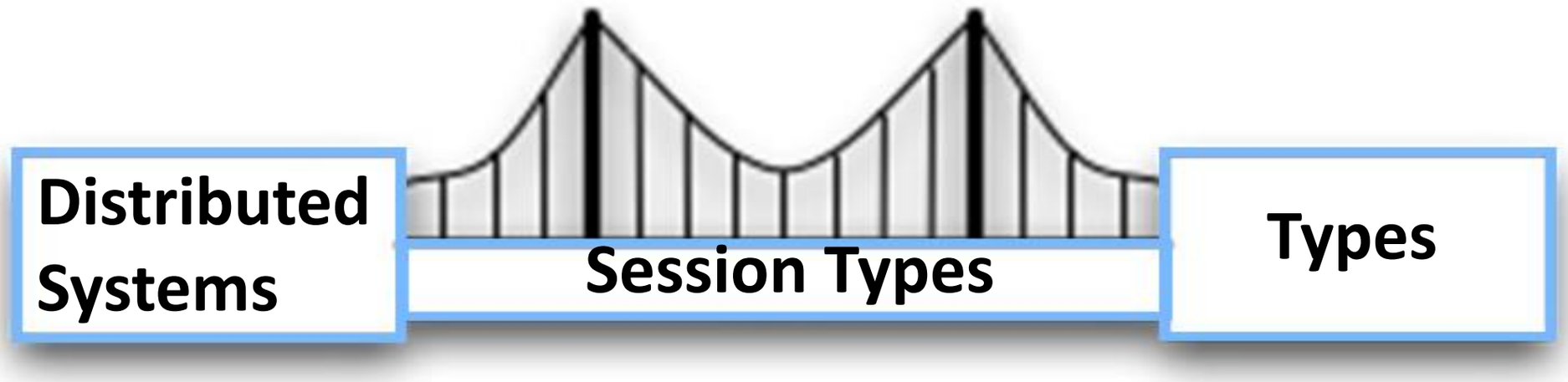
not on

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

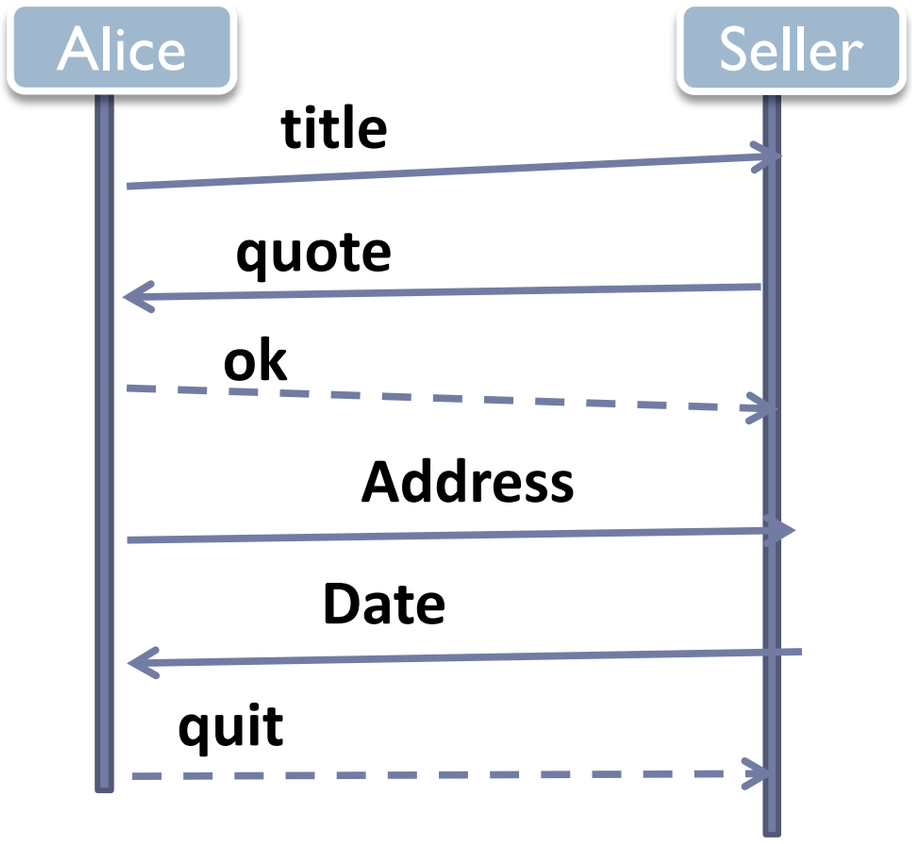
- Primitives – to build the types
  - send, receive (well , there are few more, but it boils down to these two ☺)

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

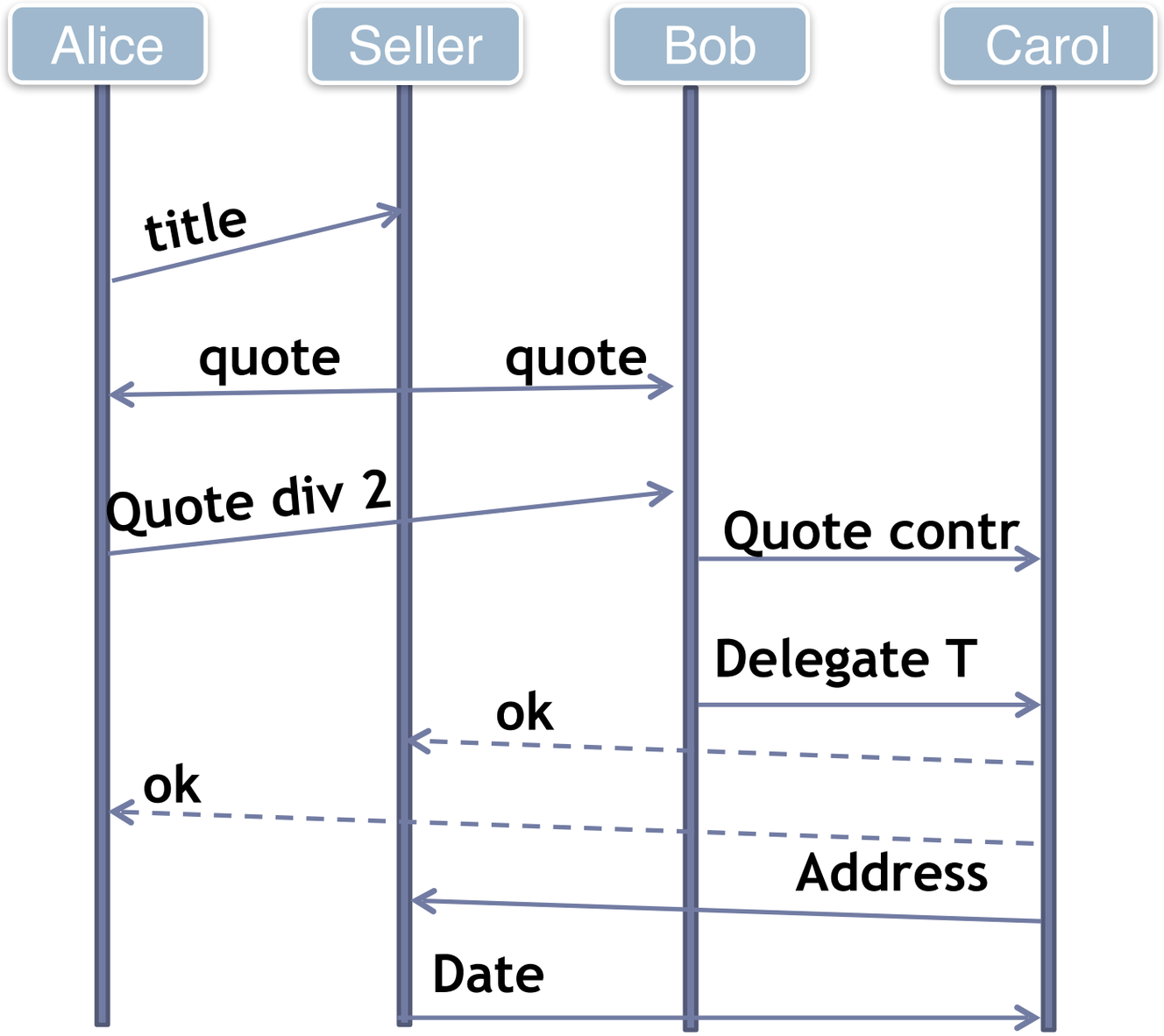
Session Fidelity

- Communication follow the described protocol

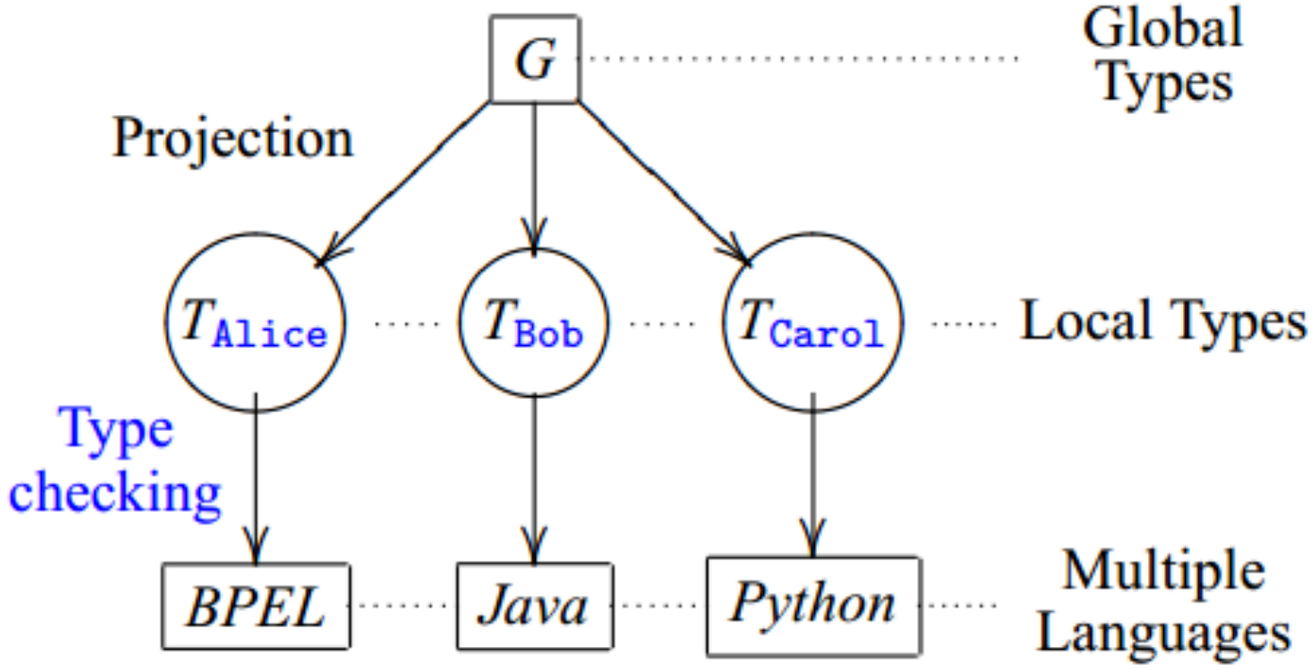
Progress

- No deadlock/ stuck in a session

Wait a minute! What if it is more than 2?



How does it work?



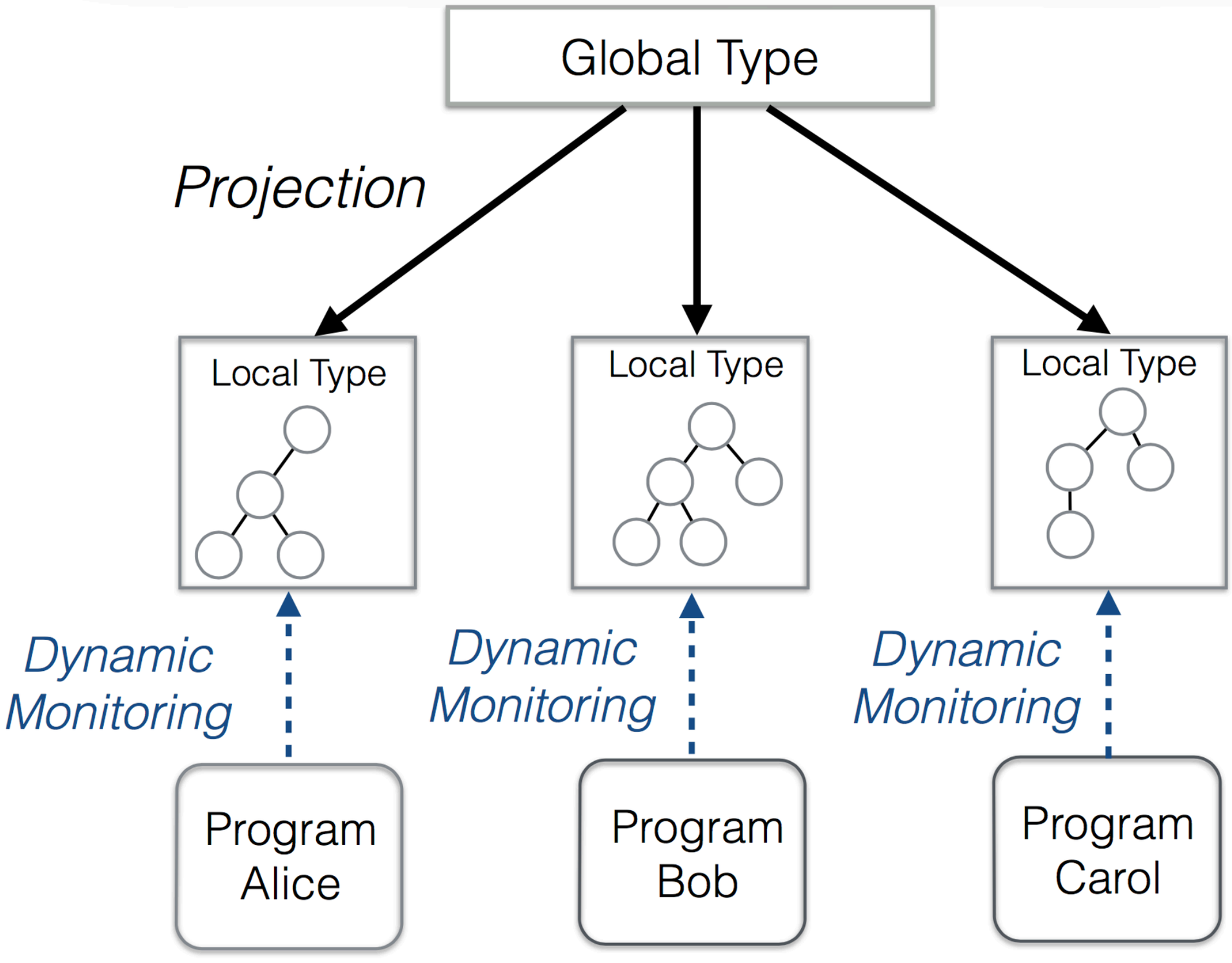
$\texttt{Alice} \to \texttt{Bob}: \langle \mathsf{Nat} \rangle.$
$\texttt{Bob} \to \texttt{Carol}: \langle \mathsf{Nat} \rangle.\mathsf{end}$

$T_{\texttt{Bob}} = ?\langle \texttt{Alice}, \mathsf{Nat} \rangle;$
$!\langle \texttt{Carol}, \mathsf{Nat} \rangle; \mathsf{end}$

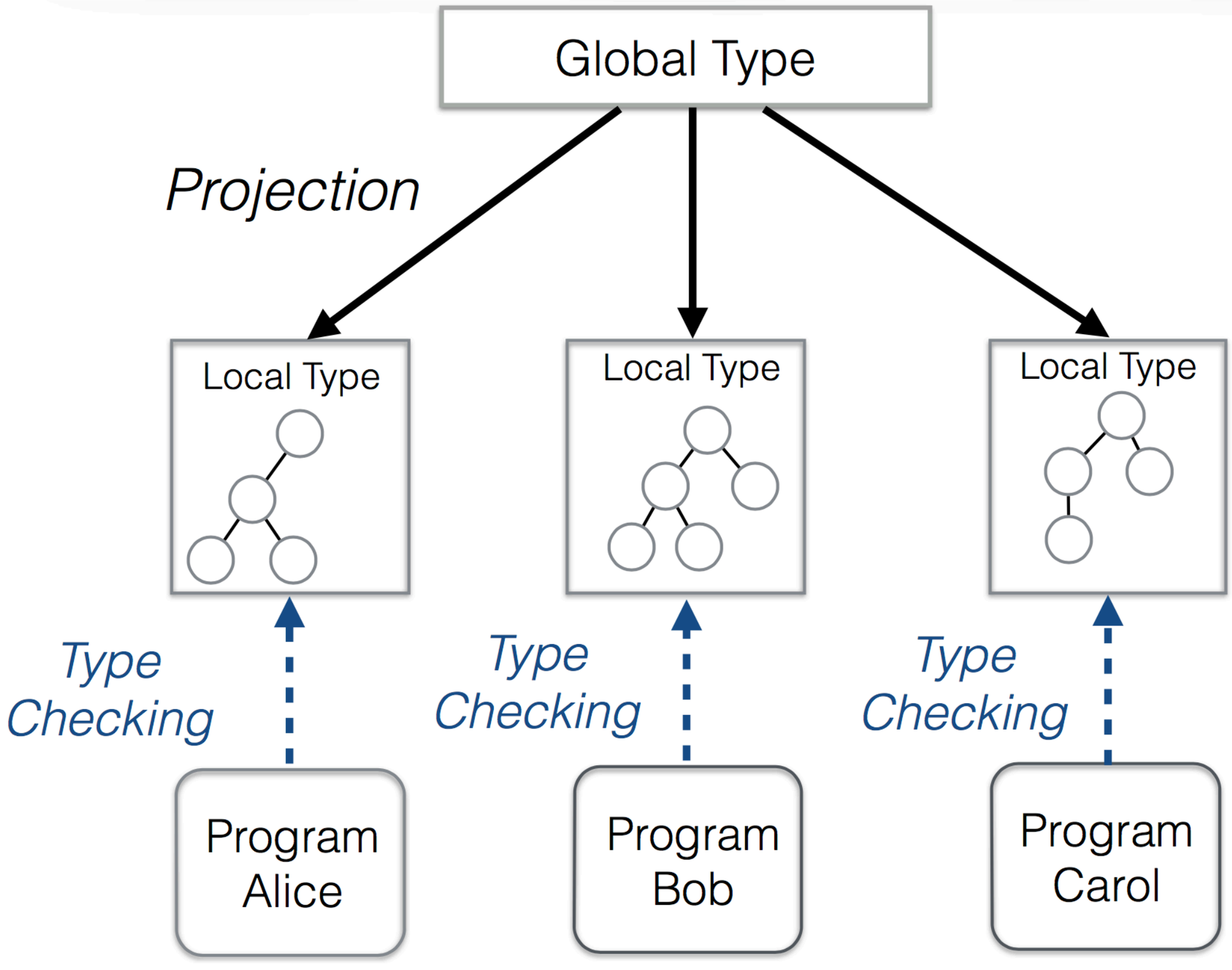
$P_{\texttt{Bob}} = s?(\texttt{Alice}, x);$
$s!\langle \texttt{Carol}, x \rangle; 0$

- Step 1: Write a Global Type
- Step 2: Write Local Programs
- Step 3: Project and Type Check Locally

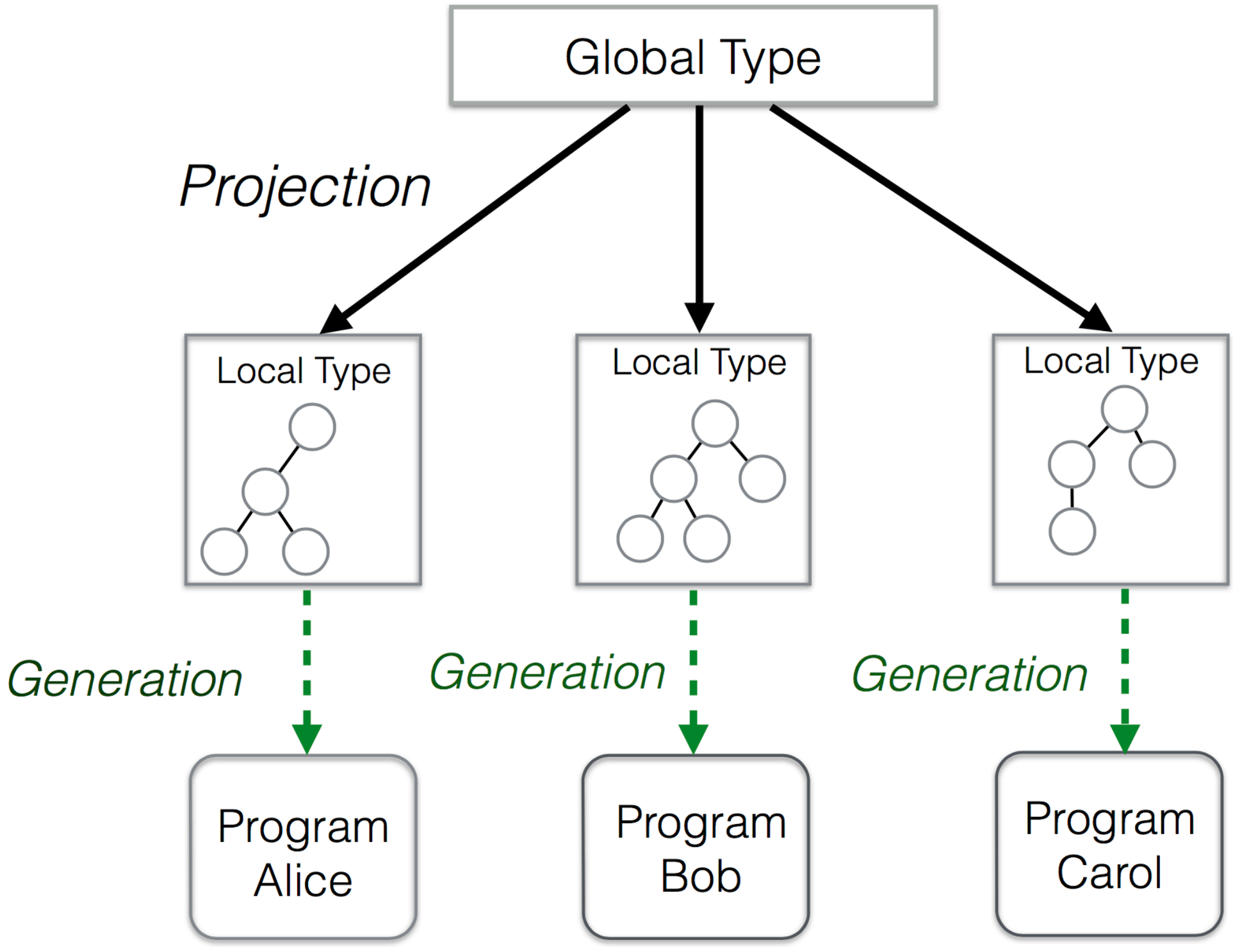
Dynamic Monitoring



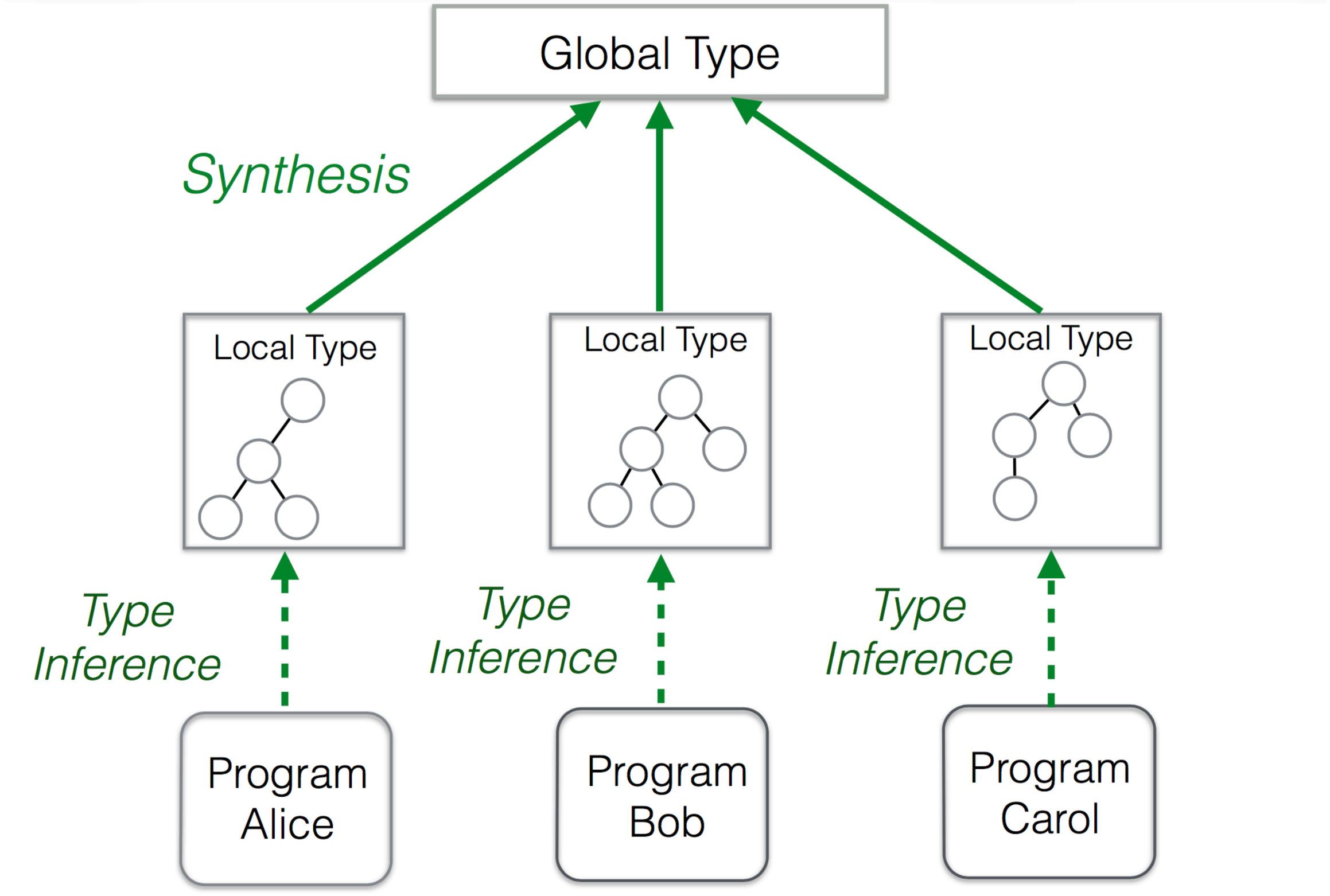
Type Checking



Code Generation



Synthesis


Global Type
Synthesis
Local Type
Local Type
Local Type
Type Inference
Type Inference
Type Inference
Program Alice
Program Bob
Program Carol

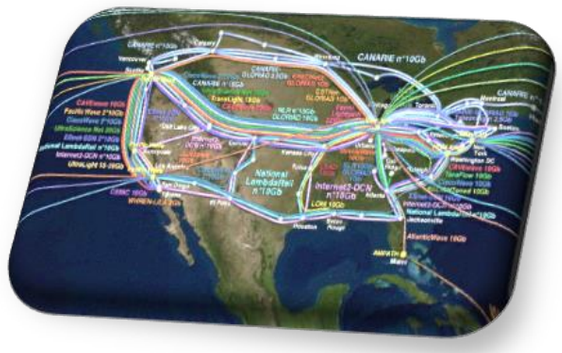
Applications

Scribble

Scala

F#

Session C



Java

ERLANG

RabbitMQ
Messaging that just works

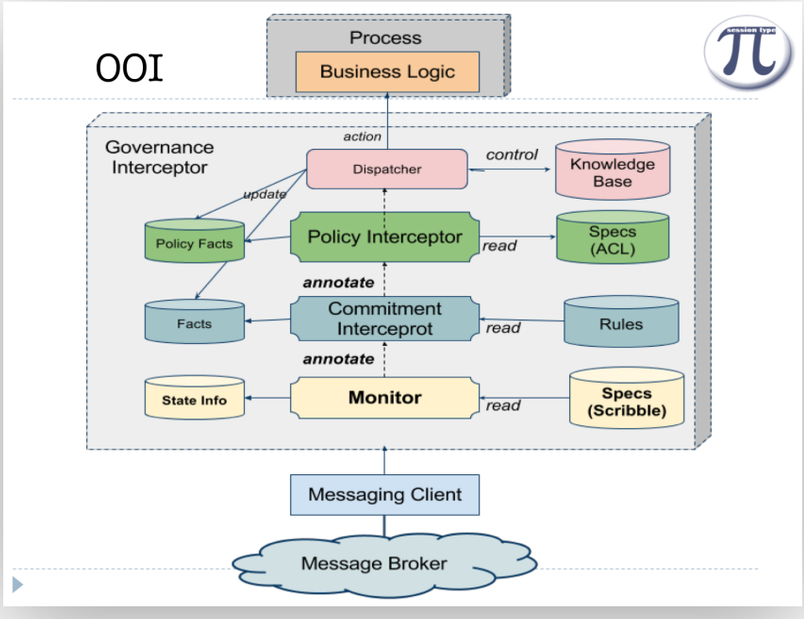
python

AMQP
Advanced Message Queuing Protocol

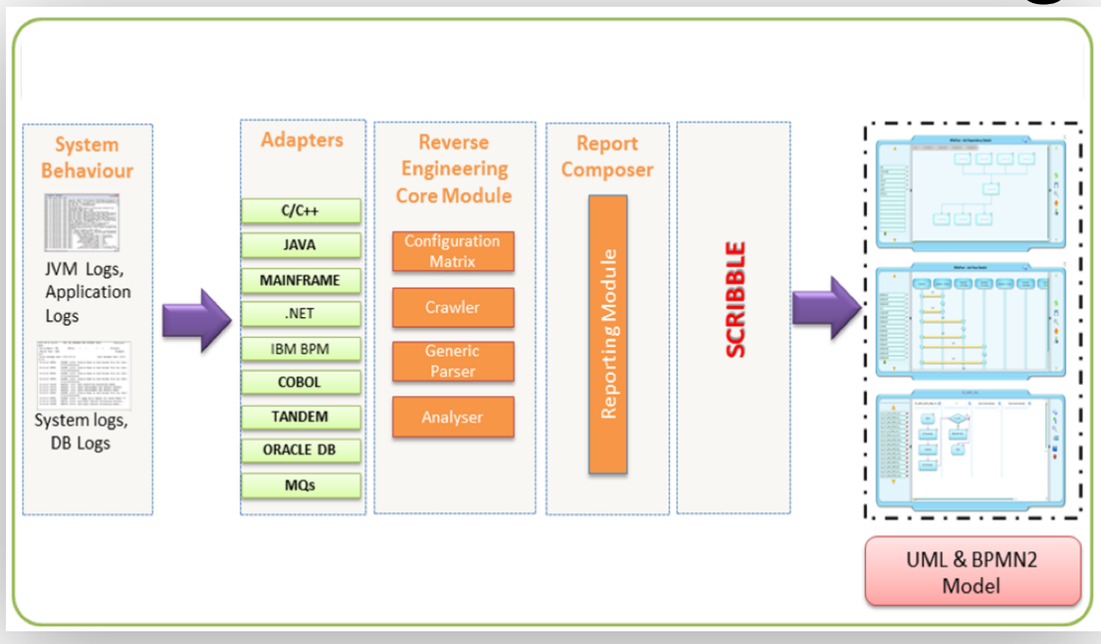
OOI
OCEAN OBSERVATORIES INITIATIVE

Session Type based Tools

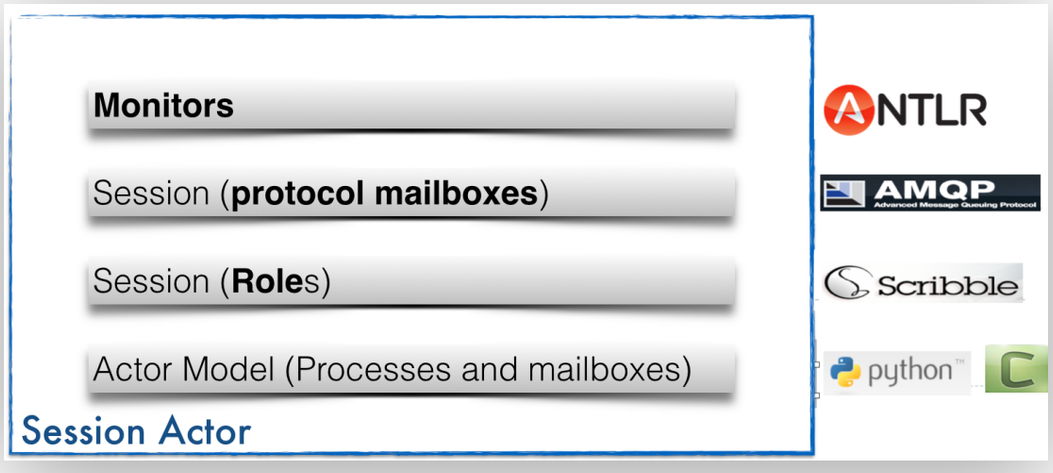
OOI Governance



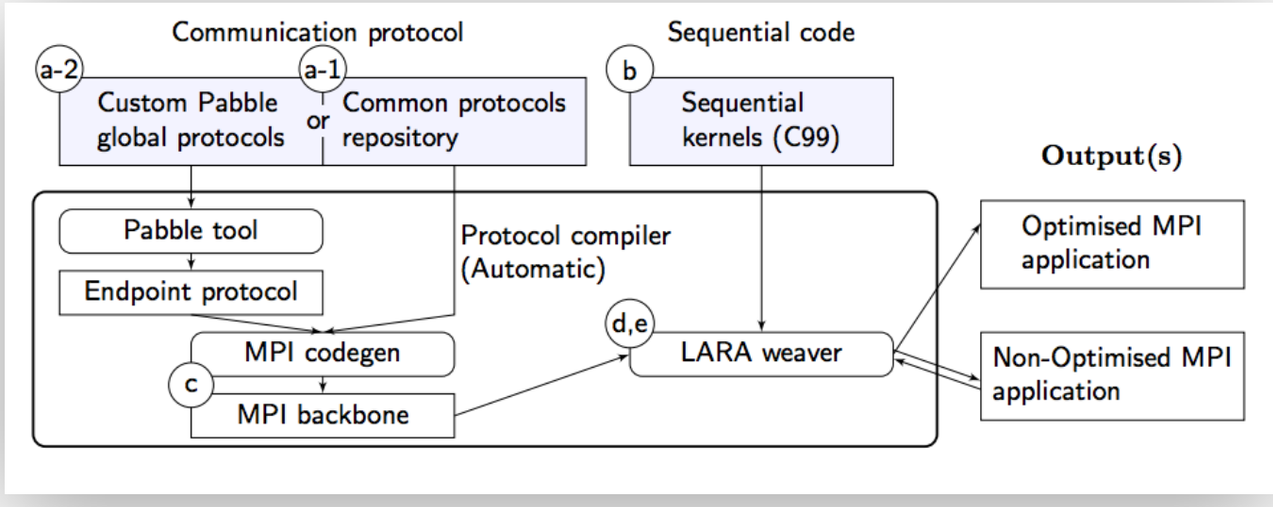
ZDLC: Process Modeling



Actor Verification

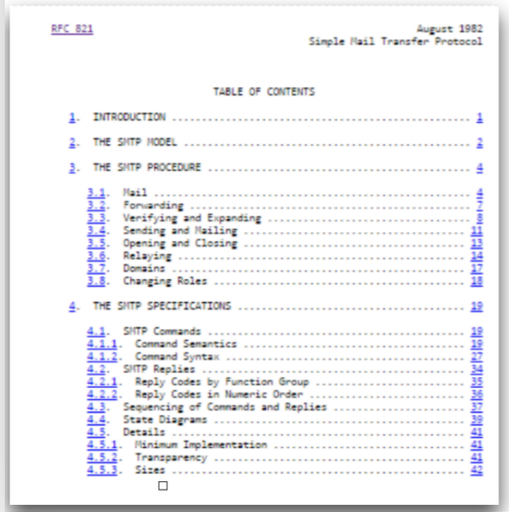


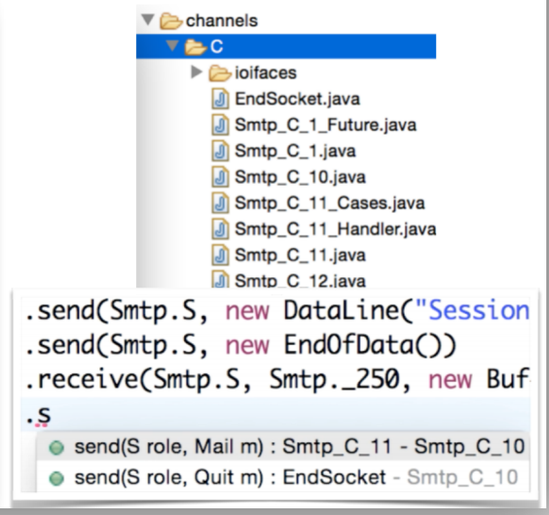
MPI code generations



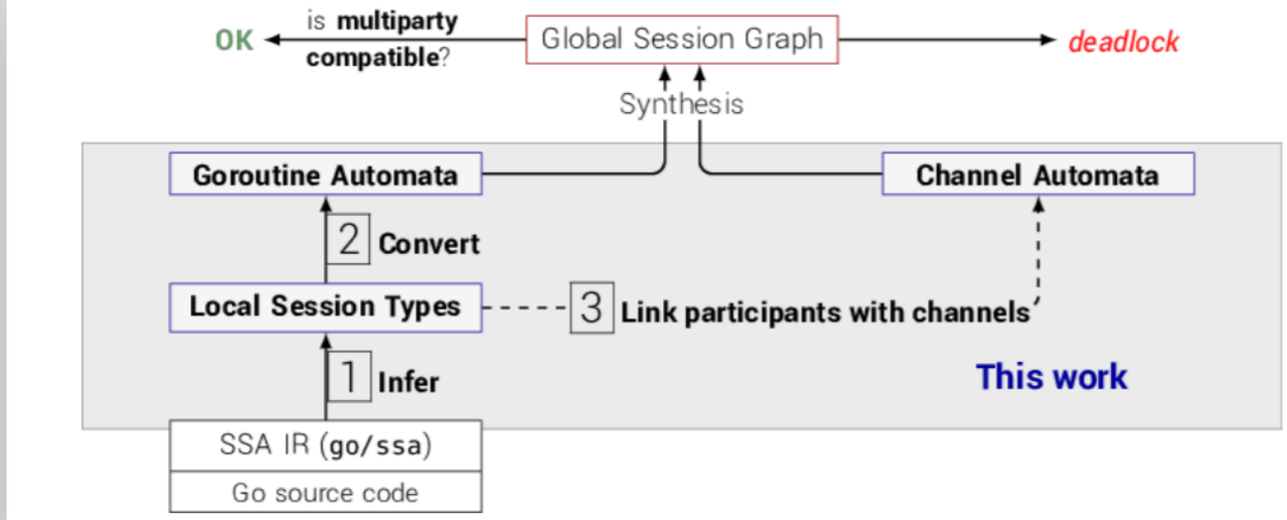
Session Type based Tools

Java API Generation [FASE'16]

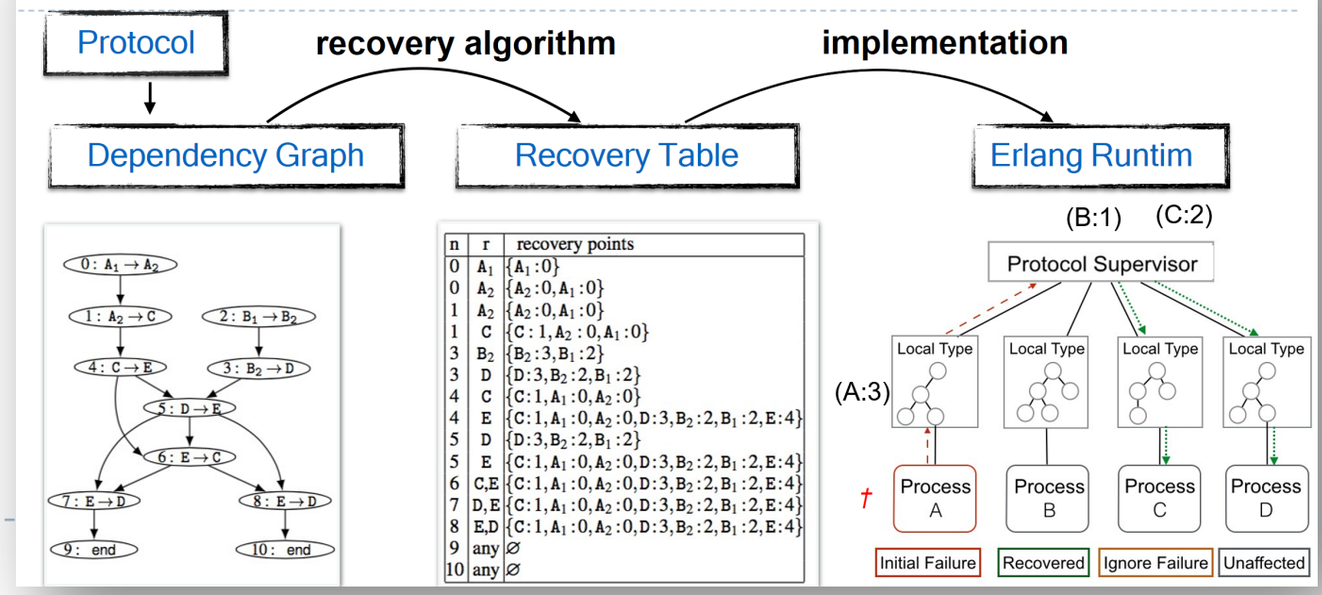




Deadlock Detection for Go [CC'16, POPL'17]



Safe Recovery for Erlang [CC'15]



www.scribble.org

Home Getting Started Documentation Downloads Community

Fork me on GitHub

Scribble: Describing Multi Party Protocols

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.

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

Scribble

```
1  module examples;
2
3  global protocol HelloWorld(role Me, role World) {
4    hello() from Me to World;
5    choice at World {
6      goodMorning() from World to Me;
7    } or {
8      goodAfternoon() from World to Me;
9    }
10 }
11
```

Hello World ▾ | Check | Protocol: examples.HelloWorld | Role: Me | Project | Generate Graph

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; }}
```

SMTP in Scribble

TABLE OF CONTENTS

1. INTRODUCTION 1

2. THE SMTP MODEL 2

3. THE SMTP PROCEDURE 4

3.1. Mail 4
3.2. Forwarding 7
3.3. Verifying and Expanding 8
3.4. Sending and Mailing 11
3.5. Opening and Closing 13
3.6. Relaying 14
3.7. Domains 17
3.8. Changing Roles 18

4. THE SMTP SPECIFICATIONS 19

4.1. SMTP Commands 19
4.1.1. Command Semantics 19
4.1.2. Command Syntax 27
4.2. SMTP Replies 34
4.2.1. Reply Codes by Function Group 35
4.2.2. Reply Codes in Numeric Order 36
4.3. Sequencing of Commands and Replies 37
4.4. State Diagrams 39
4.5. Details 41
4.5.1. Minimum Implementation 41
4.5.2. Transparency 41
4.5.3. Sizes 42

SMTP in Scribble

RFC 821 August 1982
Simple Mail Transfer Protocol

TABLE OF CONTENTS

1. INTRODUCTION 1

2. THE SMTP MODEL 2

3. THE SMTP PROCEDURE 4

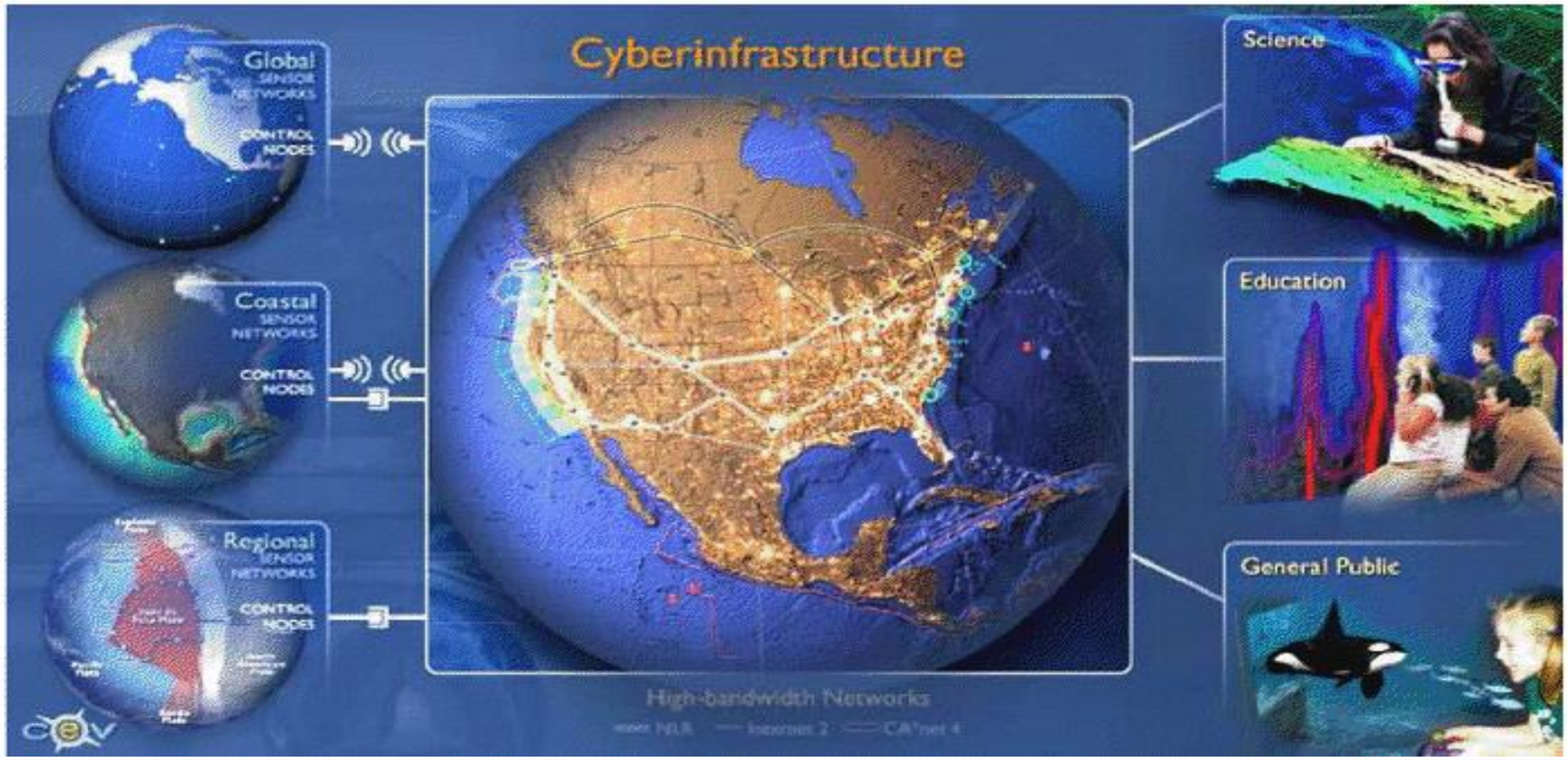
3.1. Mail 4
3.2. Forwarding 7
3.3. Verifying and Expanding 8
3.4. Sending and Mailing 11
3.5. Opening and Closing 13
3.6. Relaying 14
3.7. Domains 17
3.8. Changing Roles 18

4. THE SMTP SPECIFICATIONS 19

4.1. SMTP Commands 19
4.1.1. Command Semantics 19
4.1.2. Command Syntax 27
4.2. SMTP Replies 34
4.2.1. Reply Codes by Function Group 35
4.2.2. Reply Codes in Numeric Order 36
4.3. Sequencing of Commands and Replies 37
4.4. State Diagrams 39
4.5. Details 41
4.5.1. Minimum Implementation 41
4.5.2. Transparency 41
4.5.3. Sizes 42

Case Study: OOI

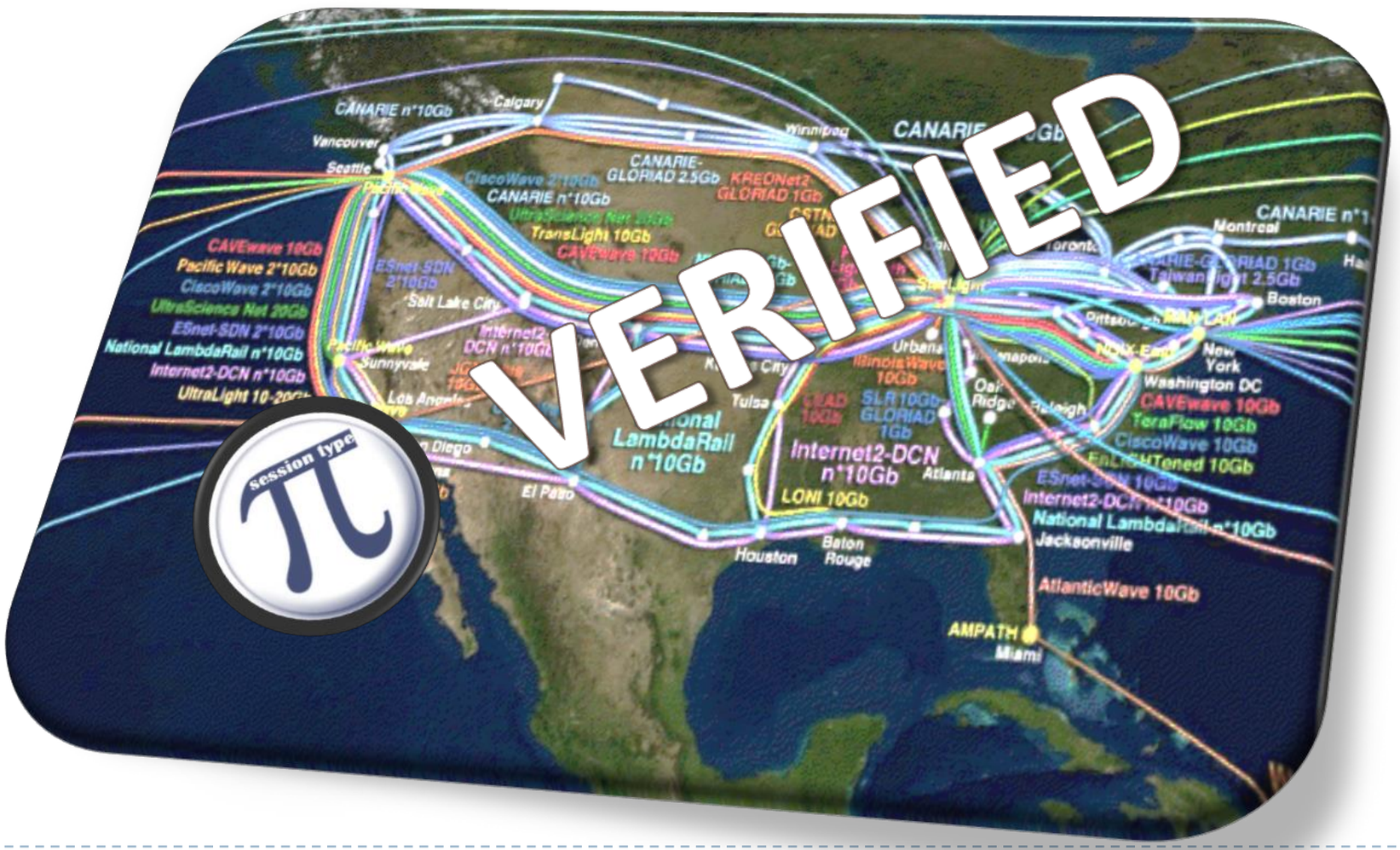
OOI aims: to deploy an infrastructure (global network) to expand the scientists' ability to remotely study the ocean



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

Checking the ocean

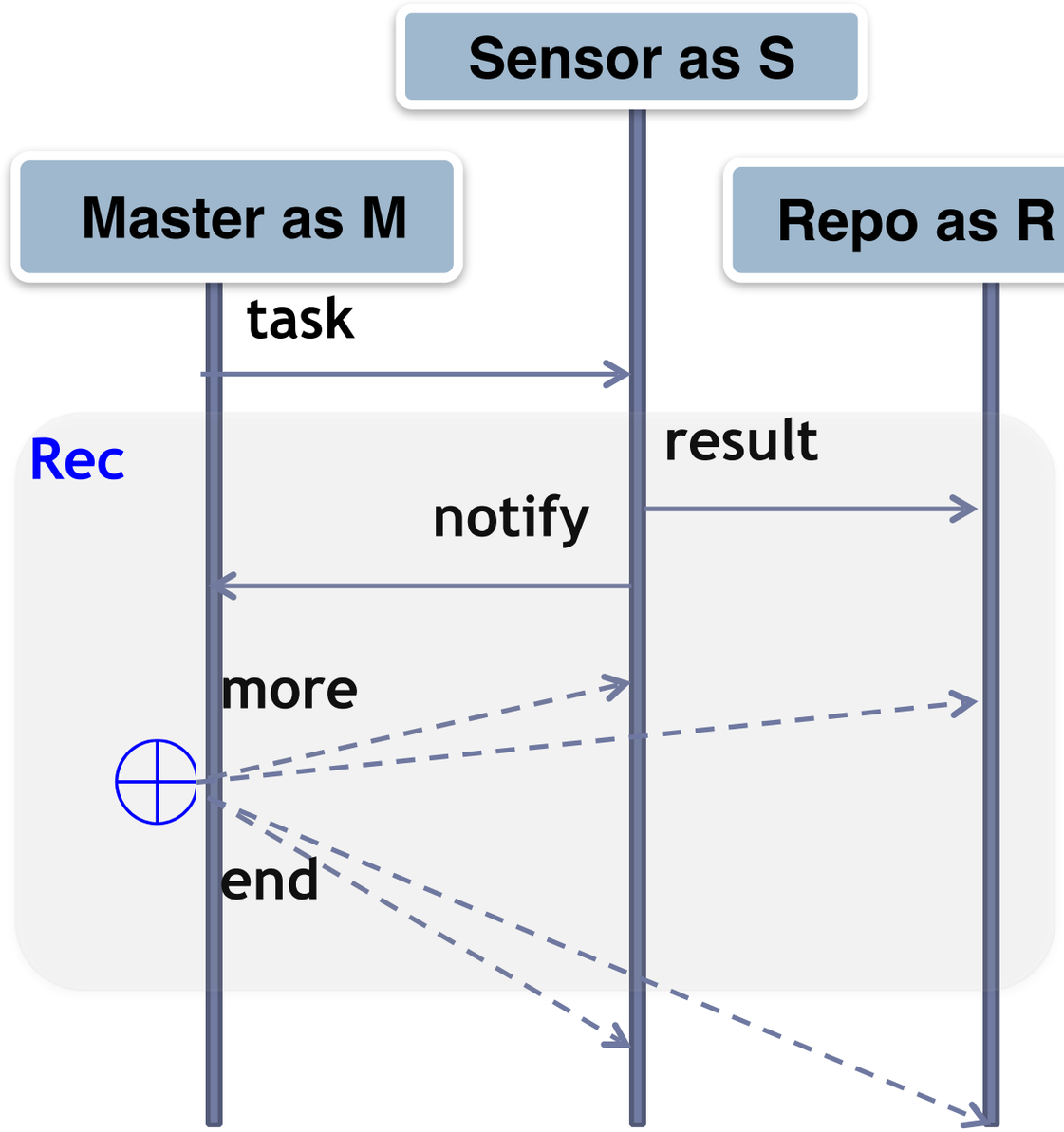
Collaboration: build session types layer on top of OOI network to guarantee global safety



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

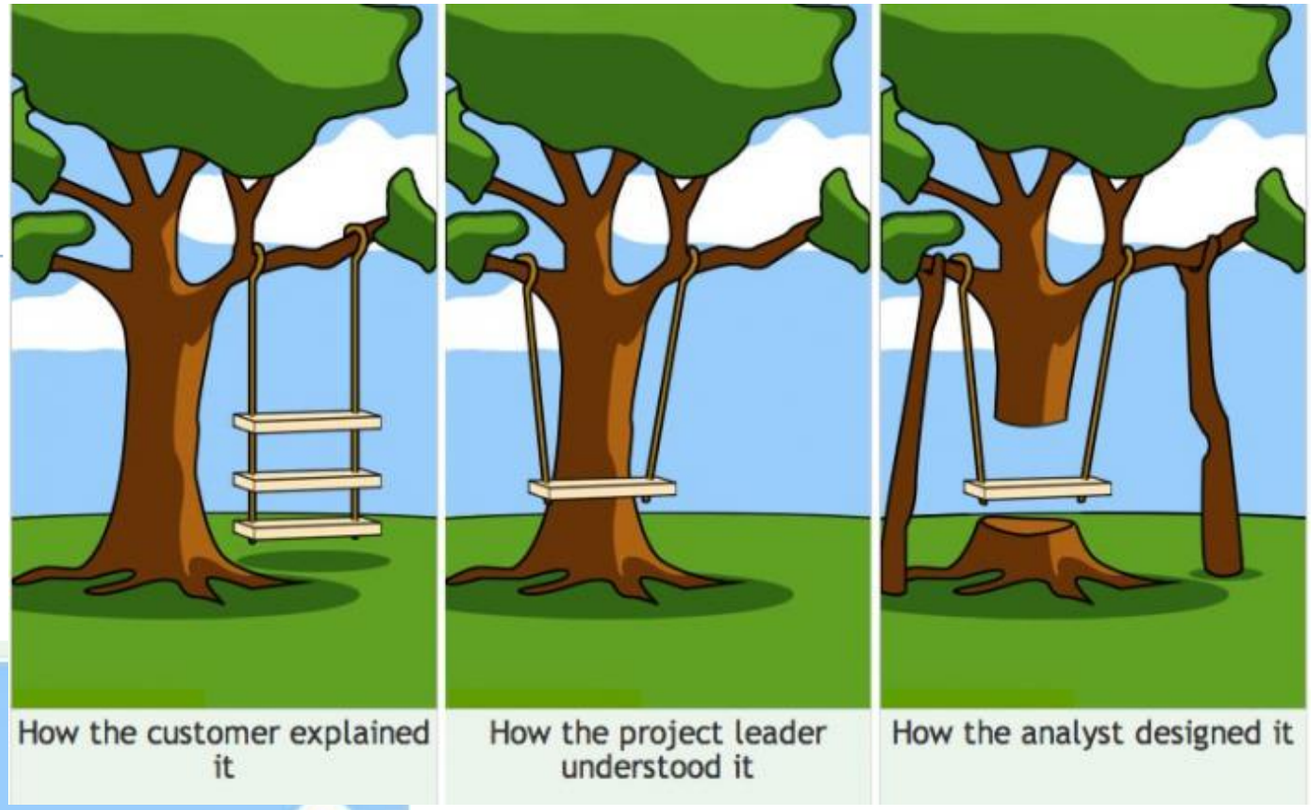
There is so much garbage here ☹

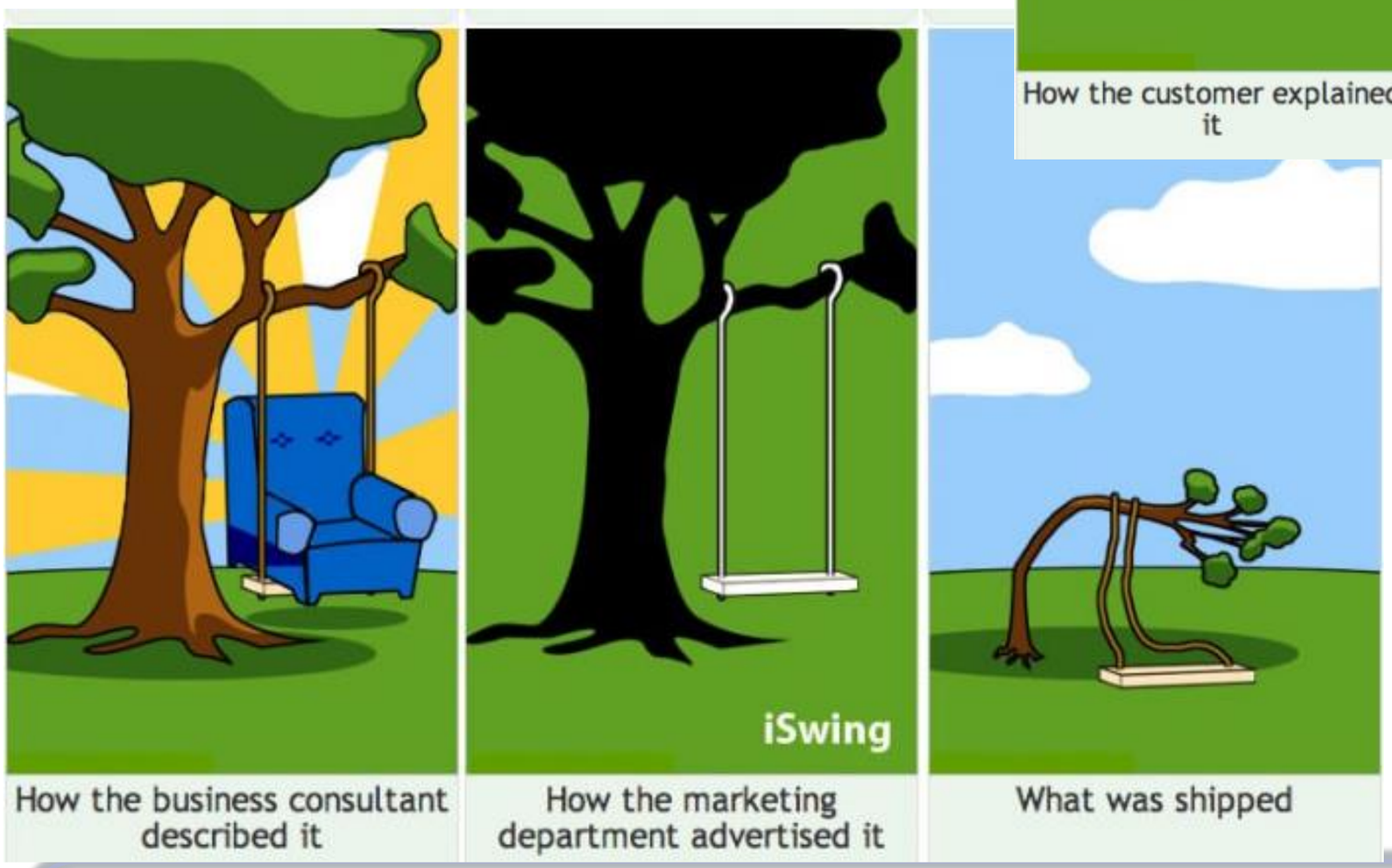
Session types prevent cluttering the network with nonsense messages.

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

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

Lost in Translation

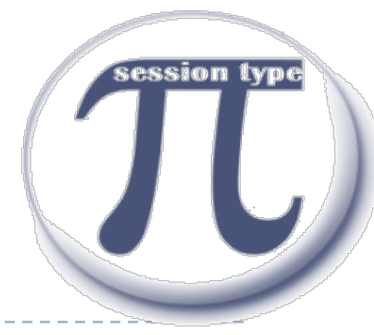




Session types can be used for Testable architectures -

where the specification match the implementation

Summary

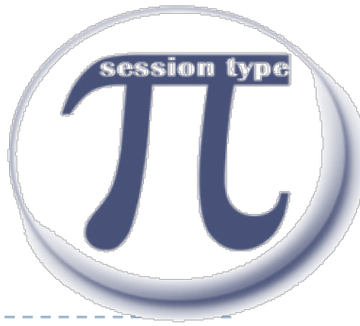


- Organise all communication actions into sessions (communication protocols)
- The result is that every distributed object can be given a type inside a session
- A checker (a type system) can judge whether 2 or more objects are compatible to communicate

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)