University of Glasgow | School of Computing Science

EPSRC
Engineering and Physical Sciences Research Council

Session Types for Reliable Distributed Systems

Simon Gay
University of Glasgow

GSSI Seminar, March 2021

University of Glasgow

From data processing to communication

University of Glasgow

The Era of Data Processing

For a long time (1950s – 1990s), most computing consisted of isolated computers doing data processing.

The importance of structured data was realised very early. The first high-level programming languages supported data structures and data types.

The Era of Data Processing

Data structure declarations in Cobol:

```
----+----1----+----2----+----3----+----4----+----5----+----6----+----7-
      *---------------------------------------------------------------*
      *    DATA-NAME                          DATA-TYPE                 *
      *---------------------------------------------------------------*
       01  PRINCIPAL                          PIC 9999.
       01  NUMBER-OF-YEARS                    PIC 99.
       01  RATE-OF-INTEREST                   PIC 99.
```

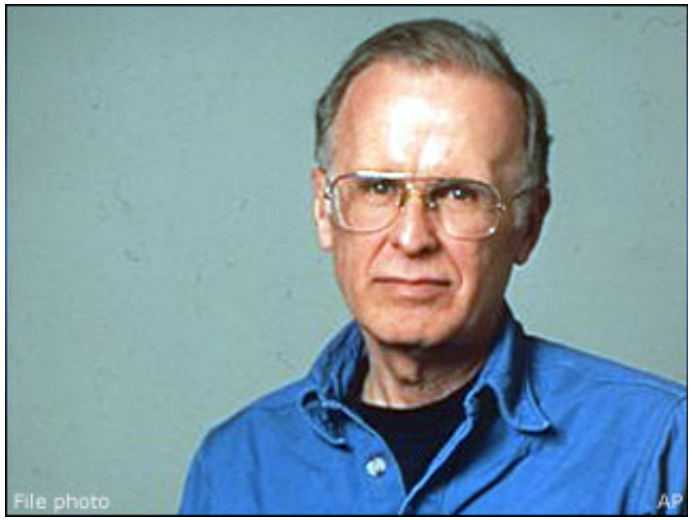
Grace Hopper

University of Glasgow

The Era of Data Processing

Data structure declarations in Fortran 77:

```
INTEGER    COLS,ROWS
PARAMETER(ROWS=12,COLS=10)
REAL       MATRIX(ROWS,COLS),VECTOR(ROWS)
```



John Backus

University of Glasgow

The Era of Data Processing

Lisp uses (dynamically typed) lists as a universal data structure:

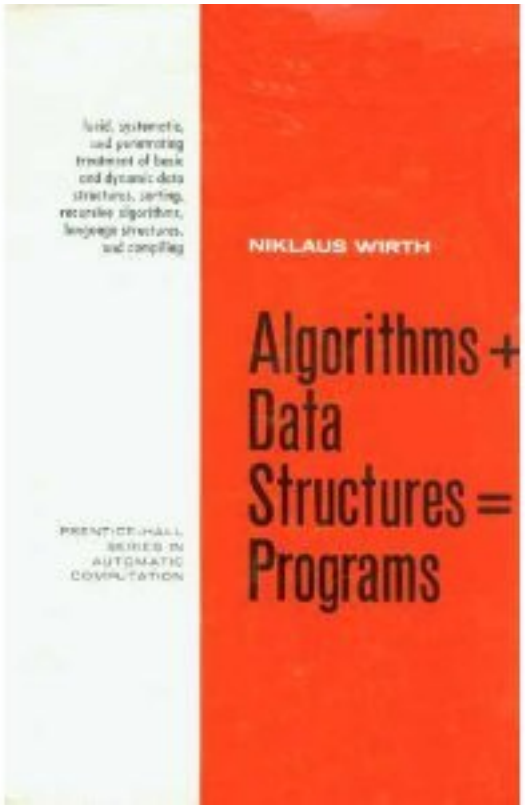
```
(cons '(1 2) '(3 4))
;Output: ((1 2) 3 4)
```

John McCarthy

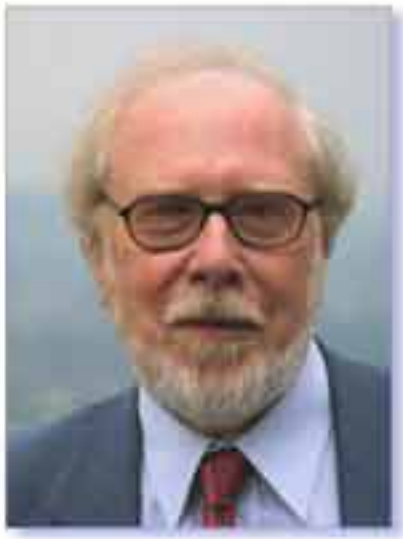
University of Glasgow

The Era of Data Processing

Niklaus Wirth, inventor of the programming language Pascal, introduced the slogan “algorithms + data structures = programs”.



Blaise Pascal



Niklaus Wirth

University of Glasgow

Data Structures and Data Types

Programming languages allow data structures to be codified as data types. Programming tools and environments use data types as the basis for analysis and verification:

- at compile time, in languages such as Java, C#, Scala, Haskell
- at run time, in languages such as Python

Example, when programming in Java with Eclipse:

- a red X if you apply an operation to the wrong data type
- a menu suggesting appropriate operations for a data type

University of Glasgow

The Era of Communication

Computing has changed. We now depend on systems of communicating programs:

- web applications and web services
- mobile apps and their connections to servers
- cloud computing
- data centres

Even within a single computer, further speed increases will depend on communicating many-core programs.

University of Glasgow

The Era of Communication

A new slogan for the era of communication:

programs + communication structures = systems

Communication structures are essential for the design of systems.

Kohei Honda suggested codifying communication structures as session types, so that they are available to programming languages and tools.

[Honda 1993; Takeuchi, Honda & Kubo 1994; Honda, Vasconcelos & Kubo 1998].

Kohei Honda

Structured communication via session types

University of Glasgow

Session Types: The Original Concept

Type-theoretic specification of communication protocols, so that protocol implementations can be verified by static type-checking.

Maths server protocol (server side)

```
T = &{ plus: ?int . ?int . !int . T,
       neg: ?int . !int . T,
       quit: end }
```

S = dual(T)

Maths server protocol (client side)

```
S = ⊕{ plus: !int . !int . ?int . S,
       neg: !int . ?int . S,
       quit: end }
```

University of Glasgow

Session Types: Assumptions

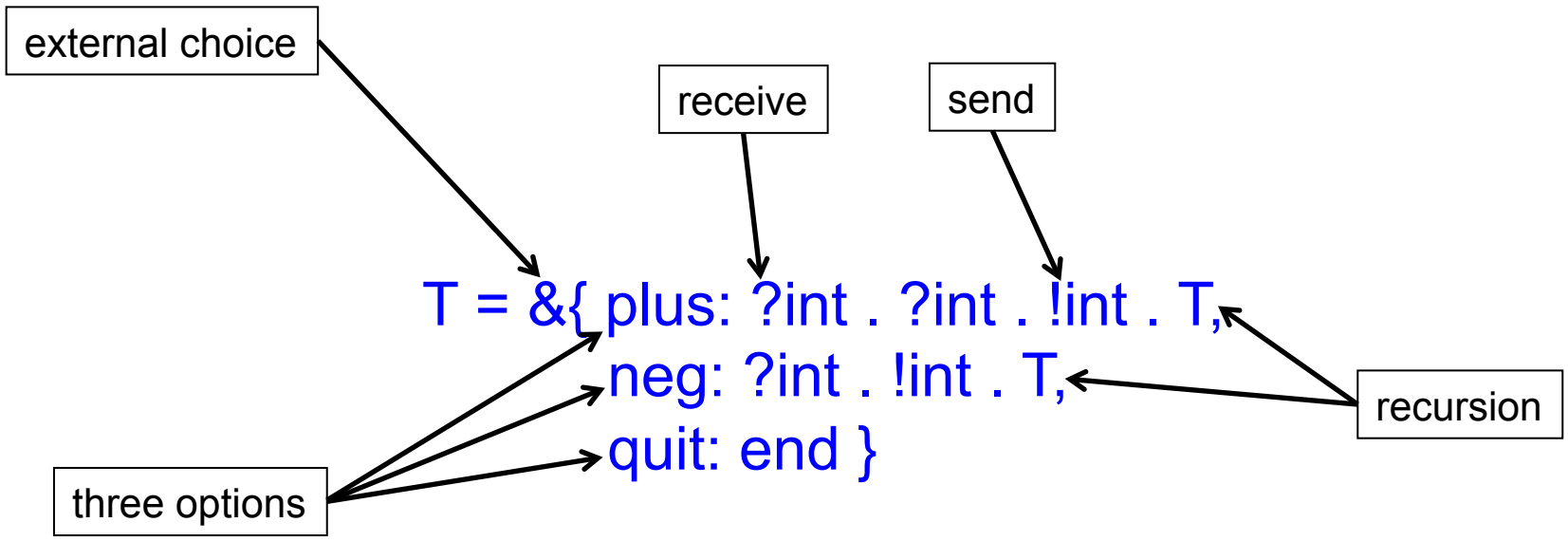
Assume that we are working in a concurrent or distributed system, with point-to-point communication channels (like pi-calculus).

Channels are bi-directional (in practice they may be implemented by pairs of uni-directional channels).

Communication may be synchronous (i.e. sender and receiver both block), or asynchronous with message queues (only receiver blocks).

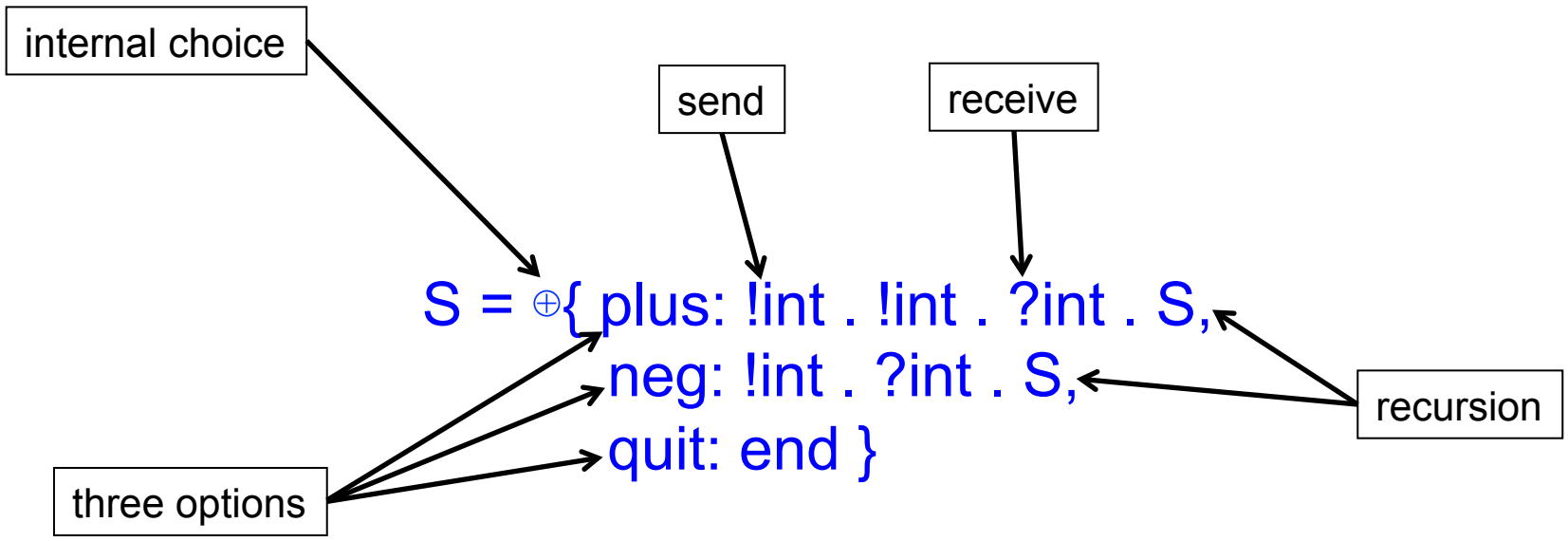
University of Glasgow

Session Types in Detail: Maths Server



University of Glasgow

Session Types in Detail: Maths Client



University of Glasgow

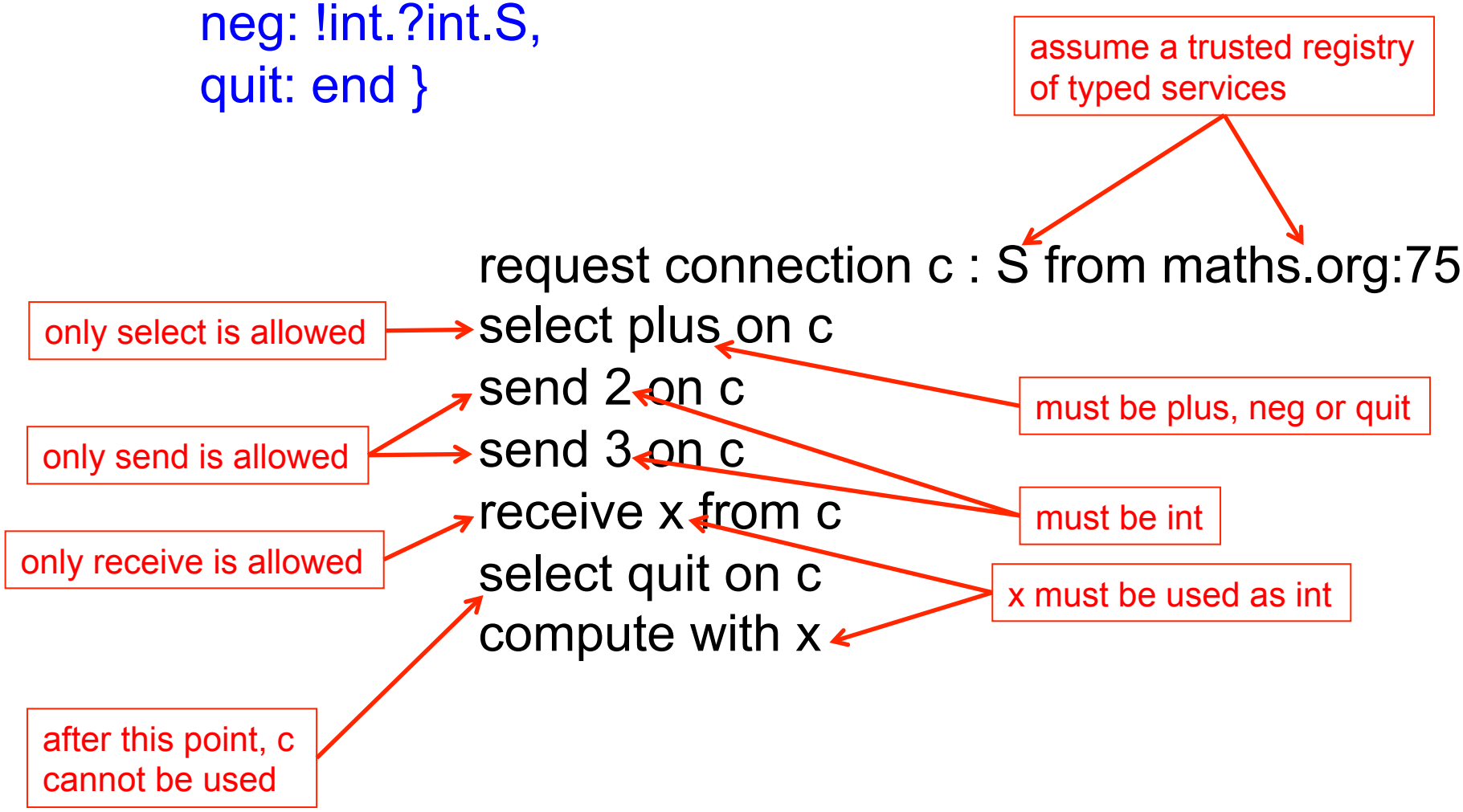
Imperative Pseudocode: Maths Client

```
S = ⊕{ plus: !int.!int.?int.S,
       neg: !int.?int.S,
       quit: end }
```

```
request connection c : S from maths.org:75
select plus on c
send 2 on c
send 3 on c
receive x from c
select quit on c
compute with x
```

Static Typechecking: Maths Client

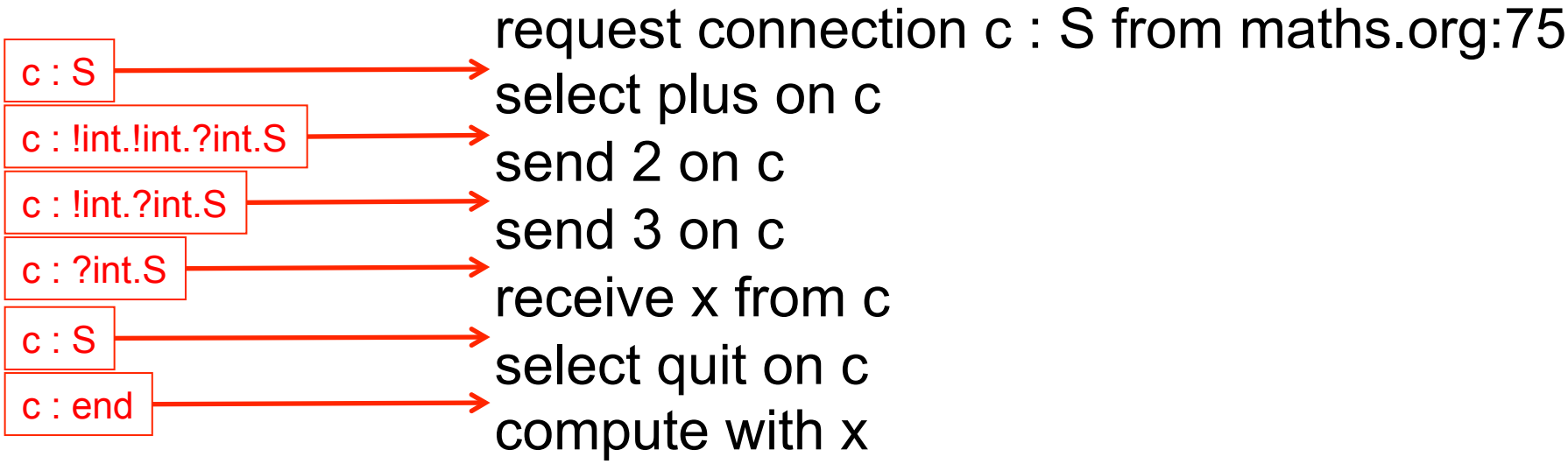
$S = \oplus\{ \text{plus: } !int.!int.?int.S,$
$\text{neg: } !int.?int.S,$
$\text{quit: end } \}$



University of Glasgow

Static Typechecking: Maths Client

S = ⊕{ plus: !int . !int . ?int . S,
neg: !int . ?int . S,
quit: end }



University of Glasgow

Imperative Pseudocode: Maths Server

```
T = &{ plus: ?int . ?int . !int . T,
       neg: ?int . !int . T,
       quit: end }
```

```
accept connection c : T on port 75
label start:
        offer on c {
                plus:   receive x from c
                        receive y from c
                        send x+y on c
                        goto start
                neg:    receive z from c
                        send –z on c
                        goto start
                quit:   break
        }
```

Static Typechecking: Maths Server

```
T = &{ plus: ?int.?int.!int.T,
       neg: ?int.!int.T,
       quit: end }
```

```
accept connection c : T on port 75
label start:
offer on c {
    plus:   receive x from c
            receive y from c
            send x+y on c
            goto start
    neg:    receive z from c
            send –z on c
            goto start
    quit:   break
}
```

only offer is allowed

all options must be present

looping is only allowed when the type of c has returned to its initial state

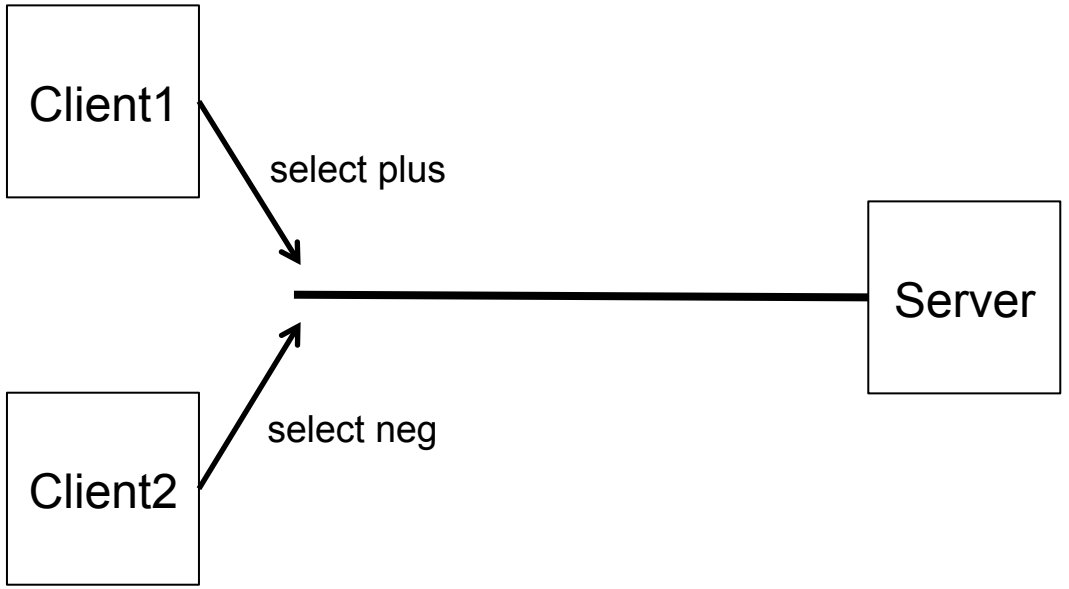
"goto statement considered typeful"

or use a recursive function

University of Glasgow

Unique Ownership of Channel Endpoints

It is essential that each endpoint of a channel is used by only one component of a system.



one select would go first, then the other would be incorrect: race condition

University of Glasgow

Linear Typing

To guarantee unique ownership of channel endpoints, session type systems use standard techniques of linearity [Girard 1987].

Specific techniques for linear type systems may be based on e.g. [Kobayashi, Pierce & Turner 1996] for pi-calculus, or [Mackie 1994] for functional languages.

Unique ownership is also guaranteed in the presence of delegation, i.e. sending a session-typed channel in a message.

A line of work following [Caires & Pfenning 2010] develops the connection between session types and linear logic, following the Curry-Howard / propositions as types paradigm.

University of Glasgow

Session Typing Guarantees…

No race conditions:

never two sends in parallel on one channel endpoint, etc

No communication mismatch:

if there is a send then there is a receive in parallel, and the message types match

Session fidelity:

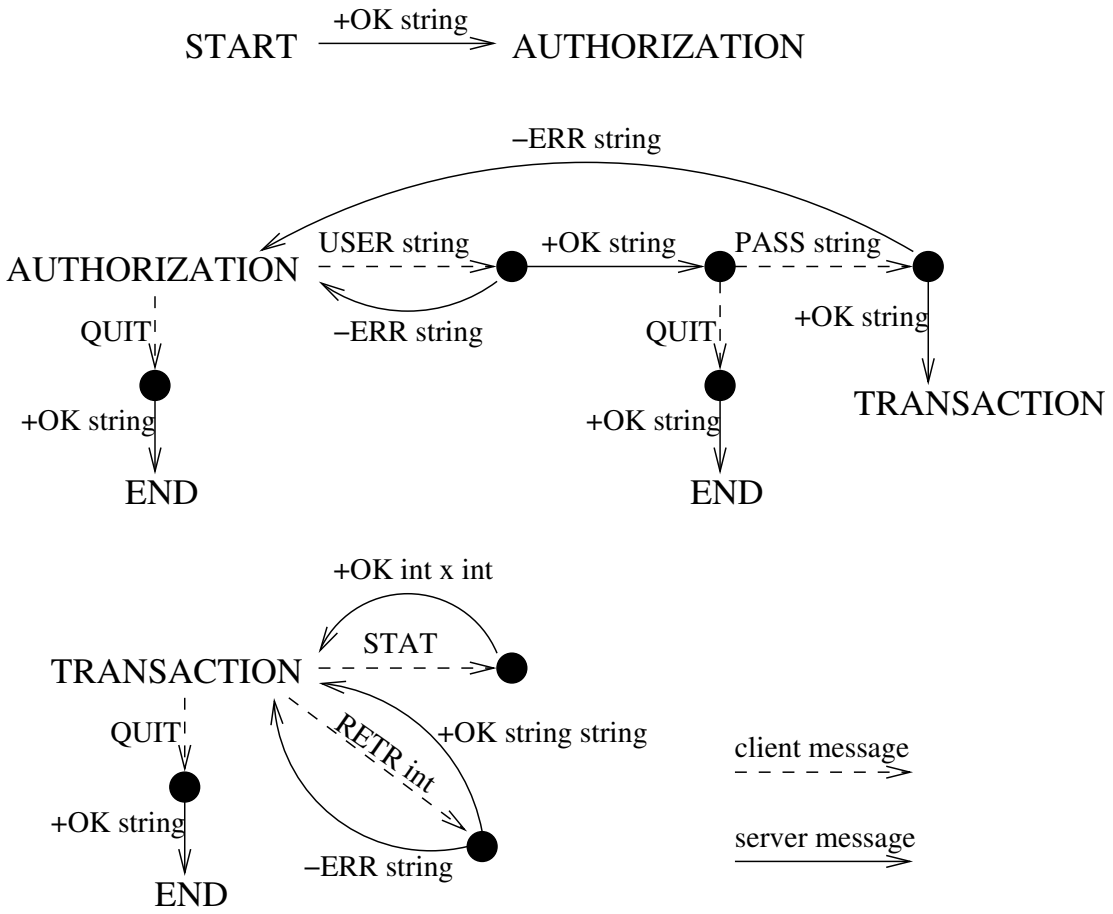
the sequence and types of messages on a channel match the type of the channel

But deadlock-freedom is not guaranteed in general.

Proofs are based on subject reduction and considering the evolution of types at each reduction step.

University of Glasgow

The POP3 Protocol (RFC 1939)



University of Glasgow

The POP3 Protocol as a Session Type

S = START, A = AUTHORIZATION, T = TRANSACTION

$$S = \oplus\{ok : !Str . A\}$$

$$A = \&\{quit : \oplus\{ok : !Str . end\},$$
$$user : ?Str . \oplus\{error : !Str . A,$$
$$ok : !Str . \&\{quit : \oplus\{ok : !Str . end\},$$
$$pass : ?Str . \oplus\{error : !Str . A,$$
$$ok : !Str . T\}\}\}\}$$

$$T = \&\{stat: \oplus\{ok: !(Int \times Int).T\},$$
$$retr: ?Int.\oplus\{ok: !Str.!Str.T,$$
$$error: !Str.T\},$$
$$quit : \oplus\{ok : !Str . end\}\}$$

University of Glasgow

The Growth of Session Types as a Field

Research on session types has developed in many directions.

- Incorporation in various programming language paradigms.
- Curry-Howard correspondence.
- Runtime monitoring as a complement to static checking.
- Generalisation from two-party to multi-party protocols.
- Gradual typing and blame.
- Connections with automata theory, time, and model-checking.
- Language implementation and tool development.

Multi-party session types

University of Glasgow

Multi-Party Session Types

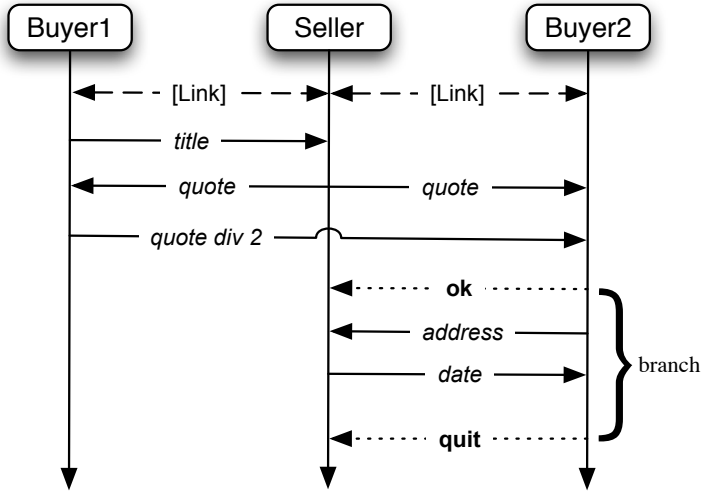
Honda, Yoshida and Carbone [2008] developed a theory of multi-party session types, generalising from the original two-party (binary) theory.

Multi-party session types provide a methodology for the design of communication-based systems, and there is an increasing amount of tool support.

Nobuko Yoshida

Marco Carbone

Example from HYC 2008



informal design

$$
\begin{array}{ll}
1 & \mathtt{B1} \to \mathtt{S} : \; s\langle \mathsf{string} \rangle. \\
2 & \mathtt{S} \to \mathtt{B1} : \; b_1\langle \mathsf{int} \rangle. \\
3 & \mathtt{S} \to \mathtt{B2} : \; b_2\langle \mathsf{int} \rangle. \\
4 & \mathtt{B1} \to \mathtt{B2} : b_2'\langle \mathsf{int} \rangle. \\
5 & \mathtt{B2} \to \mathtt{S} : \; s\{\mathsf{ok} : \mathtt{B2} \to \mathtt{S} : \; s\,\langle \mathsf{string} \rangle.\mathtt{S} \to \mathtt{B2} : \; b_2\,\langle \mathsf{date} \rangle.\mathsf{end}, \\
 & \qquad\qquad \mathsf{quit} : \mathsf{end}\}
\end{array}
$$

global type

Example from HYC 2008

1 $\mathtt{B1} \to \mathtt{S} : \; s\langle \mathsf{string} \rangle.$
2 $\mathtt{S} \to \mathtt{B1} : \; b_1\langle \mathsf{int} \rangle.$
3 $\mathtt{S} \to \mathtt{B2} : \; b_2\langle \mathsf{int} \rangle.$
4 $\mathtt{B1} \to \mathtt{B2} : b_2'\langle \mathsf{int} \rangle.$
5 $\mathtt{B2} \to \mathtt{S} : \; s\{\mathsf{ok} : \mathtt{B2} \to \mathtt{S} : \; s\langle \mathsf{string} \rangle.\mathtt{S} \to \mathtt{B2} : \; b_2\langle \mathsf{date} \rangle.\mathsf{end},$
$\mathsf{quit} : \mathsf{end}\}$

global type

endpoint projection, subject to consistency conditions

Buyer 1: S!<string> ; S?<int> ; B2!<int>

Buyer 2: S?<int> ; B1?<int> ; S⊕{ ok : S!<string> ; S?<date> ; end, quit : end }

Seller : B1?<string> ; B1!<int> ; B2!<int> ; B2&{ ok : B2?<string> ; B2!<date> ; end , quit : end }

local types

University of Glasgow

Example from HYC 2008

Global types are expressed in Scribble [Honda et al. 2007 – 2016] and there is a toolset for consistency checking, projection etc.

Local types are used for typechecking in an endpoint language, using further tools to translate between Scribble and each language.

E.g. the Mungo / StMungo tools for Java [Gay et al. 2016-2021]

University of Glasgow

But…

What about failure? Fault-tolerance? Reliability?

University of Glasgow

What about failure? What about reliability?

Most of the literature on session types ignores failure.

Even if a whole closed system is being designed with types, it's unrealistic to ignore hardware and communication failure.

Even more realistically, systems are open and components must interact with external non-designed agents.

The Stardust project (2020-2024) aims to combine the **structuring mechanism** of session types and the **reliability mechanism** of the **actor** paradigm.

University of Glasgow

The Stardust Project

Session Types for Reliable Distributed Systems

Funded by UK EPSRC 2020-2024.

University of Glasgow (SG, Phil Trinder)
University of Kent (Simon Thompson, Laura Bocchi)
Imperial College London (Nobuko Yoshida)

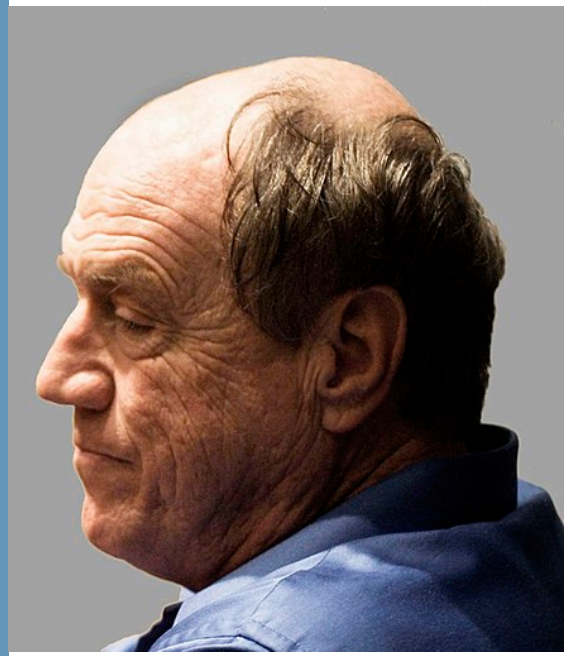
Actyx
Erlang Solutions Ltd
Lightbend
Quviq
Tata Consultancy Services

University of Glasgow

Actor Languages

Originate from the work of Carl Hewitt and Gul Agha.

Actors have private state, communicate by message-passing, respond to incoming messages to determine actions.



Carl Hewitt

Gul Agha

University of Glasgow

Erlang

The best-known actor language is Erlang.

Originally designed for scalable and reliable (99.9999% availability) telecommunications software.

Reliability is achieved by **timeouts** and **supervision**.

Supervision means detecting failure and taking action, e.g. restarting an actor.

Usually failure is detected by timeouts.

Joe Armstrong

University of Glasgow

Session Types and Erlang

One of our aims is to develop a session type system for Erlang.

- Language extension or external tool?
- How do we adapt session types from channels to mailboxes?
- How do we combine static and dynamic typing?
- How can type information guide the design of supervision trees?
- What properties can we prove about well-typed systems?

University of Glasgow

Preliminary work on session types and actors

Multiparty Session Types for Safe Runtime Adaptation in an Actor Language

Paul Harvey[1]
Rakuten Institute of Technology
paul@paul-harvey.org

Simon Fowler[1]
School of Computing Science, University of Glasgow
Simon.Fowler@glasgow.ac.uk

Ornela Dardha
School of Computing Science, University of Glasgow
Ornela.Dardha@glasgow.ac.uk

Simon J. Gay
School of Computing Science, University of Glasgow
Simon.Gay@glasgow.ac.uk

University of Glasgow

Overview

- Adaptive software is increasingly important for pervasive computing.
- Adaptation includes discovering, replacing and communicating with software components that are not part of the original system.
- Ensemble is an actor-based language with support for adaptation.
- We designed and implemented EnsembleS by adding session types to Ensemble.
- Static type checking guarantees safe runtime adaptation.

University of Glasgow

EnsembleS language features

- Imperative actor-based language.
- Channels instead of mailboxes.
- Support for adaptation.

University of Glasgow

Adaptation in EnsembleS

- Discover: locate an actor with a given interface and satisfying a given query.
- Install: spawn a new actor instance at a specified stage.
- Migrate: move an executing actor to a different stage.
- Replace: replace an executing actor with a new actor instance with the same interface.
- Interact: connect to another actor and communicate with it.

University of Glasgow

Adaptation in EnsembleS, with session types

- Discover: locate an actor with a given interface and satisfying a given query and a given session type.
- Replace: replace an executing actor with a new actor instance with the same interface and the same session type.
- Interact: connect to another actor and communicate with it, following its session type.

EnsembleS uses multi-party session types.

The implementation needs a trusted registry of typed actors.

University of Glasgow

Replacement in EnsembleS

```
1  // session and interface definitions
2  actor fastA presents accountingI
3        follows accountingSession{
4   constructor() {}
5   behaviour{
6    receive data on input;
7    quicksort(data);
8    send data on output;
9    }
10 }
11
12 actor slowA presents accountingI
13       follows accountingSession{
14  pS= new property[2] of property("",0);
15  constructor() {
16   pS[0]:= new property("serial",823);
17   pS[1]:= new property("version",2);
18   publish pS;
19  }
```

```
20  behaviour{
21   receive data on input;
22   bubblesort(data);
23   send data on output;
24 } }
25
26 query alpha() { $serial==823 && $version<4; }
27
28 actor main presents mainI {
29  constructor() { }
30  behaviour {
31  // Find the slow actors matching query
32  actor_s = discover(accountingI,
33     accountingSession, alpha());
34  // Replace them with efficient versions
35  if(actor_s[0].length > 1) {
36     replace actor_s[0] with fastA();
37  }
38 } }
```

```
type accountingI is interface(
    in { Client, int[] } input;
    out { Client, int[] } output;
)
```

```
type accountingSession is session(
    data(int[]) from Client;
    data(int[]) to Client;
)
```

University of Glasgow

Results about EnsembleS

- We have formalised a core language with a type system and operational semantics.
- A correctly-typed system satisfies the following properties.
- Type safety: the behaviour of every actor matches its session type, and communication never has a type mismatch.
- Progress: if a system stops executing then either:
  - every actor is either terminated or waiting for input, or:
  - there is an unmatched discover operation.

University of Glasgow

Conclusion

- Session types are a programming language mechanism for specifying and checking communication protocols.
- The field of session types is beginning to address issues of reliability and fault-tolerance.
- Our key idea is to combine session types and actor-based programming languages.
- The Stardust project has a great set of academic researchers and industrial partners.

University of Glasgow

THANK YOU