Session types

An Introduction

Michaelmas Term

Data types

One of the **most successful concepts** in computer science

Data types describe the **intended usage** of data

```
int x = 1;
int y = 2;
int z = x + y; // Correct :)
```

Data types

One of the **most successful concepts** in computer science

Data types describe the **intended usage** of data

```
int x = 1;
int y = 2;
int z = x + y; // Correct :)
```

Allow to **document code** and **spot errors at compile-time**

```
public int add(int a, int b) { ... }

int x = add(1, 2)   // Correct :)
int y = add(1, "Hi!") // Does not compile :)
```

Data types (cont'd)

```
public int add(int a, int b) { ... }
```

From this definition, we can tell that:

1. add takes two integers a, b
2. it does *something*, possibly **using** a, b
   - and if a,b are used, they are **only used as integers**
3. finally, it returns an integer

What about communicating systems?

Say that we need to write a **network client** implementing a **certain protocol**. Are data types helpful?

What about communicating systems?

Say that we need to write a **network client** implementing a **certain protocol**. Are data types helpful?

```
public void client(Socket s) { ... }
```

From this definition, we can tell that:

1. `client` takes a **network socket** `s`
2. it can **use** `s` **to send/receive data** to/from the server
3. it returns nothing

What about communicating systems?

Say that we need to write a **network client** implementing a **certain protocol**. Are data types helpful?

```
public void client(Socket s) { ... }
```

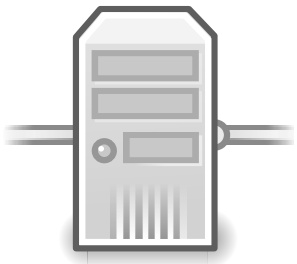
From this definition, we can tell that:

1. `client` takes a **network socket** `s`
2. it can **use** `s` **to send/receive data** to/from the server
3. it returns nothing

But **what messages are sent/received?**
And **in what order?**
Does `client` really **implement the intended protocol?**

What could possibly go wrong?

Client

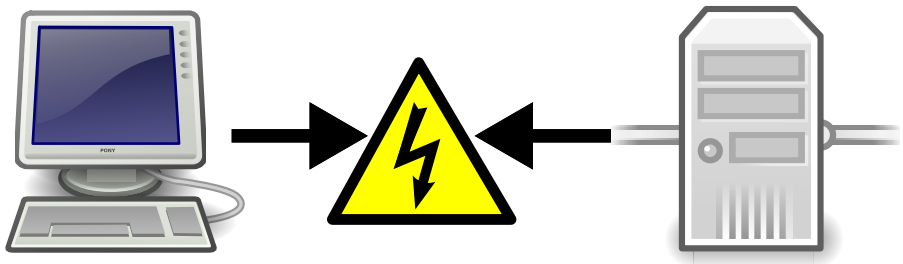


Server

E.g., consider a very simple protocol:

1. a **client** **sends two integers** to a **server**
2. then, the **server** **answers with a boolean**

What could possibly go wrong?



E.g., consider a very simple protocol:

1. a **client** **sends two integers** to a **server**
2. then, the **server** **answers with a boolean**

If one of the two does not correctly implement its part of the protocol, we can encounter **various run-time errors**:

- **mismatching inputs/outputs** ✗
- **deadlocks** ✗

From data types to session types

A **session type** formalises a **protocol** as a **structured sequence of communications**

- unlike data types, session types focus on **communication**

From data types to session types

A **session type** formalises a **protocol** as a **structured sequence of communications**

- unlike data types, session types focus on **communication**

E.g., consider again our simple protocol:

1. a **client** **sends two integers** to a **server**
2. then, the **server** **answers with a boolean**

From data types to session types

A **session type** formalises a **protocol** as a **structured sequence of communications**

- unlike data types, session types focus on **communication**

E.g., consider again our simple protocol:

1. a **client** **sends two integers** to a **server**
2. then, the **server** **answers with a boolean**

From the **client** **viewpoint**, the **session type** is:

$$\text{Session} = \mathbf{send}(\text{int}).\mathbf{send}(\text{int}).\mathbf{receive}(\text{bool})$$

Session types enforce channel usage

From the **client** **viewpoint**, the **session type** is:

$$\text{Session} = \mathbf{send}(\text{int}).\mathbf{send}(\text{int}).\mathbf{receive}(\text{bool})$$

With session types, a client looks like:

```
public void client(Session s) { ... }
```

. . . where `s` is a **communication channel** of type "Session"

Session types enforce channel usage

From the **client viewpoint**, the **session type** is:

$$\text{Session} = \mathbf{send}(\text{int}).\mathbf{send}(\text{int}).\mathbf{receive}(\text{bool})$$

With session types, a client looks like:

```
public void client(Session s) { ... }
```

. . . where `s` is a **communication channel** of type "Session"

The type determines the **intended usage of** `s`: i.e., if the code compiles, then `client` uses `s` to **send two** integers, and then **receive a** boolean

Session types and duality

From the **client** **viewpoint**, the **session type** is:

$$\text{Session} = \mathbf{send}(\text{int}).\mathbf{send}(\text{int}).\mathbf{receive}(\text{bool})$$

What about the **server**?

Session types and duality

From the **client** **viewpoint**, the **session type** is:

$$\text{Session} = \mathbf{send}(\text{int}).\mathbf{send}(\text{int}).\mathbf{receive}(\text{bool})$$

What about the **server**? Its view of the protocol is **dual:**

$$\overline{\text{Session}} = \mathbf{receive}(\text{int}).\mathbf{receive}(\text{int}).\mathbf{send}(\text{bool})$$

Session types and duality

From the **client** **viewpoint**, the **session type** is:

$$\text{Session} = \mathbf{send}(\text{int}).\mathbf{send}(\text{int}).\mathbf{receive}(\text{bool})$$

What about the **server**? Its view of the protocol is **dual:**

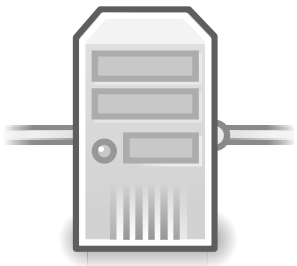
$$\overline{\text{Session}} = \mathbf{receive}(\text{int}).\mathbf{receive}(\text{int}).\mathbf{send}(\text{bool})$$

. . . and correspondingly, its code uses a dual type:

```
public void server(Session̄ s) { ... }
```

Session-typed interactions never go wrong

Client

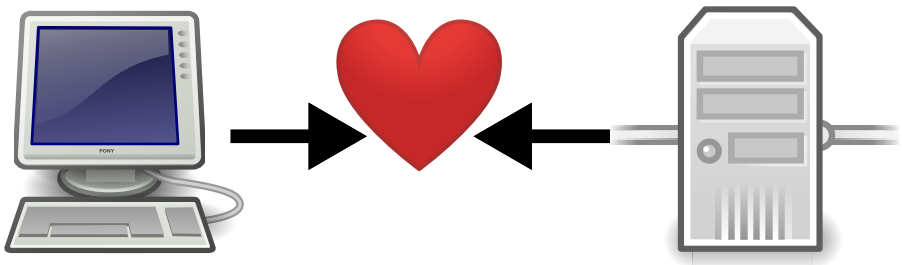


Server

Now, **at compile time**, we can verify that **client** and **server** implement **dual sessions types**...

$$\begin{aligned} \text{Session} &= \mathbf{send}(\text{int}).\mathbf{send}(\text{int}).\mathbf{receive}(\text{bool}) \\ \overline{\text{Session}} &= \mathbf{receive}(\text{int}).\mathbf{receive}(\text{int}).\mathbf{send}(\text{bool}) \end{aligned}$$

Session-typed interactions never go wrong



Now, **at compile time**, we can verify that **client** and **server** implement **dual sessions types**...

$$\begin{aligned} \text{Session} &= \mathbf{send}(\text{int}).\mathbf{send}(\text{int}).\mathbf{receive}(\text{bool}) \\ \overline{\text{Session}} &= \mathbf{receive}(\text{int}).\mathbf{receive}(\text{int}).\mathbf{send}(\text{bool}) \end{aligned}$$

...and this guarantees **correct interaction at run-time**:

- **communication safety** — no mismatching input/output ✓
- **progress** — no deadlocks ✓

Session types in theory...

In this course, you will learn the **basic theory of session types**...

- i.e., their use in the π-calculus

Session types in theory... and practice


In this course, you will learn the **basic theory of session types**...

- i.e., their use in the π-calculus

...and experiment with **practical applications** in **Scala**

- using a tool called **Scribble** —

`https://github.com/scribble/scribble-java`

`https://nuscr.dev/nuscr/`

`https://github.com/nuscr/nuscr`

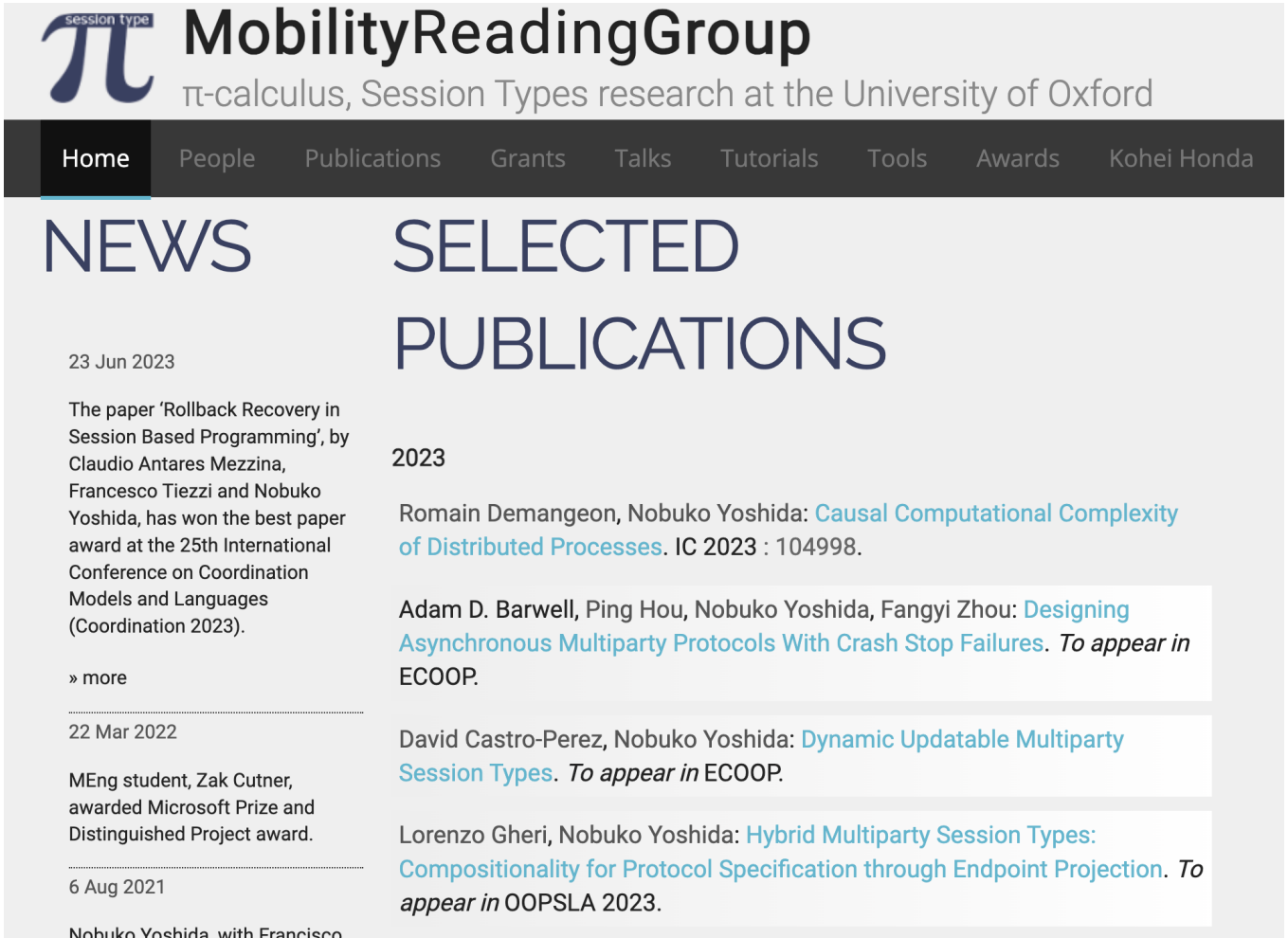
DEMO:

Scribble and **RFC 5321**
("Simple" Mail Transfer Protocol)

The Mobility Reading Group at Oxford

Our research group is **specialised in π-calculus** and **session types** — both **theory** and **applications**

https://mrg.cs.ox.ac.uk/



MobilityReadingGroup

π-calculus, Session Types research at the University of Oxford

Home People Publications Grants Talks Tutorials Tools Awards Kohei Honda

NEWS

23 Jun 2023

The paper 'Rollback Recovery in Session Based Programming', by Claudio Antares Mezzina, Francesco Tiezzi and Nobuko Yoshida, has won the best paper award at the 25th International Conference on Coordination Models and Languages (Coordination 2023).

» more

22 Mar 2022

MEng student, Zak Cutner, awarded Microsoft Prize and Distinguished Project award.

6 Aug 2021

Nobuko Yoshida, with Francisco

SELECTED PUBLICATIONS

2023

Romain Demangeon, Nobuko Yoshida: Causal Computational Complexity of Distributed Processes. IC 2023 : 104998.

Adam D. Barwell, Ping Hou, Nobuko Yoshida, Fangyi Zhou: Designing Asynchronous Multiparty Protocols With Crash Stop Failures. *To appear in* ECOOP.

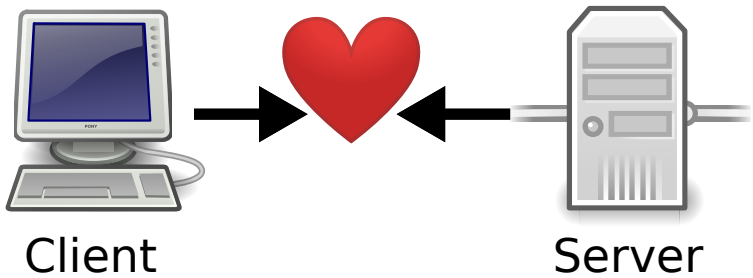
David Castro-Perez, Nobuko Yoshida: Dynamic Updatable Multiparty Session Types. *To appear in* ECOOP.

Lorenzo Gheri, Nobuko Yoshida: Hybrid Multiparty Session Types: Compositionality for Protocol Specification through Endpoint Projection. *To appear in* OOPSLA 2023.

From client-server...

Session types can formalise **complex client-server protocols**...

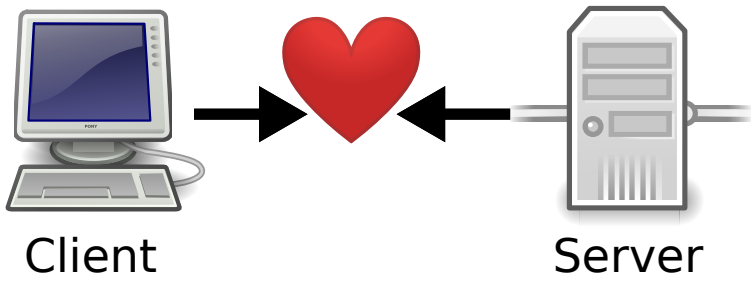
- SMTP
- HTTP
- POP3
- ...



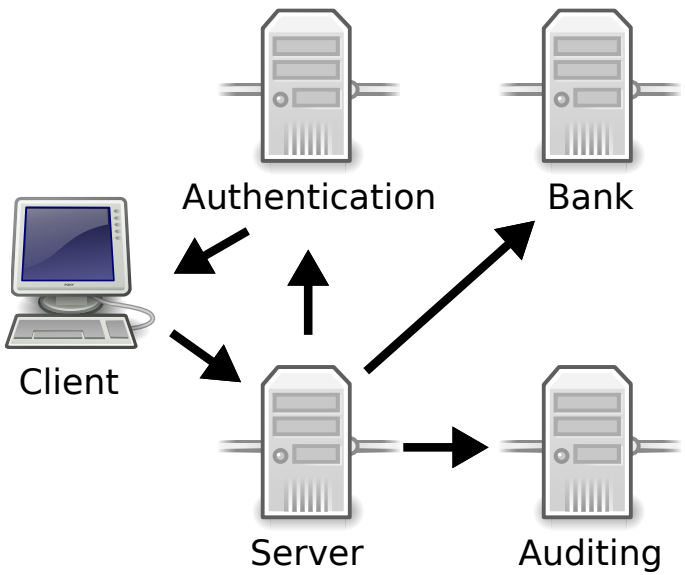
From client-server... to multiparty protocols

Session types can formalise **complex client-server protocols**...

- SMTP
- HTTP
- POP3
- ...

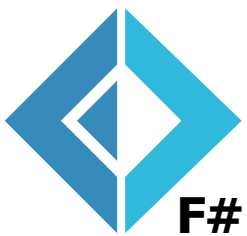


...and also extend to **multiparty protocols**



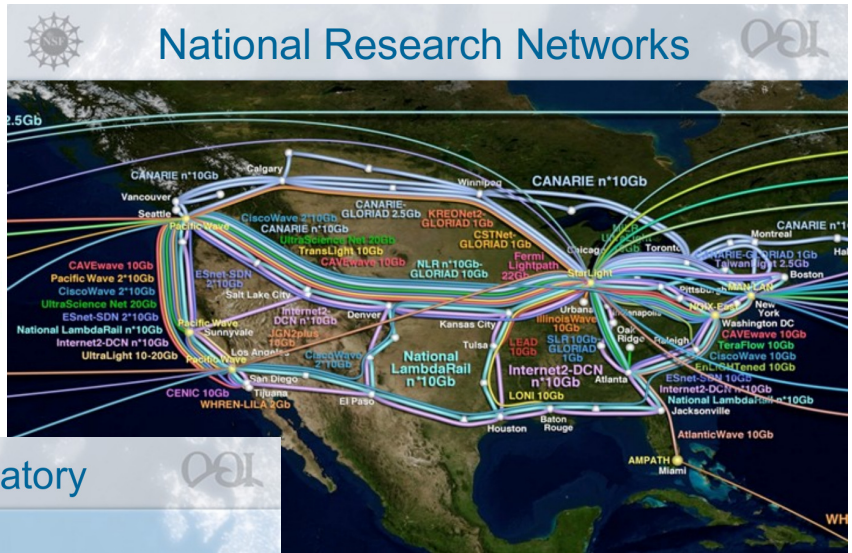
Other programming languages

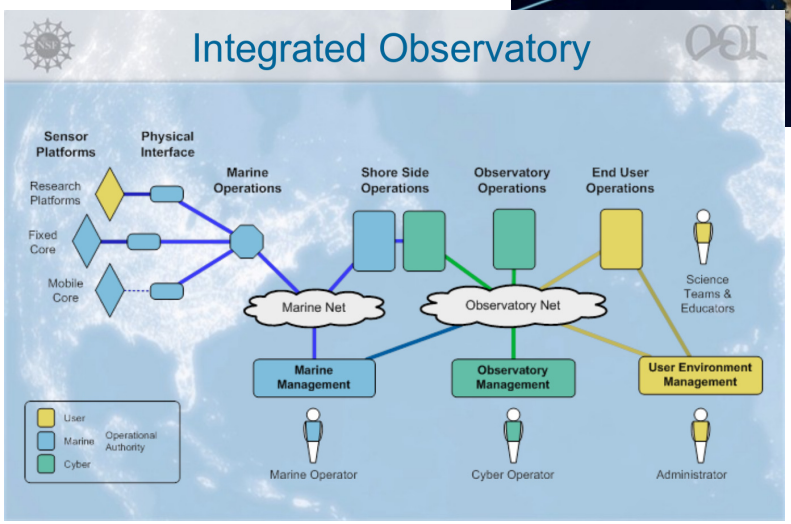
(Elements of) the session types theory have been **implemented in many languages**, e.g.:



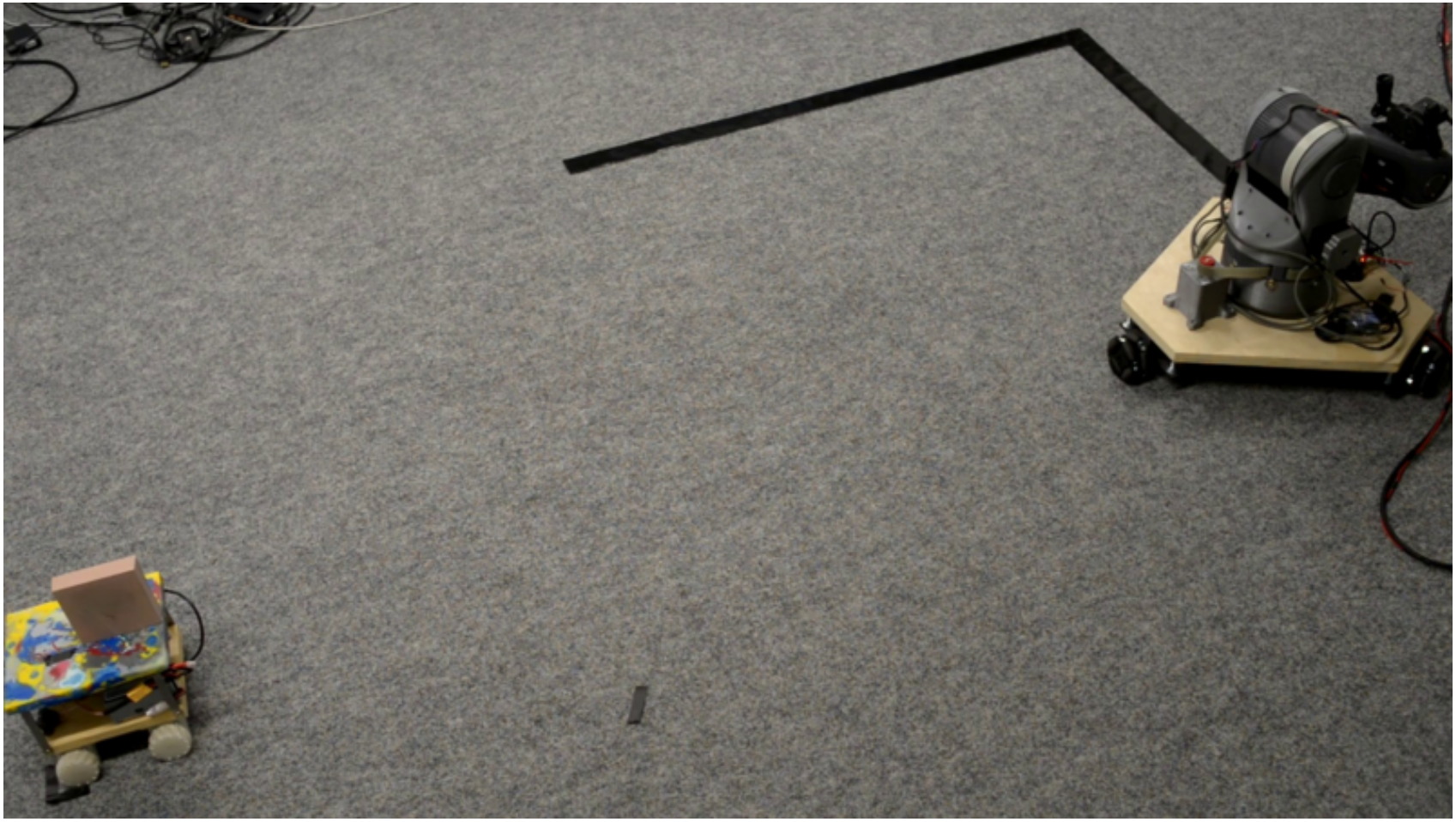
Case study (I): Ocean Observatories Initiative

Collaboration: use session types on top of OOI network to **guarantee global safety**

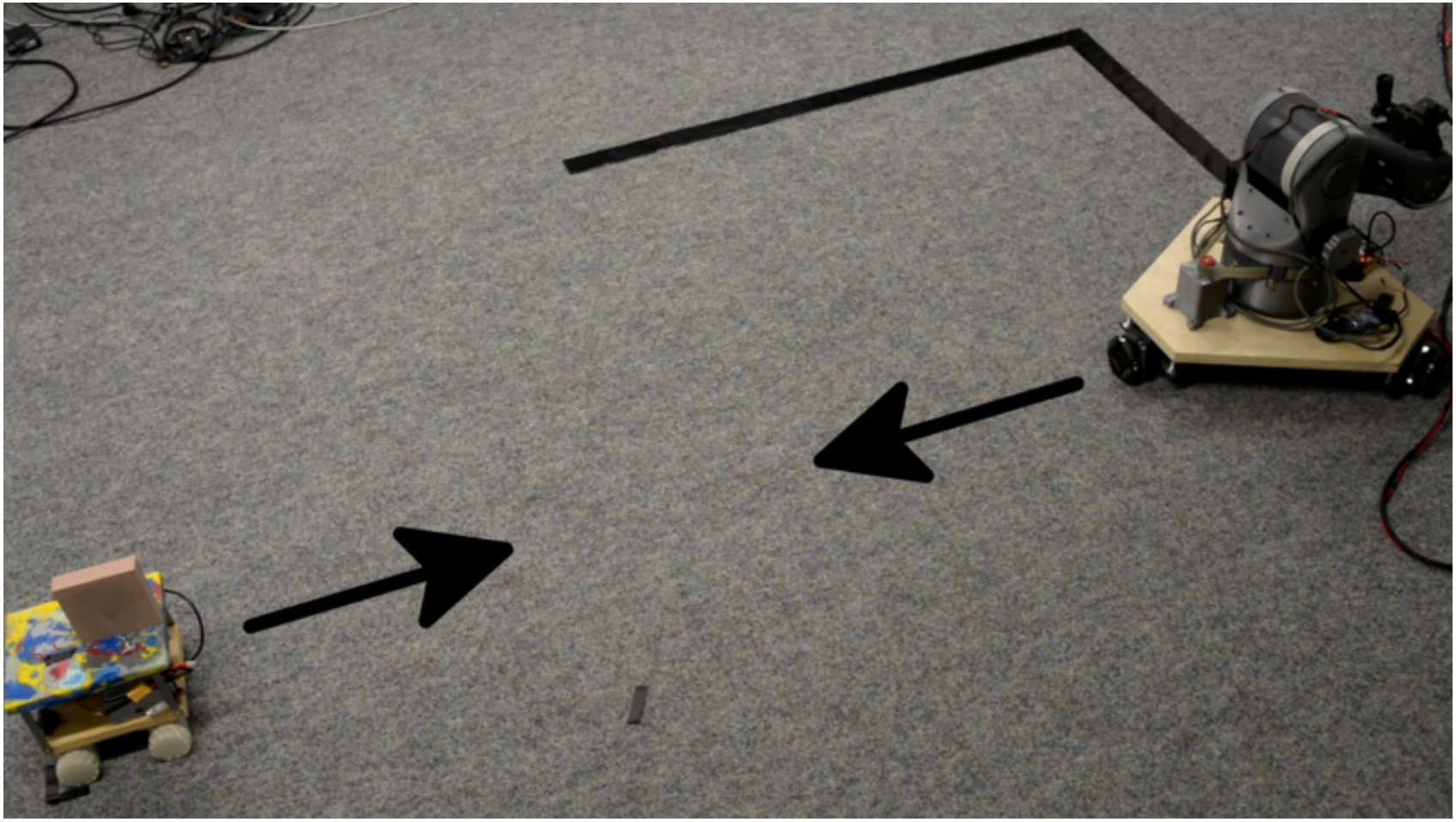




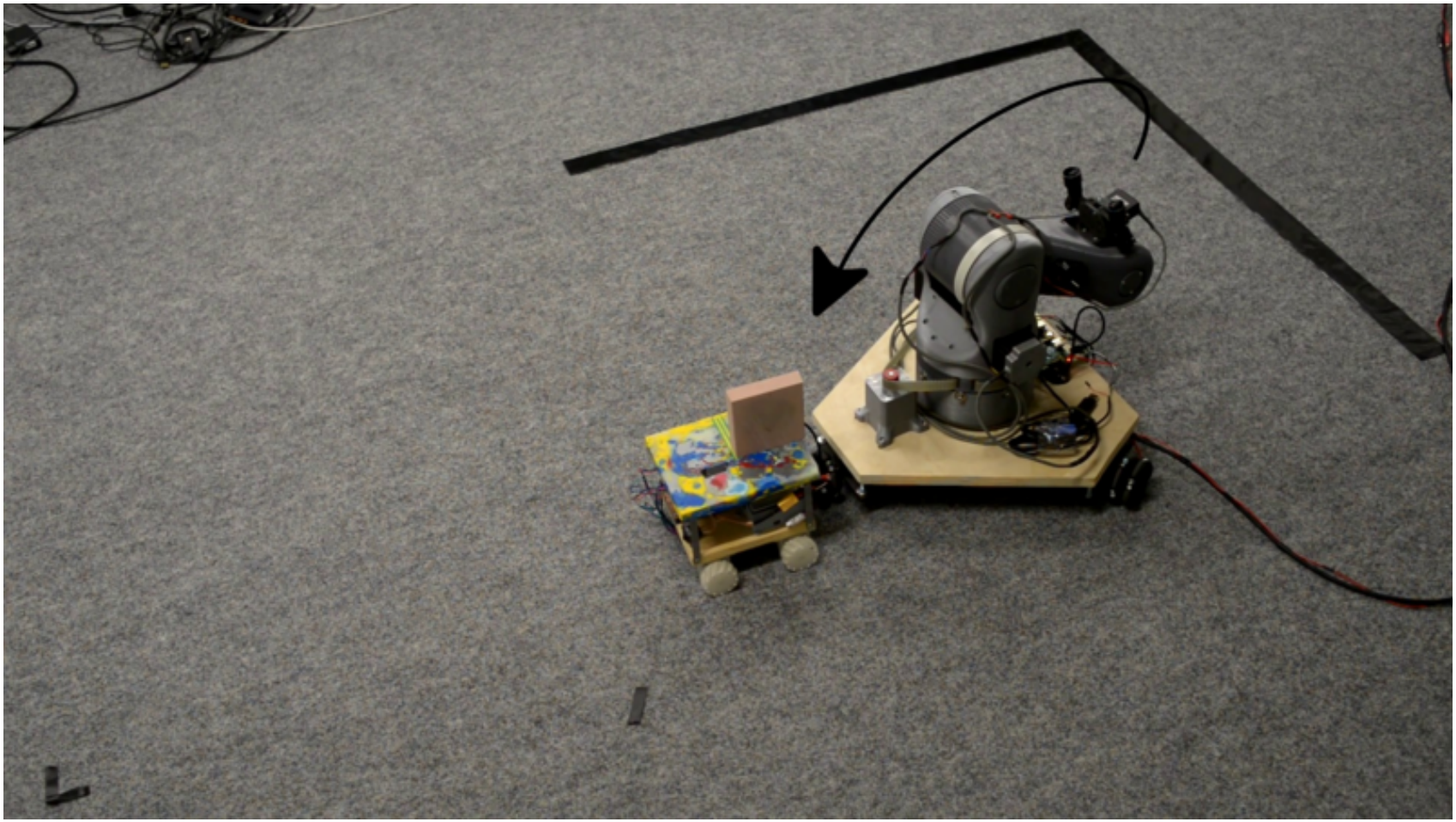
Case study (II): Motion Session Types for Robotics Interactions



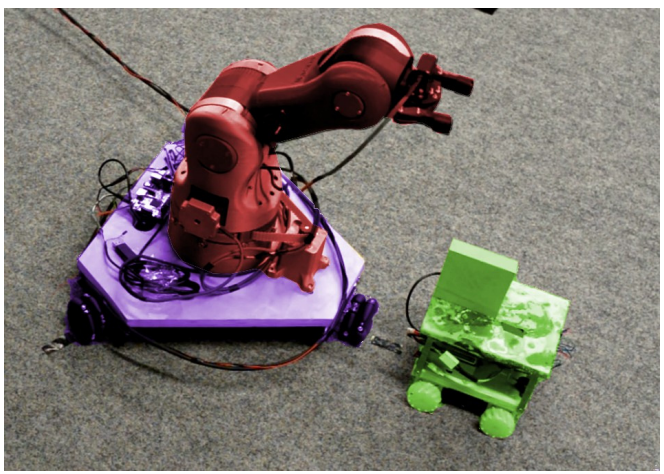
Case study (II): Motion Session Types for Robotics Interactions



Case study (II): Motion Session Types for Robotics Interactions



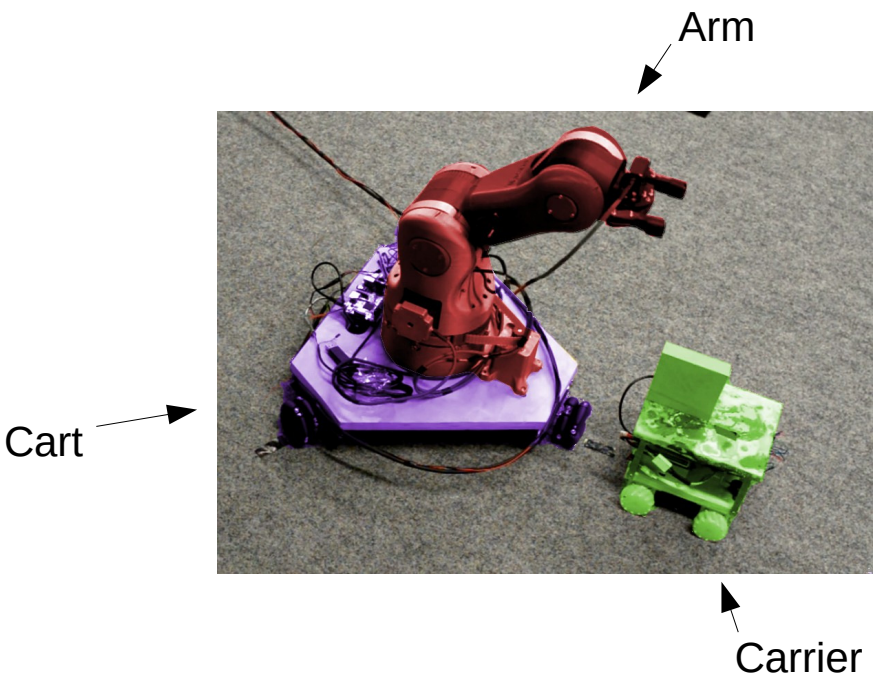
Case study (II): Motion Session Types for Robotics Interactions

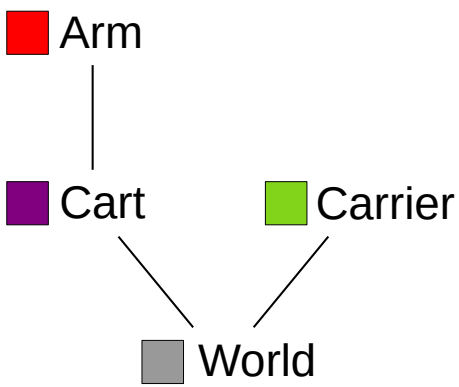


Cart & Arm:

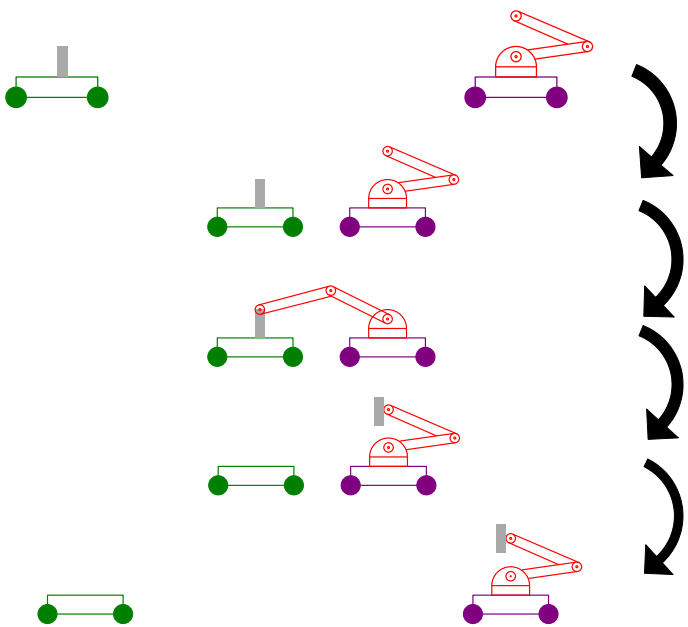
Two robots attached together that act as “one” robot (communication)

Case study (II): Motion Session Types for Robotics Interactions





Case study (II): Motion Session Types for Robotics Interactions



$\mathsf{Cart} \to \mathsf{Arm} : \mathit{fold}(\mathtt{unit}).\mathsf{dt}\langle \mathsf{Cart} : \mathsf{idle}, \mathsf{Carrier} : \mathsf{idle}, \mathsf{Arm} : \mathsf{fold} \rangle.$

$\mathsf{Arm} \to \mathsf{Cart} : \mathit{ok}(\mathtt{unit}).\mathsf{Cart} \to \mathsf{Carrier} : \mathit{ok}(\mathtt{unit}).$

$\mathsf{dt}\langle \mathsf{Cart} : \mathsf{move}, \mathsf{Carrier} : \mathsf{move}, \mathsf{Arm} : \mathsf{idle} \rangle.$

$\mathsf{Carrier} \to \mathsf{Cart} : \mathit{ok}(\mathtt{unit}).\mathsf{Cart} \to \mathsf{Arm} : \mathit{grab}(\mathtt{unit}).$

$\mathsf{dt}\langle \mathsf{Cart} : \mathsf{idle}, \mathsf{Carrier} : \mathsf{idle}, \mathsf{Arm} : \mathsf{grip} \rangle.$

$\mathsf{Arm} \to \mathsf{Cart} : \mathit{ok}(\mathtt{unit}).\mathsf{Cart} \to \mathsf{Carrier} : \mathit{ok}(\mathtt{unit}).$

$\mathsf{dt}\langle \mathsf{Cart} : \mathsf{move}, \mathsf{Carrier} : \mathsf{move}, \mathsf{Arm} : \mathsf{idle} \rangle.$

$\mathsf{Cart} \to \mathsf{Arm} : \mathit{done}(\mathtt{unit}).\mathsf{Cart} \to \mathsf{Carrier} : \mathit{done}(\mathtt{unit}).\mathtt{end}$

Recent Student Projects

Looking for **student projects** or **PhDs**? Here are some from previous years:

- Complexity Analyses of Subtyping of Session Types
  - **Published at PLACES'24**
- Implementing Flexible Multiparty Session Types in **TypeScript**
  - **Published at ECOOP'22**
- Implementing Asynchronous Multiparty Session Types in **Rust**
  - **Published at PPoPP'22**
- Safe Session Web Programming in **TypeScript**
  - **Published at CC'21**
- Efficient runtime for concurrent programs in **Dotty / Scala 3**
  - **Published at PLDI'19**
- Session type providers in **F#**
  - **Published at CC'18**

Student Project Topics

Looking for **research topics** from this course? Here are some of our proposed topics:

- Integration of session types with various programming languages: e.g. **Scala**, **Rust**, **Go**, **Haskell**, **OCaml**, **Typescript**, **F#**, **PureScript**, . . .
- Applications of session types for **cyberphysical systems**, **robotics** and **cyber security**.
- . . . and we are **open to suggestions!**

We are looking for **MSC project and PhD students**.

Conclusion

Session types allow to:

1. **formalise protocols** for concurrent/distributed systems
2. **verify concurrent programs** at **compile-time**
3. and also **implement automatic run-time monitoring**

This leads to two main uses:

- spot **protocol violations** and **deadlocks** at **compile-time**
- and detect and **report protocol violations** at **run-time**

Conclusion

Session types allow to:

1. **formalise protocols** for concurrent/distributed systems
2. **verify concurrent programs** at **compile-time**
3. and also **implement automatic run-time monitoring**

This leads to two main uses:

- spot **protocol violations** and **deadlocks** at **compile-time**
- and detect and **report protocol violations** at **run-time**

For papers and more info, visit:

https://mrg.cs.ox.ac.uk/

. . . and get in touch for enquiries and research topics!

Questions?