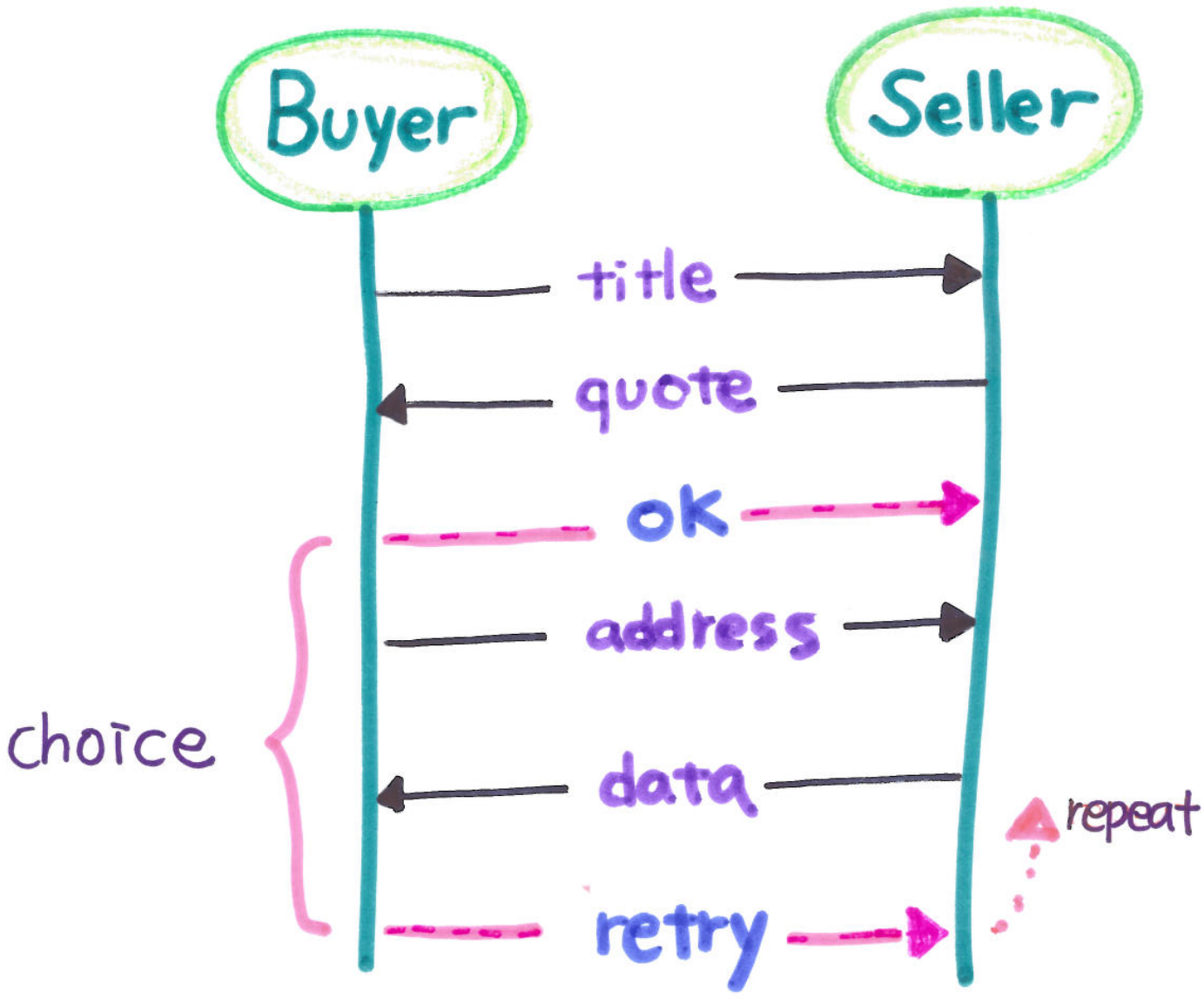
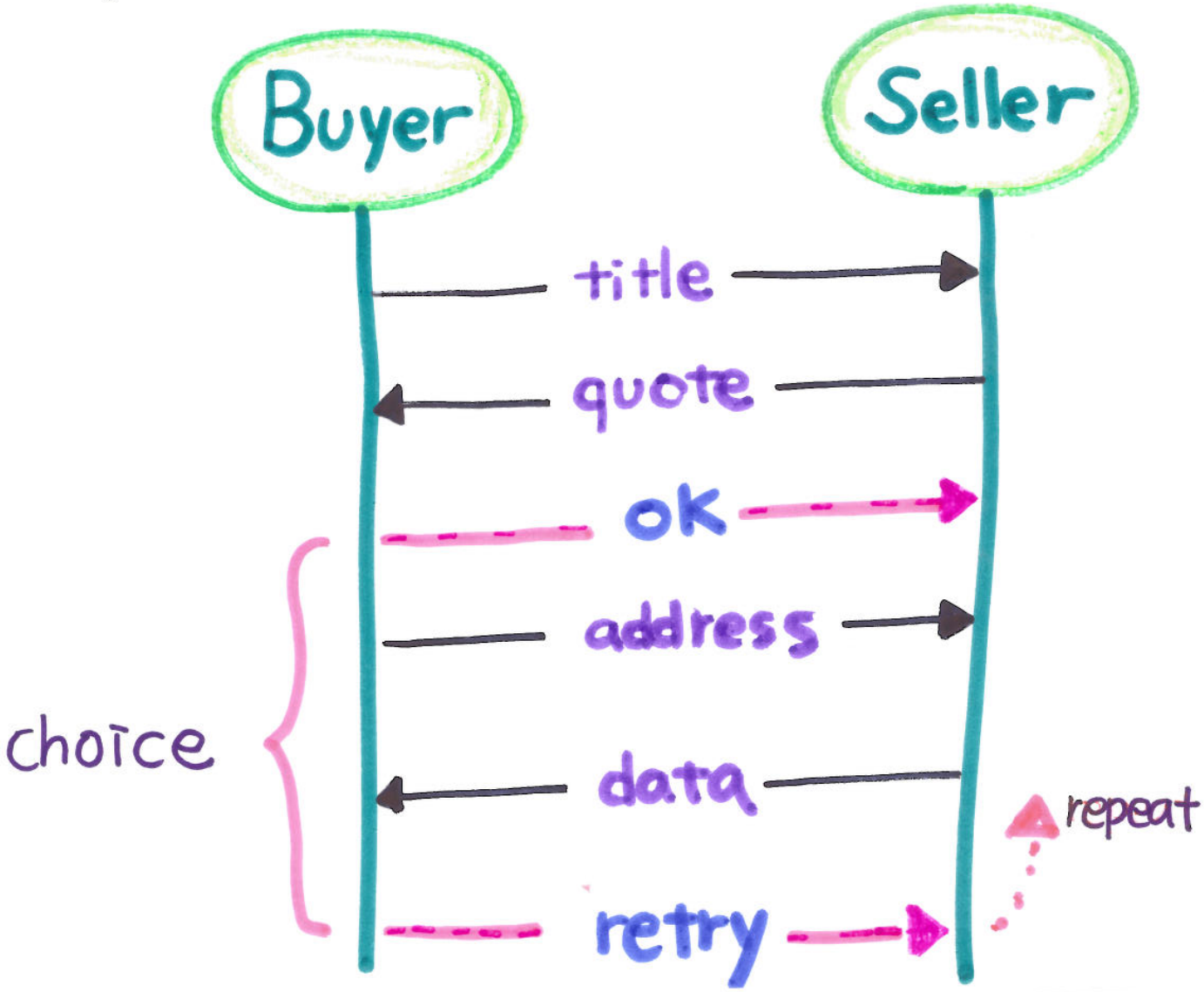
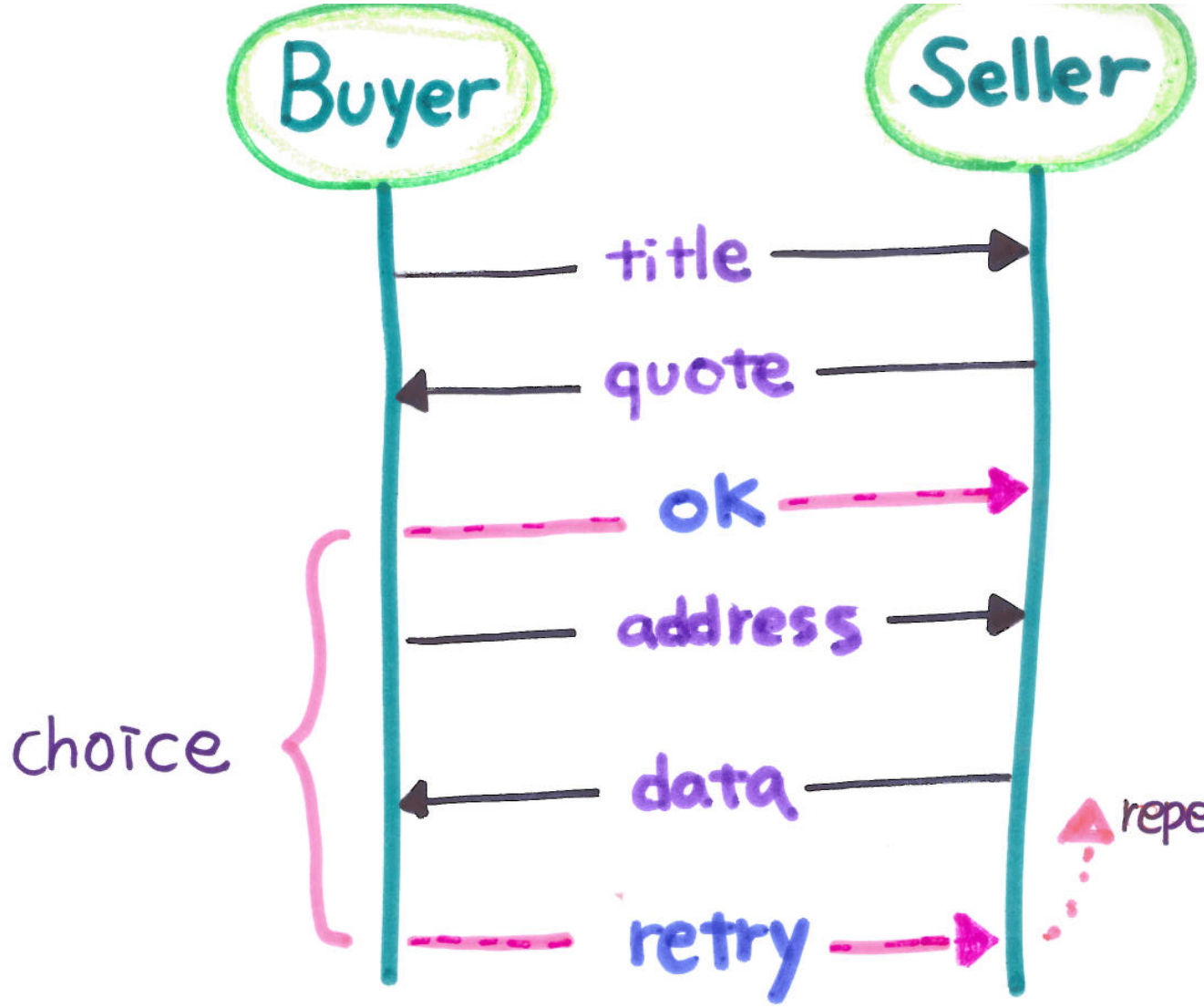
Binary Session Types: Buyer - Seller Protocol



Binary Session Types: Buyer - Seller Protocol



μt. ! Title ; ? Quote ; ! { ok : ! Add ; ? Date , retry : t }



P has T

Q has $\overline{T}$

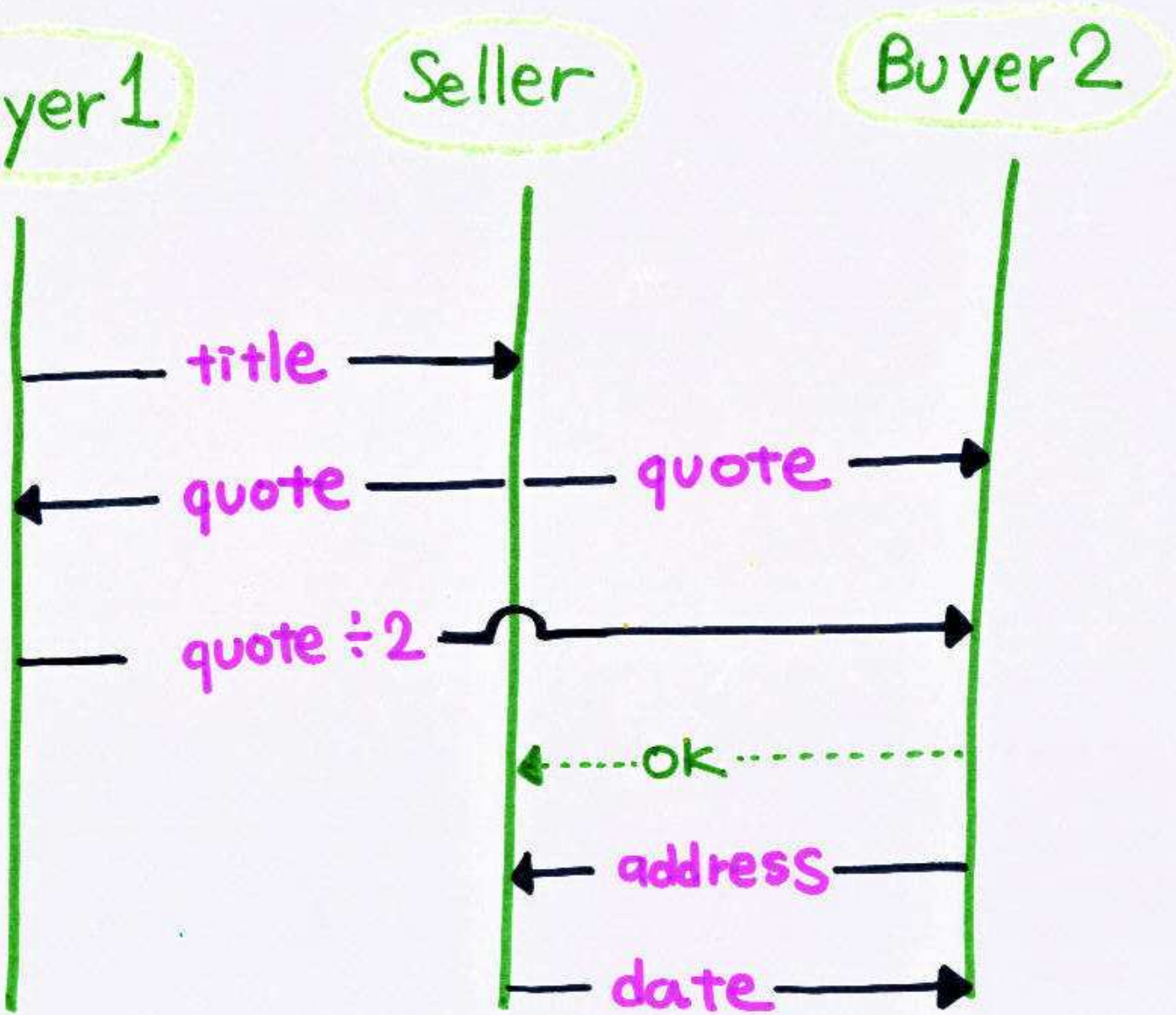
dual

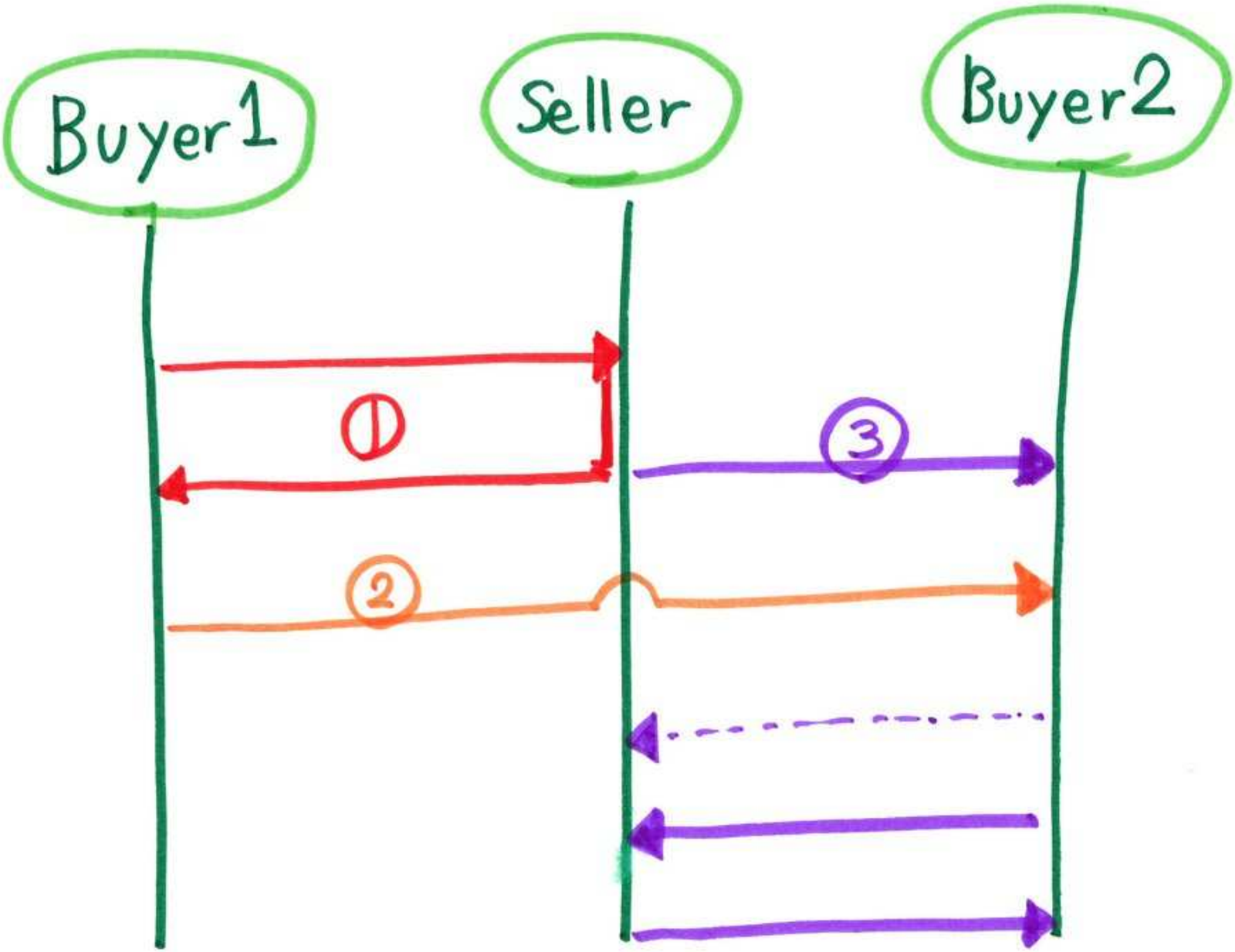
P | Q typable

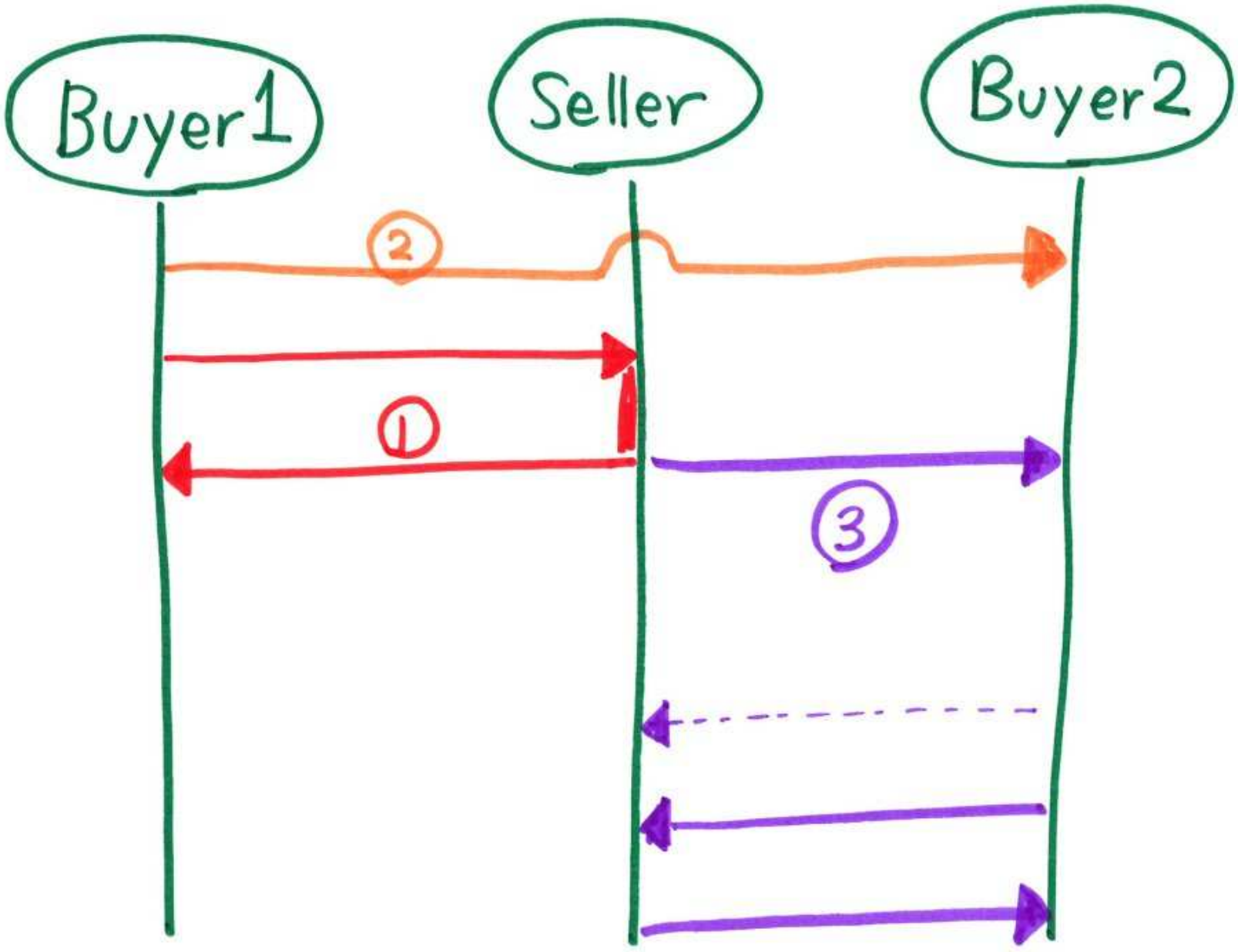
$\mu t.!\ \text{Title}\ ;\ ?\ \text{Quote}\ ;\ !\ \{\mathbf{ok}: !\ \text{Add}\ ;\ ?\ \text{Date}\ ,\ \mathbf{retry}: t\}$

$\mu t.?\ \text{Title}\ ;\ !\ \text{Quote}\ ;\ ?\ \{\mathbf{ok}: ?\ \text{Add}\ ;\ !\ \text{Date}\ ,\ \mathbf{retry}: t\}$

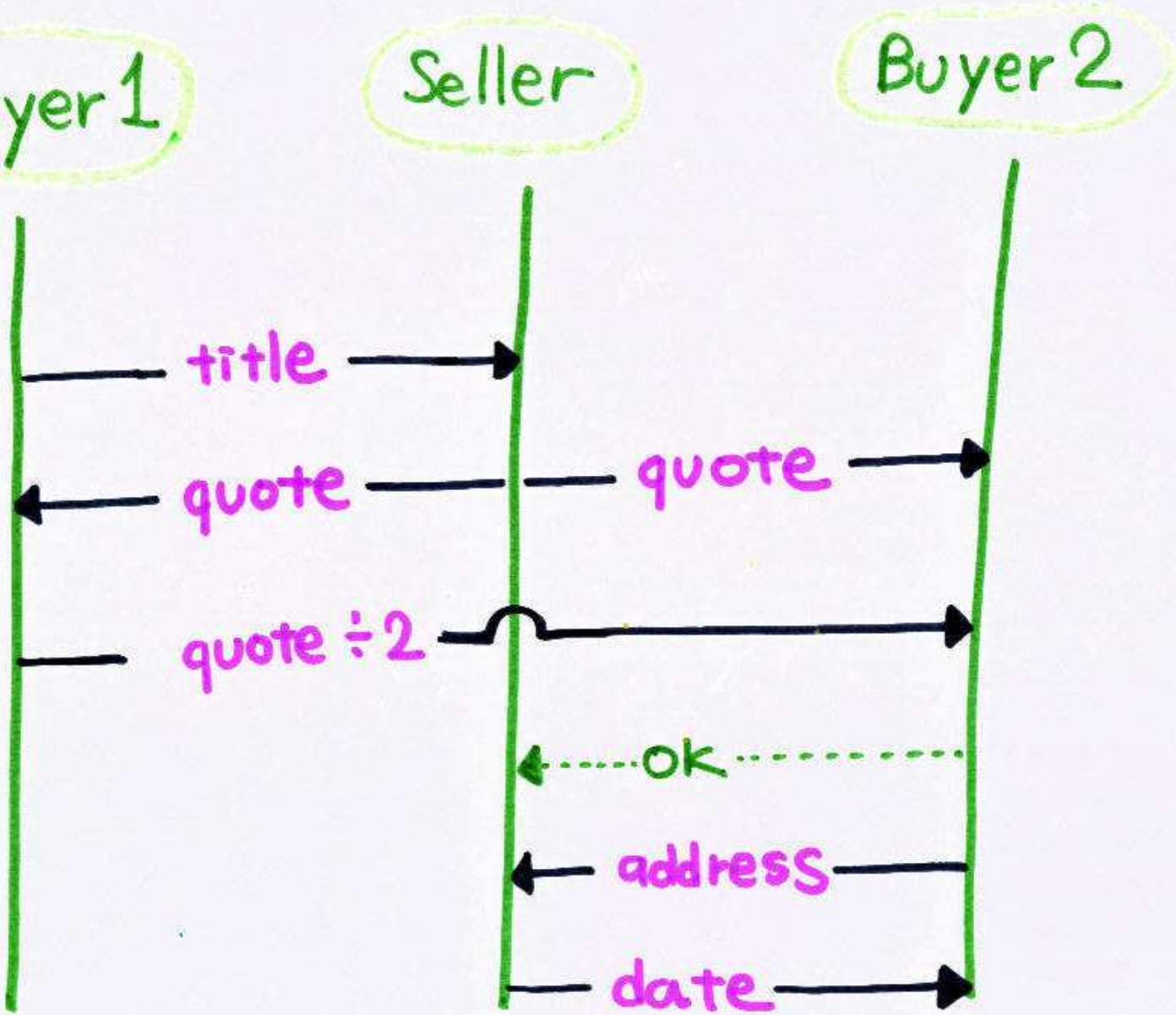
Multiparty Session Types

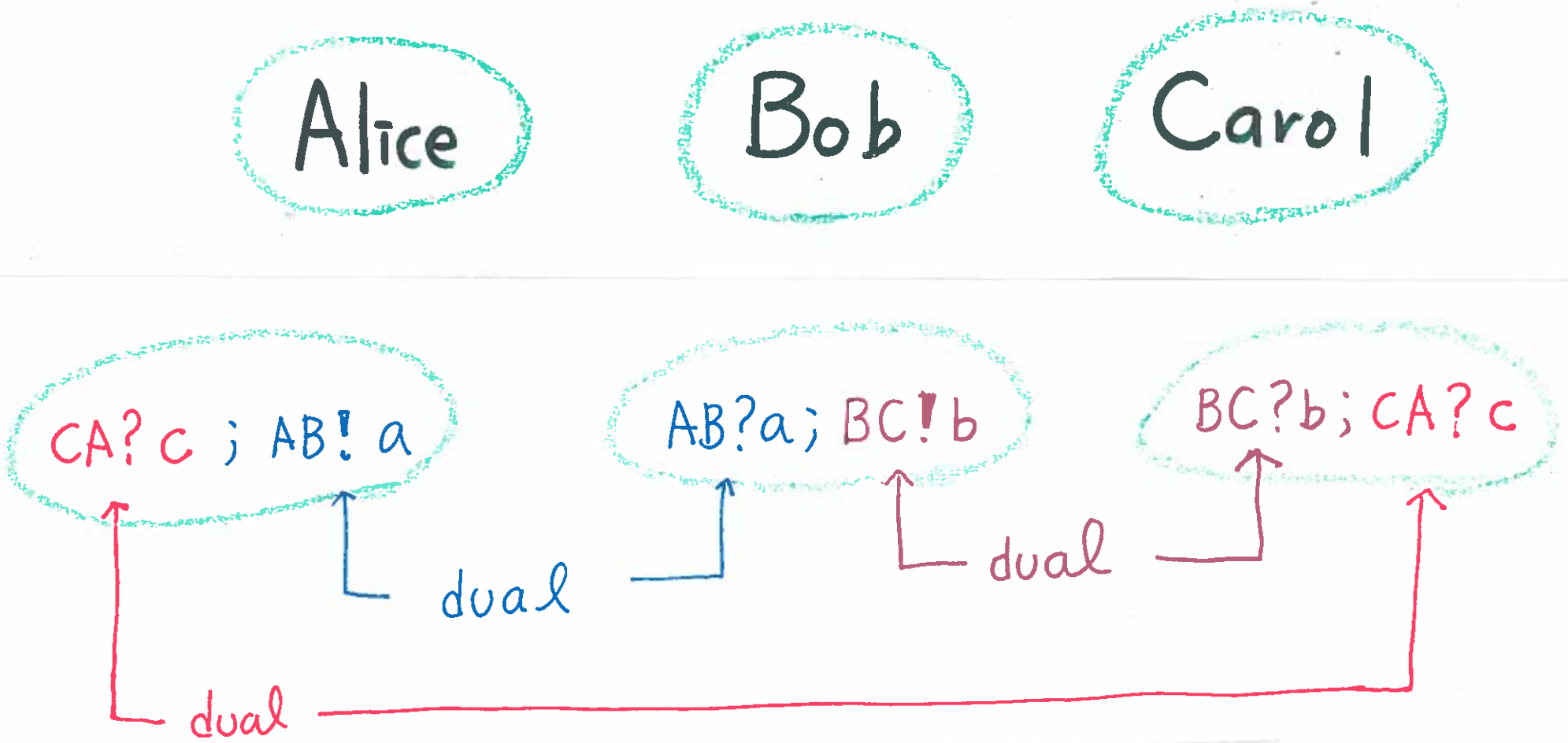



Buyer1
Seller
Buyer2
1
3
2


Buyer1
Seller
Buyer2
2
1
3

Multiparty Session Types



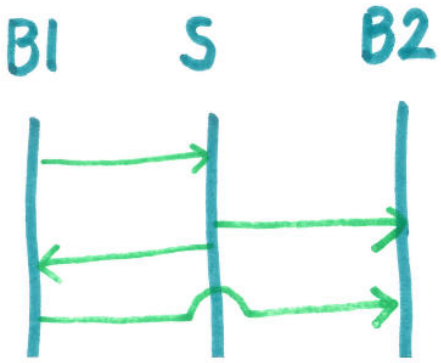


3 dual pairs

If you use binary Session Types ....

Deadlock!

Multiparty Session Types [Honda, Yoshida, Carbone 2008]



G

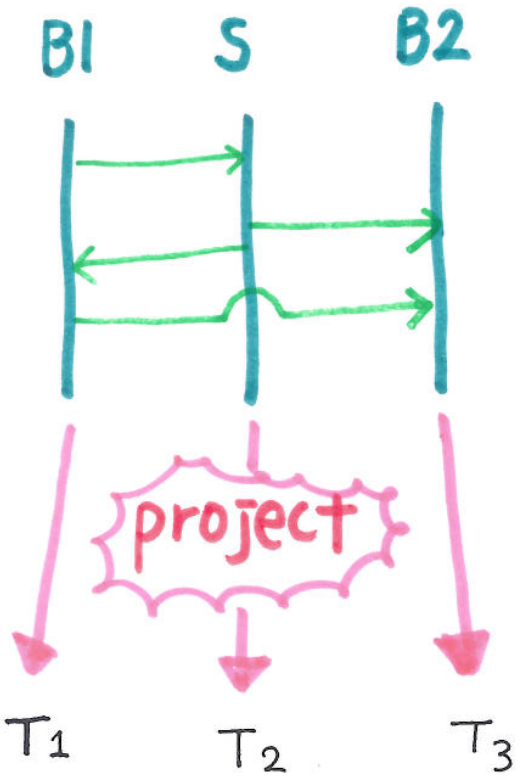
B1 → S Int.

S → B2 Char

STEP 1

Write Global Type

Multiparty Session Types [Honda, Yoshida, Carbone 2008]



G

$B1 \rightarrow S$ Int.

$S \rightarrow B2$ Char

T

$B_1?$ Int. $B_2!$ Char

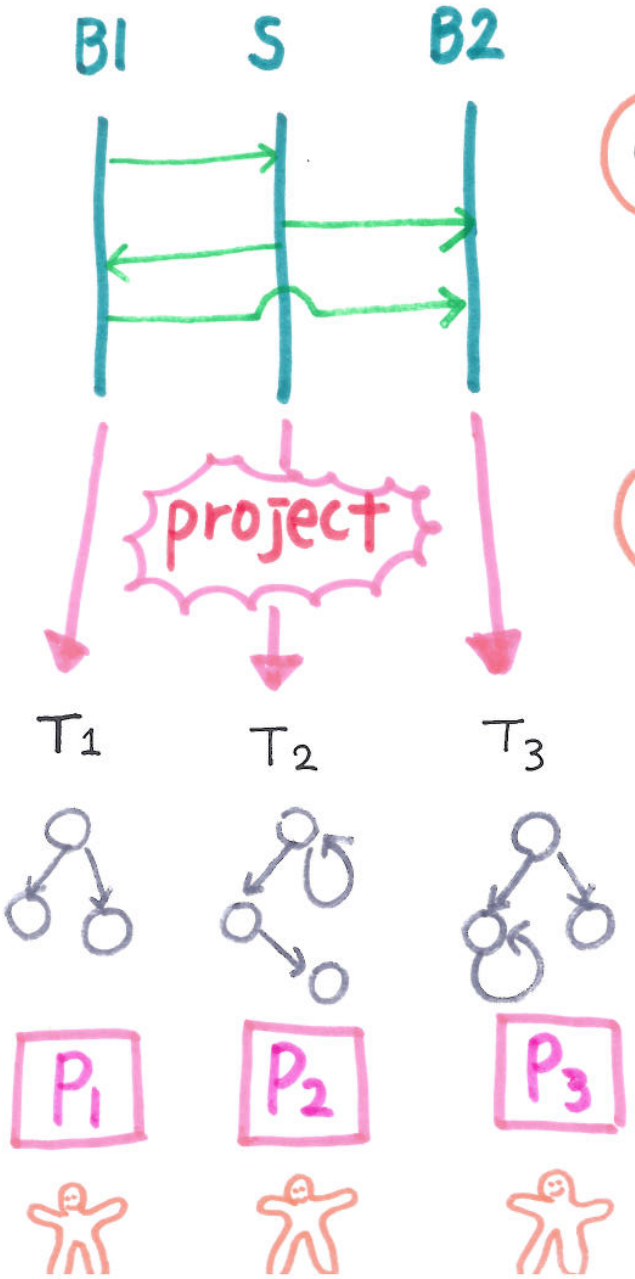
STEP 1

Write Global Type

STEP 2

Project to Local Types

Multiparty Session Types [Honda, Yoshida, Carbone 2008]



G $B1 \rightarrow S$ Int.
$S \rightarrow B2$ Char

T $B_1?\text{Int}.\, B_2!\text{Char}$

P $B_1?(x).\, B_2!\langle\text{"apple"}\rangle$

STEP 1
Write Global Type

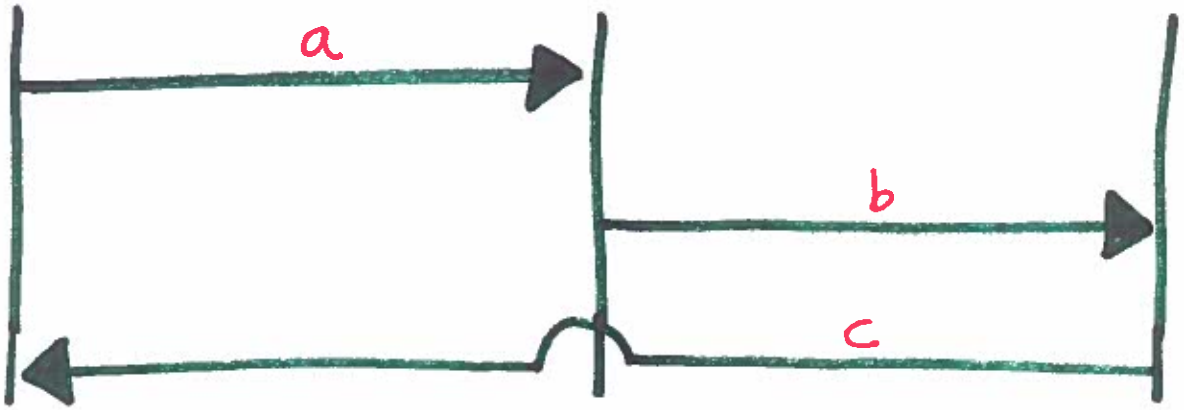
STEP 2
Project to Local Type

STEP 3
- Static Check
- Generate Code
- Run-time Check

Alice

Bob

Carol



Global Type

Projection

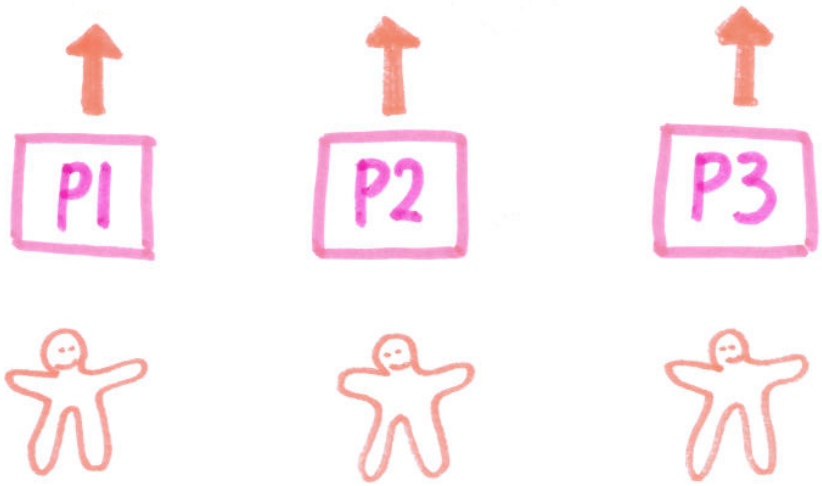
Alice AB!a; CA?c

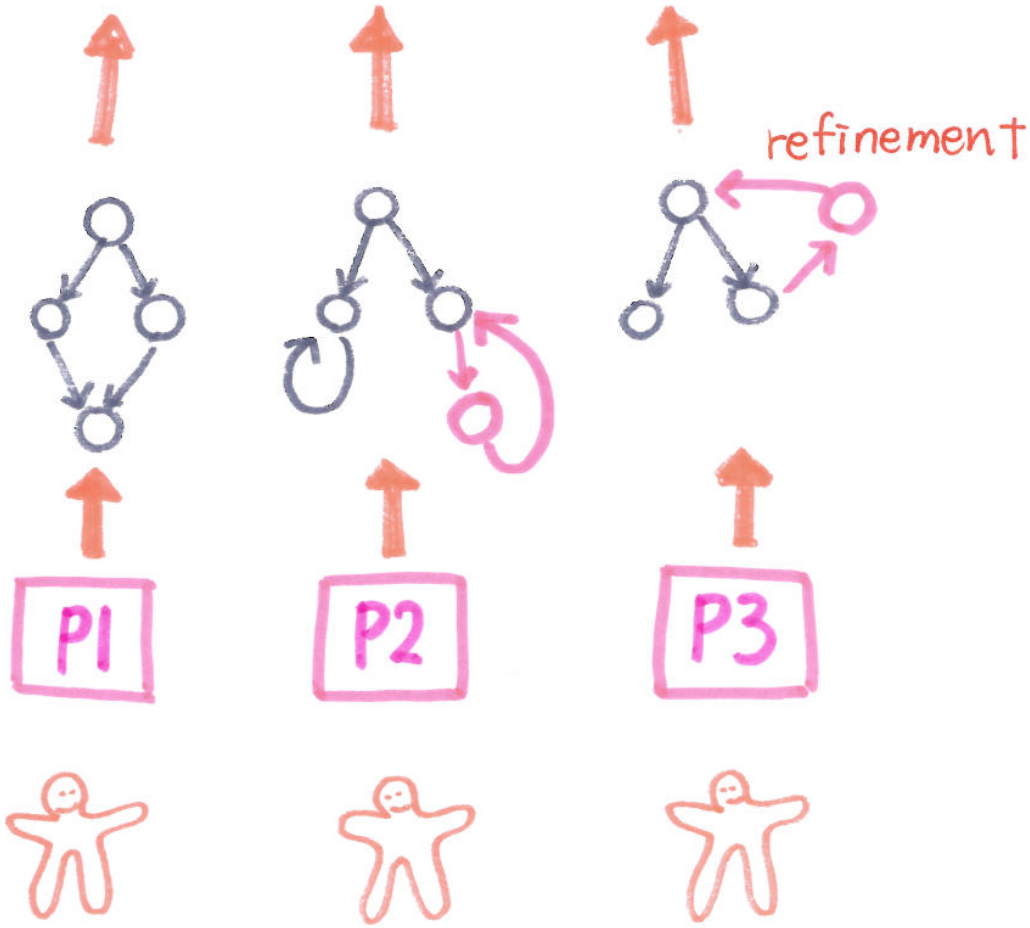
Bob AB?a; BC!b

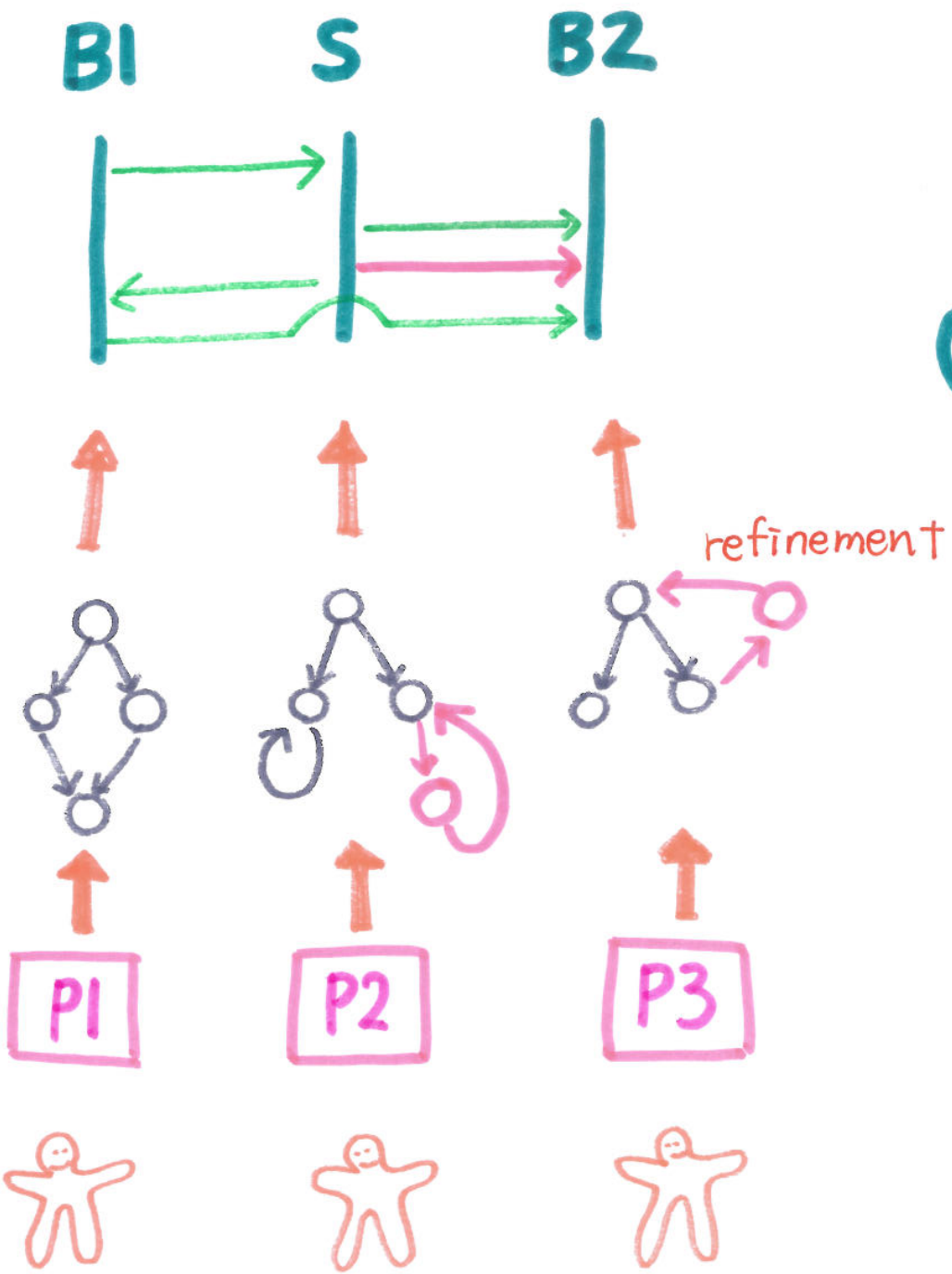
Carol BC?b; CA!c;

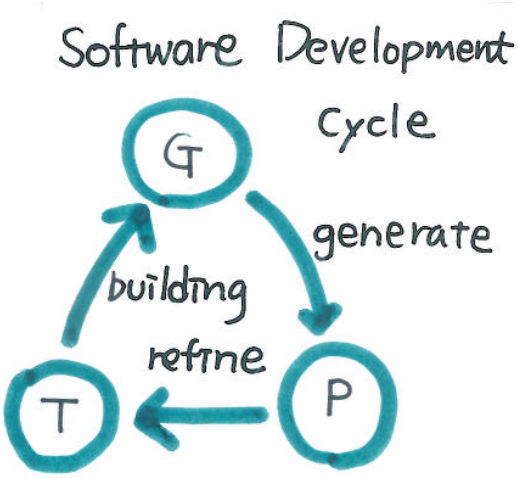
LOCAL TYPES

NO Deadlock


P1
P2
P3


refinement
P1
P2
P3



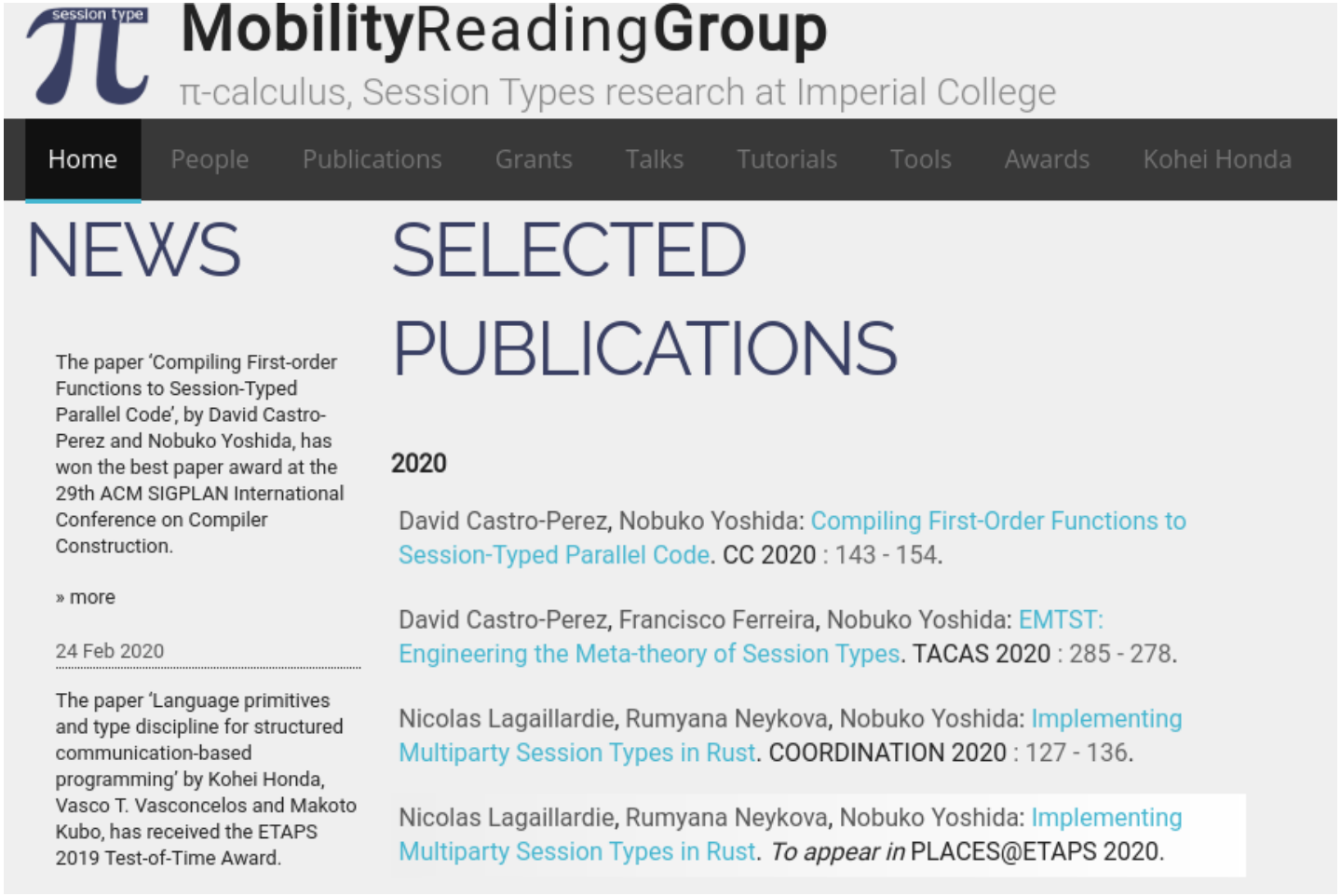


- Optimisation
- refinement
- Inference
- Testing

The Mobility Reading Group at Imperial College

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

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



- **TACAS'20**: **Coq Mechanisation**. EMTST: Engineering the Meta-theory of Session Types. David Castro-Perez, Francisco Ferreira and NY
- **ESOP'20**: **Theory**. Exploring Type-Level Bisimilarity towards More Expressive Multiparty Session Types, Sung-Shik Jongmans and NY
- **CC'20**: **Parallel Programming** (**Best Paper Award**) Compiling First-Order Functions to Session-Typed Parallel Code, David Castro-Perez and NY
- **ECOOP'20**: **Go Language**. Static Race Detection and Mutex Safety and Liveness for Go Programs, Julia Gabet and NY
- **ECOOP'20**: **OCaml**. Multiparty Session Programming with Global Protocol Combinators, Keigo Imai, Rumyana Neykova, NY and Shoji Yuen
- **OOPSLA'20**: **Cost Analysis using Multiparty Session Types**. CAMP: Cost-Aware Multiparty Session Protocol, David Castro-Perez and NY
- **OOPSLA'20**: **Go Language**. Featherweight Go, Robert Griesemer and Raymond Hu and Wen Kokke and Julien Lange and Ian Lance Taylor and Bernardo Toninho and Philip Wadler and NY.
- **OOPSLA'20**: **Robotics**. Multiparty Motion Coordination: From Choreographies to Robotics Programs, Rupak Majumdar, NY and Damien Zufferey.
- **OOPSLA'20**: **F⋆**. Statically Verified Refinements for Multiparty Protocols, Fangyi Zhou, Francisco Ferreira, Raymond Hu, Rumyana Neykova and NY

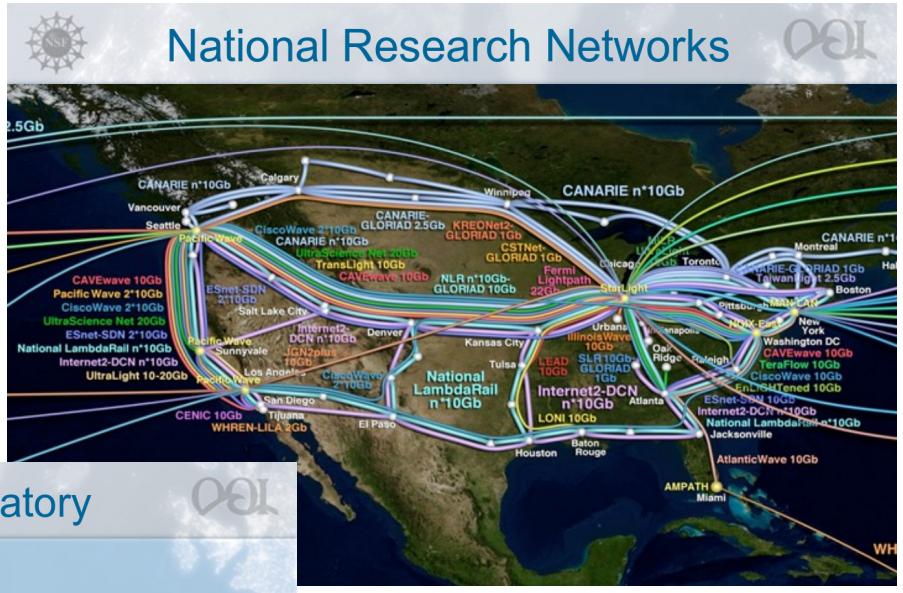
Case study (I): Network Protocols with Scribble

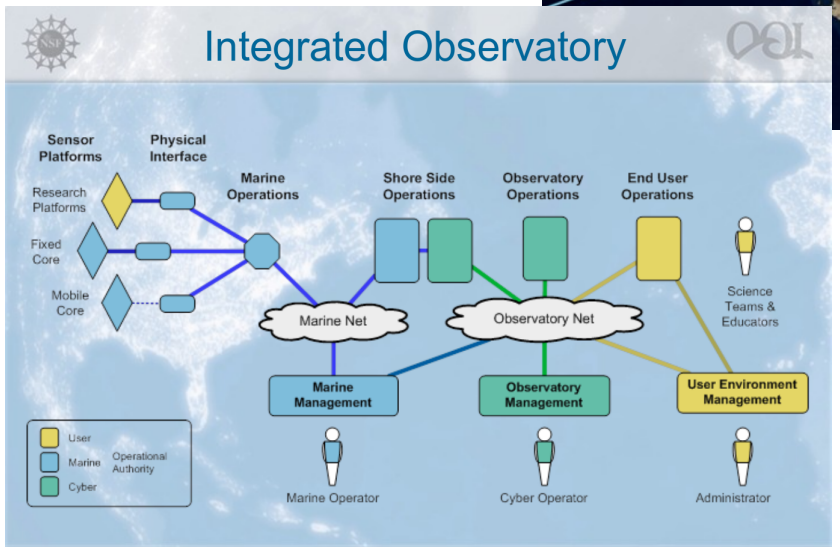
DEMO:

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

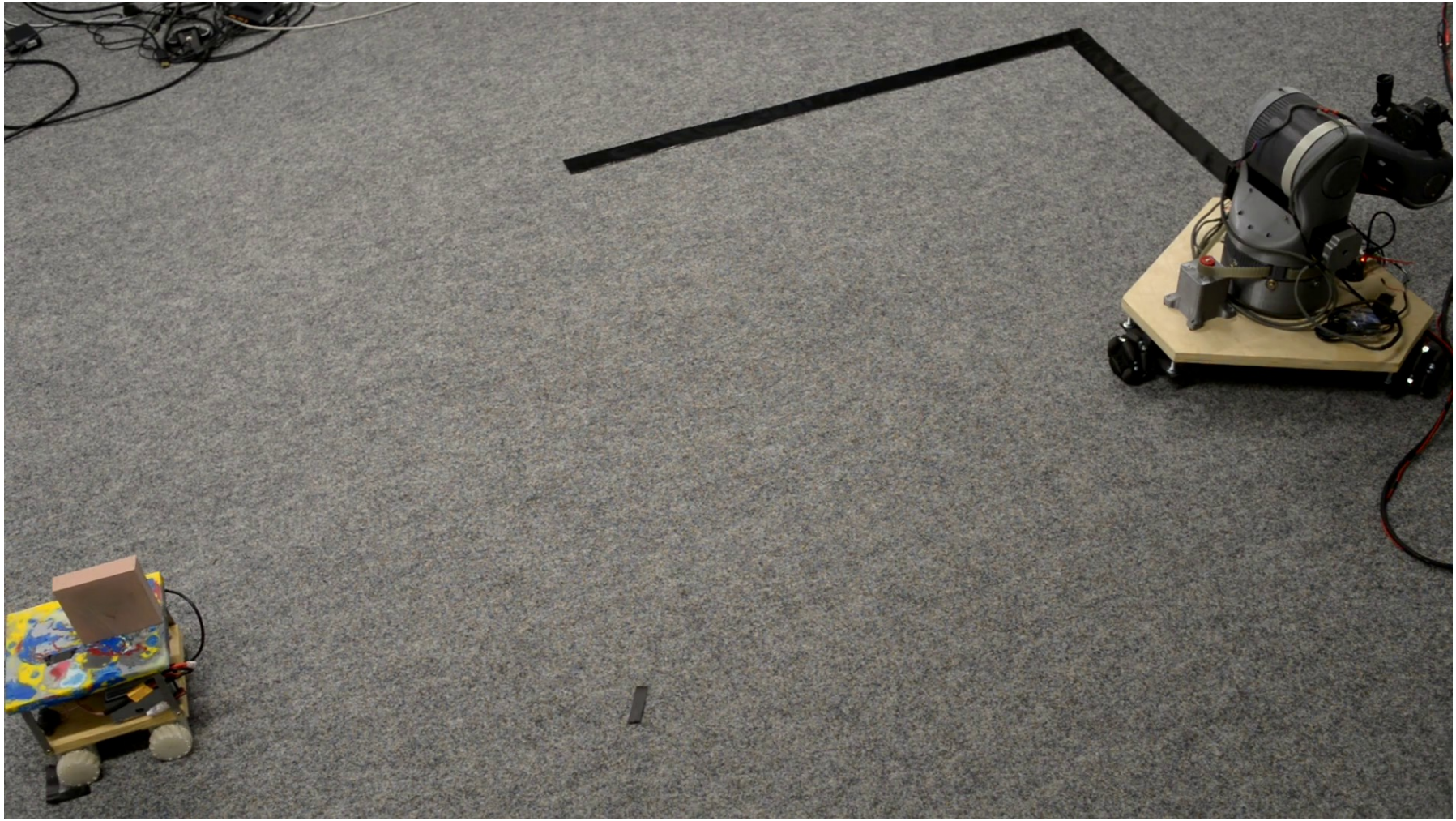
Case study (II): Ocean Observatories Initiative

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

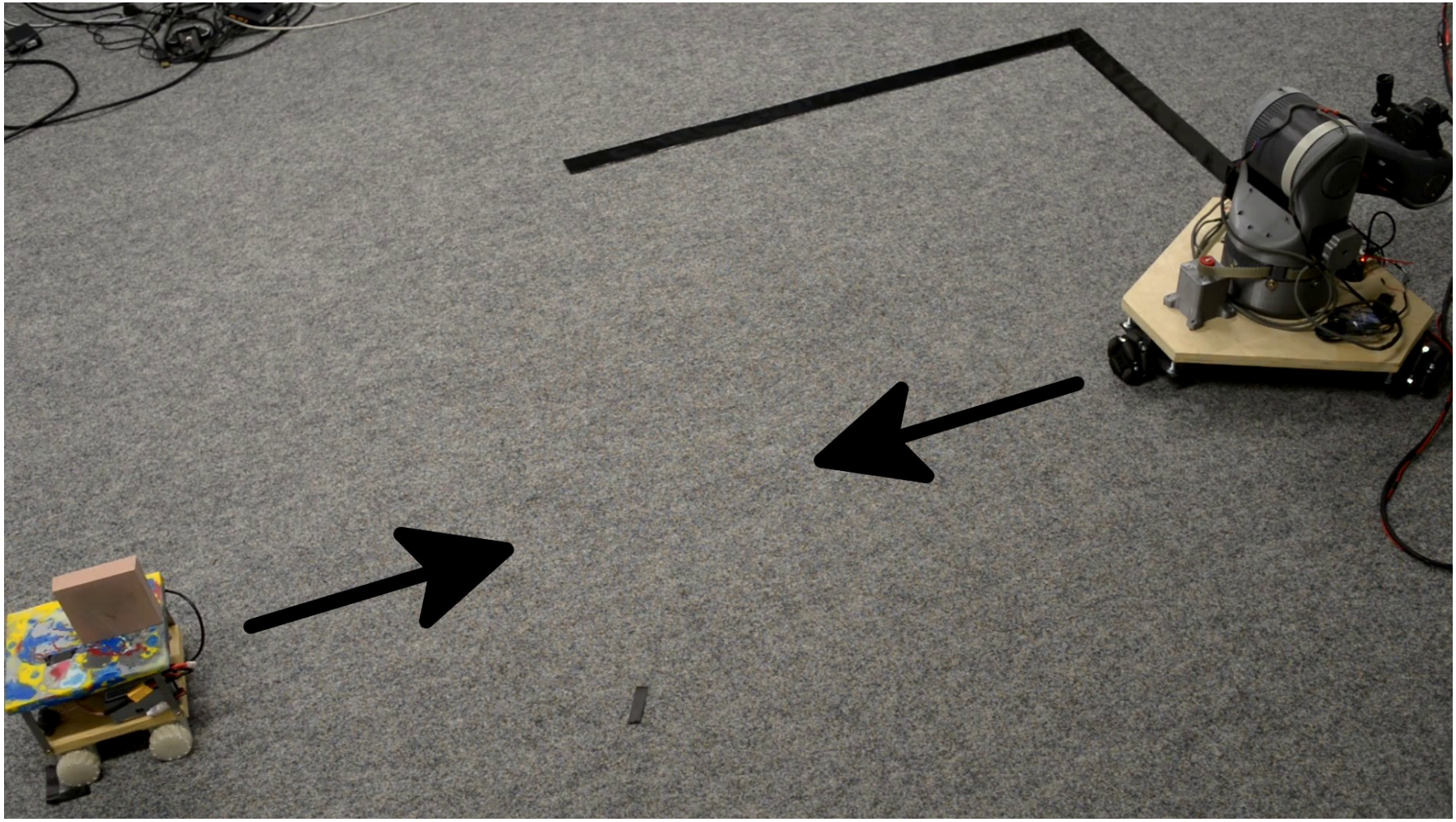




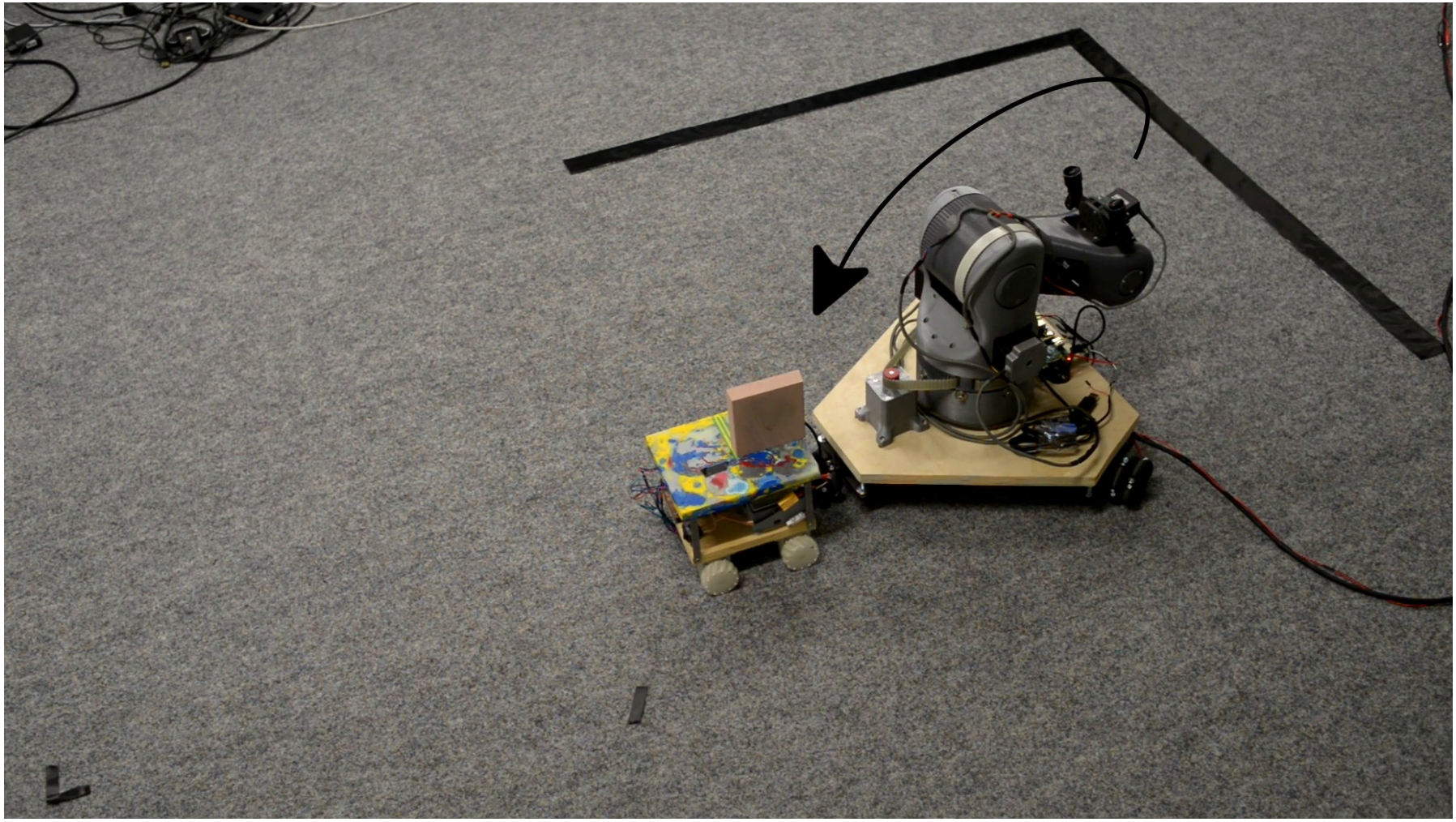
Case study (III): Motion Session Types for Robotics Interactions [ECOOP'19,OOPSLA'20]



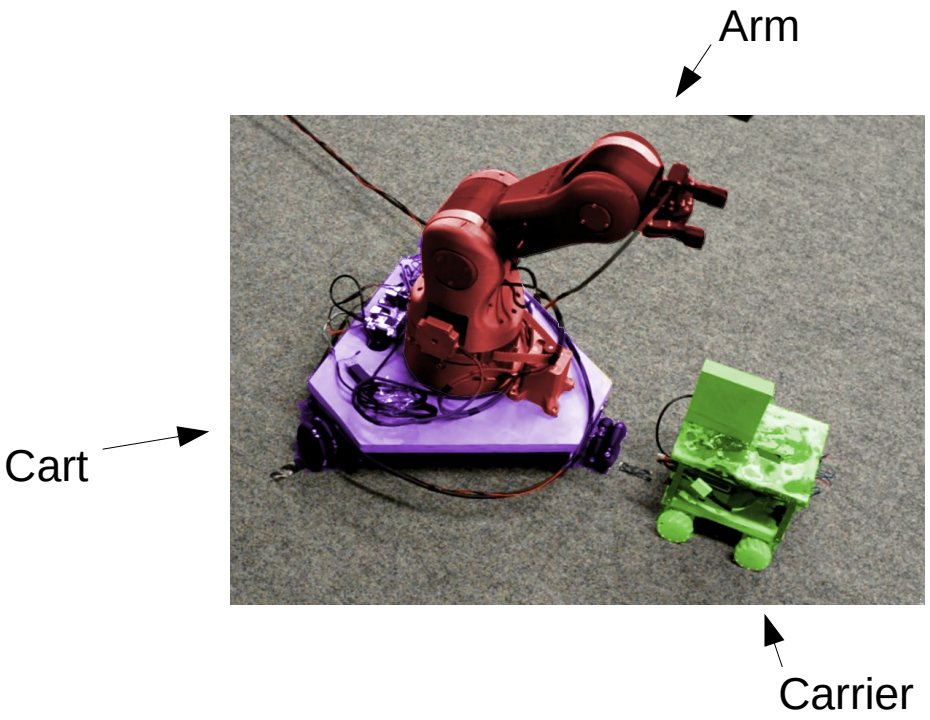
Case study (III): Motion Session Types for Robotics Interactions [ECOOP'19,OOPSLA'20]



Case study (III): Motion Session Types for Robotics Interactions [ECOOP'19,OOPSLA'20]



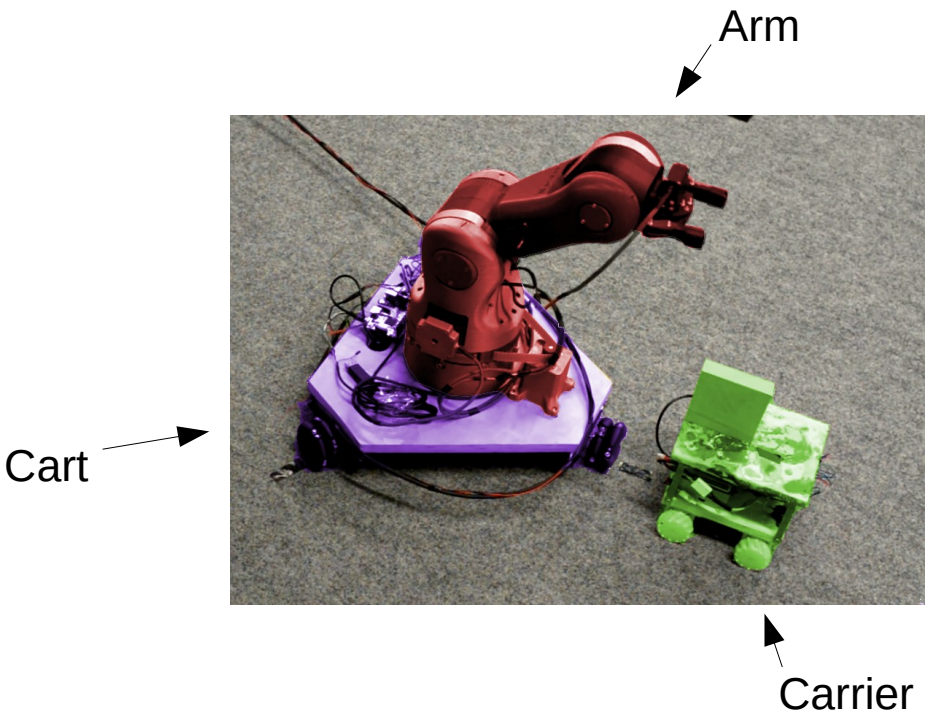
Case study (III): Motion Session Types for Robotics Interactions [ECOOP'19,OOPSLA'20]

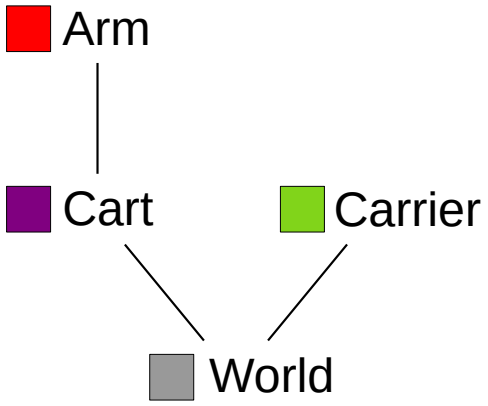


Cart & Arm:

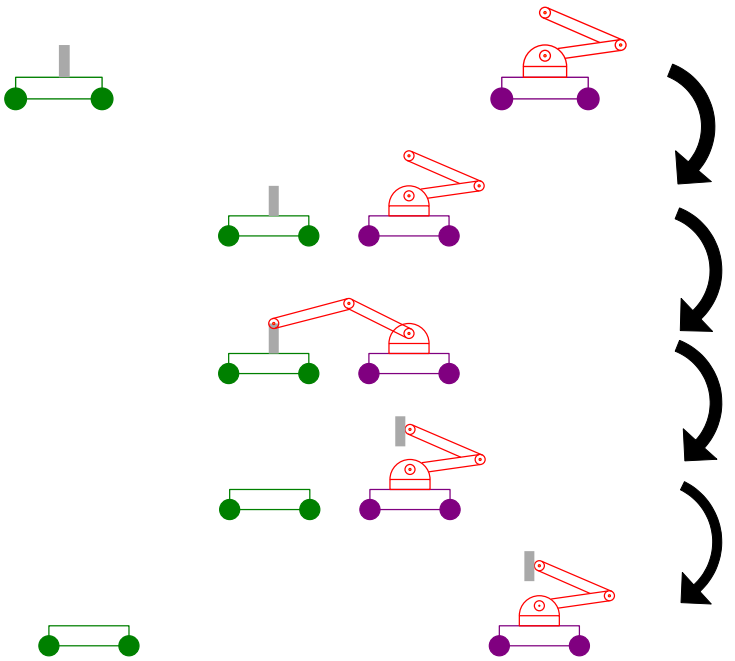
Two robots attached together that act as "one" robot (communication)

Case study (III): Motion Session Types for Robotics Interactions [ECOOP'19,OOPSLA'20]





Case study (III): Motion Session Types for Robotics Interactions [ECOOP'19,OOPSLA'20]



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

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

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

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

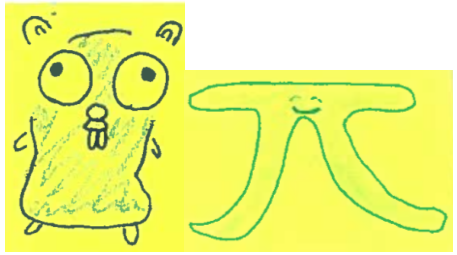
$\mathsf{dt}\langle \mathsf{Cart} : \mathsf{idle}, \mathsf{Carrier} : \mathsf{idle}, \mathsf{Arm} : \mathsf{grip}\rangle.$

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

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

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

Case Study IV Golang research



Go is a popular industrial systems language developed by **Google** (5th GitHub language). Go's design is inspired by **CSP**, but more akin to **the π-calculus**.

Our MRG group has several research projects involving **Go**:

- Static verification framework for **Go** [**CC'16**,**POPL'17**, **ICSE'18**,**ECOOP'20**].
- Distributed programming in **Go** based on MPST [**POPL'19**]
- Introducing **polymorphism** in **Go**, in collaboration with Google [**OOPSLA'20**].

Case Study IV Golang research

Robert Griesemer

Ian Lance Taylor

I want to thank you and your team for all the type theory work on Go so far – it really helped clarify our understanding to a massive degree. So thanks! (Robert Griesemer)

Conclusion

Session types allow to:

1. **formalise protocols** for concurrent/distributed systems
2. **verify concurrent and distributed 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 and distributed 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**

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

. . . and get in touch for enquiries and a **reading list**!
(**n.yoshida@imperial.ac.uk**)

Questions?