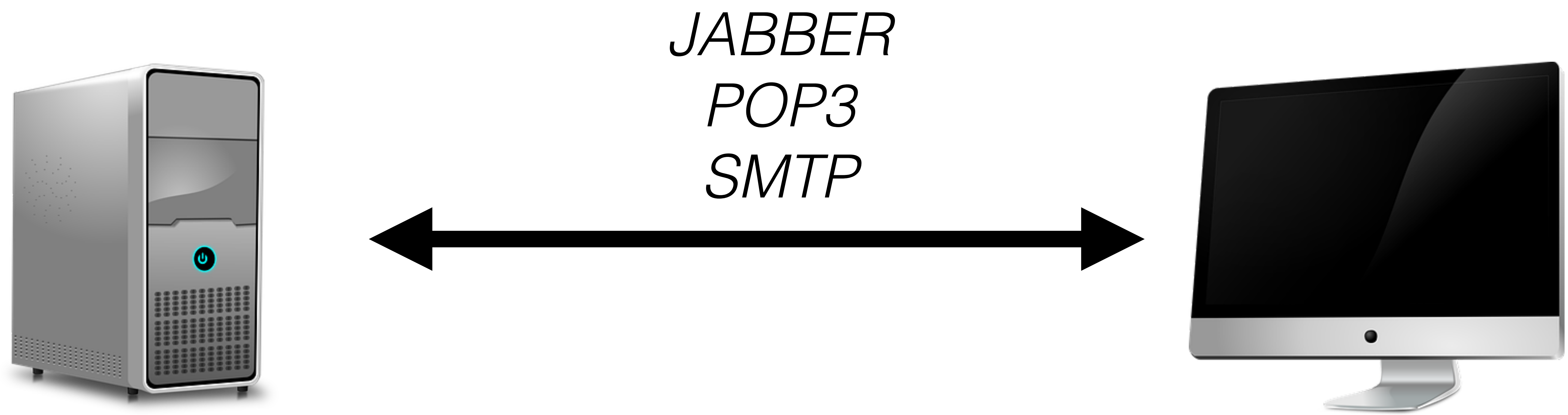
Practical Session Types

Christophe De Troyer
Christophe Scholliers
Wolfgang De Meuter

Vrije Universiteit Brussel
Universiteit Gent

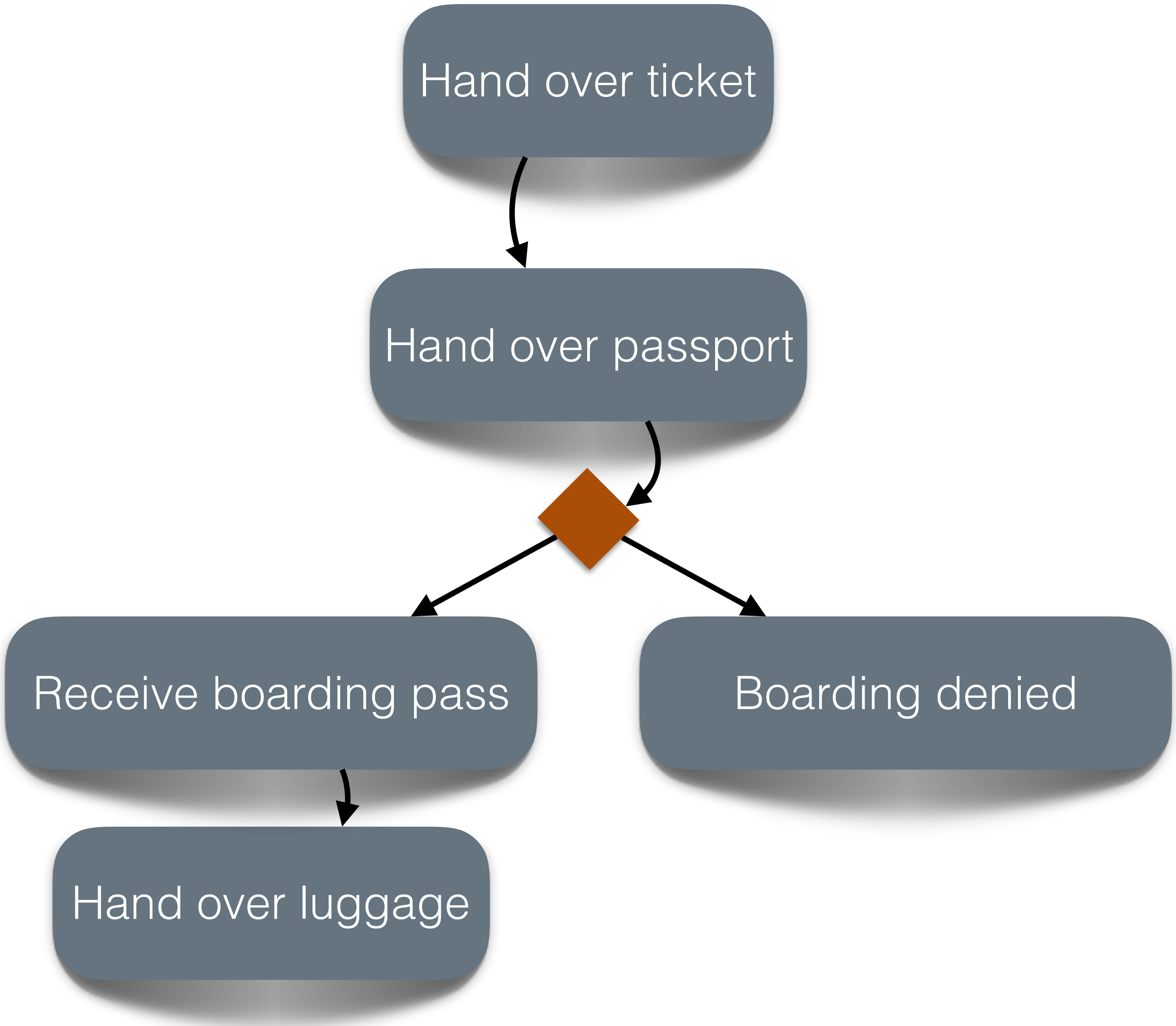
Session Types

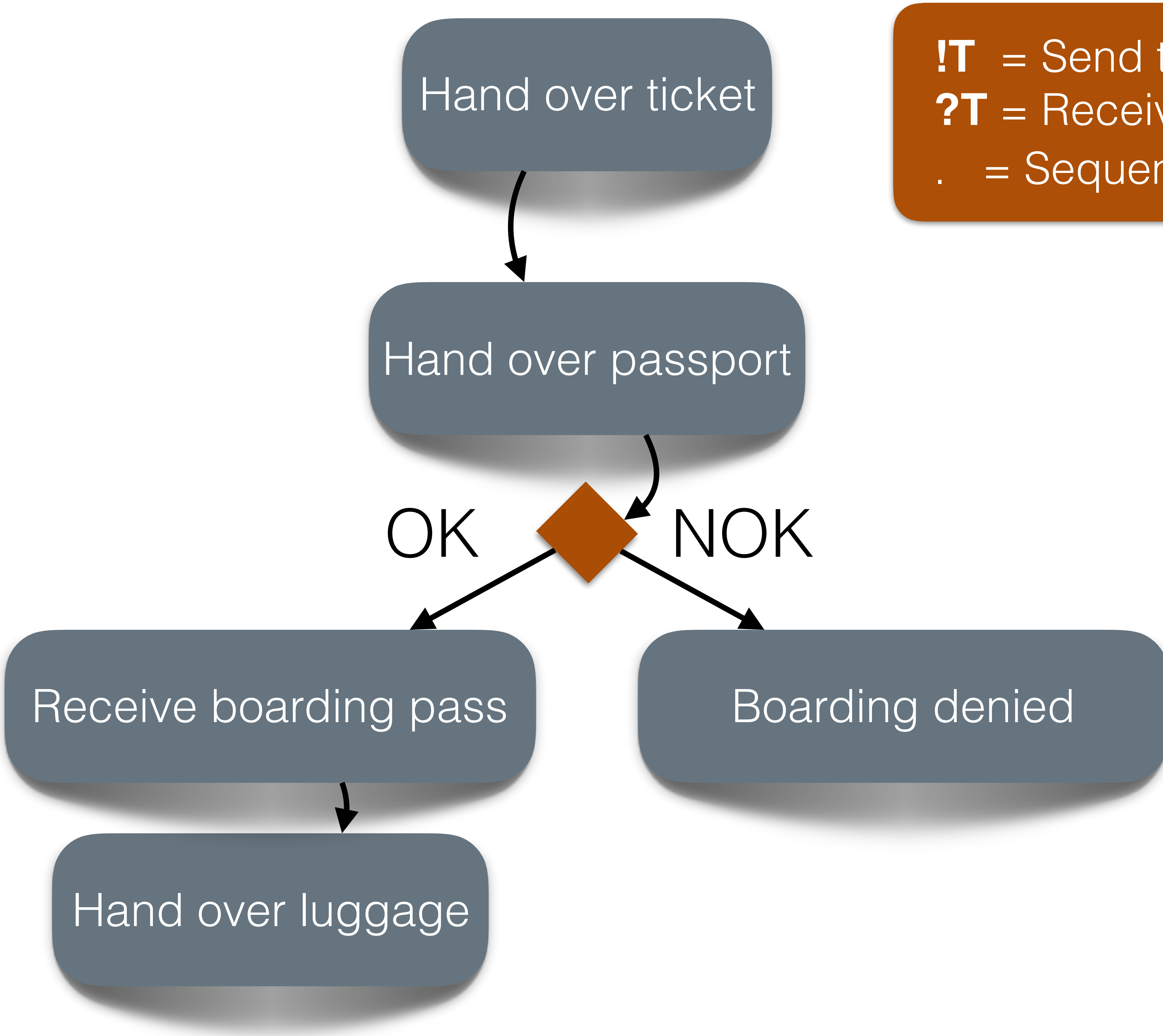


State of the art

Calculi	Programming Languages
Functional Calculi	Haskell, Scala, OCaml, Links, Sill, C, Erlang, Go, Rust, Java
π-calculus	
Object-Oriented	

Running Example + Session Types in a Nutshell


Hand over ticket
Hand over passport
Receive boarding pass
Boarding denied
Hand over luggage



!T = Send type T
?T = Receive type T
. = Sequence

```
!Ticket.!Passport.&<OK : ?BoardingPass.!Luggage.End,
                    NOK: ?String.End
>
```

Duality

```
!Ticket.!Passport.&<OK : ?BoardingPass.!Luggage.End,
                    NOK: ?String.End                          >
```

!T → ?T

?T → !T

&<..> → ⊕<..>

⊕ <..> → &<..>

```
?Ticket.?Passport.⊕<OK : !BoardingPass.?Luggage.End,
                    NOK: !String.End                          >
```

Modularity is difficult

Types

Logic

```
sestyp Checkin = ⊕<NORMAL: !Ticket.!Passport,
                   FAST  : !Barcode          >.&<OK : ?BoardingPass.!Luggage.End,
                                                NOK: End                         >

sendTicket airport = let ticket = grabTicket in send ticket airport

sendPassport airport = let passport = grabPassport in send passport airport

getBoardingPass airport = return (receive airport)

sendLuggage airport = let luggage = grabFromTrunk in send luggage airport ;

checkin airport = select NORMAL airport ;
                        sendTicket ; sendPassport ;
                  case airport
                    OK  = let boardingpass = getBoardingPass airport
                          in
                            sendLuggage airport ;
                            close airport
                    NOK = let error = receive airport
                          in
                            close airport ;
                            error (format “Go back home: %s” error)
```

Refactor

```
sestyp Checkin = ⊕<NORMAL: !Ticket.!Passport,
                   FAST  : !Barcode          >.&<OK : ?BoardingPass.!Luggage.End,
                                                NOK: End                         >

sendTicket :: [ß:Checkin] Chan ß
              ->
              [ß:!Passport.&<OK : ?BoardingPass.!Luggage.End, NOK: End >] Chan ß
sendTicket airport = let ticket = grabTicket in send ticket airport

sendPassport :: [ß:!Passport.&<OK : ?BoardingPass.!Luggage.End, NOK: End >] Chan ß
                ->
                [ß:&<OK : ?BoardingPass.!Luggage.End, NOK: End >] Chan ß
sendPassport airport = let passport = grabPassport in send passport airport

getBoardingPass :: [ß:?BoardingPass.!Luggage.End] Chan ß
                   ->
                   [ß:!Luggage.End] Chan ß
getBoardingPass airport = return (receive airport)

sendLuggage :: [ß:!Luggage.End] Chan ß -> [ß:End] Chan ß
sendLuggage airport = let luggage = grabFromTrunk in send luggage airport ;

checkin :: [ß:Checkin] Chan ß -> [ß:/] ()
checkin airport = select NORMAL airport ;
                        sendTicket ; sendPassport ;
                  case airport
                    OK  = let boardingpass = getBoardingPass airport
                          in
                            sendLuggage airport ;
                            close airport
                    NOK = let error = receive airport
                          in
                            close airport ;
                            error (format “Go back home: %s” error)
```

Type information explodes

Improving modularity at the type level in session type programs

- Polymorphism for full session types
- Pattern Matching
- Session types in System F

Session Types in practice

A FUNCTION WORKING ON TYPES

Session Typed Function

```
checkin :: [ß:S] Chan ß -> [ß:S’] Bool
```

Syntax of prototype implementation

A SAMPLE PROGRAM

```
session Checkin = !Ticket.!Passport.&<OK : ?BoardingPass.!Luggage.End,
                                       NOK: End                       >

checkin :: [ß:Checkin] Chan ß -> [ß:End] Chan ß
checkin airport = let ticket   = grabTicket
                      passport = grabPassport
                      luggage  = grabFromTrunk
                  in
                    send ticket airport ;
                    send passport airport ;
                    case airport
                       OK  = let boardingpass = receive airport
                             in
                               send luggage airport ;
                               close airport
                       NOK = let err = receive airport
                             in
                               close airport ;
                               error (format “Go back home: %s” err)
```

A SAMPLE PROGRAM

```
session Checkin = !Ticket.!Passport.&<OK : ?BoardingPass.!Luggage.End,
                                         NOK: End                       >

checkin :: [ß:Checkin] Chan ß -> [ß:End] Chan ß
checkin airport = let ticket   = grabTicket
                      passport = grabPassport
                      luggage  = grabFromTrunk
                  in
                    send ticket airport ;
                    send passport airport ;
                    case airport
                       OK  = let boardingpass = receive airport
                             in
                               send luggage airport ;
                               close airport
                       NOK = let err = receive airport
                             in
                               close airport ;
                               error (format “Go back home: %s” err)
```

A SAMPLE PROGRAM

```
session Checkin = !Ticket.!Passport.&<OK : ?BoardingPass.!Luggage.End,
                                        NOK: End                        >

checkin :: [ß:Checkin] Chan ß -> [ß:End] Chan ß
checkin airport = let ticket   = grabTicket
                      passport = grabPassport
                      luggage  = grabFromTrunk
                  in
                    send ticket airport ;
                    send passport airport ;
                    case airport
                       OK  = let boardingpass = receive airport
                             in
                               send luggage airport ;
                               close airport
                       NOK = let err = receive airport
                             in
                               close airport ;
                               error (format “Go back home: %s” err)
```

=

```
sendTicket :: [ß:Checkin] Chan ß
              ->
              [ß:!Passport.&<OK : ?BoardingPass.!Luggage.End, NOK:
End >] Chan ß
sendTicket airport = let ticket = grabTicket in send ticket airport
```

Modularity is difficult

Types

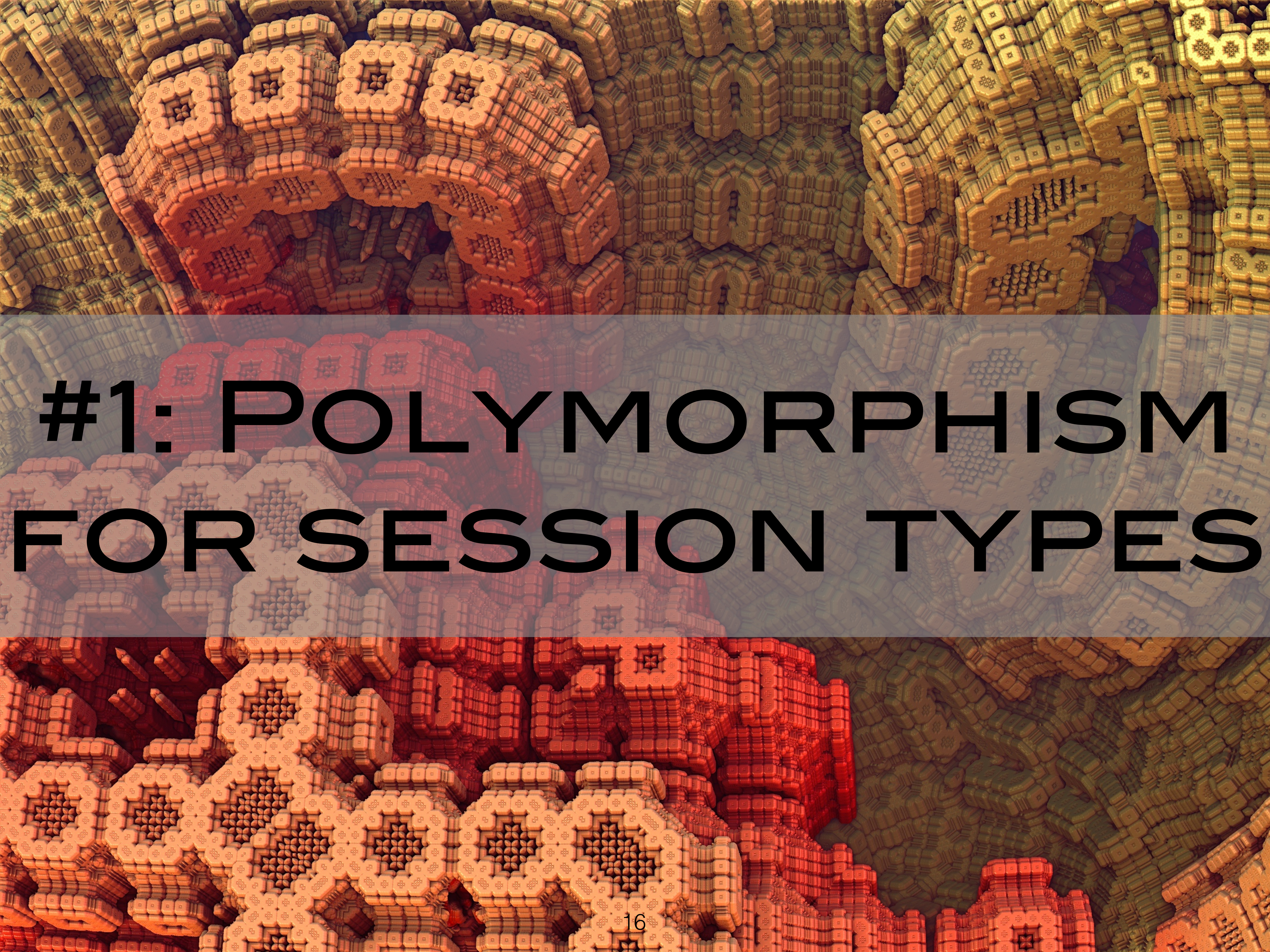
Logic

```
sestyp Checkin = ⊕<NORMAL: !Ticket.!Passport,
                    FAST  : !Barcode          >.&<OK : ?BoardingPass.!Luggage.End,
                                                 NOK: End                       >

sendTicket airport = let ticket = grabTicket in send ticket airport

sendPassport airport = let passport = grabPassport in send passport airport

getBoardingPass airport = return (receive airport)

sendLuggage airport = let luggage = grabFromTrunk in send luggage airport ;

checkin airport = select NORMAL airport ;
                        sendTicket ; sendPassport ;
                  case airport
                    OK  = let boardingpass = getBoardingPass airport
                          in
                            sendLuggage airport ;
                            close airport
                    NOK = let error = receive airport
                          in
                            close airport ;
                            error (format “Go back home: %s” error)
```

Refactor

```
sestyp Checkin = ⊕<NORMAL: !Ticket.!Passport,
                    FAST  : !Barcode          >.&<OK : ?BoardingPass.!Luggage.End,
                                                 NOK: End                       >

sendTicket :: [ß:Checkin] Chan ß
              ->
              [ß:!Passport.&<OK : ?BoardingPass.!Luggage.End, NOK: End >] Chan ß
sendTicket airport = let ticket = grabTicket in send ticket airport

sendPassport :: [ß:!Passport.&<OK : ?BoardingPass.!Luggage.End, NOK: End >] Chan ß
                ->
                [ß:&<OK : ?BoardingPass.!Luggage.End, NOK: End >] Chan ß
sendPassport airport = let passport = grabPassport in send passport airport

getBoardingPass :: [ß:?BoardingPass.!Luggage.End] Chan ß
                   ->
                   [ß:!Luggage.End] Chan ß
getBoardingPass airport = return (receive airport)

sendLuggage :: [ß:!Luggage.End] Chan ß -> [ß:End] Chan ß
sendLuggage airport = let luggage = grabFromTrunk in send luggage airport ;

checkin :: [ß:Checkin] Chan ß -> [ß:/] ()
checkin airport = select NORMAL airport ;
                        sendTicket ; sendPassport ;
                  case airport
                    OK  = let boardingpass = getBoardingPass airport
                          in
                            sendLuggage airport ;
                            close airport
                    NOK = let error = receive airport
                          in
                            close airport ;
                            error (format “Go back home: %s” error)
```

Type information explodes

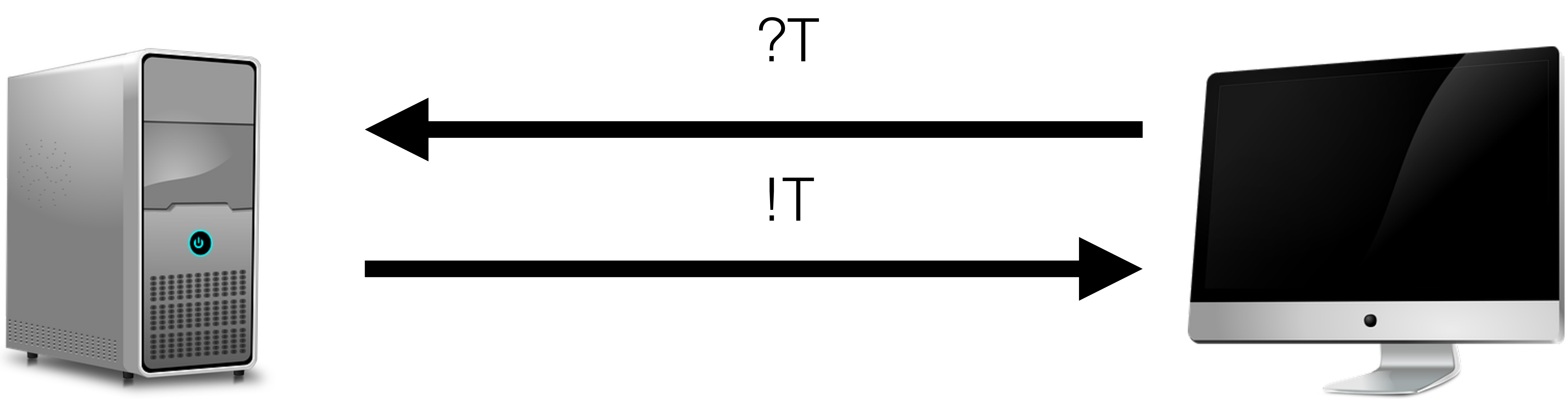


#1: Polymorphism for session types

Polymorphism

Receive and send back

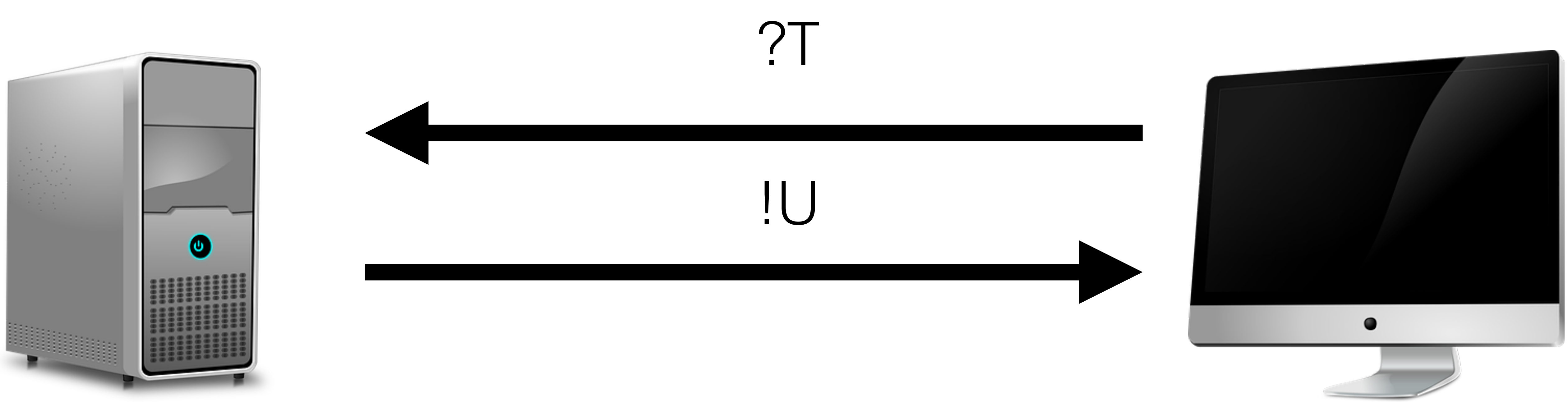
```
sendBack :: ForAll T . [ß: ?T.!T.End] Chan ß -> [ß:End]()
```



POLYMORPHISM

Apply f to the received value and send it back

```
sendBack :: ForAll T,U . [ß: ?T.!U.End] Chan ß -> (T -> U) -> [ß:End]()
sendBack chn f = let incoming = receive chn
                     outgoing = f incoming
                 in
                   send outgoing chn
```



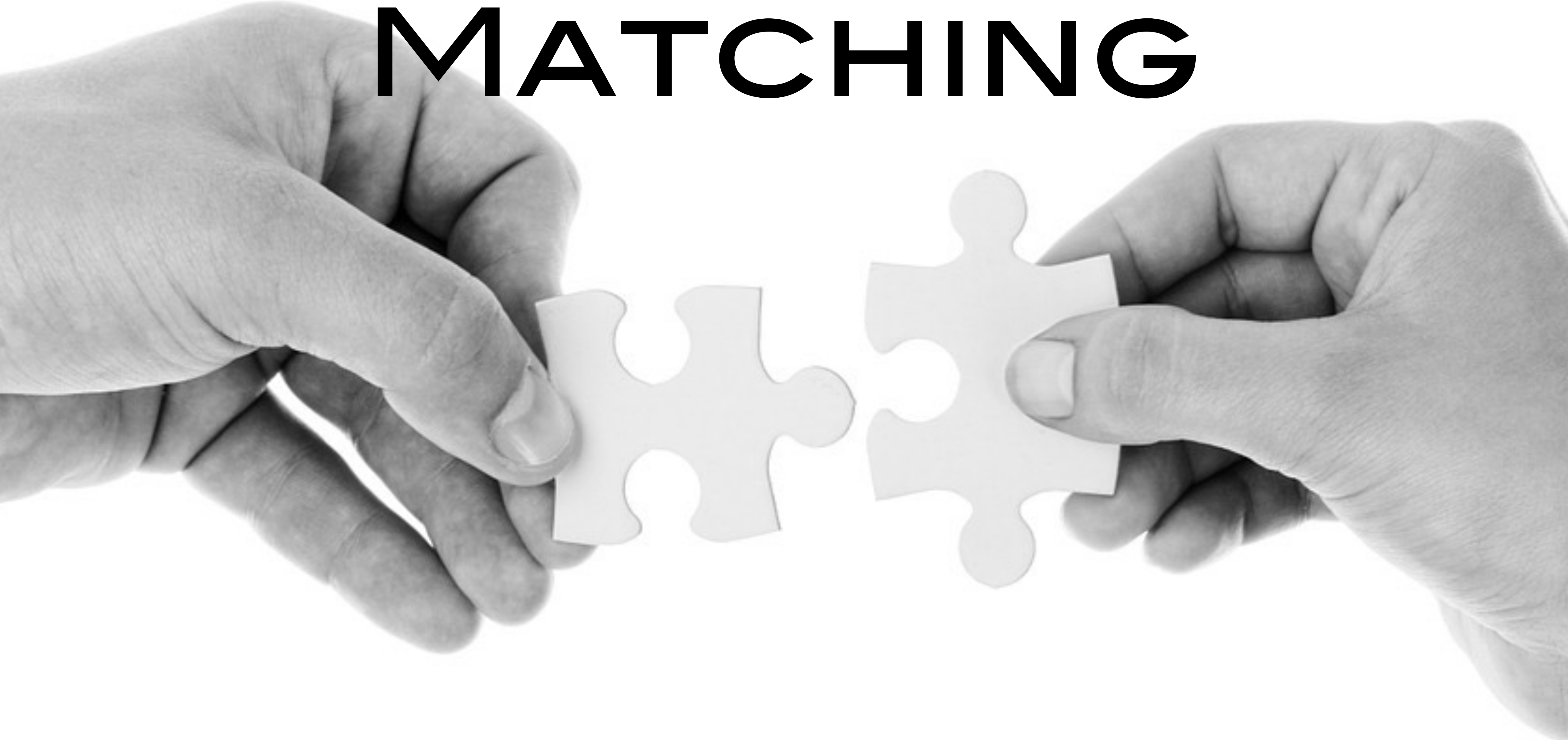
POLYMORPHISM

```
session Checkin = ⊕<NORMAL: !Ticket.!Passport,
                    FAST   : !Barcode          >.&<OK : ?BoardingPass.!Luggage.End,
                                                   NOK: End                          >

customsHandbag :: [ß:?HandBag.⊕<OK: !HandBag, ERR: !Reason>.End] Chan ß -> [ß:End] ()
customsHandbag conveyorbelt = let handbag = receive conveyorbelt
                              in
                                if (containsLiquids? handbag)
                                  then select NOK conveyorbelt
                                       send “Not allowed!”
                                  else send handbag conveyorbelt

customsLuggage :: [ß:?Luggage.⊕<OK: !Luggage, ERR: !Reason>.End] Chan ß -> [ß:End] ()
customsLuggage conveyorbelt = let luggage = receive conveyorbelt
                              in
                                if (containsDrugs? handbag)
                                  then select NOK conveyorbelt
                                       send “Not allowed!”
                                  else send handbag conveyorbelt
```

Polymorphism for Session Types

```
session Checkin = ⊕<NORMAL: !Ticket.!Passport,
                    FAST  : !Barcode          >.&<OK : ?BoardingPass.!Luggage.End,
                                                  NOK: End                      >

checkWith :: [ß:?T.⊕<OK: !T, ERR: !Reason>.End] Chan ß -> (T -> Bool) -> [ß:End] ()
checkWith conveyorbelt f = let t = receive conveyorbelt
                           in
                             if (f t)
                               then select ERR conveyorbelt
                                    send “Not allowed!”
                                    else send t conveyorbelt

customsHandbag :: [ß:?HandBag.⊕<OK: !HandBag.End, ERR: !Reason>.End] Chan ß -> [ß:End] ()
customsHandbag conveyorbelt = checkWith conveyorbelt containsLiquids?

customsLuggage :: [ß:?Luggage.⊕<OK: !Luggage, ERR: !Reason>.End] Chan ß -> [ß:End] ()
customsLuggage conveyorbelt = checkWith conveyorbelt containsDrugs?
```

#2: Pattern Matching



Enforce Structural constraints

“As long as the session type wants me to send an Int”

```
f :: [ß: S.!Int.U] Chan ß -> [ß:U] ()
```

“As long as the channel offers these two choices”

```
h :: [ß: &<CH1: S, CH2: T>.U] Chan ß -> [ß:U] ()
```

A sample program

```
sendTicket :: [ß:Checkin] Chan ß
              ->
              [ß:!Passport.&<OK : ?BoardingPass.!Luggage.End, NOK: End >] Chan ß
sendTicket airport = let ticket = grabTicket in send ticket airport

sendPassport :: [ß:!Passport.&<OK : ?BoardingPass.!Luggage.End, NOK: End >] Chan ß
                ->
                [ß:&<OK : ?BoardingPass.!Luggage.End, NOK: End >] Chan ß
sendPassport airport = let passport = grabPassport in send passport airport

getBoardingPass :: [ß:?BoardingPass.!Luggage.End] Chan ß
                   ->
                   [ß:!Luggage.End] Chan ß
getBoardingPass airport = return (receive airport)
```

A BETTER SAMPLE PROGRAM

```
[ß:⊕<FAST: S, NORMAL: S’>.S”]
```

```
selectFastTrack :: [ß:⊕<FAST: S, NORMAL: S’>.S”] Chan ß -> [ß:S.S”] Chan ß
selectFastTrack airport = select FAST airport

sendTicket :: [ß:!Ticket.S] Chan ß -> [ß:S] Chan ß
sendTicket airport = let ticket = grabTicket in send ticket airport

sendPassport :: [ß:!Passport.S] Chan ß -> [ß:S] Chan ß
sendPassport airport = let passport = grabPassport in send passport airport

getBoardingPass :: [ß:?BoardingPass.S] Chan ß -> [ß:S] BoardingPass
getBoardingPass airport = return (receive airport)
```

```
[ß:?BoardingPass.S]
```

Thank you!

- Help session types transform with program
- Reduce type overhead
- Proof of Concept implementation
    - Polymorphism for full session types
    - Pattern matching (Structural Constraints)
    - System F with session types
- All images taken from pixabay.com