Capable GV

Capabilities for Session Types in GV

Magdalena J. Latifa and **Ornela Dardha**

TYPES 2022

Oilthigh Ghlaschu

VIA VERITAS VITA

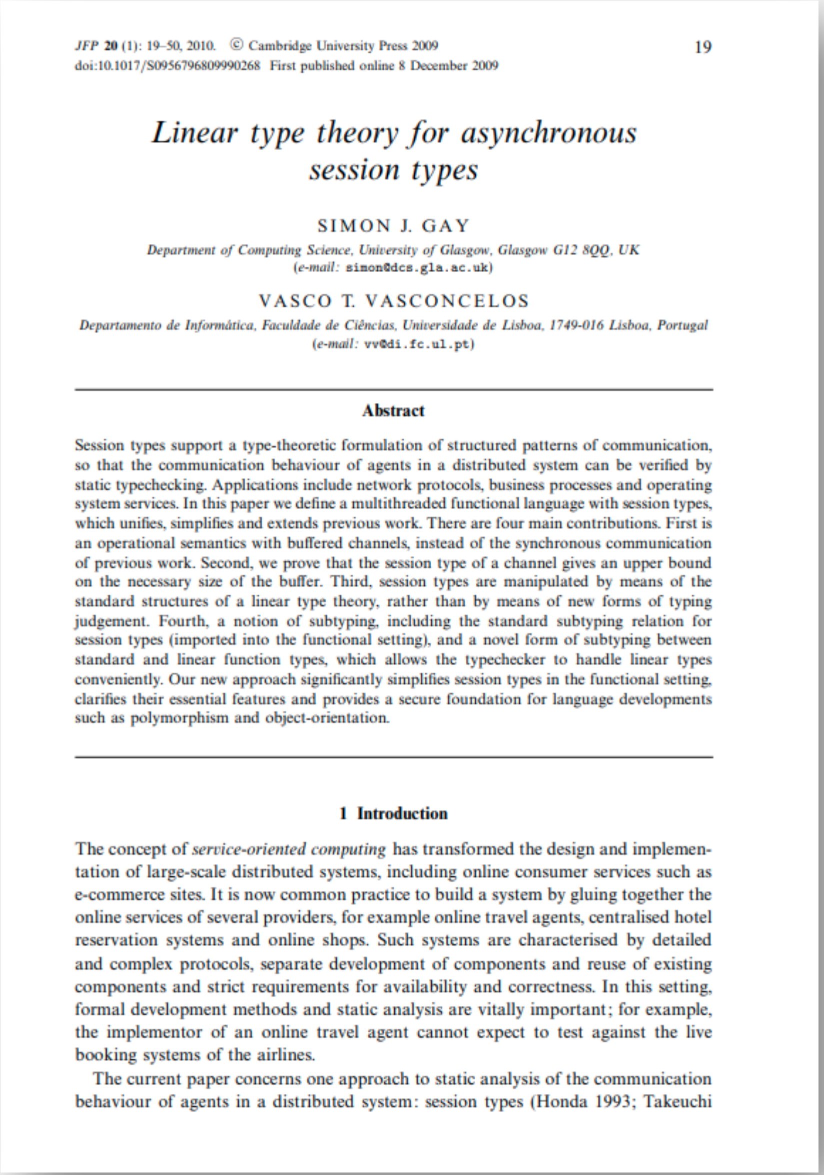
CoLab
Concurrency Lab

Session types

- Communication formalism
- Guarantees: communication safety, session fidelity and privacy
- Rely on ***linearity***

GV

- Functional calculus with session types
- Correspondence to classical linear logic
- Programs are well behaved
- Practical suitability
- Many extensions have been developed


JFP **20** (1): 19–50, 2010. © Cambridge University Press 2009 19
doi:10.1017/S0956796809990268 First published online 8 December 2009

Linear type theory for asynchronous session types

SIMON J. GAY
Department of Computing Science, University of Glasgow, Glasgow G12 8QQ, UK
(*e-mail:* `simon@dcs.gla.ac.uk`)

VASCO T. VASCONCELOS
Departamento de Informática, Faculdade de Ciências, Universidade de Lisboa, 1749-016 Lisboa, Portugal
(*e-mail:* `vv@di.fc.ul.pt`)

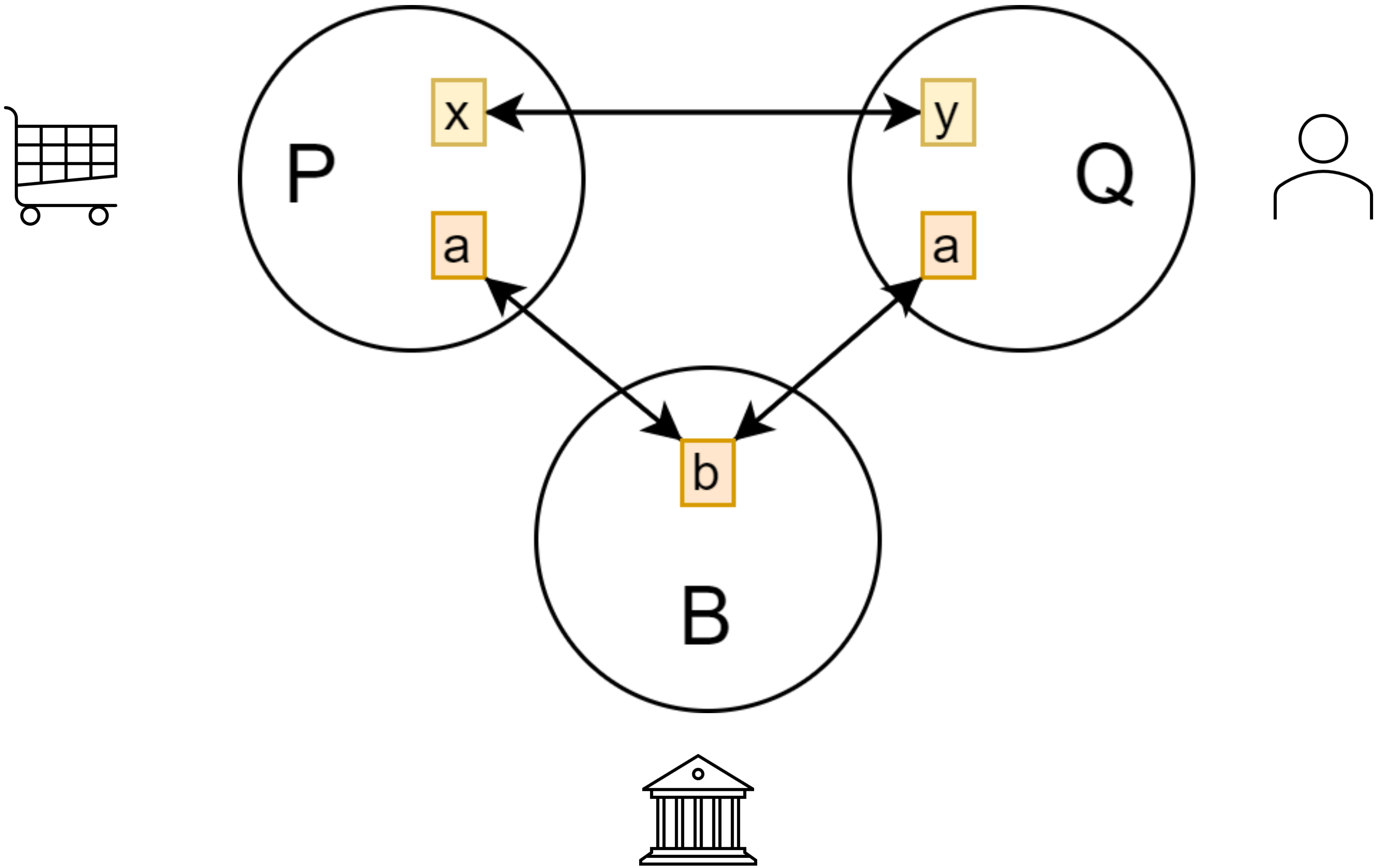
Abstract

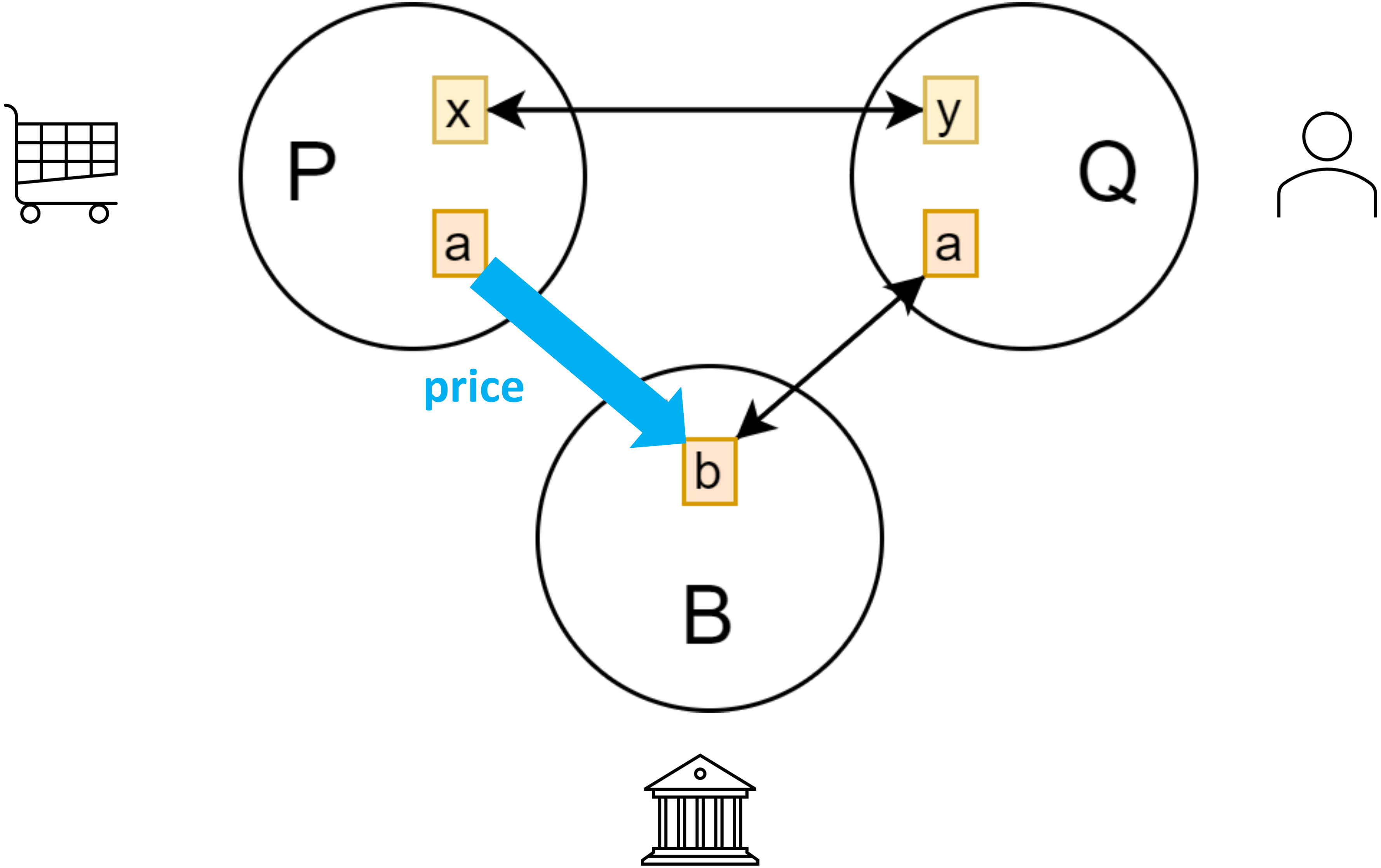
Session types support a type-theoretic formulation of structured patterns of communication, so that the communication behaviour of agents in a distributed system can be verified by static typechecking. Applications include network protocols, business processes and operating system services. In this paper we define a multithreaded functional language with session types, which unifies, simplifies and extends previous work. There are four main contributions. First is an operational semantics with buffered channels, instead of the synchronous communication of previous work. Second, we prove that the session type of a channel gives an upper bound on the necessary size of the buffer. Third, session types are manipulated by means of the standard structures of a linear type theory, rather than by means of new forms of typing judgement. Fourth, a notion of subtyping, including the standard subtyping relation for session types (imported into the functional setting), and a novel form of subtyping between standard and linear function types, which allows the typechecker to handle linear types conveniently. Our new approach significantly simplifies session types in the functional setting, clarifies their essential features and provides a secure foundation for language developments such as polymorphism and object-orientation.

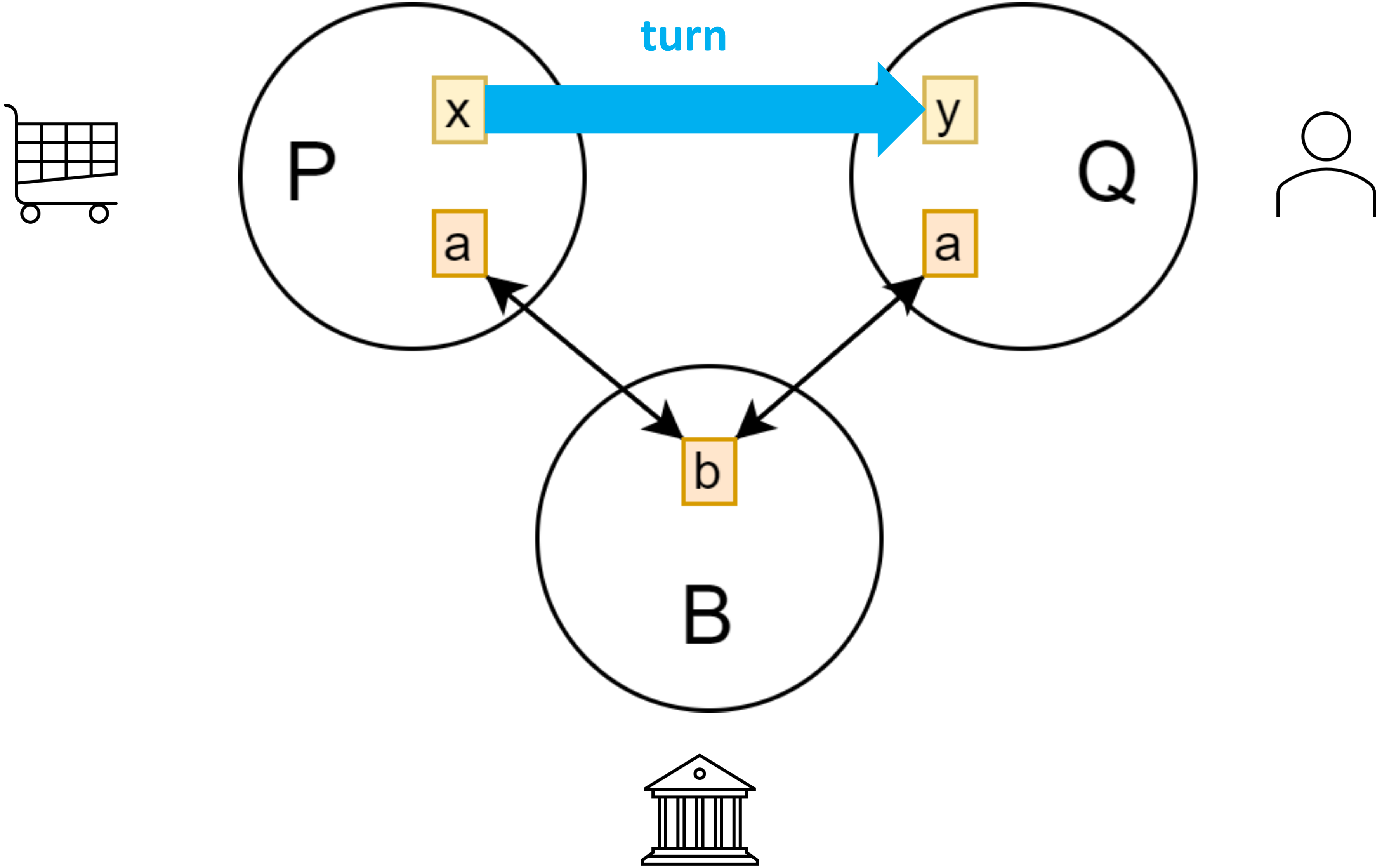
1 Introduction

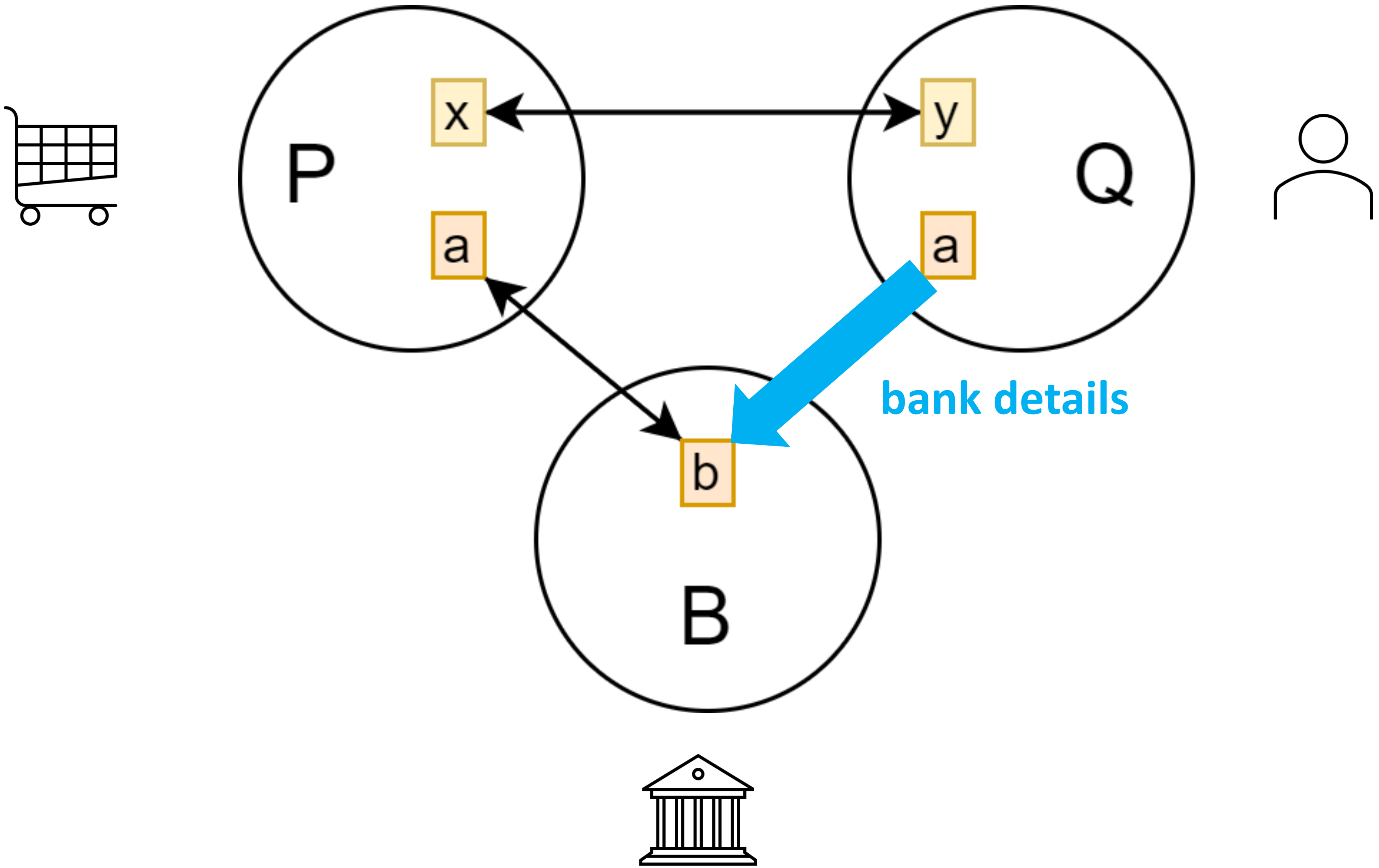
The concept of *service-oriented computing* has transformed the design and implementation of large-scale distributed systems, including online consumer services such as e-commerce sites. It is now common practice to build a system by gluing together the online services of several providers, for example online travel agents, centralised hotel reservation systems and online shops. Such systems are characterised by detailed and complex protocols, separate development of components and reuse of existing components and strict requirements for availability and correctness. In this setting, formal development methods and static analysis are vitally important; for example, the implementor of an online travel agent cannot expect to test against the live booking systems of the airlines.

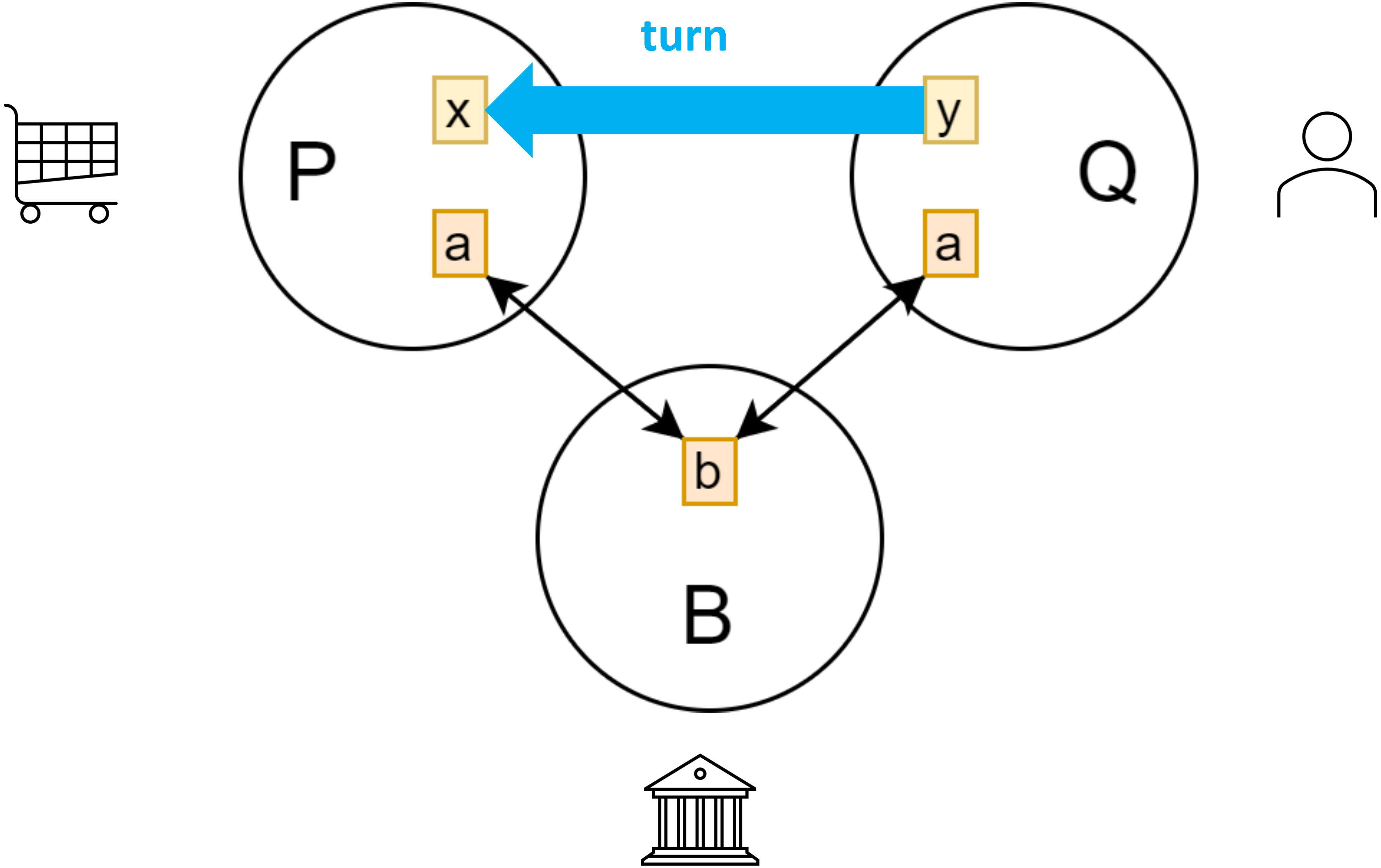
The current paper concerns one approach to static analysis of the communication behaviour of agents in a distributed system: session types (Honda 1993; Takeuchi

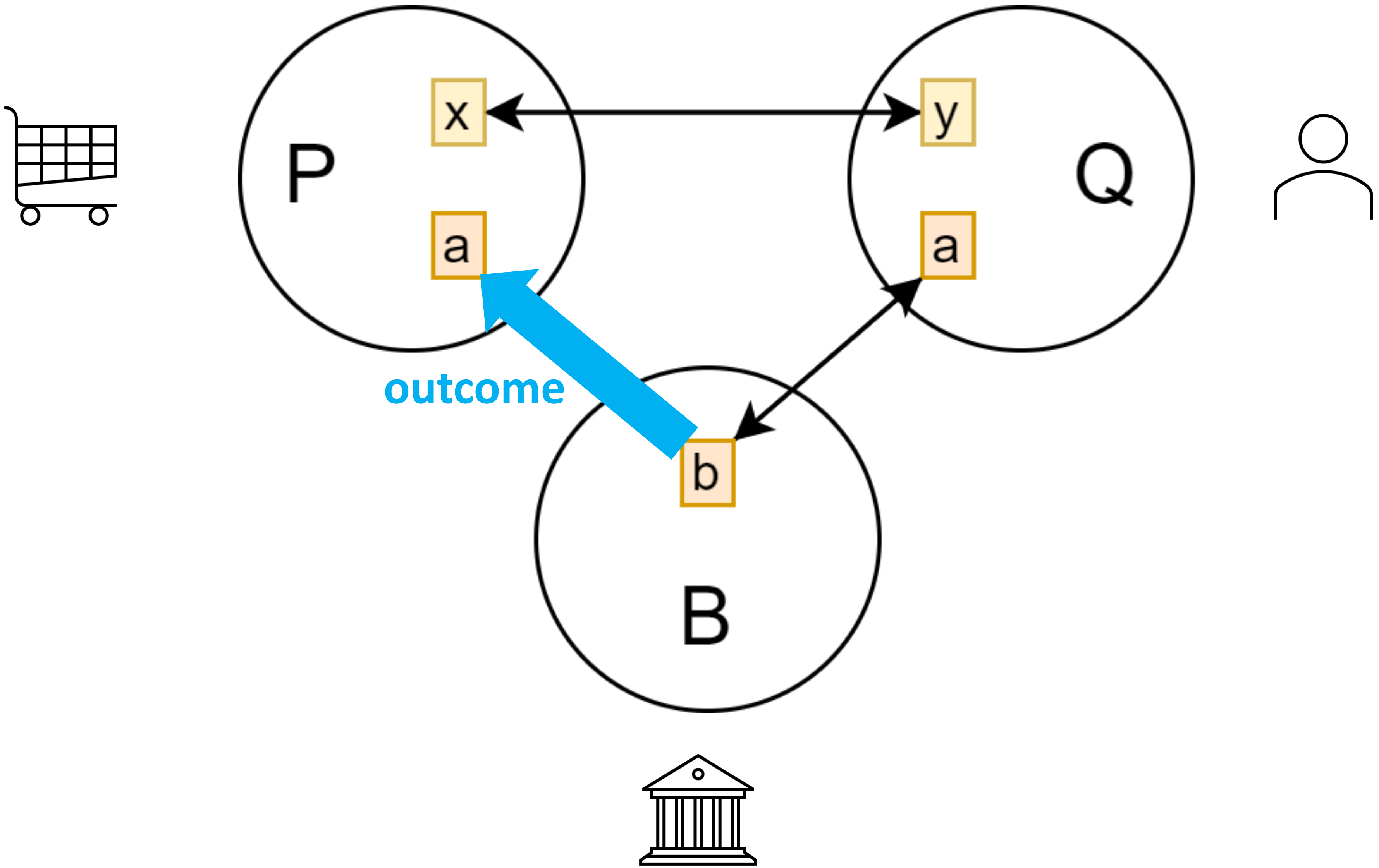

x
y
P
Q
a
a
b
B


P
x
a
y
Q
a
price
b
B


turn
x
y
P
Q
a
a
b
B


P
x
a
y
Q
a
bank details
b
B


turn
x
y
P
Q
a
a
b
B



x
y
P
Q
a
a
outcome
b
B

PQB

- Not currently possible in any of the current GVs
- Can be implemented via simply passing the channel endpoint but then neither P nor Q have the guarantee they are communicating directly with B
- For example, P could pretend to send over channel a and instead send over a new channel and snoop on Q's details

Channel sharing

- The solution to the PQB problem
- Channel endpoints can be accessed by multiple processes
- Need to enforce linearity for communication safety…

Capabilities

- **Split** channels into *channel endpoints* and their *capabilities*

- Allows the *endpoints to be unrestricted* by keeping *capabilities linear*

- Technique originated in region-based memory management

Resource Sharing via Capability-Based Multiparty Session Types *

A. Laura Voinea[0000-0003-4482-205X], Ornela Dardha[0000-0001-9927-7875], and Simon J. Gay[0000-0003-3033-9091]

School of Computing Science, University of Glasgow, United Kingdom
a.voinea.1@research.gla.ac.uk
{Ornela.Dardha, Simon.Gay}@glasgow.ac.uk

Abstract. *Multiparty Session Types (MPST)* are a type formalism used to model communication protocols among components in distributed systems, by specifying *type* and *direction* of data transmitted. It is standard for multiparty session type systems to use access control based on *linear* or *affine* types. While useful in offering strong guarantees of communication safety and session fidelity, linearity and affinity run into the well-known problem of inflexible programming, excluding scenarios that make use of shared channels or need to store channels in shared data structures.

In this paper, we develop *capability-based resource sharing* for multiparty session types. In this setting, channels are split into two entities, the channel itself and the capability of using it. This gives rise to a more flexible session type system, which allows channel references to be shared and stored in persistent data structures. We illustrate our type system through a producer-consumer case study. Finally, we prove that the resulting language satisfies type safety.

Keywords: session types · sharing · concurrent programming

1 Introduction

In the present era of communication-centric software systems, it is increasingly recognised that the structure of communication is an essential aspect of system design. *(Multiparty) session types* [18,19,31] allow communication structures to be codified as type definitions in programming languages, which can be exploited by compilers, development environments and runtime systems, for compile-time analysis or runtime monitoring. A substantial and ever-growing literature on session types and, more generally, behavioural types [20] provides a rich theoretical foundation, now being applied to a range of programming languages [1,16].

* Supported by the UK EPSRC grant EP/K034413/1, "From Data Types to Session Types: A Basis for Concurrency and Distribution (ABCD)", by the EU HORIZON 2020 MSCA RISE project 778233 "BehAPI: Behavioural Application Program Interfaces", and by an EPSRC PhD studentship.

Capable GV

- Incorporates capabilities into the linear setting of GV
- Based predominantly on constructs from PGV rather than other GV extension to take advantage of cyclic structure
- Static elements: terms, types, typing rules
- The rest in progress…

CGV types

$$T, U ::= T \times U \mid T + U \mid tr\,(\rho) \mid [\rho\,(S)] \mid 1 \mid T\,(C) \multimap (C')\,U$$

$$\Gamma ::= \varnothing \mid \Gamma, x : T$$

$$\Delta ::= \varnothing \mid \Delta, \rho$$

$$C ::= \varnothing \mid C \otimes \rho\,(S)$$

CGV types

$$T, U ::= T \times U \mid T + U \mid tr\,(\rho) \mid [\rho\,(S)] \mid 1 \mid T\,(C) \multimap (C')\,U$$

$$\Gamma ::= \varnothing \mid \Gamma, x : T$$

$$\Delta ::= \varnothing \mid \Delta, \rho$$

$$C ::= \varnothing \mid C \otimes \rho\,(S)$$

CGV types

$$T, U ::= T \times U \mid T + U \mid tr(\rho) \mid [\rho\,(S)] \mid 1 \mid T\,(C) \multimap (C')\,U$$

$$\Gamma ::= \varnothing \mid \Gamma, x : T$$

$$\Delta ::= \varnothing \mid \Delta, \rho$$

$$C ::= \varnothing \mid C \otimes \rho\,(S)$$

Typing judgements

- $\Gamma;\ \Delta;\ \vdash\ V : T$
  under typing environment Γ and capability environment Δ,
  term V is of type T

- $\Gamma;\ \Delta;\ C \vdash\ M : T \rhd\ C'$
  under typing environment Γ and capability environment Δ and with
  capability set C, term M is of type T and produces capability set C'

Typing judgements

- $\Gamma;\ \Delta;\ \vdash\ V : T$

  under typing environment Γ and capability environment Δ,

  term V is of type T

- $\Gamma;\ \Delta;\ C\ \vdash\ M : T\ \rhd\ C'$

  under typing environment Γ and capability environment Δ and with

  capability set C, term M is of type T and produces capability set C'

CGV terms

$$V, W ::= () \mid x \mid \lambda x.M \mid (V, W) \mid \text{inl } V \mid \text{inr } V$$

$$\begin{aligned} L, M, N ::= \; & V\,W \mid \text{return } V \mid \text{let } x = M \text{ in } N \\ \mid \; & \text{let } (x, y) = V \text{ in } M \mid \text{let } () = V \text{ in } M \\ \mid \; & \text{case } L \; \{\text{inl } x \mapsto M; \text{inr } y \mapsto N\} \\ \mid \; & \text{new} \mid \text{send } V \mid \text{recv } V \mid \text{close } V \\ \mid \; & \text{inact } V \mid \text{act } V \mid \text{spawn } M \end{aligned}$$

CGV terms

V, W ::= () | x | $\lambda x\,.M$ | (V, W) | inl V | inr V

L, M, N ::= $V\,W$ | return V | let x = M in N

| let (x, y) = V in M | let () = V in M

| case L {inl $x \mapsto M$; inr $y \mapsto N$ }

| new | send V | recv V | close V

| inact V | act V | spawn M

$$\textsc{T-Recv}\quad \frac{\Gamma; \Delta \vdash V : tr(\rho)}{\Gamma; \Delta; C \otimes \rho(?T.S) \vdash \mathrm{recv}\ V : T \triangleright C \otimes \rho(S)}$$

$$\textsc{T-Send}\quad \frac{\Gamma; \Delta \vdash V : T \times tr(\rho)}{\Gamma; \Delta; C \otimes \rho(!T.S) \vdash \mathrm{send}\ V : 1 \triangleright C \otimes \rho(S)}$$

T-Recv

$$\frac{\Gamma; \Delta \vdash \boxed{V : tr(\rho)}}{\Gamma; \Delta; C \otimes \boxed{\rho(?T.S)} \vdash \mathrm{recv}\ V : \boxed{T} \triangleright C \otimes \boxed{\rho(S)}}$$

T-Send

$$\frac{\Gamma; \Delta \vdash V : T \times tr(\rho)}{\Gamma; \Delta; C \otimes \rho(!T.S) \vdash \mathrm{send}\ V : 1 \triangleright C \otimes \rho(S)}$$

T-Recv

$$\frac{\Gamma; \Delta \vdash V : tr(\rho)}{\Gamma; \Delta; C \otimes \rho(?T.S) \vdash \mathrm{recv}\ V : T \triangleright C \otimes \rho(S)}$$

T-Send

$$\frac{\Gamma; \Delta \vdash V : T \times tr(\rho)}{\Gamma; \Delta; C \otimes \rho(!T.S) \vdash \mathrm{send}\ V : 1 \triangleright C \otimes \rho(S)}$$

CGV terms

V, W ::= $()$ | x | $\lambda x.M$ | (V, W) | inl V | inr V

L, M, N ::= $V\,W$ | return V | let x = M in N

| let (x, y) = V in M | let $()$ = V in M

| case L {inl $x \mapsto M$; inr $y \mapsto N$ }

| new | send V | recv V | close V

| inact V | act V | spawn M

T-Inact

$$\frac{\Gamma; \Delta \vdash V : tr(\rho)}{\Gamma; \Delta; C \otimes \rho(S) \vdash \text{inact } V : [\rho(S)] \triangleright C}$$

T-Act

$$\frac{\Gamma; \Delta \vdash V : [\rho(S)]}{\Gamma; \Delta; C \vdash \text{act } V : 1 \triangleright C \otimes \rho(S)}$$

T-INACT

$$\frac{\Gamma; \Delta \vdash V : tr(\rho)}{\Gamma; \Delta; C \otimes \rho(S) \vdash \text{inact } V : [\rho(S)] \triangleright C}$$

T-ACT

$$\frac{\Gamma; \Delta \vdash V : [\rho(S)]}{\Gamma; \Delta; C \vdash \text{act } V : 1 \triangleright C \otimes \rho(S)}$$

T-Inact

$$\frac{\Gamma; \Delta \vdash V : tr(\rho)}{\Gamma; \Delta; C \otimes \rho(S) \vdash \text{inact}\ V : [\rho(S)] \triangleright C}$$

T-Act

$$\frac{\Gamma; \Delta \vdash V : \boxed{[\rho(S)]}}{\Gamma; \Delta; \boxed{C} \vdash \text{act}\ V : 1 \triangleright \boxed{C \otimes \rho(S)}}$$

CGV terms

$$V, W ::= () \mid x \mid \lambda x.M \mid (V, W) \mid \text{inl } V \mid \text{inr } V$$

$$L, M, N ::= V\,W \mid \text{return } V \mid \text{let } x = M \text{ in } N$$
$$\mid \text{let } (x, y) = V \text{ in } M \mid \text{let } () = V \text{ in } M$$
$$\mid \text{case } L \,\{\text{inl } x \mapsto \rightarrow M; \text{inr } y \mapsto \rightarrow N\}$$
$$\mid \text{new} \mid \text{send } V \mid \text{recv } V \mid \text{close } V$$
$$\mid \text{inact } V \mid \text{act } V \mid \text{spawn } M$$

T-LetBind

$$\frac{\Gamma_1; \Delta_1; C \vdash M : T \triangleright C' \qquad \Gamma_2, x : T; \Delta_2; C' \vdash N : U \triangleright C''}{\Gamma_1 \circ \Gamma_2; \Delta_1, \Delta_2; C \vdash \mathsf{let}\ x = M\ \mathsf{in}\ N : U \triangleright C''}$$

T-Spawn

$$\frac{\Gamma; \Delta; C_s \vdash M : 1 \triangleright \varnothing}{\Gamma; \Delta; C \otimes C_s \vdash \mathsf{spawn}\ M : 1 \triangleright C}$$

T-LetBind

$$\frac{\Gamma_1; \Delta_1; \boxed{C} \vdash M : T \triangleright \boxed{C'} \qquad \Gamma_2, x : T; \Delta_2; \boxed{C'} \vdash N : U \triangleright \boxed{C''}}{\Gamma_1 \circ \Gamma_2; \Delta_1, \Delta_2; \boxed{C} \vdash \mathrm{let}\ x = M \ \mathrm{in}\ N : U \triangleright \boxed{C''}}$$

T-Spawn

$$\frac{\Gamma; \Delta; C_s \vdash M : 1 \triangleright \varnothing}{\Gamma; \Delta; C \otimes C_s \vdash \mathrm{spawn}\ M : 1 \triangleright C}$$

T-LetBind

$$\frac{\Gamma_1; \Delta_1; C \vdash M : T \triangleright C' \qquad \Gamma_2, x : T; \Delta_2; C' \vdash N : U \triangleright C''}{\Gamma_1 \circ \Gamma_2; \Delta_1, \Delta_2; C \vdash \mathrm{let}\ x = M\ \mathrm{in}\ N : U \triangleright C''}$$

T-Spawn

$$\frac{\Gamma; \Delta; \boxed{C_s} \vdash M : 1 \triangleright \varnothing}{\Gamma; \Delta; \boxed{C \otimes C_s} \vdash \mathrm{spawn}\ M : 1 \triangleright \boxed{C}}$$

Conclusion

- CGV allows channel sharing - PQB is typeable
- Linearity of communication is enforced via type-end-effect system
- Operational semantics in the works
- Expecting to preserve subject reduction but introduce deadlocks
- Can restore deadlock-freedom e.g. via priorities

Thank you!