Session-Typed Concurrent Programming
Lecture 1

Stephanie Balzer
Carnegie Mellon University

OPLSS 2021
June 23, 2021

About this class

About this class

Session-type concurrent programming

- concurrency (as opposed to parallelism)
- nondeterminism

About this class

Session-type concurrent programming

- concurrency (as opposed to parallelism)
- nondeterminism

Roadmap

- message-passing concurrent programming
- session types as types for message-passing concurrency
- linear logic and session types
- manifest sharing (controlled form of aliasing)
- deadlock-freedom

Terminology that will be meaningful to you

Terminology that will be meaningful to you

progress

session type

contraction

weakening

intuitionism

linear logic

higher-order channels

affine

aliasing

Curry-Howard correspondence

identity

preservation

cut

deadlock-freedom

session fidelity

pi-calculus

sequent calculus

Learning objectives

Learning objectives

- How to program in a message-passing, concurrent style

Learning objectives

- How to program in a message-passing, concurrent style
- What session types are about

Learning objectives

- How to program in a message-passing, concurrent style
- What session types are about
- Benefits of linear logic for programming

Learning objectives

- How to program in a message-passing, concurrent style
- What session types are about
- Benefits of linear logic for programming
- How to accommodate sharing in a logically motivated way

Learning objectives

- How to program in a message-passing, concurrent style
- What session types are about
- Benefits of linear logic for programming
- How to accommodate sharing in a logically motivated way
- How to reason about deadlocks in the presence of aliasing

Hands-on session

Hands-on session

Tutorial by Soares Chen (Ruo Fei)

- Friday (6/25) and Saturday (6/26) from 12:20 pm - 1:50 pm

Hands-on session

Tutorial by Soares Chen (Ruo Fei)

- Friday (6/25) and Saturday (6/26) from 12:20 pm - 1:50 pm

Ferrite session type library in Rust

- writing session-typed programs in Rust
- support of linear and shared session types

Hands-on session

Tutorial by Soares Chen (Ruo Fei)

- Friday (6/25) and Saturday (6/26) from 12:20 pm - 1:50 pm

Ferrite session type library in Rust

- writing session-typed programs in Rust
- support of linear and shared session types

What you'll learn

- techniques used for session types embedding
- how to use the library
- practice with prepared exercises

Message-passing concurrent programming

Message-passing programming model

Message-passing programming model

Computation by a processes that exchange messages along channels

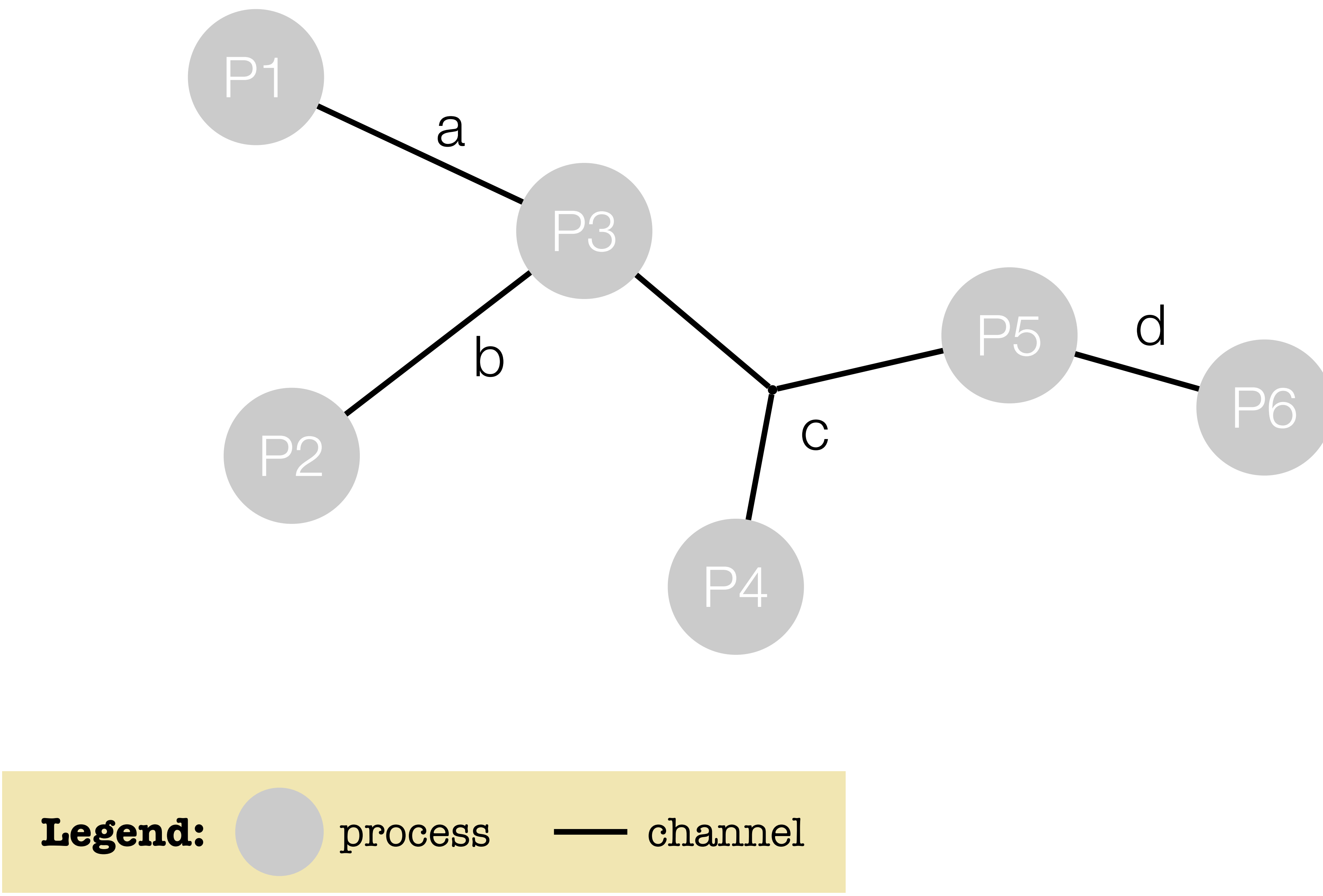
Message-passing programming model

Computation by a processes that exchange messages along channels

Legend: process

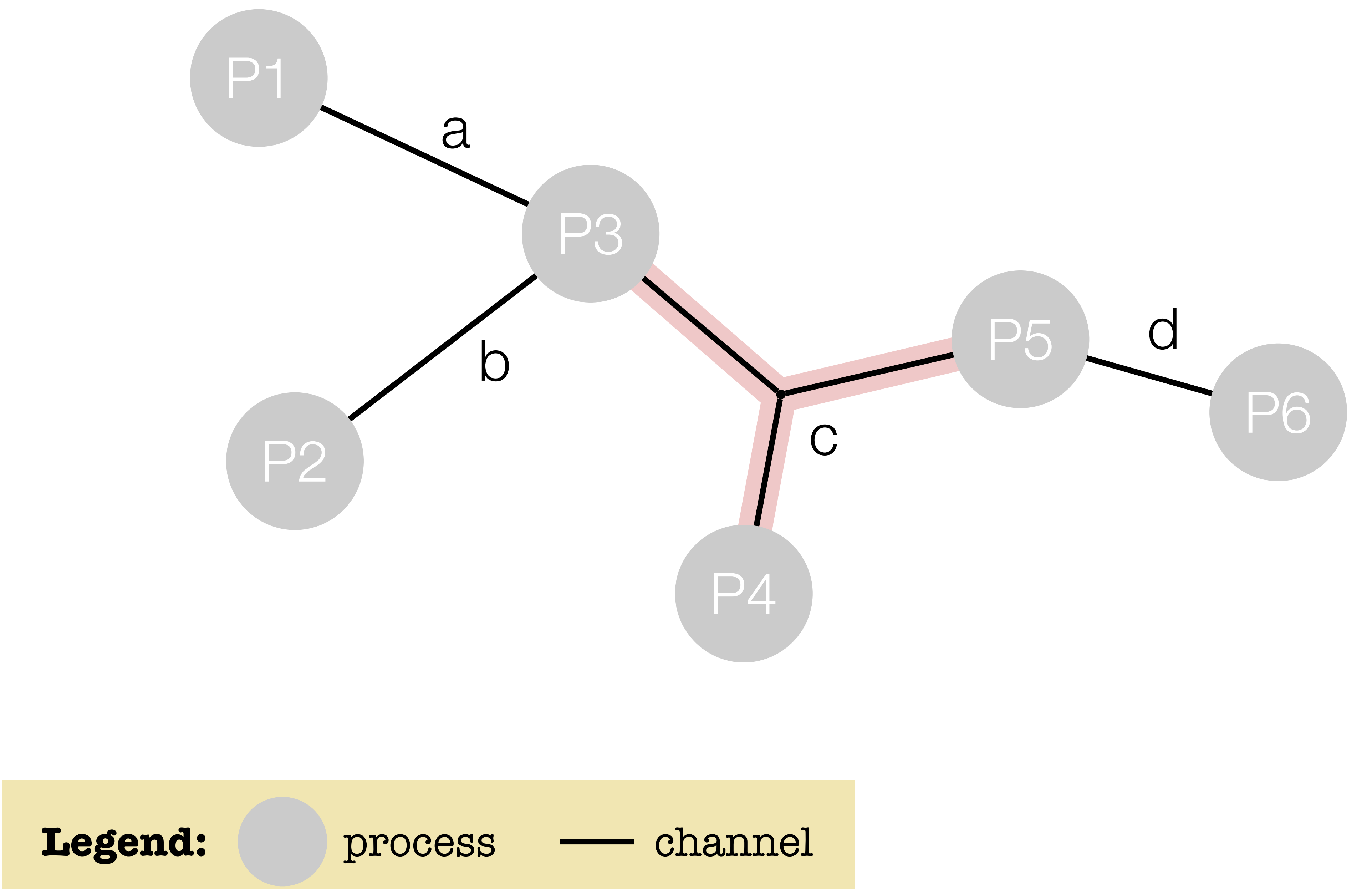
Message-passing programming model

Computation by a processes that exchange messages along channels



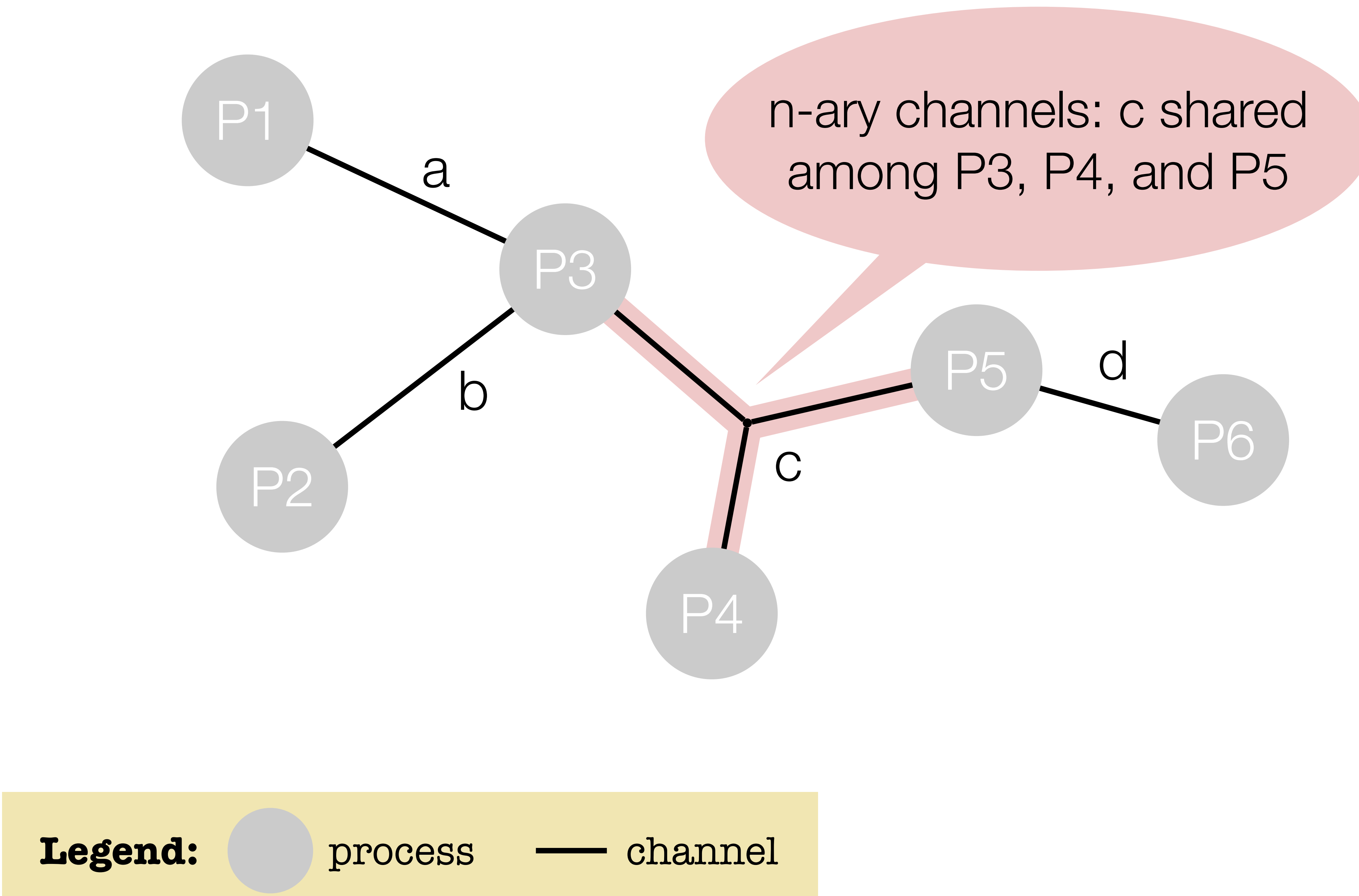
Message-passing programming model

Computation by a processes that exchange messages along channels



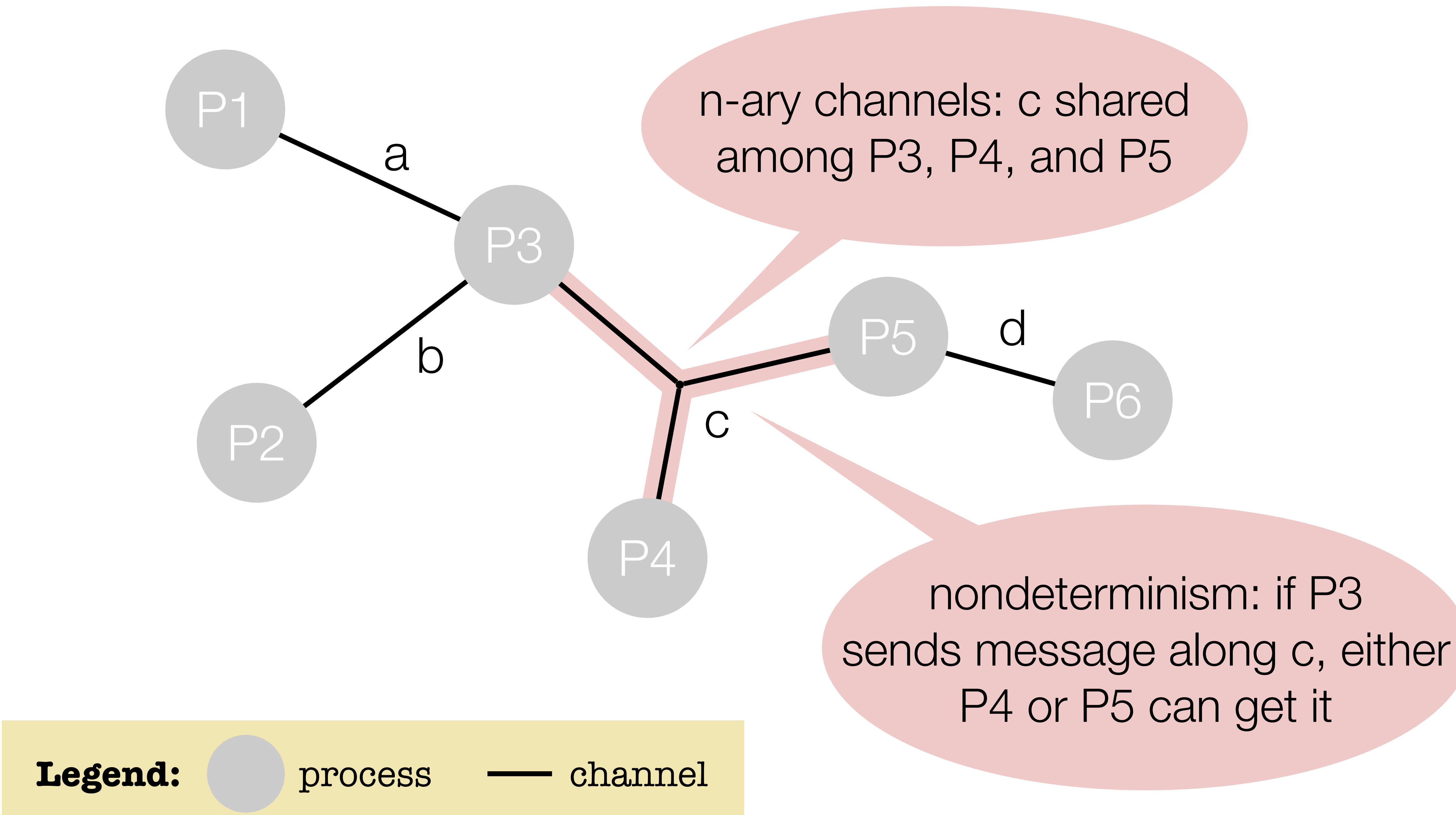
Message-passing programming model

Computation by a processes that exchange messages along channels



Message-passing programming model

Computation by a processes that exchange messages along channels



Message-passing programming model

Computation by a processes that exchange messages along channels

Message-passing programming model

Computation by a processes that exchange messages along channels

→ **underlying formal model: process calculus (e.g., pi-calculus)**

Robin Milner. Communicating and mobile systems: the pi-calculus. Cambridge University Press, 1999.

Message-passing programming model

Computation by a processes that exchange messages along channels

- underlying formal model: process calculus (e.g., pi-calculus)
  - universality: encoding of lambda-calculus into pi-calculus

Robin Milner. Communicating and mobile systems: the pi-calculus. Cambridge University Press, 1999.

Robin Milner. Functions as processes. Mathematical Structures in Computer Science, 1992.

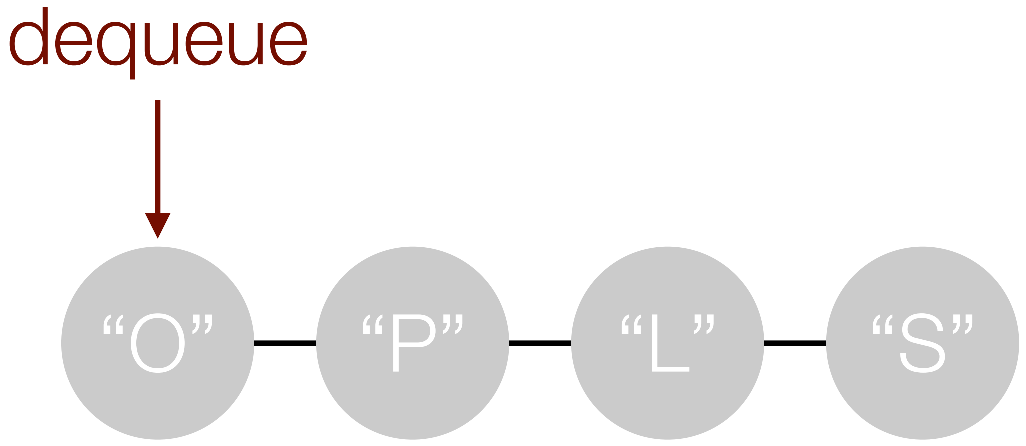
A message-passing queue

A message-passing queue

Queue of character processes:

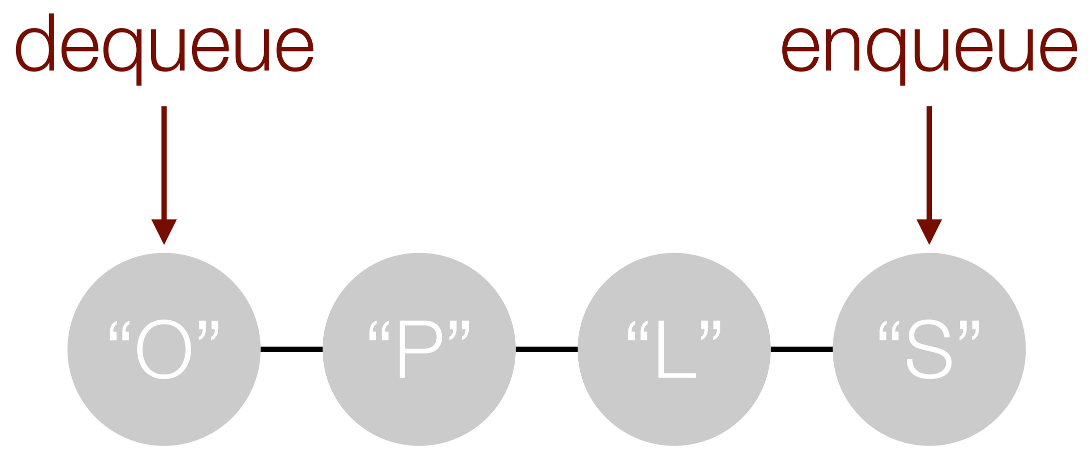
A message-passing queue

Queue of character processes:



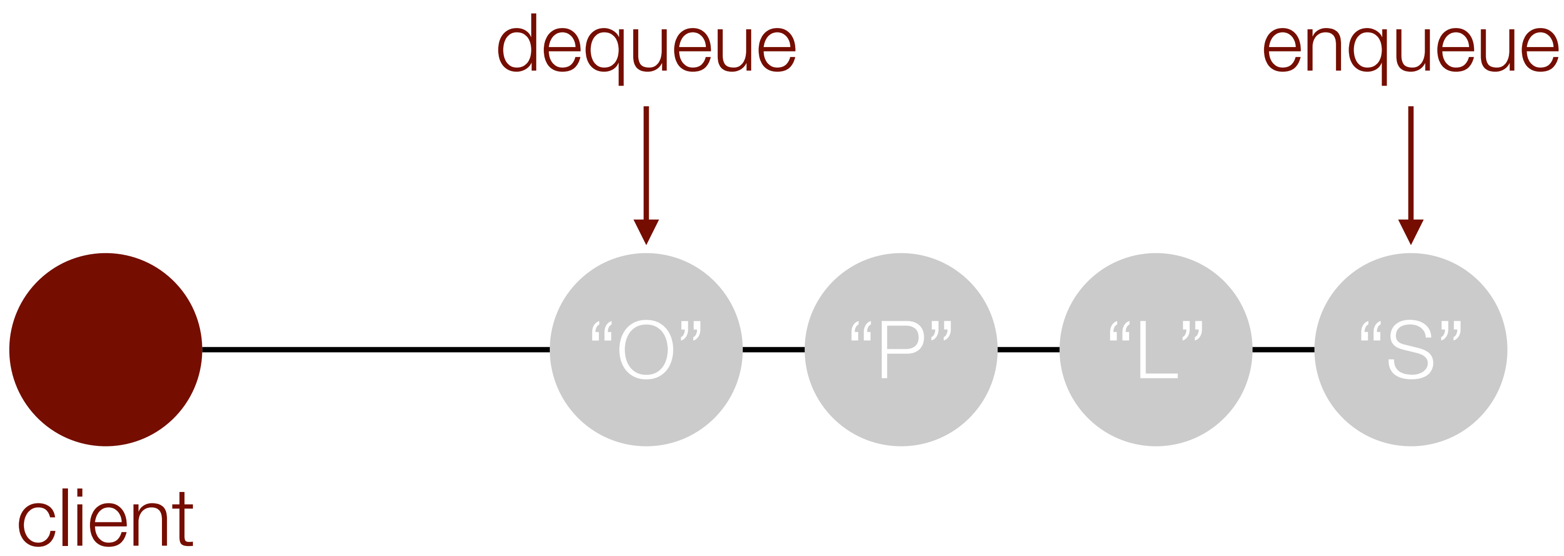
A message-passing queue

Queue of character processes:



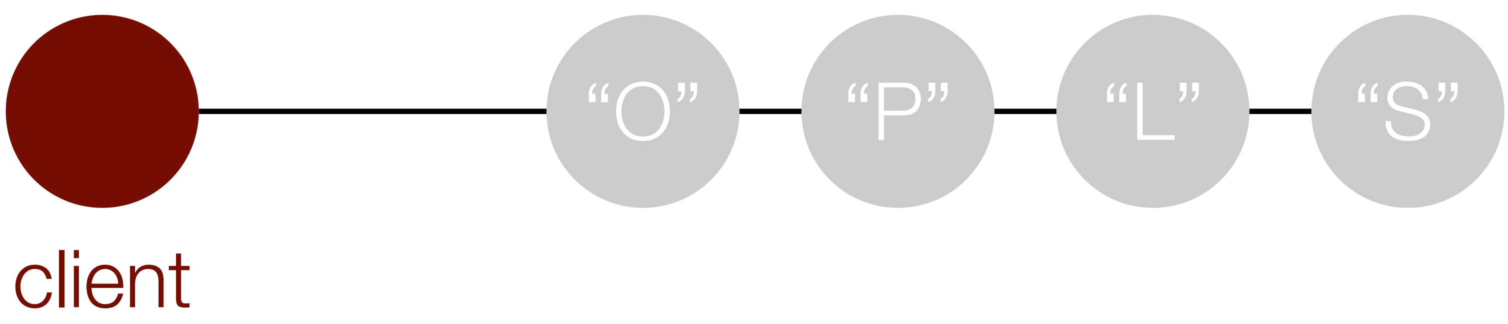
A message-passing queue

Queue of character processes:



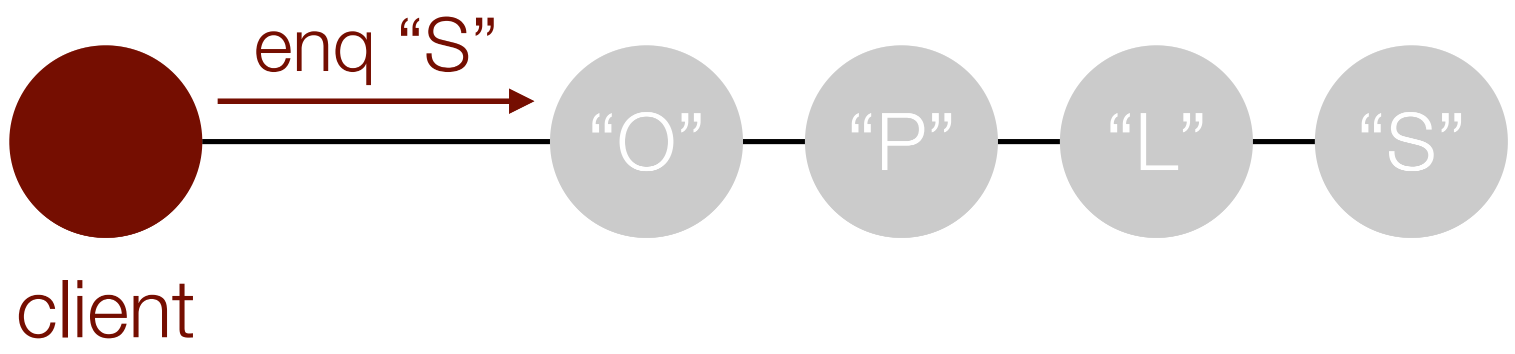
A message-passing queue

Queue of character processes:



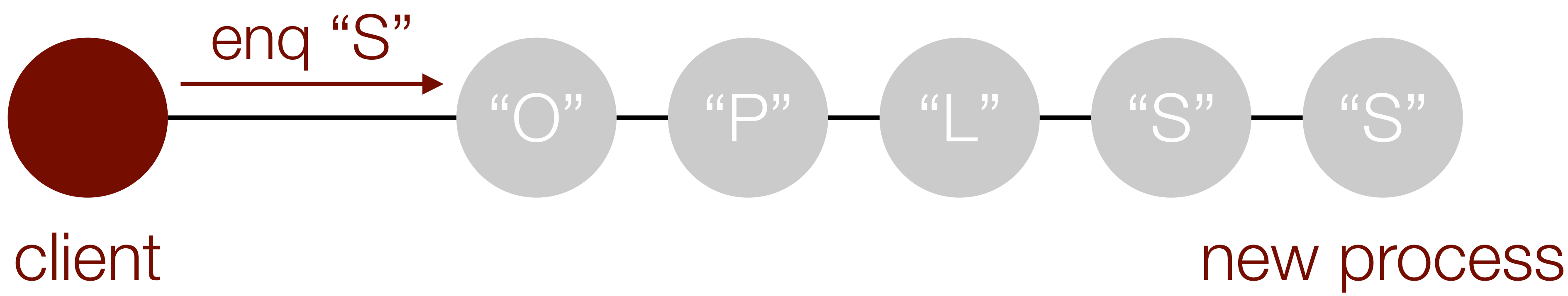
A message-passing queue

Queue of character processes:



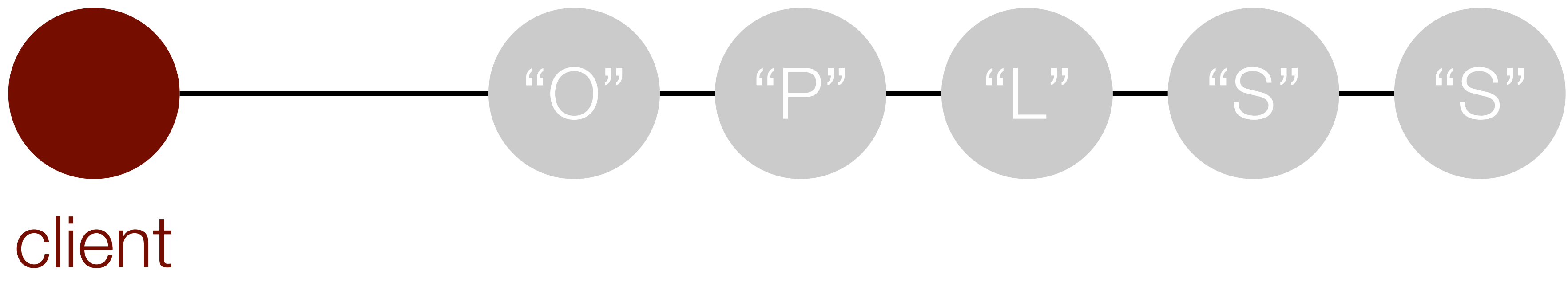
A message-passing queue

Queue of character processes:



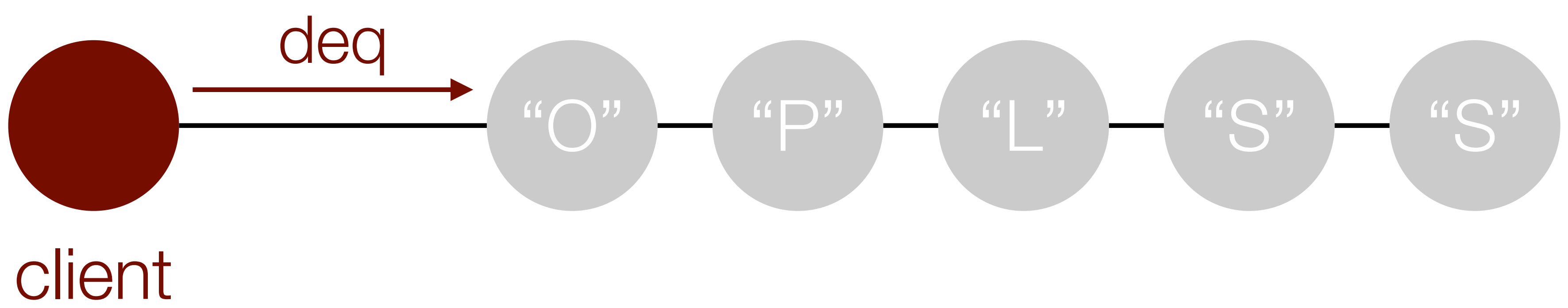
A message-passing queue

Queue of character processes:



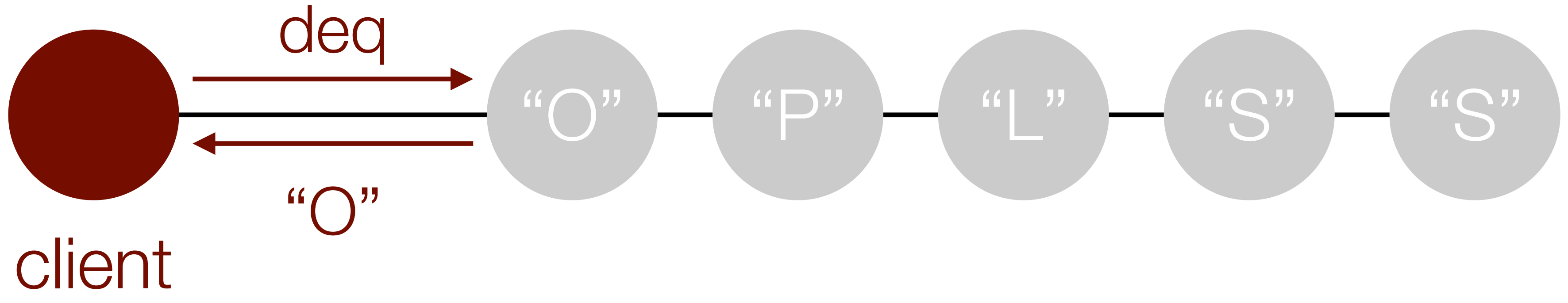
A message-passing queue

Queue of character processes:



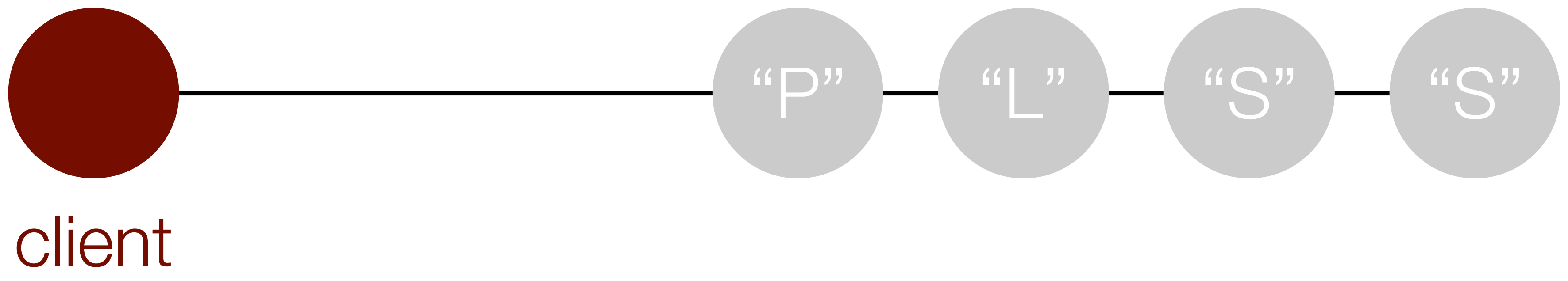
A message-passing queue

Queue of character processes:



A message-passing queue

Queue of character processes:



A message-passing queue

A message-passing queue

Here, we've exchanged basic values (e.g., characters).

A message-passing queue

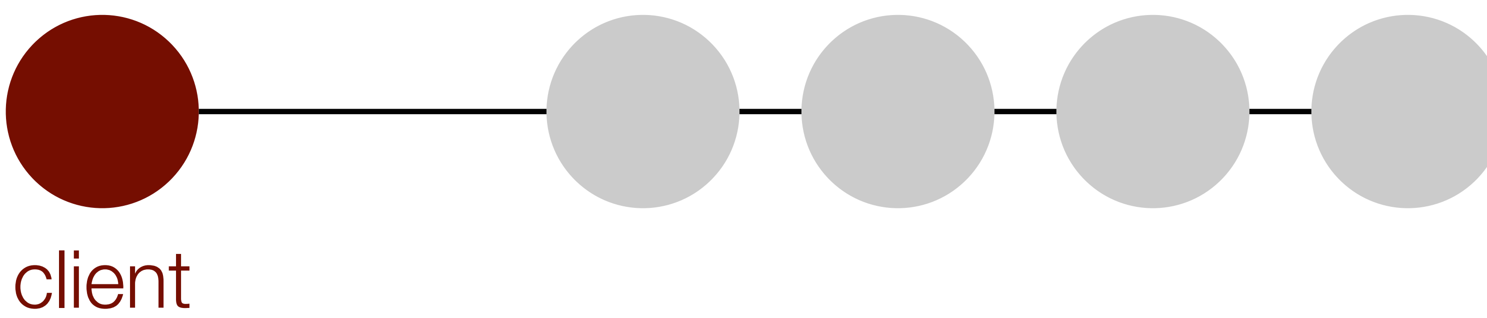
Here, we’ve exchanged basic values (e.g., characters).

In original pi-calculus, only channel references can be exchanged.

A message-passing queue

Here, we've exchanged basic values (e.g., characters).

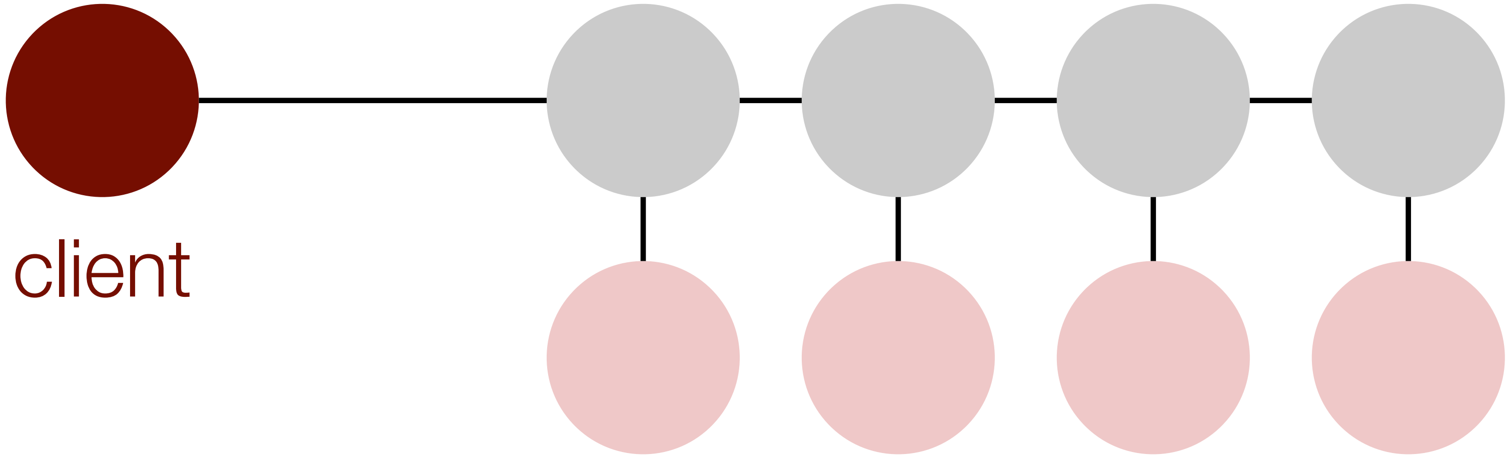
In original pi-calculus, only channel references can be exchanged.



A message-passing queue

Here, we've exchanged basic values (e.g., characters).

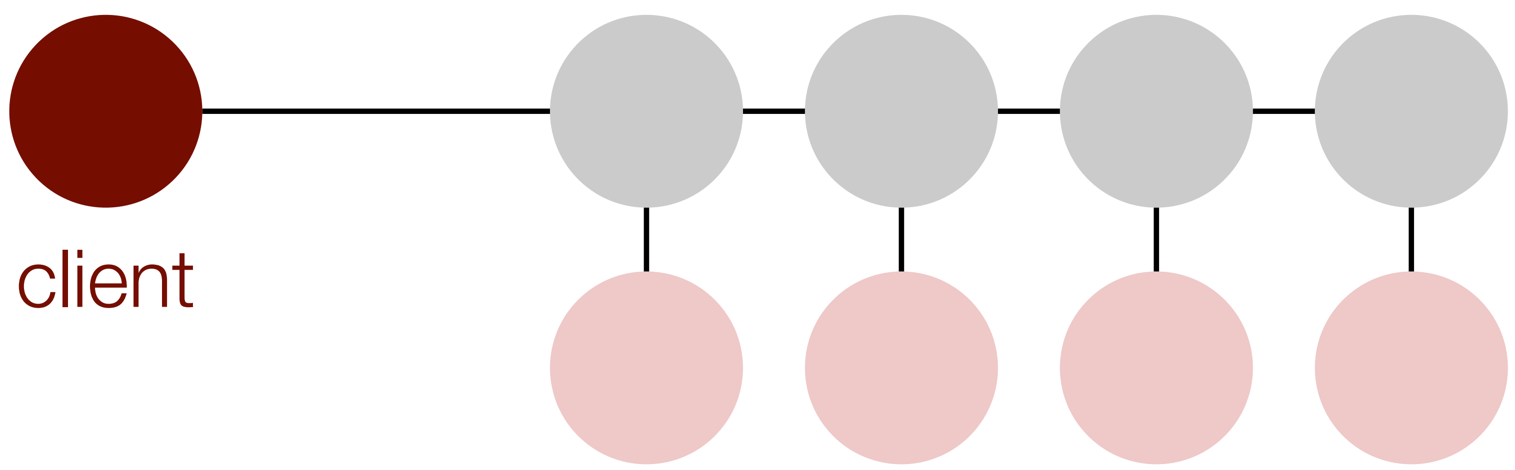
In original pi-calculus, only channel references can be exchanged.



A message-passing queue

Here, we've exchanged basic values (e.g., characters).

In original pi-calculus, only channel references can be exchanged.

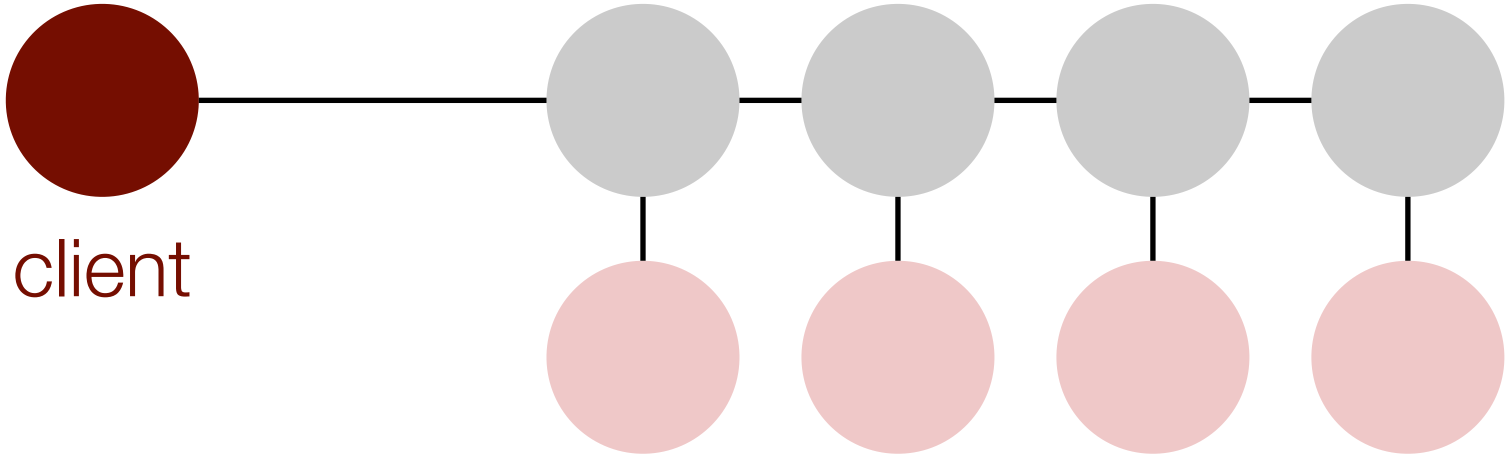


“mobility” in pi-calculus

A message-passing queue

Here, we've exchanged basic values (e.g., characters).

In original pi-calculus, only channel references can be exchanged.



"mobility" in pi-calculus

"higher-order channels" in session types

Session types

Types for protocols of message exchange

Types for protocols of message exchange

Session types

Kohei Honda. Types for dyadic interaction. CONCUR 1993.

Types for protocols of message exchange

Session types

$$
\begin{array}{lcl}
A & \triangleq & ?[T].A' \mid ![T].A' \mid \\
 & & \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
 & & \mathsf{end} \mid X \mid \mu X.A' \\
T & \triangleq & A \mid \mathsf{int} \mid \ldots
\end{array}
$$

Kohei Honda. Types for dyadic interaction. CONCUR 1993.

Types for protocols of message exchange

Session types

$$
\begin{array}{lll}
A & \triangleq & ?[T].A' \mid ![T].A' \mid \\
 & & \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
 & & \mathsf{end} \mid X \mid \mu X.A' \\
T & \triangleq & A \mid \mathsf{int} \mid \ldots
\end{array}
$$

input: receive message of type T, continue as type A’

Kohei Honda. Types for dyadic interaction. CONCUR 1993.

Types for protocols of message exchange

Session types

$$
\begin{aligned}
A &\triangleq ?[T].A' \mid ![T].A' \mid \\
&\quad \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
&\quad \mathsf{end} \mid X \mid \mu X.A' \\
T &\triangleq A \mid \mathsf{int} \mid \ldots
\end{aligned}
$$

input: receive message of type T, continue as type A’

types can be session (higher-order channels) or basic types

Kohei Honda. Types for dyadic interaction. CONCUR 1993.

Types for protocols of message exchange

Session types

$$
\begin{array}{lll}
A & \triangleq & ?[T].A' \mid ![T].A' \mid \\
 & & \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
 & & \mathsf{end} \mid X \mid \mu X.A' \\
T & \triangleq & A \mid \mathsf{int} \mid \ldots
\end{array}
$$

Kohei Honda. Types for dyadic interaction. CONCUR 1993.

Types for protocols of message exchange

Session types

$$
\begin{aligned}
A \quad & \triangleq \quad ?[T].A' \mid ![T].A' \mid \\
& \quad\;\; \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
& \quad\;\; \mathsf{end} \mid X \mid \mu X.A' \\
T \quad & \triangleq \quad A \mid \mathsf{int} \mid \ldots
\end{aligned}
$$

output: send message of type T, continue as type A’

Kohei Honda. Types for dyadic interaction. CONCUR 1993.

Types for protocols of message exchange

Session types

$$
\begin{array}{lll}
A & \triangleq & ?[T].A' \mid ![T].A' \mid \\
 & & \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
 & & \mathsf{end} \mid X \mid \mu X.A' \\
T & \triangleq & A \mid \mathsf{int} \mid \ldots
\end{array}
$$

Kohei Honda. Types for dyadic interaction. CONCUR 1993.

Types for protocols of message exchange

Session types

$$
\begin{array}{lll}
A & \triangleq & ?[T].A' \mid ![T].A' \mid \\
 & & \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
 & & \mathsf{end} \mid X \mid \mu X.A' \\
T & \triangleq & A \mid \mathsf{int} \mid \ldots
\end{array}
$$

external choice: receive label l_i, continue as type A_i

Kohei Honda. Types for dyadic interaction. CONCUR 1993.

Types for protocols of message exchange

Session types

$$
\begin{array}{lll}
A & \triangleq & ?[T].A' \mid ![T].A' \mid \\
 & & \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
 & & \mathsf{end} \mid X \mid \mu X.A' \\
T & \triangleq & A \mid \mathsf{int} \mid \ldots
\end{array}
$$

Kohei Honda. Types for dyadic interaction. CONCUR 1993.

Types for protocols of message exchange

Session types

$$\begin{array}{lll} A & \triangleq & ?[T].A' \mid ![T].A' \mid \\ & & \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\ & & \mathsf{end} \mid X \mid \mu X.A' \\ T & \triangleq & A \mid \mathsf{int} \mid \ldots \end{array}$$

internal choice: send label l_i, continue as type A_i

Kohei Honda. Types for dyadic interaction. CONCUR 1993.

Types for protocols of message exchange

Session types

$$
\begin{array}{lll}
A & \triangleq & ?[T].A' \mid ![T].A' \mid \\
 & & \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
 & & \mathsf{end} \mid X \mid \mu X.A' \\
T & \triangleq & A \mid \mathsf{int} \mid \ldots
\end{array}
$$

Kohei Honda. Types for dyadic interaction. CONCUR 1993.

Types for protocols of message exchange

Session types

$$
\begin{array}{lll}
A & \triangleq & ?[T].A' \mid ![T].A' \mid \\
 & & \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
 & & \mathsf{end} \mid X \mid \mu X.A' \\
T & \triangleq & A \mid \mathsf{int} \mid \ldots
\end{array}
$$

termination: close session and terminate

Kohei Honda. Types for dyadic interaction. CONCUR 1993.

Types for protocols of message exchange

Session types

$$
\begin{array}{lll}
A & \triangleq & ?[T].A' \mid ![T].A' \mid \\
 & & \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
 & & \mathsf{end} \mid X \mid \mu X.A' \\
T & \triangleq & A \mid \mathsf{int} \mid \ldots
\end{array}
$$

Kohei Honda. Types for dyadic interaction. CONCUR 1993.

Types for protocols of message exchange

Session types

$$
\begin{array}{lll}
A & \triangleq & ?[T].A' \mid ![T].A' \mid \\
 & & \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
 & & \mathsf{end} \mid X \mid \mu X.A' \\
T & \triangleq & A \mid \mathsf{int} \mid \ldots
\end{array}
$$

recursive session types

Kohei Honda. Types for dyadic interaction. CONCUR 1993.

Types for protocols of message exchange

Session types

$$
\begin{array}{lll}
A & \triangleq & ?[T].A' \mid ![T].A' \mid \\
 & & \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
 & & \mathsf{end} \mid X \mid \mu X.A' \\
T & \triangleq & A \mid \mathsf{int} \mid \ldots
\end{array}
$$

Kohei Honda. Types for dyadic interaction. CONCUR 1993.

Types for protocols of message exchange

Session types

$$
\begin{array}{lll}
A & \triangleq & ?[T].A' \mid ![T].A' \mid \\
 & & \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
 & & \mathsf{end} \mid X \mid \mu X.A' \\
T & \triangleq & A \mid \mathsf{int} \mid \ldots
\end{array}
$$

Queue session type:

Kohei Honda. Types for dyadic interaction. CONCUR 1993.

Types for protocols of message exchange

Session types

$$
\begin{aligned}
A \quad &\triangleq \quad ?[T].A' \mid ![T].A' \mid \\
&\qquad \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
&\qquad \mathsf{end} \mid X \mid \mu X.A' \\
T \quad &\triangleq \quad A \mid \mathsf{int} \mid \ldots
\end{aligned}
$$

Queue session type:

$$
\begin{aligned}
\mathsf{queue} = \&\{&\mathsf{enq} : ?[\mathsf{char}].\mathsf{queue}, \\
&\mathsf{deq} : \oplus\{\mathsf{none} : \mathsf{end}, \mathsf{some} : ![\mathsf{char}].\mathsf{queue}\}\}
\end{aligned}
$$

Kohei Honda. Types for dyadic interaction. CONCUR 1993.

Queue session type in action

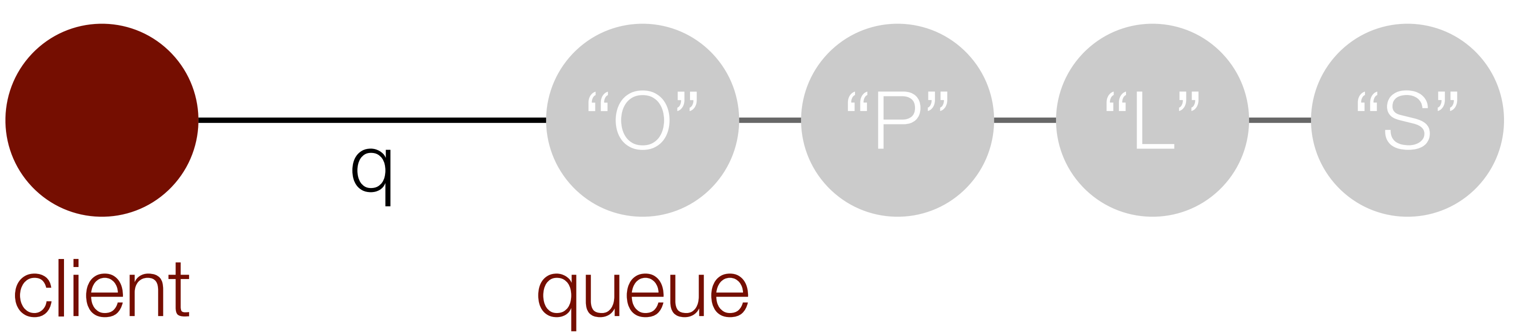
Queue session type:

$$\mathsf{queue} = \&\{\mathsf{enq} : ?[\mathsf{char}].\mathsf{queue},$$
$$\mathsf{deq} : \oplus\{\mathsf{none} : \mathsf{end}, \mathsf{some} : ![\mathsf{char}].\mathsf{queue}\}\}$$

Queue session type in action

Queue session type:

$$\mathsf{queue} = \&\{\mathsf{enq} : ?[\mathsf{char}].\mathsf{queue},\; \mathsf{deq} : \oplus\{\mathsf{none} : \mathsf{end}, \mathsf{some} : ![\mathsf{char}].\mathsf{queue}\}\}$$

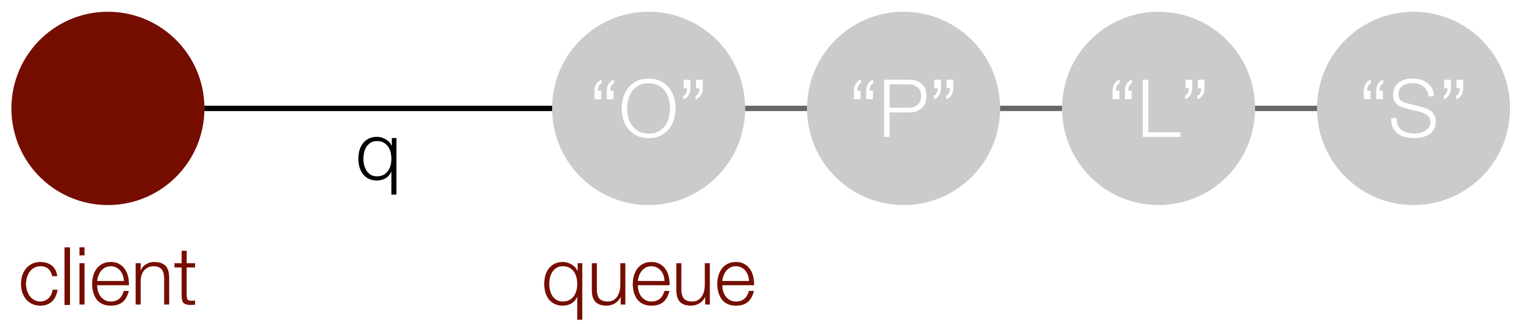


Queue session type in action

Queue session type:

$$\mathsf{queue} = \&\{\mathsf{enq} : ?[\mathsf{char}].\mathsf{queue},\\ \mathsf{deq} : \oplus\{\mathsf{none} : \mathsf{end}, \mathsf{some} : ![\mathsf{char}].\mathsf{queue}\}\}$$

Type of channel q:

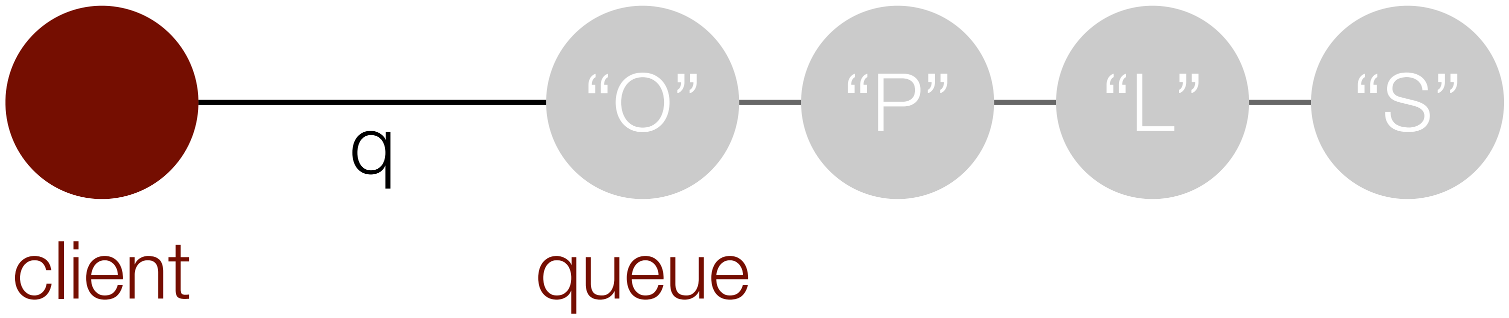


Queue session type in action

Queue session type:

$$\text{queue} = \&\{\text{enq} : ?[\text{char}].\text{queue}, \\ \text{deq} : \oplus\{\text{none} : \text{end}, \text{some} : ![\text{char}].\text{queue}\}\}$$

Type of channel q: queue

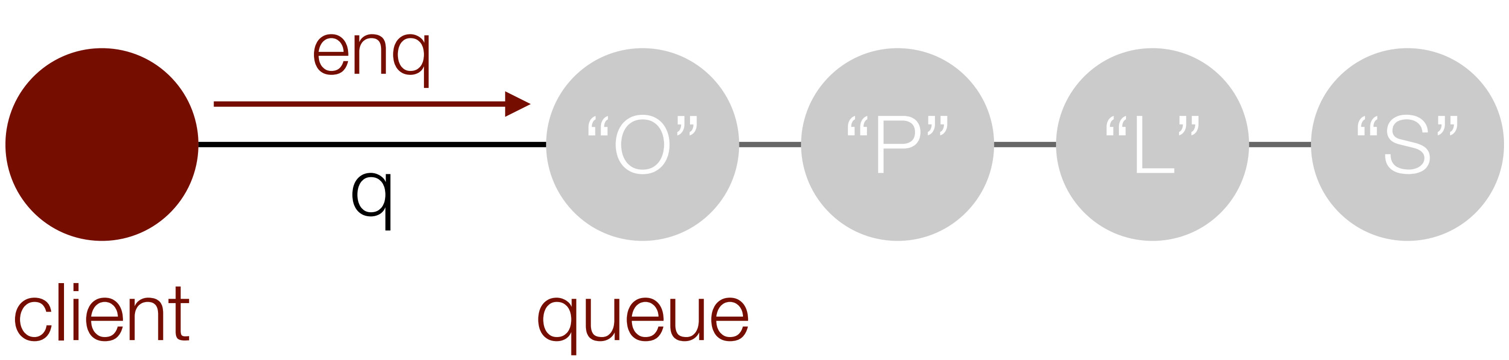


Queue session type in action

Queue session type:

$$\mathsf{queue} = \&\{\mathsf{enq} : ?[\mathsf{char}].\mathsf{queue},\\ \mathsf{deq} : \oplus\{\mathsf{none} : \mathsf{end}, \mathsf{some} : ![\mathsf{char}].\mathsf{queue}\}\}$$

Type of channel q: queue

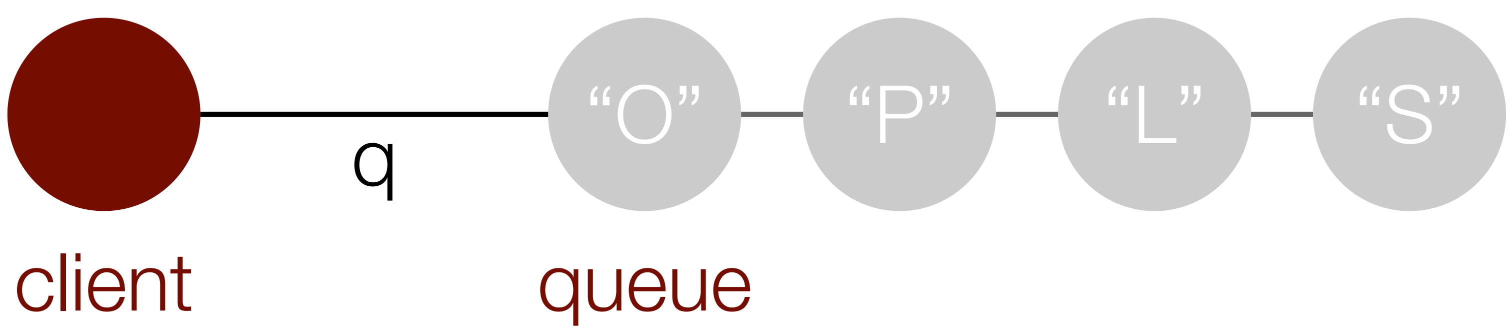


Queue session type in action

Queue session type:

$$\mathsf{queue} = \&\{\mathsf{enq} : ?[\mathsf{char}].\mathsf{queue}, \\ \mathsf{deq} : \oplus\{\mathsf{none} : \mathsf{end}, \mathsf{some} : ![\mathsf{char}].\mathsf{queue}\}\}$$

Type of channel q: $?[\mathsf{char}].\mathsf{queue}$

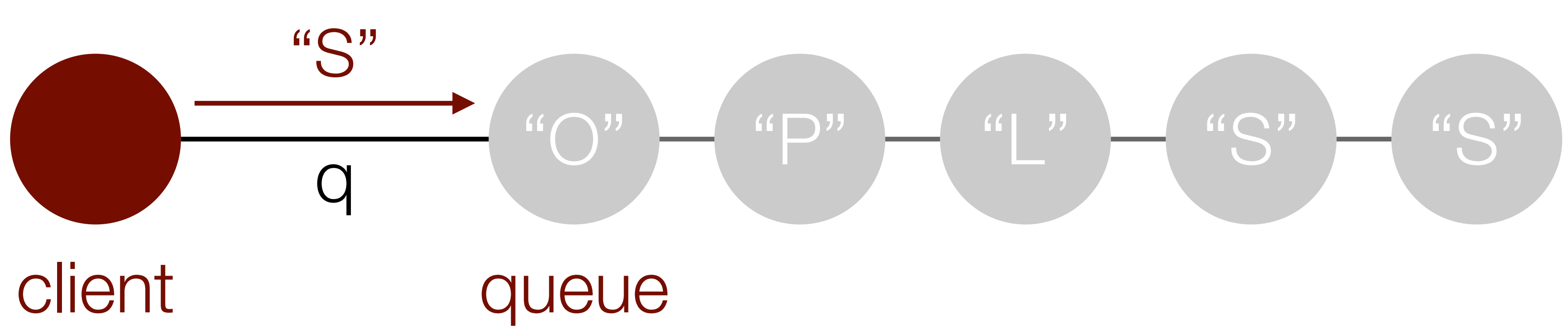


Queue session type in action

Queue session type:

$$\text{queue} = \&\{\text{enq} : ?[\text{char}].\text{queue},\\ \quad \text{deq} : \oplus\{\text{none} : \text{end}, \text{some} : ![\text{char}].\text{queue}\}\}$$

Type of channel q: $?[\text{char}].\text{queue}$

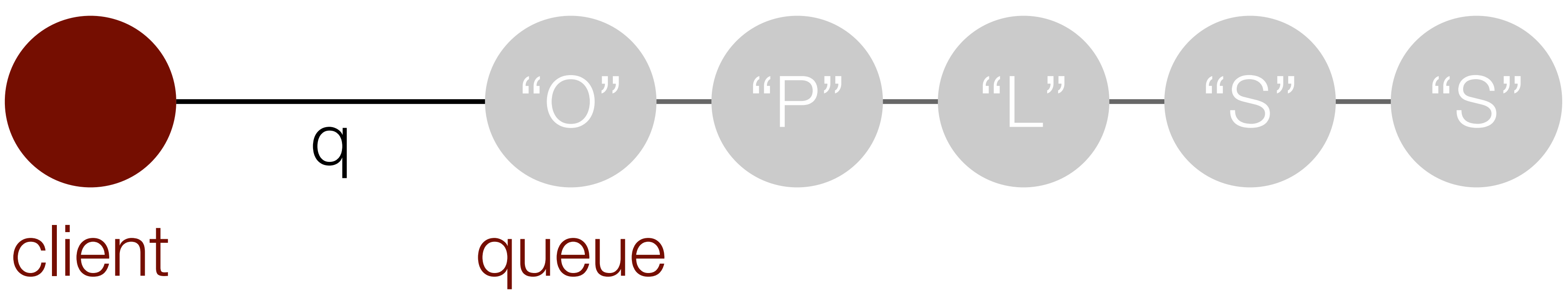


Queue session type in action

Queue session type:

$$\mathsf{queue} = \&\{\mathsf{enq} : ?[\mathsf{char}].\mathsf{queue},\ \mathsf{deq} : \oplus\{\mathsf{none} : \mathsf{end}, \mathsf{some} : ![\mathsf{char}].\mathsf{queue}\}\}$$

Type of channel q: queue

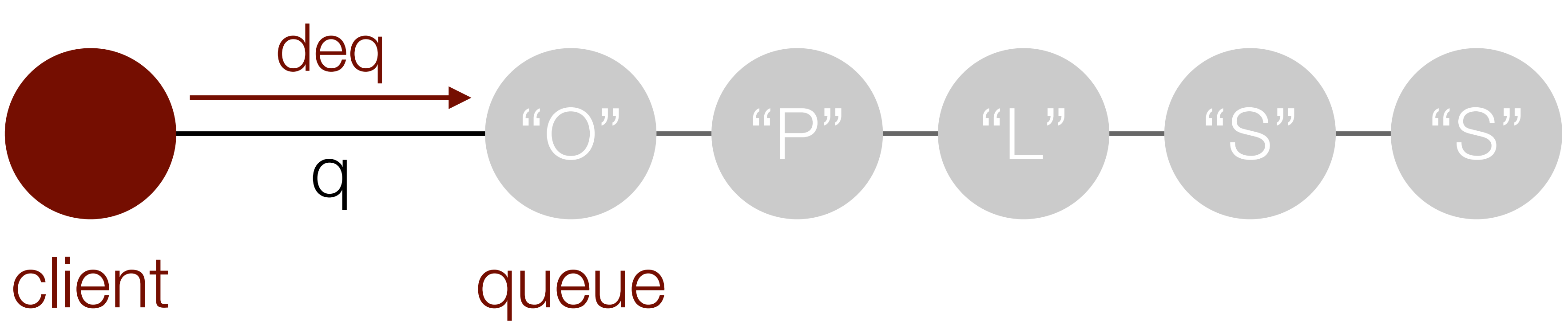


Queue session type in action

Queue session type:

$$\mathsf{queue} = \&\{\mathsf{enq} : ?[\mathsf{char}].\mathsf{queue}, \\ \mathsf{deq} : \oplus\{\mathsf{none} : \mathsf{end}, \mathsf{some} : ![\mathsf{char}].\mathsf{queue}\}\}$$

Type of channel q: queue

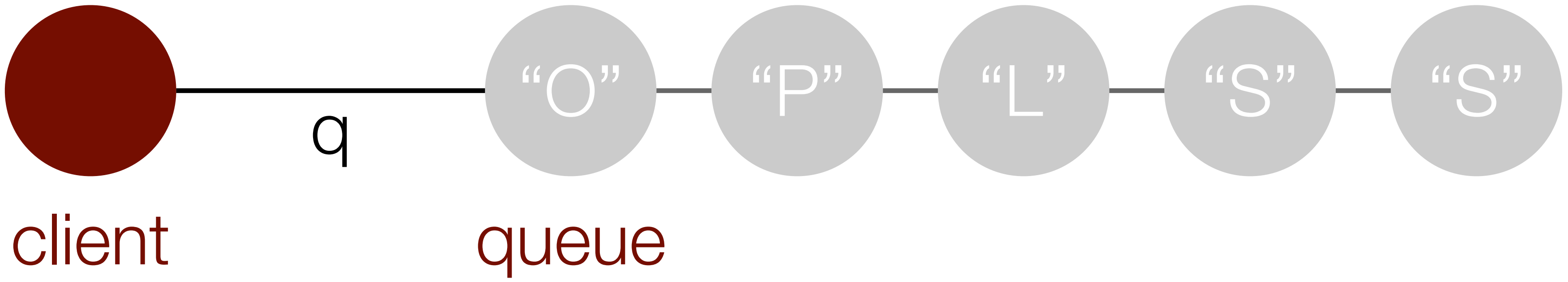


Queue session type in action

Queue session type:

$$\mathsf{queue} = \&\{\mathsf{enq} : ?[\mathsf{char}].\mathsf{queue}, \\ \mathsf{deq} : \oplus\{\mathsf{none} : \mathsf{end}, \mathsf{some} : ![\mathsf{char}].\mathsf{queue}\}\}$$

Type of channel q: $\oplus\{\mathsf{none} : \mathsf{end}, \mathsf{some} : ![\mathsf{char}].\mathsf{queue}\}$

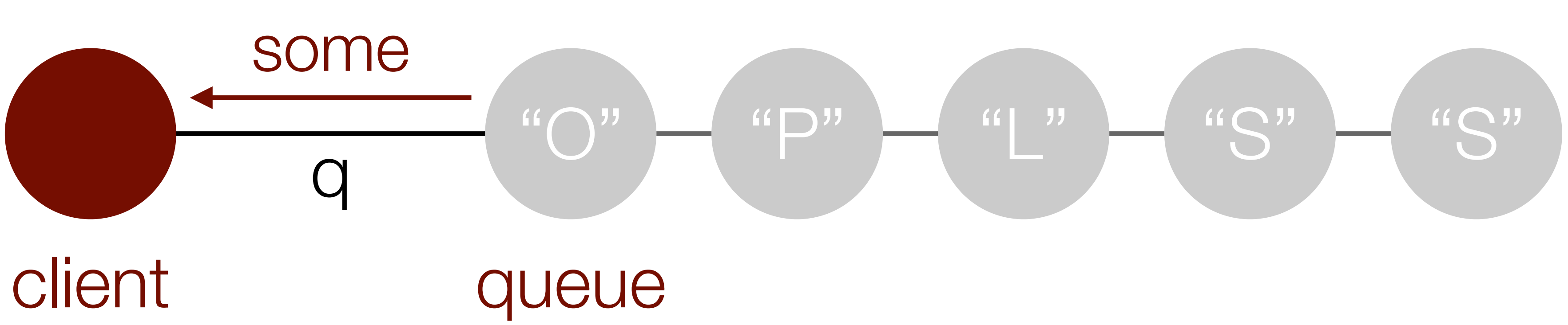


Queue session type in action

Queue session type:

$$\text{queue} = \&\{\text{enq} : ?[\text{char}].\text{queue}, \\ \text{deq} : \oplus\{\text{none} : \text{end}, \text{some} : ![\text{char}].\text{queue}\}\}$$

Type of channel q: $\oplus\{\text{none} : \text{end}, \text{some} : ![\text{char}].\text{queue}\}$

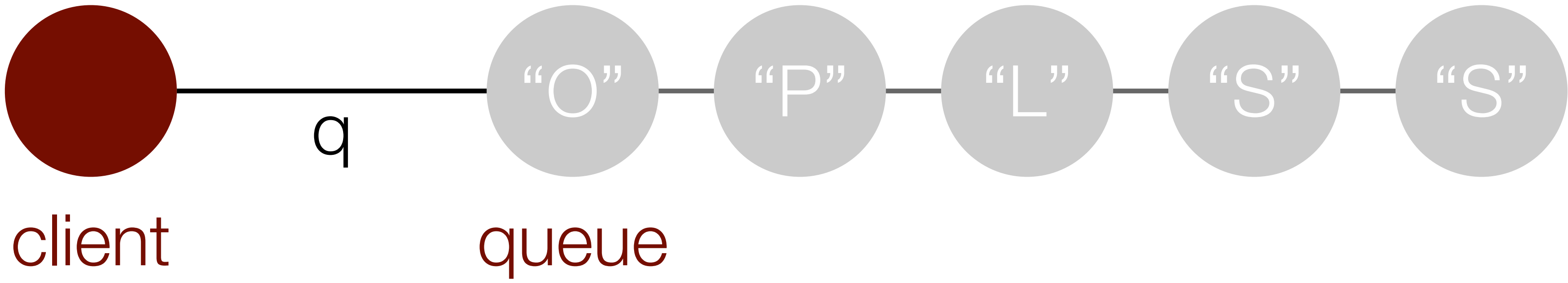


Queue session type in action

Queue session type:

$$\begin{aligned}\mathsf{queue} = \&\{&\mathsf{enq} : ?[\mathsf{char}].\mathsf{queue},\\ &\mathsf{deq} : \oplus\{\mathsf{none} : \mathsf{end}, \mathsf{some} : ![\mathsf{char}].\mathsf{queue}\}\}\end{aligned}$$

Type of channel q: $![\mathsf{char}].\mathsf{queue}$

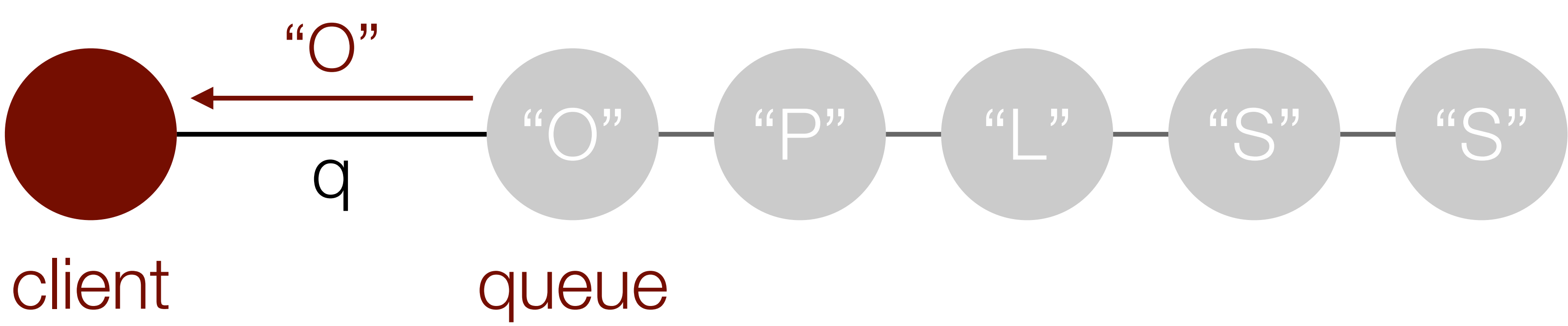


Queue session type in action

Queue session type:

$$\text{queue} = \&\{\text{enq} : ?[\text{char}].\text{queue}, \\ \text{deq} : \oplus\{\text{none} : \text{end}, \text{some} : ![\text{char}].\text{queue}\}\}$$

Type of channel q: $![\text{char}].\text{queue}$

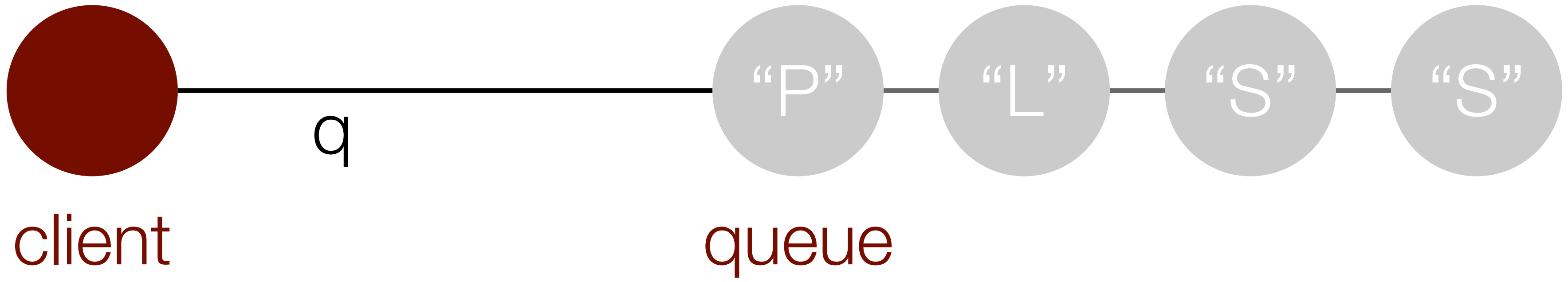


Queue session type in action

Queue session type:

$$\mathsf{queue} = \&\{\mathsf{enq} : ?[\mathsf{char}].\mathsf{queue}, \quad \mathsf{deq} : \oplus\{\mathsf{none} : \mathsf{end}, \mathsf{some} : ![\mathsf{char}].\mathsf{queue}\}\}$$

Type of channel q: queue

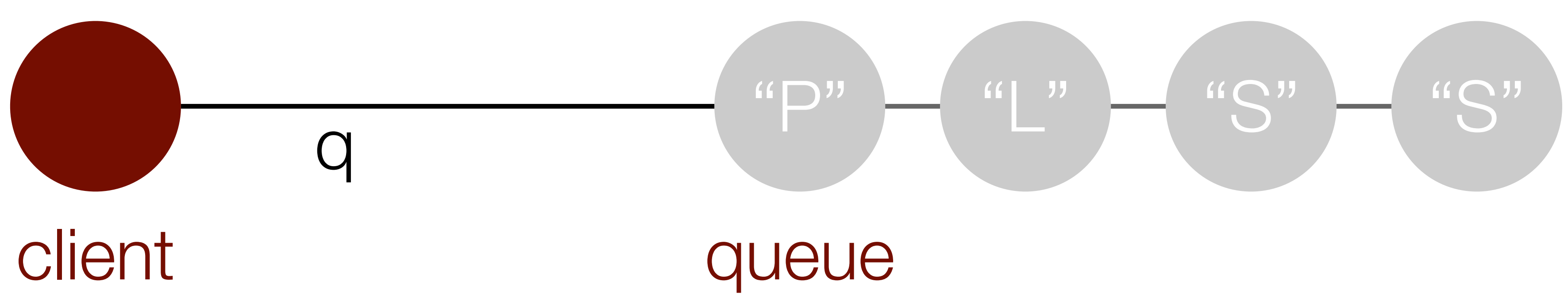


Queue session type in action

Queue session type:

$$\begin{aligned} \mathsf{queue} = \&\{ & \mathsf{enq} : ?[\mathsf{char}].\mathsf{queue}, \\ & \mathsf{deq} : \oplus\{\mathsf{none} : \mathsf{end}, \mathsf{some} : ![\mathsf{char}].\mathsf{queue}\}\} \end{aligned}$$

Type of channel q: queue



➡ type of channel/process changes with message exchange

Protocol verification

Protocol verification

➡ session types ensure protocol adherence by type-checking

Protocol verification

- session types ensure protocol adherence by type-checking
  - session fidelity (a.k.a. preservation)

Protocol verification

session types ensure protocol adherence by type-checking

session fidelity (a.k.a. preservation)

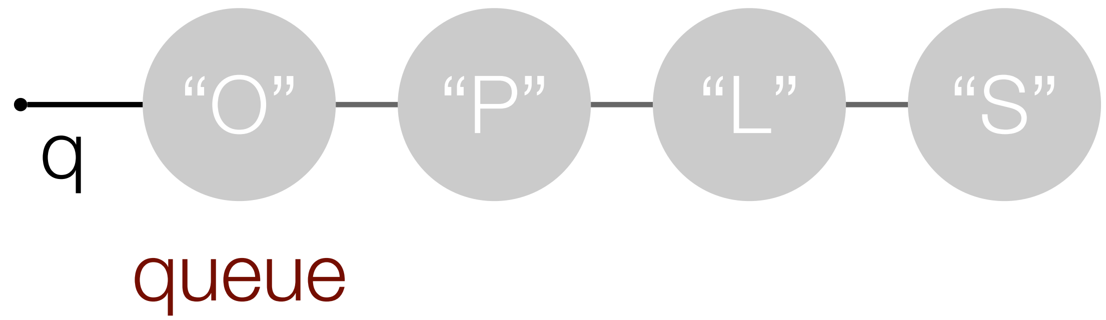
Challenges for preservation:

Protocol verification

session types ensure protocol adherence by type-checking

session fidelity (a.k.a. preservation)

Challenges for preservation:

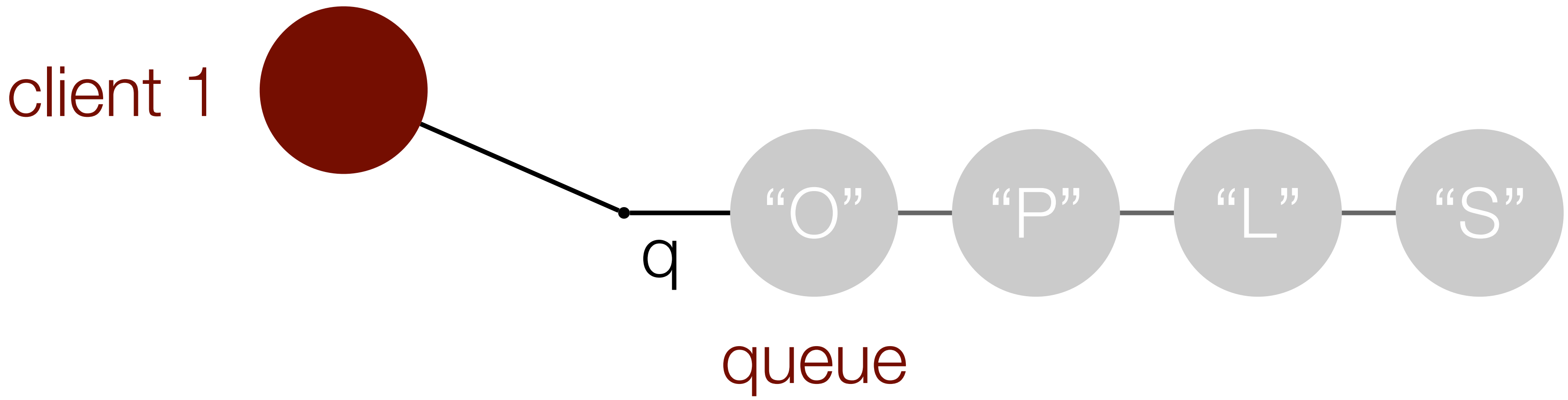


Protocol verification

session types ensure protocol adherence by type-checking

session fidelity (a.k.a. preservation)

Challenges for preservation:

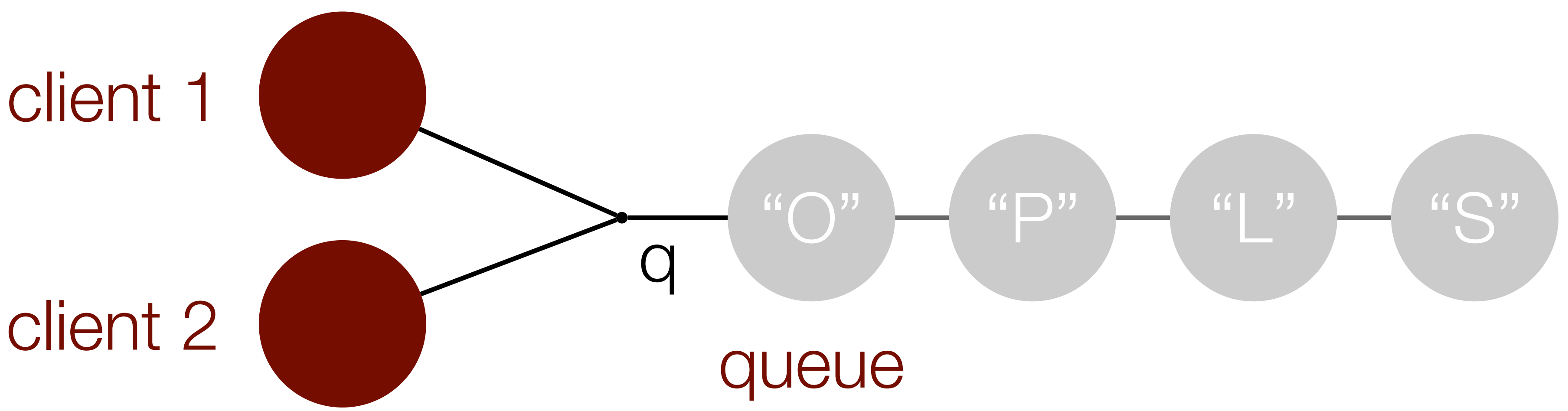


Protocol verification

session types ensure protocol adherence by type-checking

session fidelity (a.k.a. preservation)

Challenges for preservation:

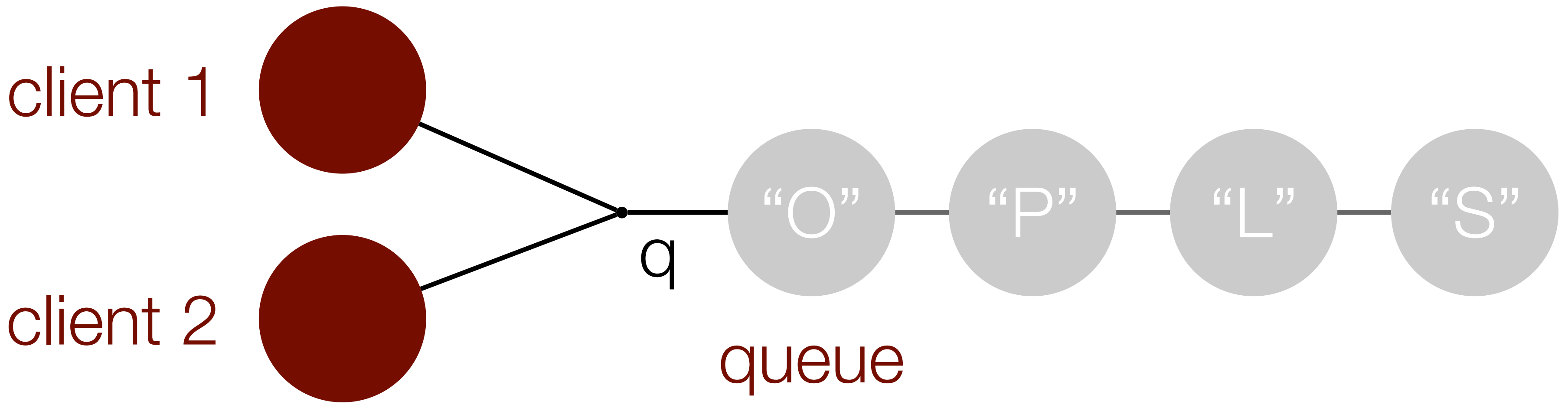


Protocol verification

session types ensure protocol adherence by type-checking

session fidelity (a.k.a. preservation)

Challenges for preservation:



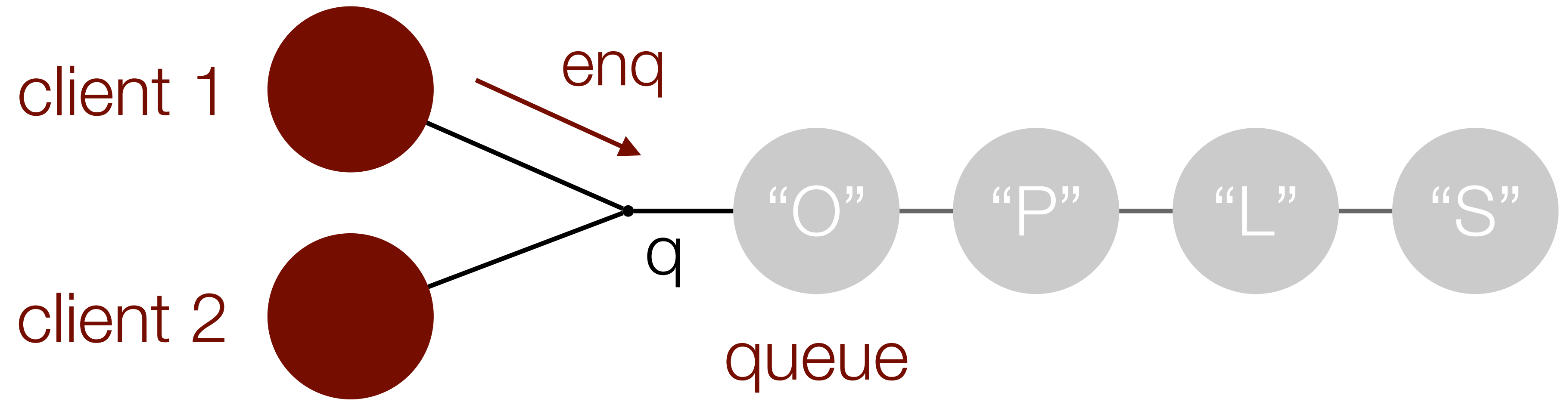
q: queue

Protocol verification

session types ensure protocol adherence by type-checking

session fidelity (a.k.a. preservation)

Challenges for preservation:



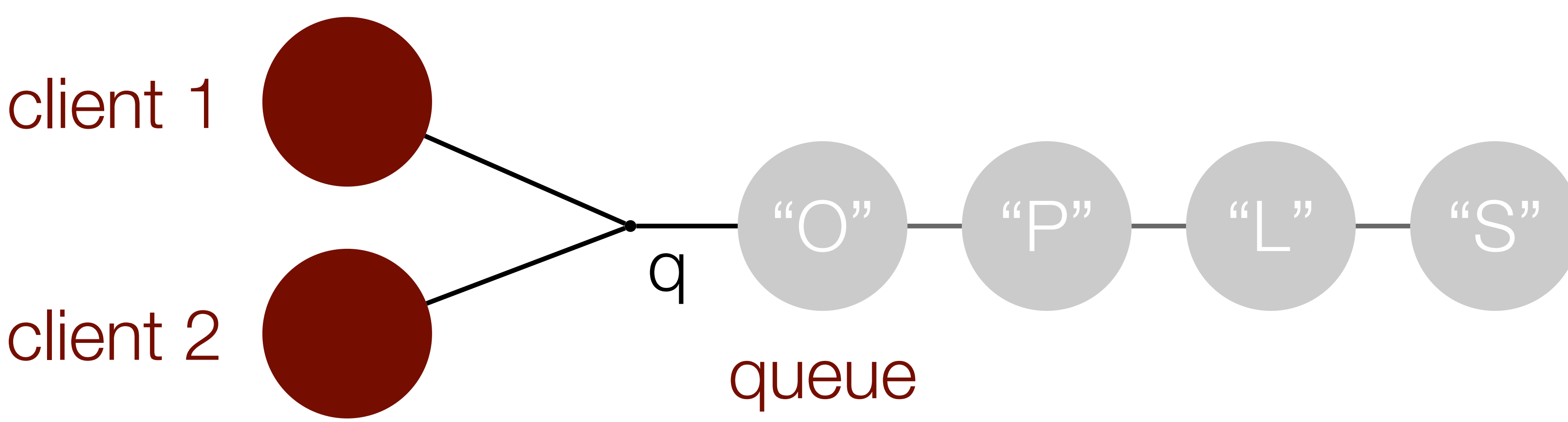
q: `queue`

Protocol verification

session types ensure protocol adherence by type-checking

session fidelity (a.k.a. preservation)

Challenges for preservation:



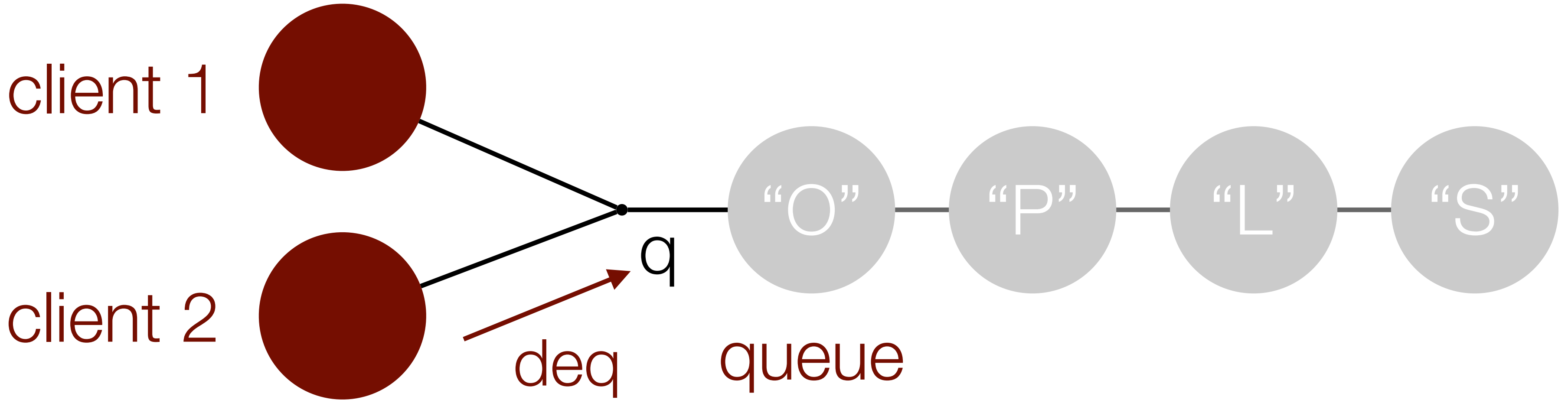
q: $?[\mathsf{char}].\mathsf{queue}$

Protocol verification

➡ session types ensure protocol adherence by type-checking

➡ session fidelity (a.k.a. preservation)

Challenges for preservation:



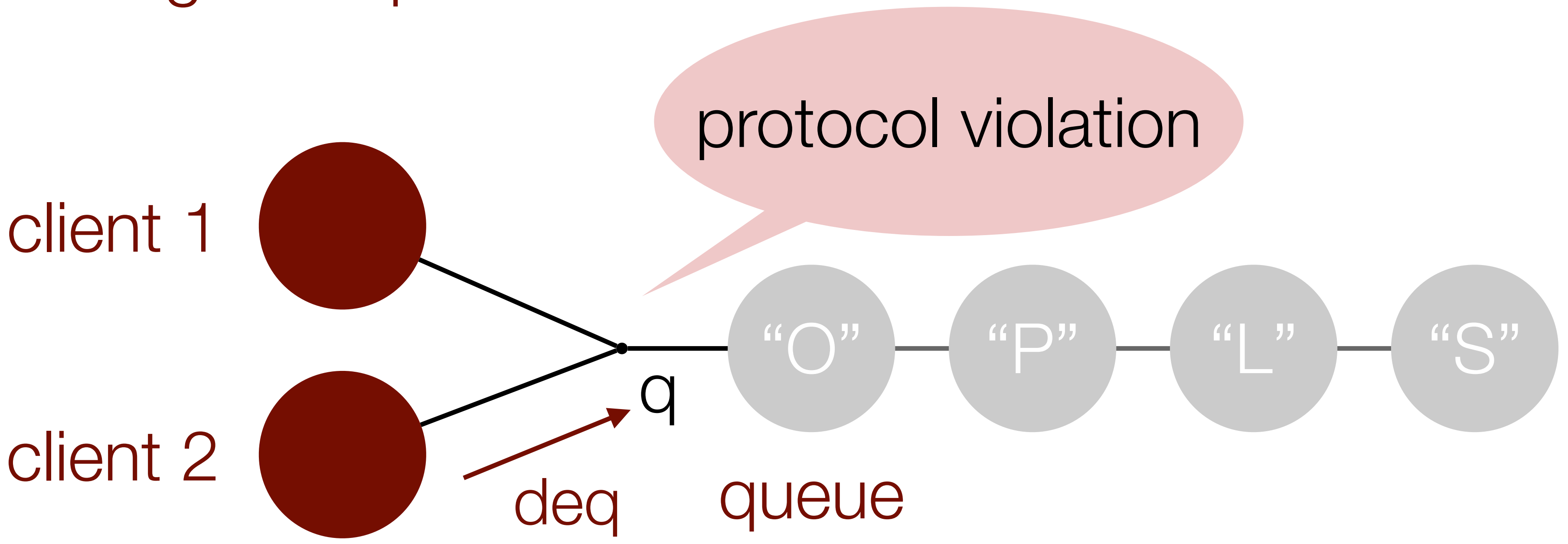
q: $?[\mathsf{char}].\mathsf{queue}$

Protocol verification

session types ensure protocol adherence by type-checking

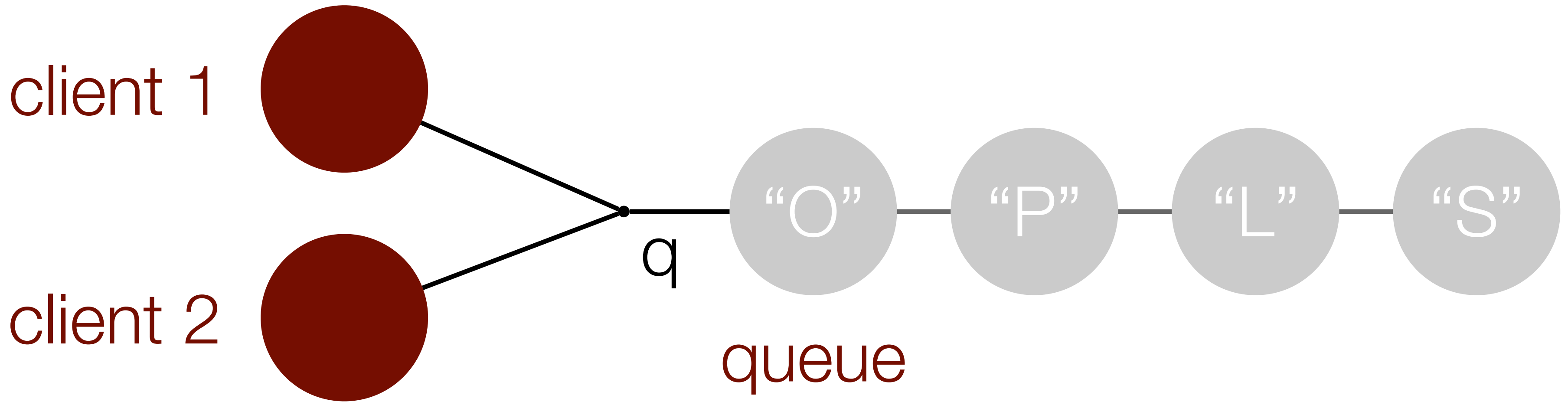
session fidelity (a.k.a. preservation)

Challenges for preservation:



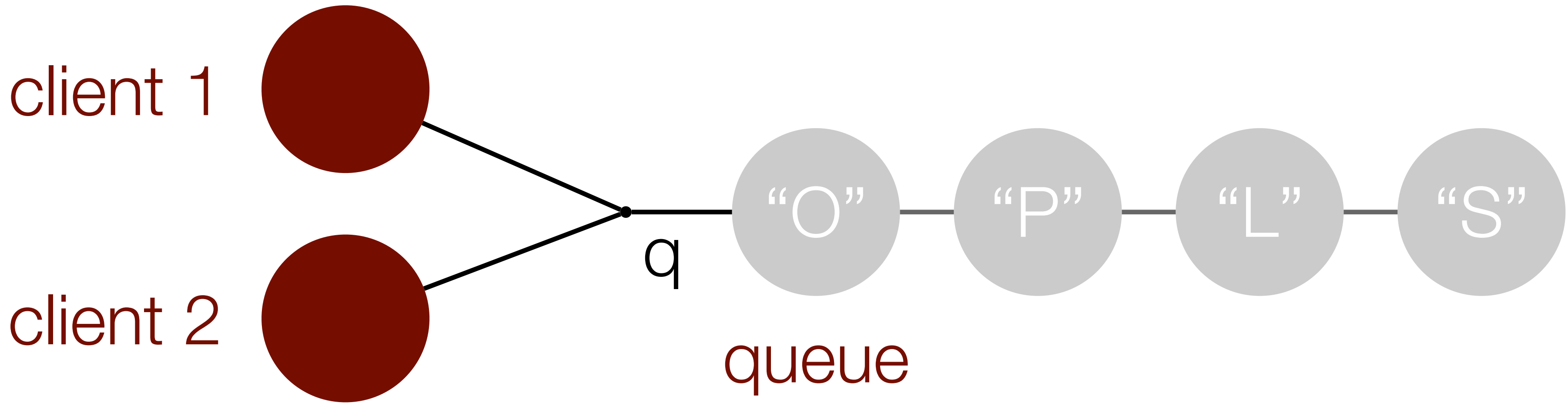
q: ?[char].queue

Protocol verification



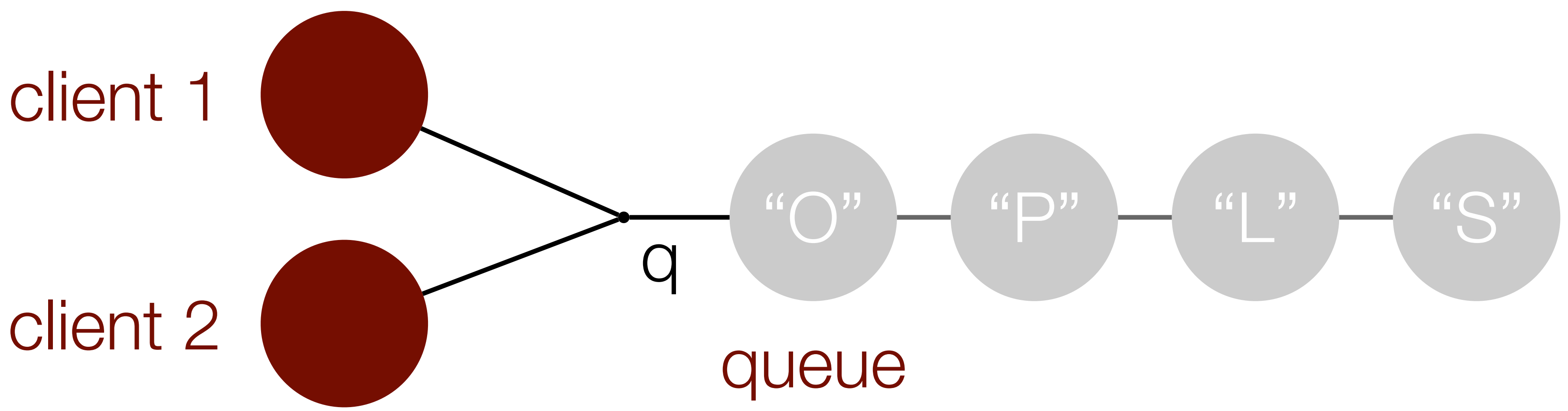
Protocol verification

Preservation: expectation for type of client and provider match



Protocol verification

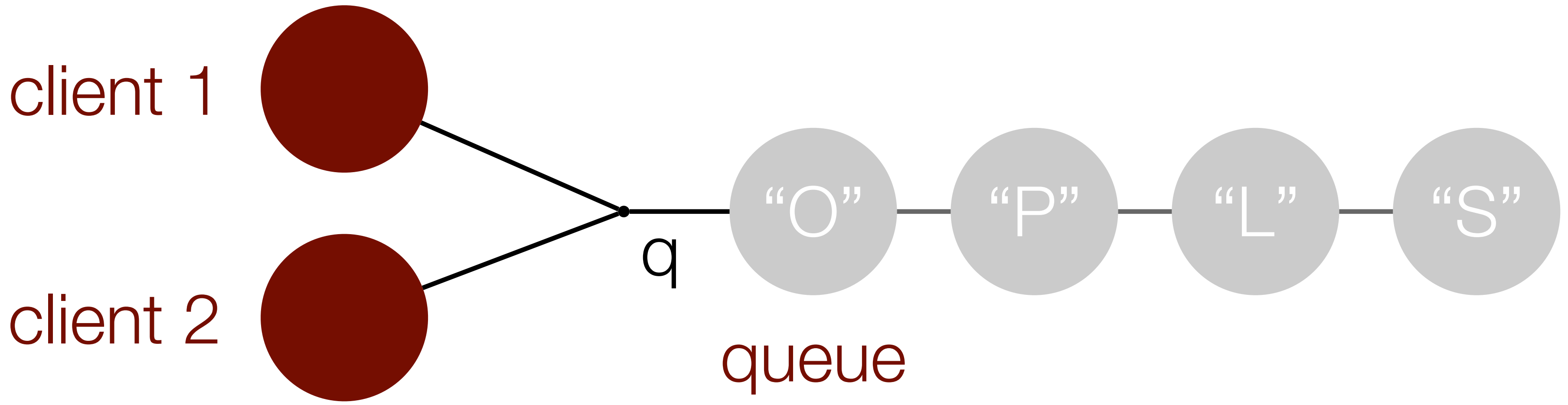
Preservation: expectation for type of client and provider match



Strategies for recovery:

Protocol verification

Preservation: expectation for type of client and provider match

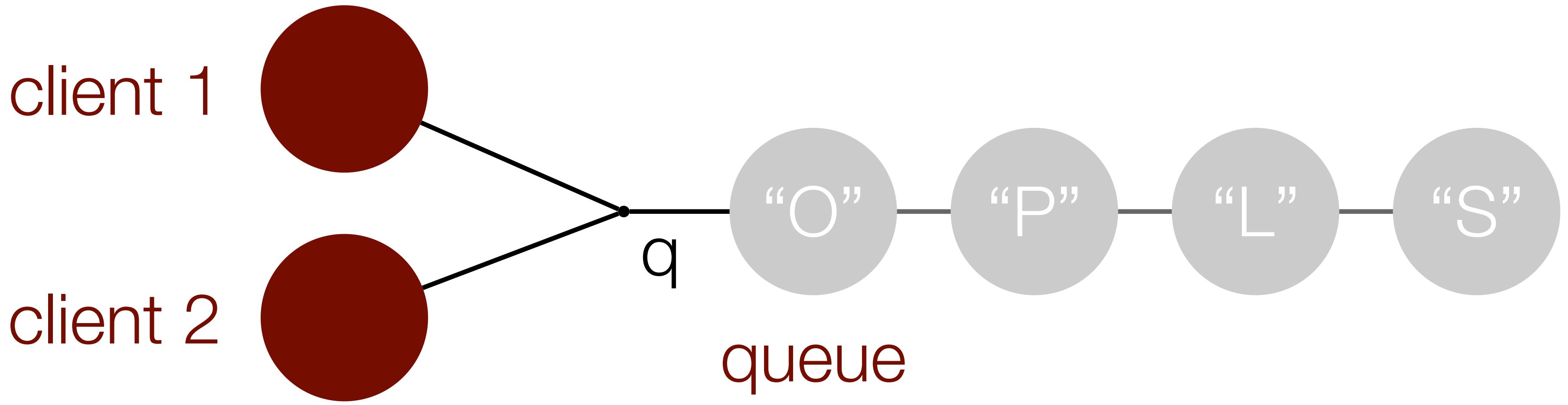


Strategies for recovery:

- **employ linearity/ownership to restrict to single client**

Protocol verification

Preservation: expectation for type of client and provider match

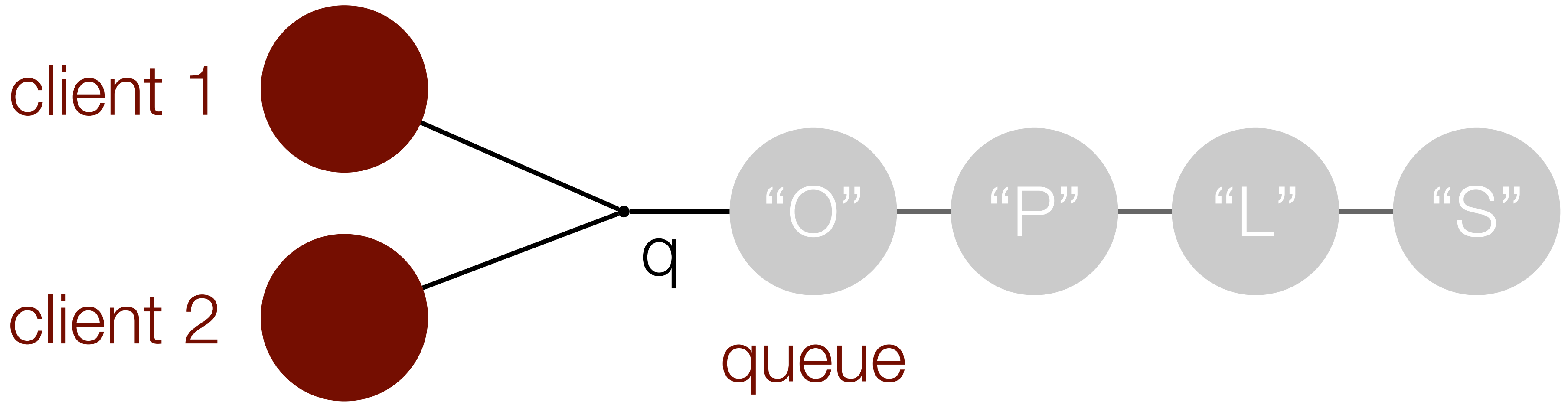


Strategies for recovery:

- employ linearity/ownership to restrict to single client
  - disallow multiple clients

Protocol verification

Preservation: expectation for type of client and provider match



Strategies for recovery:

- employ linearity/ownership to restrict to single client
  - disallow multiple clients
- allow multiple clients but control aliasing (manifest sharing)

Intuitionistic linear logic session types

Linear logic from a programming perspective

Linear logic from a programming perspective

Linear logic is a so-called substructural logic that tracks ownership.

Linear logic from a programming perspective

Linear logic is a so-called substructural logic that tracks ownership.

→ we first discover characteristics programmatically, then revisit them formally

Linear logic from a programming perspective

Linear logic is a so-called substructural logic that tracks ownership.

→ we first discover characteristics programmatically, then revisit them formally

Types:

A, B	$\triangleq$	$A \otimes B$	multiplicative conjunction	"channel output"
		$A \multimap B$	multiplicative implication	"channel input"
		$A \mathbin{\&} B$	additive conjunction	"external choice"
		$A \oplus B$	additive disjunction	"internal choice"
		$\mathbf{1}$	unit for $\otimes$	"termination"

Linear logic from a programming perspective

Linear logic is a so-called substructural logic that tracks ownership.

➡ we first discover characteristics programmatically, then revisit them formally

Types:

A, B	$\triangleq$	$A \otimes B$	multiplicative conjunction	"channel output"
		$A \multimap B$	multiplicative implication	"channel input"
		$A \& B$	additive conjunction	"external choice"
		$A \oplus B$	additive disjunction	"internal choice"
		$\mathbf{1}$	unit for $\otimes$	"termination"

➡ for simplicity, we restrict to binary external/internal choice and to higher-order channels

Linear logic from a programming perspective

Types:

A, B	$\triangleq$	$A \otimes B$	multiplicative conjunction	“channel output”
		$A \multimap B$	multiplicative implication	“channel input”
		$A \& B$	additive conjunction	“external choice”
		$A \oplus B$	additive disjunction	“internal choice”
		$\mathbf{1}$	unit for $\otimes$	“termination”

Linear logic from a programming perspective

Types:

A, B	$\triangleq$	$A \otimes B$	multiplicative conjunction	"channel output"
		$A \multimap B$	multiplicative implication	"channel input"
		$A \mathbin{\&} B$	additive conjunction	"external choice"
		$A \oplus B$	additive disjunction	"internal choice"
		$\mathbf{1}$	unit for $\otimes$	"termination"

Queue session type:

Linear logic from a programming perspective

Types:

A, B	$\triangleq$	$A \otimes B$	multiplicative conjunction	"channel output"
		$A \multimap B$	multiplicative implication	"channel input"
		$A \mathbin{\&} B$	additive conjunction	"external choice"
		$A \oplus B$	additive disjunction	"internal choice"
		$\mathbf{1}$	unit for $\otimes$	"termination"

Queue session type:

$$\begin{aligned} \mathsf{queue}\ A = \mathbin{\&}\{ & \mathsf{enq} : A \multimap \mathsf{queue}\ A, \\ & \mathsf{deq} : \oplus\{\mathsf{none} : \mathbf{1}, \mathsf{some} : A \otimes \mathsf{queue}\ A\}\} \end{aligned}$$

Linear logic from a programming perspective

Types:

A, B	$\triangleq$	$A \otimes B$	multiplicative conjunction	"channel output"
		$A \multimap B$	multiplicative implication	"channel input"
		$A \,\&\, B$	additive conjunction	"external choice"
		$A \oplus B$	additive disjunction	"internal choice"
		$\mathbf{1}$	unit for $\otimes$	"termination"

Queue session type:

polymorphic

$$\begin{aligned}\mathsf{queue}\ A = \&\{&\mathsf{enq} : A \multimap \mathsf{queue}\ A,\\ &\mathsf{deq} : \oplus\{\mathsf{none} : \mathbf{1}, \mathsf{some} : A \otimes \mathsf{queue}\ A\}\}\end{aligned}$$

Typing judgment and rules

Typing judgment and rules

Intuitionistic linear sequent:

$$x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A)$$

Typing judgment and rules

Intuitionistic linear sequent:

$$x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A)$$

"Process P offers a session of type A along channel x using session A_1*, ...,* A_n *provided along channels* x_1*, ...,* x_n*."*

Typing judgment and rules

antecedent

Intuitionistic linear sequent:

$$x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A)$$

“Process P offers a session of type A along channel x using session A_1, …, A_n provided along channels x_1, …, x_n.”

Typing judgment and rules

antecedent

succedent

Intuitionistic linear sequent:

$$x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A)$$

"Process P offers a session of type A along channel x using session A_1, …, A_n provided along channels x_1, …, x_n."

Typing judgment and rules

Intuitionistic linear sequent:

$$x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A)$$

"Process P offers a session of type A along channel x using session A_1, …, A_n provided along channels x_1, …, x_n."

Inference rule:

$$\frac{\Delta' \vdash Q :: (x : A')}{\Delta \vdash P; Q :: (x : A)}$$

Typing judgment and rules

Intuitionistic linear sequent:

$$x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A)$$

"Process P offers a session of type A along channel x using session A_1, ..., A_n provided along channels x_1, ..., x_n."

Inference rule:

conclusion $$\frac{\Delta' \vdash Q :: (x : A')}{\Delta \vdash P; Q :: (x : A)}$$

Typing judgment and rules

Intuitionistic linear sequent:

$$x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A)$$

"Process P offers a session of type A along channel x using session A_1, …, A_n provided along channels x_1, …, x_n."

Inference rule:

premise / conclusion

$$\frac{\Delta' \vdash Q :: (x : A')}{\Delta \vdash P; Q :: (x : A)}$$

Typing judgment and rules

Intuitionistic linear sequent:

$$x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A)$$

"Process P offers a session of type A along channel x using session A_1, …, A_n provided along channels x_1, …, x_n."

Inference rule:

$$\frac{\Delta' \vdash Q :: (x : A')}{\Delta \vdash P; Q :: (x : A)}$$

premise (top), conclusion (bottom) — ↑ bottom-up reading

Typing judgment and rules

Intuitionistic linear sequent:

$$x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A)$$

"Process P offers a session of type A along channel x using session A_1, ..., A_n provided along channels x_1, ..., x_n."

Inference rule:

premise

conclusion

$$\frac{\Delta' \vdash Q :: (x : A')}{\Delta \vdash P; Q :: (x : A)}$$

bottom-up reading

current

Typing judgment and rules

Intuitionistic linear sequent:

$$x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A)$$

"Process P offers a session of type A along channel x using session A_1, …, A_n provided along channels x_1, …, x_n."

Inference rule:

continuation

premise $\Delta' \vdash Q :: (x : A')$

conclusion $\Delta \vdash P; Q :: (x : A)$

$$\frac{\Delta' \vdash Q :: (x : A')}{\Delta \vdash P; Q :: (x : A)}$$

↑ bottom-up reading

current

Typing judgment and rules

Intuitionistic linear sequent:

$$x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A)$$

"Process P offers a session of type A along channel x using session A_1, …, A_n provided along channels x_1, …, x_n."

Inference rule:

premise

conclusion

$$\frac{\Delta' \vdash Q :: (x : A')}{\Delta \vdash P; Q :: (x : A)}$$

bottom-up reading

Left and right rules:

$$\frac{\Delta', x : B \vdash Q :: (z : C)}{\Delta, x : A \diamond B \vdash P; Q :: (z : C)} \diamond_L \qquad \frac{\Delta' \vdash Q :: (x : B)}{\Delta \vdash P; Q :: (x : A \diamond B)} \diamond_R$$

Typing judgment and rules

Intuitionistic linear sequent:

$$x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A)$$

"Process P offers a session of type A along channel x using session A_1, …, A_n provided along channels x_1, …, x_n."

Inference rule:

$$\begin{array}{rc} \text{premise} & \Delta' \vdash Q :: (x : A') \\ \hline \text{conclusion} & \Delta \vdash P; Q :: (x : A) \end{array}$$

bottom-up reading

Left and right rules:

client

$$\frac{\Delta', x : B \vdash Q :: (z : C)}{\Delta, x : A \diamond B \vdash P; Q :: (z : C)} \diamond_L \qquad \frac{\Delta' \vdash Q :: (x : B)}{\Delta \vdash P; Q :: (x : A \diamond B)} \diamond_R$$

Typing judgment and rules

Intuitionistic linear sequent:

$$x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A)$$

"Process P offers a session of type A along channel x using session A_1, …, A_n provided along channels x_1, …, x_n."

Inference rule:

$$\text{premise} \quad \frac{\Delta' \vdash Q :: (x : A')}{\Delta \vdash P; Q :: (x : A)} \quad \text{conclusion}$$

↑ bottom-up reading

Left and right rules:

client

provider

$$\frac{\Delta', x : B \vdash Q :: (z : C)}{\Delta, x : A \diamond B \vdash P; Q :: (z : C)} \diamond_L \qquad \frac{\Delta' \vdash Q :: (x : B)}{\Delta \vdash P; Q :: (x : A \diamond B)} \diamond_R$$

Multiplicative conjunction - channel output

Multiplicative conjunction - channel output

$$\frac{\vdash\ P :: (x : \quad\ )}{\vdash\ \mathsf{send}\ x\ y; P :: (x : A \otimes B)}\ \otimes_R$$

Multiplicative conjunction - channel output

$$\frac{\vdash\ P :: (x : B)}{\vdash\ \mathsf{send}\ x\ y; P :: (x : A \otimes B)}\ \otimes_R$$

Multiplicative conjunction - channel output

$$\frac{\vdash\ P :: (x : B)}{\Delta, y : A \vdash \mathsf{send}\ x\ y; P :: (x : A \otimes B)} \otimes_R$$

Multiplicative conjunction - channel output

$$\frac{\Delta \vdash P :: (x : B)}{\Delta, y : A \vdash \mathsf{send}\ x\ y; P :: (x : A \otimes B)} \otimes_R$$

Multiplicative conjunction - channel output

we have lost y!

$$\frac{\Delta \vdash P :: (x : B)}{\Delta, y : A \vdash \mathsf{send}\ x\ y; P :: (x : A \otimes B)} \otimes_R$$

Multiplicative conjunction - channel output

$$\frac{\Delta \vdash P :: (x : B)}{\Delta, y : A \vdash \mathsf{send}\ x\ y; P :: (x : A \otimes B)} \otimes_R$$

$$\frac{\Delta, \qquad\qquad\qquad \vdash Q_y :: (z : C)}{\Delta, x : A \otimes B \vdash y \leftarrow \mathsf{recv}\ x; Q_y :: (z : C)} \otimes_L$$

Multiplicative conjunction - channel output

$$\frac{\Delta \vdash P :: (x : B)}{\Delta, y : A \vdash \mathsf{send}\ x\ y; P :: (x : A \otimes B)} \otimes_R$$

$$\frac{\Delta, x : B \qquad\qquad \vdash Q_y :: (z : C)}{\Delta, x : A \otimes B \vdash y \leftarrow \mathsf{recv}\ x; Q_y :: (z : C)} \otimes_L$$

Multiplicative conjunction - channel output

$$\frac{\Delta \vdash P :: (x : B)}{\Delta, y : A \vdash \mathsf{send}\ x\ y; P :: (x : A \otimes B)} \otimes_R$$

$$\frac{\Delta, x : B, y : A \vdash Q_y :: (z : C)}{\Delta, x : A \otimes B \vdash y \leftarrow \mathsf{recv}\ x; Q_y :: (z : C)} \otimes_L$$

Multiplicative implication - channel input

Multiplicative implication - channel input

$$\frac{\Delta \qquad\qquad\qquad \vdash P_y :: (x : \qquad)}{\Delta \vdash y \leftarrow \mathsf{recv}\ x; P_y :: (x : A \multimap B)} \multimap_R$$

Multiplicative implication - channel input

$$\frac{\Delta \qquad\qquad \vdash P_y :: (x : B)}{\Delta \vdash y \leftarrow \mathsf{recv}\ x; P_y :: (x : A \multimap B)} \multimap_R$$

Multiplicative implication - channel input

$$\frac{\Delta, y : A \vdash P_y :: (x : B)}{\Delta \vdash y \leftarrow \mathsf{recv}\ x; P_y :: (x : A \multimap B)} \multimap_R$$

Multiplicative implication - channel input

$$\frac{\Delta, y : A \vdash P_y :: (x : B)}{\Delta \vdash y \leftarrow \mathsf{recv}\ x; P_y :: (x : A \multimap B)} \multimap_R$$

$$\frac{\Delta, x : \quad \vdash Q :: (z : C)}{\Delta, x : A \multimap B \quad\quad \vdash \mathsf{send}\ x\ y; Q :: (z : C)} \multimap_L$$

Multiplicative implication - channel input

$$\frac{\Delta, y : A \vdash P_y :: (x : B)}{\Delta \vdash y \leftarrow \mathsf{recv}\ x; P_y :: (x : A \multimap B)} \multimap_R$$

$$\frac{\Delta, x : B \vdash Q :: (z : C)}{\Delta, x : A \multimap B \quad \vdash \mathsf{send}\ x\ y; Q :: (z : C)} \multimap_L$$

Multiplicative implication - channel input

$$\frac{\Delta, y : A \vdash P_y :: (x : B)}{\Delta \vdash y \leftarrow \textsf{recv}\ x; P_y :: (x : A \multimap B)} \multimap_R$$

$$\frac{\Delta, x : B \vdash Q :: (z : C)}{\Delta, x : A \multimap B, y : A \vdash \textsf{send}\ x\ y; Q :: (z : C)} \multimap_L$$

Multiplicative implication - channel input

$$\frac{\Delta, y : A \vdash P_y :: (x : B)}{\Delta \vdash y \leftarrow \mathsf{recv}\ x; P_y :: (x : A \multimap B)} \multimap_R$$

we have lost y!

$$\frac{\Delta, x : B \vdash Q :: (z : C)}{\Delta, x : A \multimap B, y : A \vdash \mathsf{send}\ x\ y; Q :: (z : C)} \multimap_L$$

Additive disjunction — internal choice

Additive disjunction — internal choice

$$\frac{\quad \vdash P :: (x : \quad\ )}{\Delta \vdash x.\mathsf{inl}; P :: (x : A \oplus B)} \oplus_{R_1}$$

$$\frac{\quad \vdash P :: (x : \quad\ )}{\Delta \vdash x.\mathsf{inr}; P :: (x : A \oplus B)} \oplus_{R_2}$$

Additive disjunction — internal choice

$$\frac{\vdash\; P :: (x : A)}{\Delta \;\vdash\; x.\mathsf{inl}; P :: (x : A \oplus B)}\;\oplus_{R_1}$$

$$\frac{\vdash\; P :: (x : \quad\;)}{\Delta \;\vdash\; x.\mathsf{inr}; P :: (x : A \oplus B)}\;\oplus_{R_2}$$

Additive disjunction — internal choice

$$\frac{\vdash\ P :: (x : A)}{\Delta \vdash\ x.\mathsf{inl}; P :: (x : A \oplus B)} \oplus_{R_1}$$

$$\frac{\vdash\ P :: (x : B)}{\Delta \vdash\ x.\mathsf{inr}; P :: (x : A \oplus B)} \oplus_{R_2}$$

Additive disjunction — internal choice

$$\frac{\Delta \vdash P :: (x : A)}{\Delta \vdash x.\mathsf{inl}; P :: (x : A \oplus B)} \oplus_{R_1}$$

$$\frac{\Delta \vdash P :: (x : B)}{\Delta \vdash x.\mathsf{inr}; P :: (x : A \oplus B)} \oplus_{R_2}$$

Additive disjunction — internal choice

$$\frac{\Delta \vdash P :: (x : A)}{\Delta \vdash x.\mathsf{inl}; P :: (x : A \oplus B)} \oplus_{R_1}$$

$$\frac{\Delta \vdash P :: (x : B)}{\Delta \vdash x.\mathsf{inr}; P :: (x : A \oplus B)} \oplus_{R_2}$$

$$\frac{\Delta, x : \quad \vdash Q_1 :: (z : C) \qquad \Delta, x : \quad \vdash Q_2 :: (z : C)}{\Delta, x : A \oplus B \vdash \mathsf{case}\, x\, \mathsf{of}(Q_1, Q_2) :: (z : C)} \oplus_L$$

Additive disjunction — internal choice

$$\frac{\Delta \vdash P :: (x : A)}{\Delta \vdash x.\mathsf{inl}; P :: (x : A \oplus B)} \oplus_{R_1}$$

$$\frac{\Delta \vdash P :: (x : B)}{\Delta \vdash x.\mathsf{inr}; P :: (x : A \oplus B)} \oplus_{R_2}$$

$$\frac{\Delta, x : A \vdash Q_1 :: (z : C) \qquad \Delta, x : \quad \vdash Q_2 :: (z : C)}{\Delta, x : A \oplus B \vdash \mathsf{case}\, x\, \mathsf{of}\,(Q_1, Q_2) :: (z : C)} \oplus_{L}$$

Additive disjunction — internal choice

$$\frac{\Delta \vdash P :: (x : A)}{\Delta \vdash x.\mathsf{inl}; P :: (x : A \oplus B)} \oplus_{R_1}$$

$$\frac{\Delta \vdash P :: (x : B)}{\Delta \vdash x.\mathsf{inr}; P :: (x : A \oplus B)} \oplus_{R_2}$$

$$\frac{\Delta, x : A \vdash Q_1 :: (z : C) \qquad \Delta, x : B \vdash Q_2 :: (z : C)}{\Delta, x : A \oplus B \vdash \mathsf{case}\, x\, \mathsf{of}(Q_1, Q_2) :: (z : C)} \oplus_L$$

Additive disjunction — internal choice

$$\frac{\Delta \vdash P :: (x : A)}{\Delta \vdash x.\mathsf{inl}; P :: (x : A \oplus B)} \oplus_{R_1}$$

$$\frac{\Delta \vdash P :: (x : B)}{\Delta \vdash x.\mathsf{inr}; P :: (x : A \oplus B)} \oplus_{R_2}$$

$$\frac{\Delta, x : A \vdash Q_1 :: (z : C) \qquad \Delta, x : B \vdash Q_2 :: (z : C)}{\Delta, x : A \oplus B \vdash \mathsf{case}\, x\, \mathsf{of}(Q_1, Q_2) :: (z : C)} \oplus_L$$

internal choice: provider chooses. Generalize to n-ary choice

Additive conjunction — external choice

Additive conjunction — external choice

$$\frac{\vdash\ P_1 :: (x : \quad) \qquad\qquad \vdash\ P_2 :: (x : \quad)}{\Delta\ \vdash\ \mathsf{case}\, x\, \mathsf{of}\, (P_1, P_2) :: (x : A \,\&\, B)}\ \&_R$$

Additive conjunction — external choice

$$\frac{\vdash\ P_1 :: (x : A) \qquad\qquad \vdash\ P_2 :: (x : \quad)}{\Delta \vdash \mathsf{case}\, x\, \mathsf{of}\, (P_1, P_2) :: (x : A \,\&\, B)}\ \&_R$$

Additive conjunction — external choice

$$\frac{\vdash\ P_1 :: (x : A) \qquad\qquad \vdash\ P_2 :: (x : B)}{\Delta\ \vdash\ \mathsf{case}\, x\, \mathsf{of}\, (P_1, P_2) :: (x : A \,\&\, B)}\ \&_R$$

Additive conjunction — external choice

$$\frac{\Delta \vdash P_1 :: (x : A) \qquad \Delta \vdash P_2 :: (x : B)}{\Delta \vdash \mathsf{case}\, x\, \mathsf{of}\, (P_1, P_2) :: (x : A \,\&\, B)} \;\&_R$$

Additive conjunction — external choice

$$\frac{\Delta \vdash P_1 :: (x : A) \qquad \Delta \vdash P_2 :: (x : B)}{\Delta \vdash \mathsf{case}\, x \,\mathsf{of}\, (P_1, P_2) :: (x : A \,\&\, B)} \;\&_R$$

$$\frac{\Delta, x : \quad\;\; \vdash Q :: (z : C)}{\Delta, x : A \,\&\, B \vdash x.\mathsf{inl}; Q :: (z : C)} \;\&_{L_1}$$

$$\frac{\Delta, x : \quad\;\; \vdash Q :: (z : C)}{\Delta, x : A \,\&\, B \vdash x.\mathsf{inr}; Q :: (z : C)} \;\&_{L_2}$$

Additive conjunction — external choice

$$\frac{\Delta \vdash P_1 :: (x : A) \qquad \Delta \vdash P_2 :: (x : B)}{\Delta \vdash \mathsf{case}\, x\, \mathsf{of}\, (P_1, P_2) :: (x : A \mathbin{\&} B)} \;\&_R$$

$$\frac{\Delta, x : A \vdash Q :: (z : C)}{\Delta, x : A \mathbin{\&} B \vdash x.\mathsf{inl}; Q :: (z : C)} \;\&_{L_1}$$

$$\frac{\Delta, x : \quad \vdash Q :: (z : C)}{\Delta, x : A \mathbin{\&} B \vdash x.\mathsf{inr}; Q :: (z : C)} \;\&_{L_2}$$

Additive conjunction — external choice

$$\frac{\Delta \vdash P_1 :: (x : A) \qquad \Delta \vdash P_2 :: (x : B)}{\Delta \vdash \mathsf{case}\, x\, \mathsf{of}\, (P_1, P_2) :: (x : A \mathbin{\&} B)} \;\&_R$$

$$\frac{\Delta, x : A \vdash Q :: (z : C)}{\Delta, x : A \mathbin{\&} B \vdash x.\mathsf{inl}; Q :: (z : C)} \;\&_{L_1}$$

$$\frac{\Delta, x : B \vdash Q :: (z : C)}{\Delta, x : A \mathbin{\&} B \vdash x.\mathsf{inr}; Q :: (z : C)} \;\&_{L_2}$$

Additive conjunction — external choice

$$\frac{\Delta \vdash P_1 :: (x : A) \qquad \Delta \vdash P_2 :: (x : B)}{\Delta \vdash \textsf{case}\, x\, \textsf{of}\, (P_1, P_2) :: (x : A \,\&\, B)} \,\&_R$$

$$\frac{\Delta, x : A \vdash Q :: (z : C)}{\Delta, x : A \,\&\, B \vdash x.\textsf{inl}; Q :: (z : C)} \,\&_{L_1}$$

$$\frac{\Delta, x : B \vdash Q :: (z : C)}{\Delta, x : A \,\&\, B \vdash x.\textsf{inr}; Q :: (z : C)} \,\&_{L_2}$$

→ external choice: client chooses. Generalize to n-ary choice