Session-Typed Concurrent Programming[1]

Stephanie Balzer
Carnegie Mellon University

POPL 2019 TutorialFest

1 Supported by a Mozilla Research Grant and an NSF grant (No. CCF-1718267)

Content of tutorial

Message-passing programming and session types

- types for typing such programs

Linear logic and session types

- benefits and limitations

Manifest sharing

- controlled sharing for concurrency (non-determinism)

Manifest sharing and the pi-calculus

- expressiveness of untyped asynchronous pi-calculus recovered

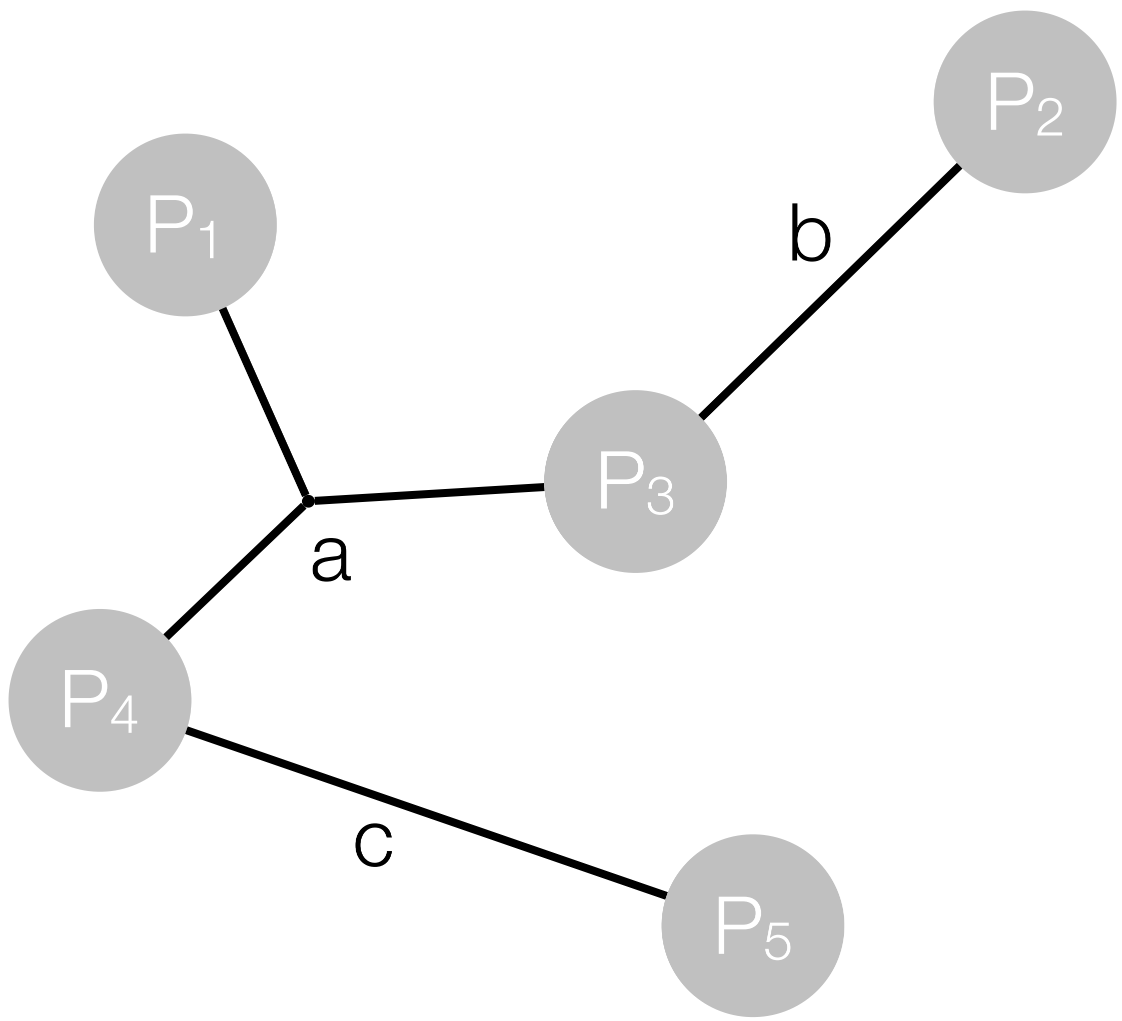
Current & future research

Learning objectives of tutorial

- How to program with session types
- What linearity is good for in programming
- How logic can guide programming language design
- Expressiveness of session-typed programming
- Hands-on experience with Concurrent C0

Session-typed message-passing programming

Message-passing concurrent programming



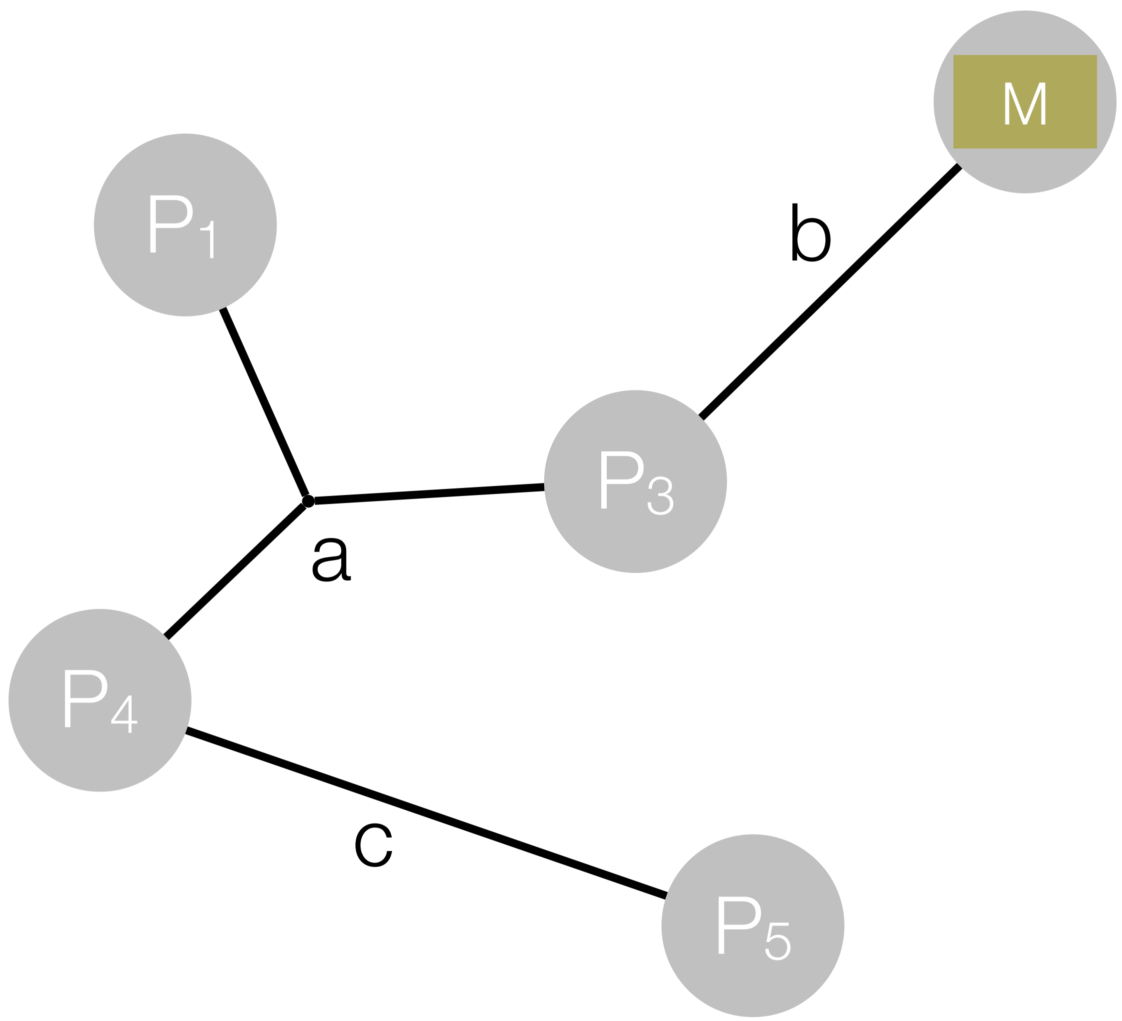
Program: network of processes connected by channels

Computation: by message exchange along channels

N-ary channels: e.g., a connects P_1, P_2, and P_3

Legend: process — channel — message

Message-passing concurrent programming



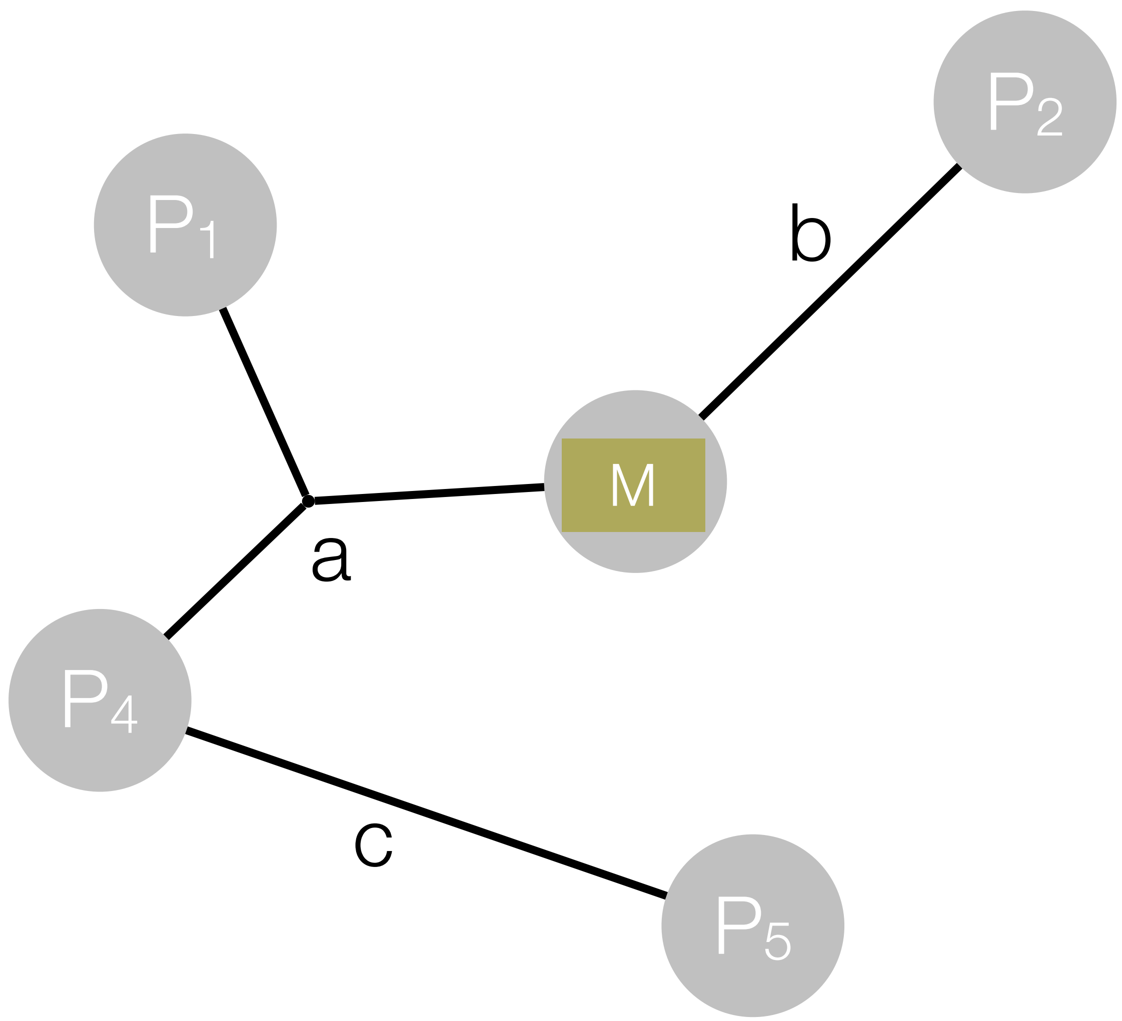
Program: network of processes connected by channels

Computation: by message exchange along channels

N-ary channels: e.g., a connects P_1, P_2, and P_3

Legend: process — channel — message

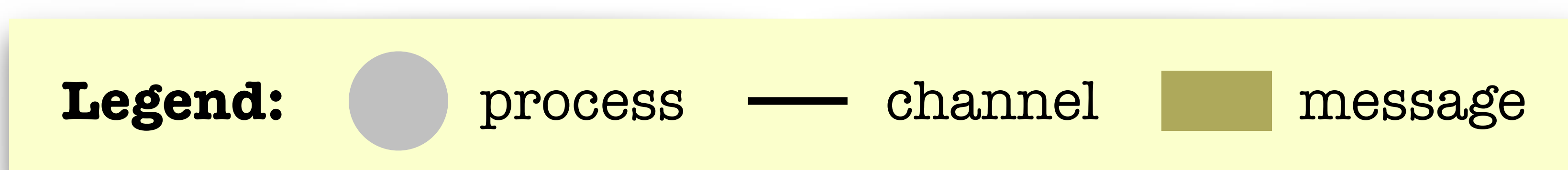
Message-passing concurrent programming



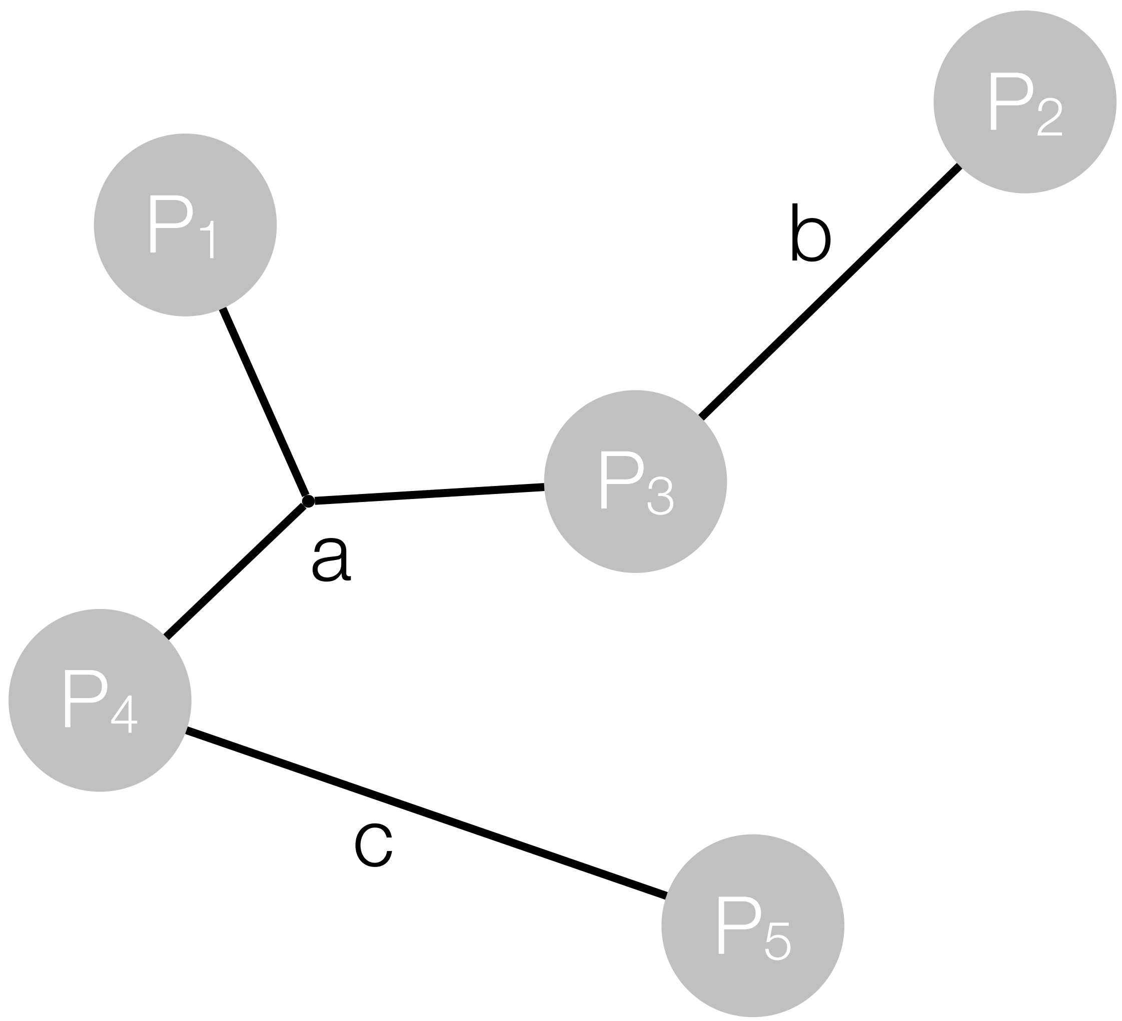
Program: network of processes connected by channels

Computation: by message exchange along channels

N-ary channels: e.g., a connects P_1, P_2, and P_3



Message-passing concurrent programming



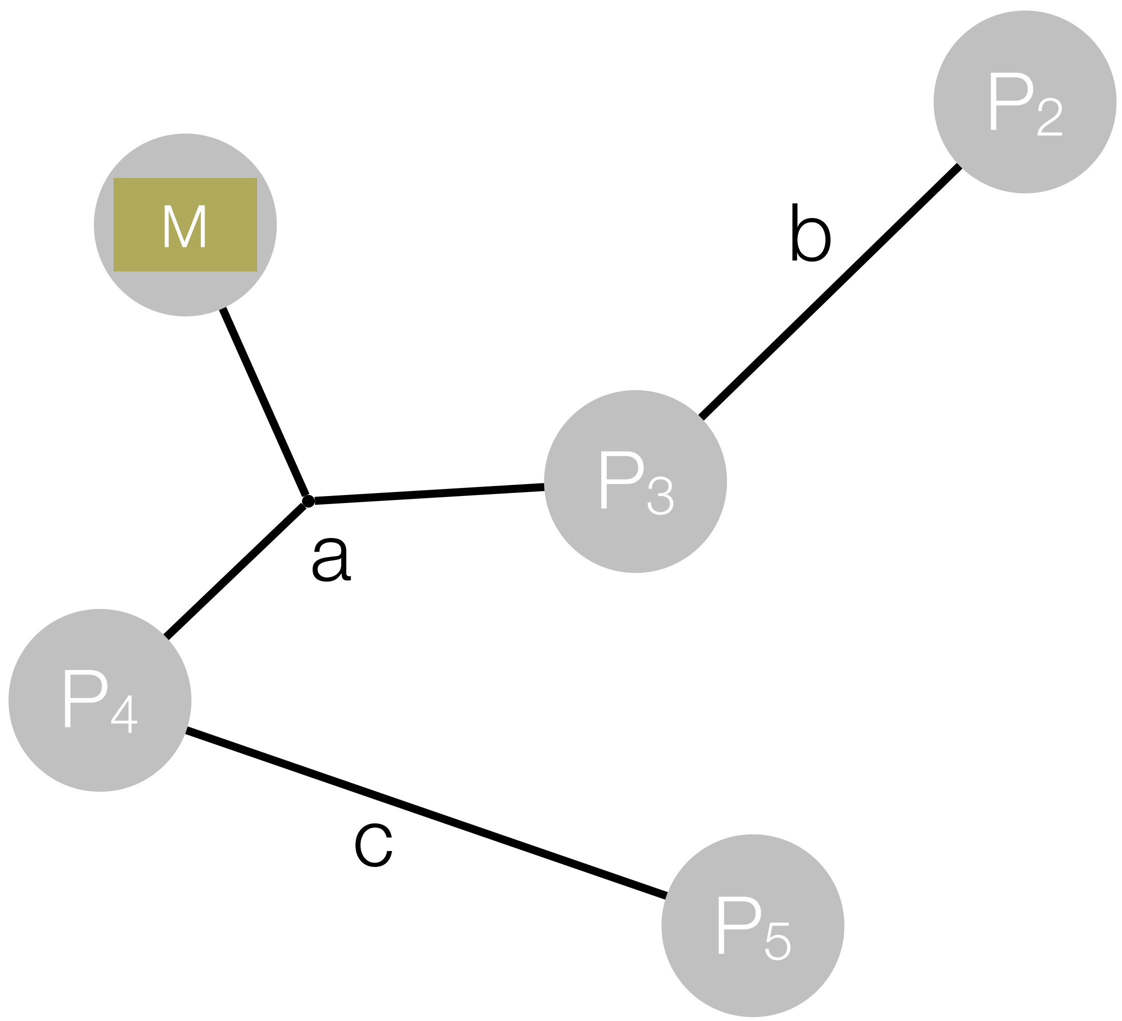
Program: network of processes connected by channels

Computation: by message exchange along channels

N-ary channels: e.g., a connects P_1, P_2, and P_3

Legend: process — channel — message

Message-passing concurrent programming



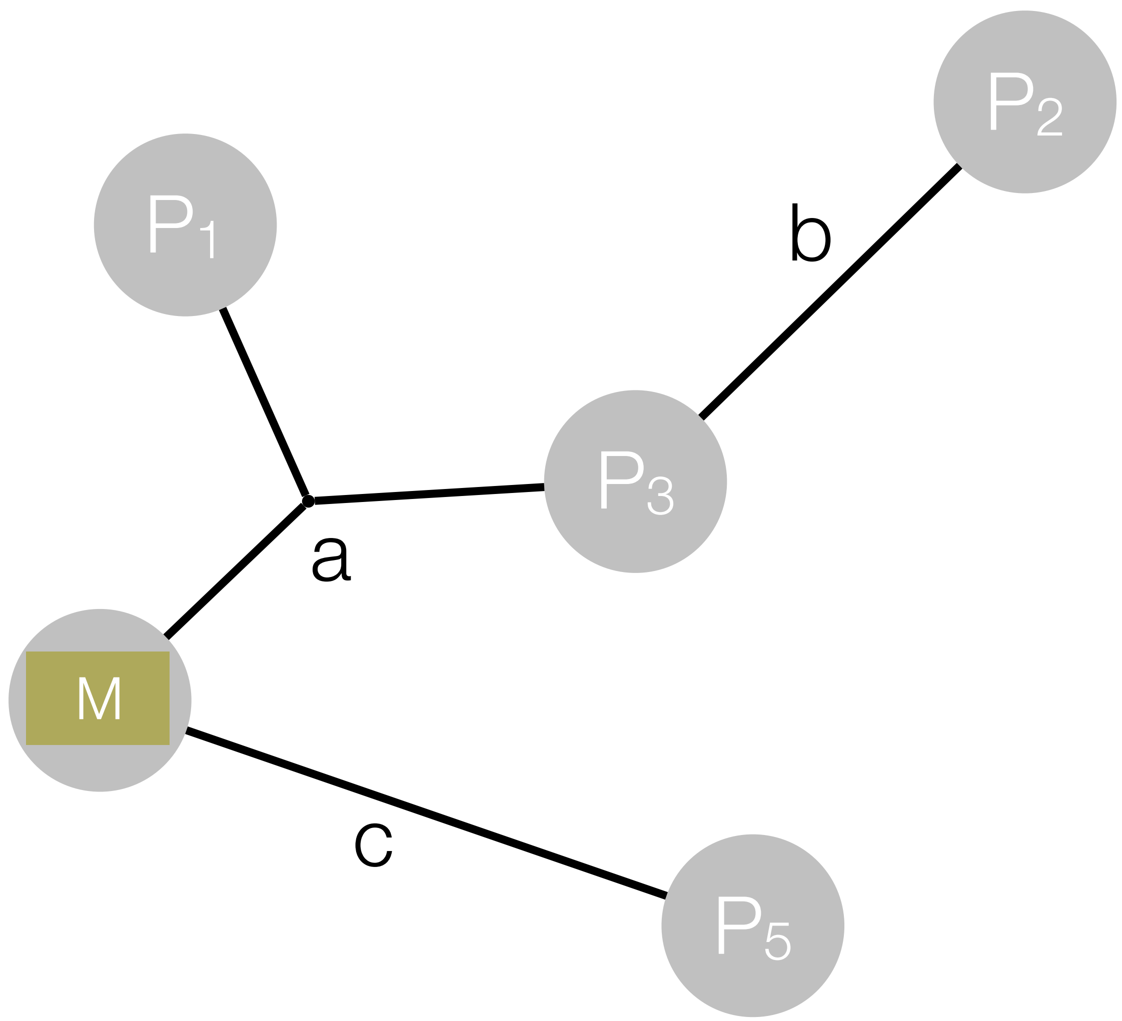
Program: network of processes connected by channels

Computation: by message exchange along channels

N-ary channels: e.g., a connects P_1, P_2, and P_3

Legend: process — channel — message

Message-passing concurrent programming



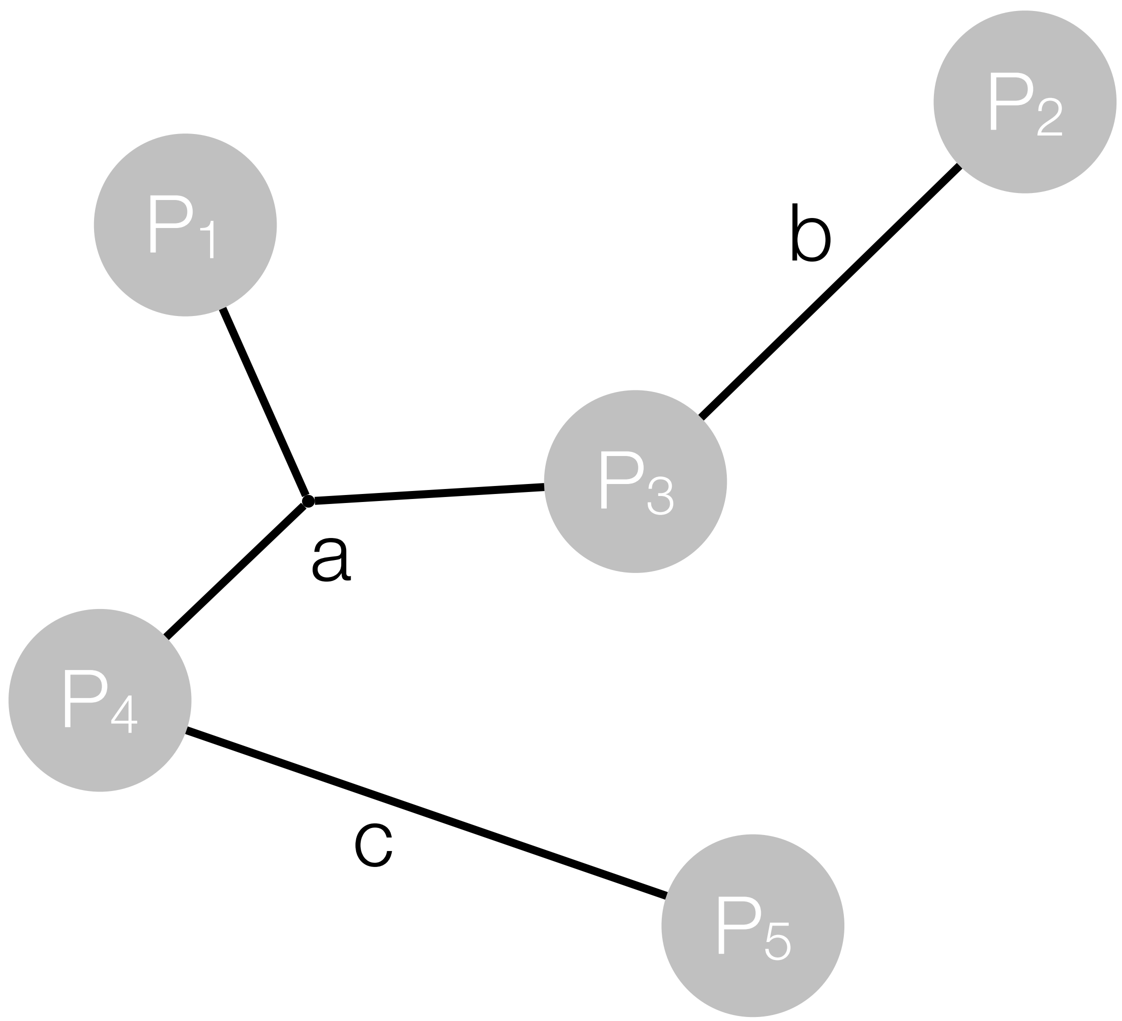
Program: network of processes connected by channels

Computation: by message exchange along channels

N-ary channels: e.g., a connects P_1, P_2, and P_3

Legend: process channel message

Message-passing concurrent programming



Program: network of processes connected by channels

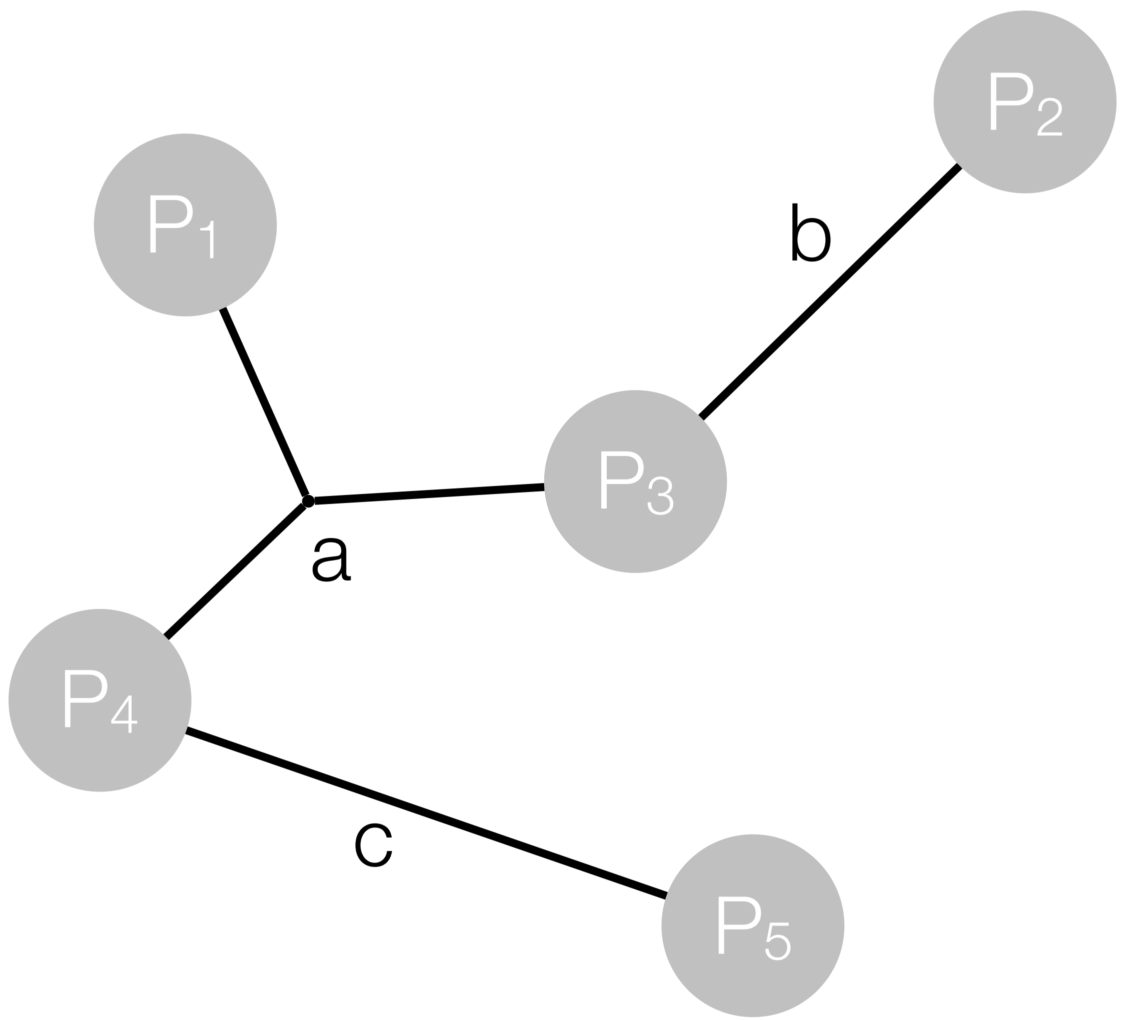
Computation: by message exchange along channels

N-ary channels: e.g., a connects P_1, P_2, and P_3

→ non-determinism

Legend: process — channel — message

Message-passing concurrent programming



Program: network of processes connected by channels

Computation: by message exchange along channels

N-ary channels: e.g., a connects P_1, P_2, and P_3

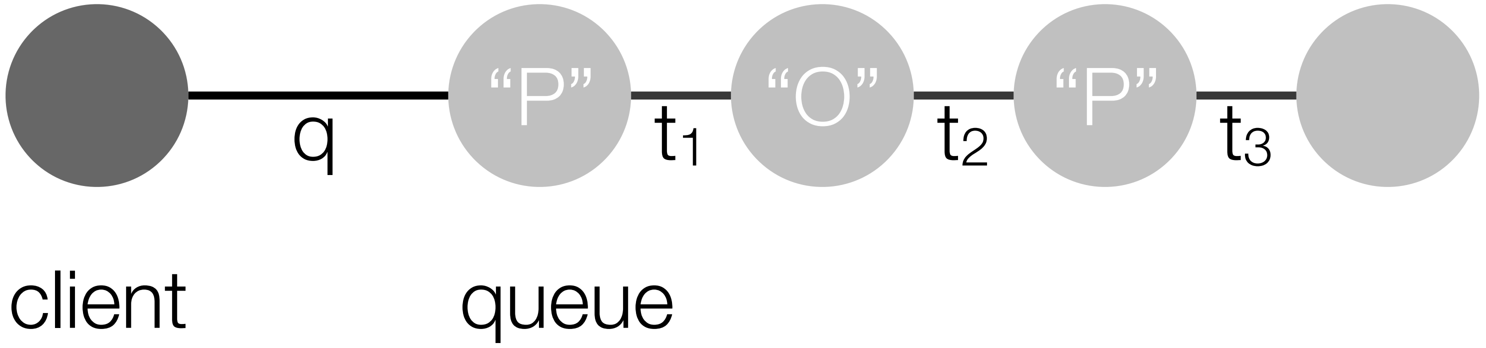
→ non-determinism

→ formal model: pi-calculus

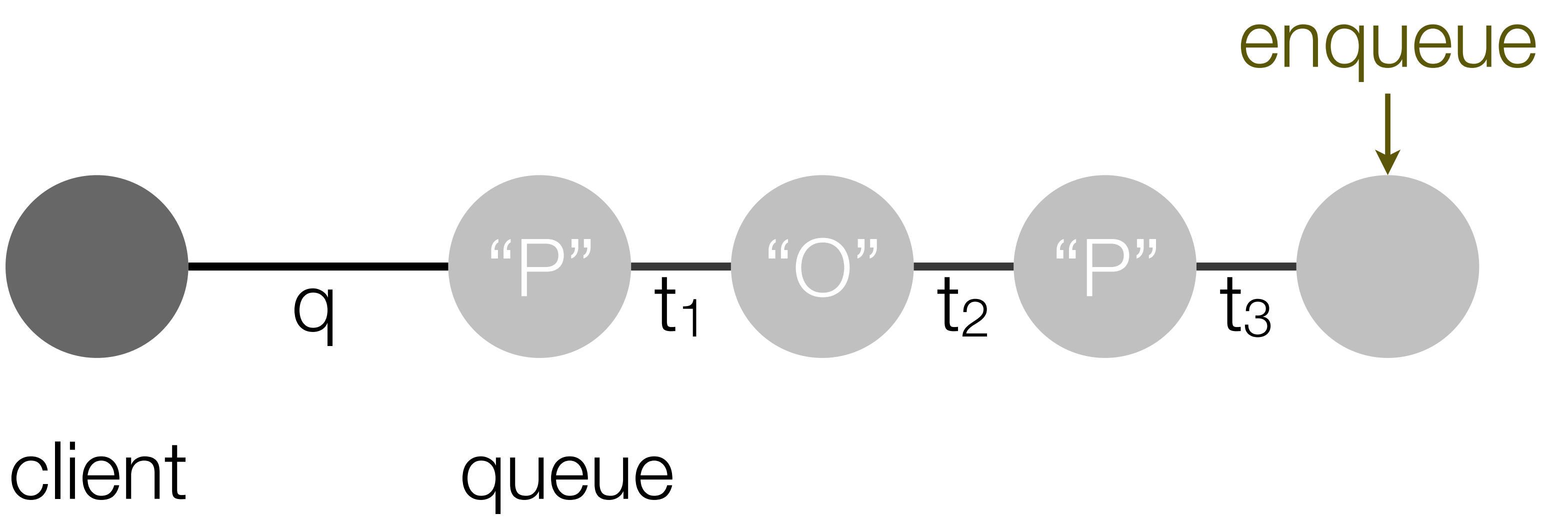
[Milner 1992]

Legend: process — channel — message

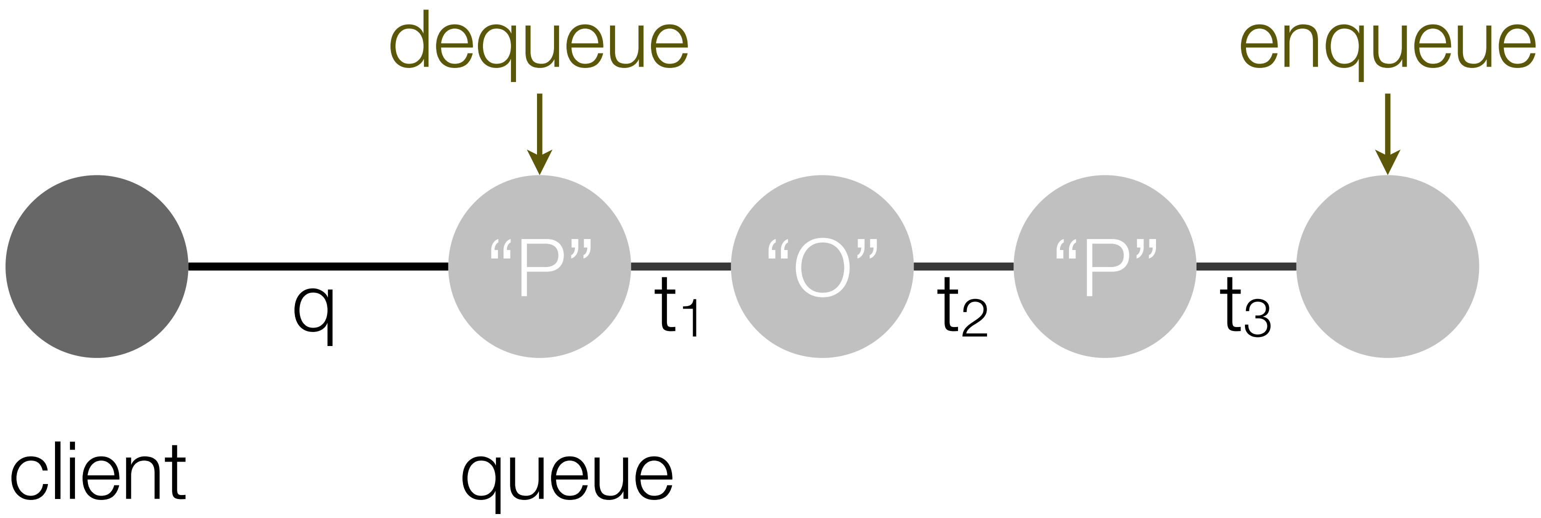
Example: queue



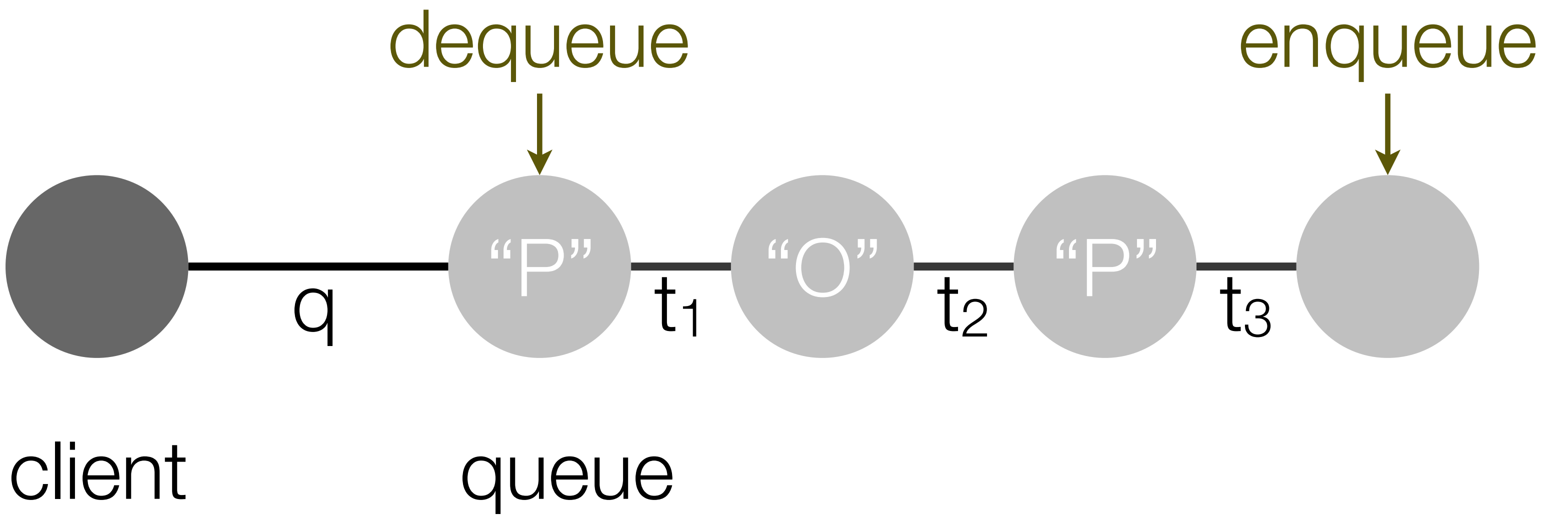
Example: queue



Example: queue

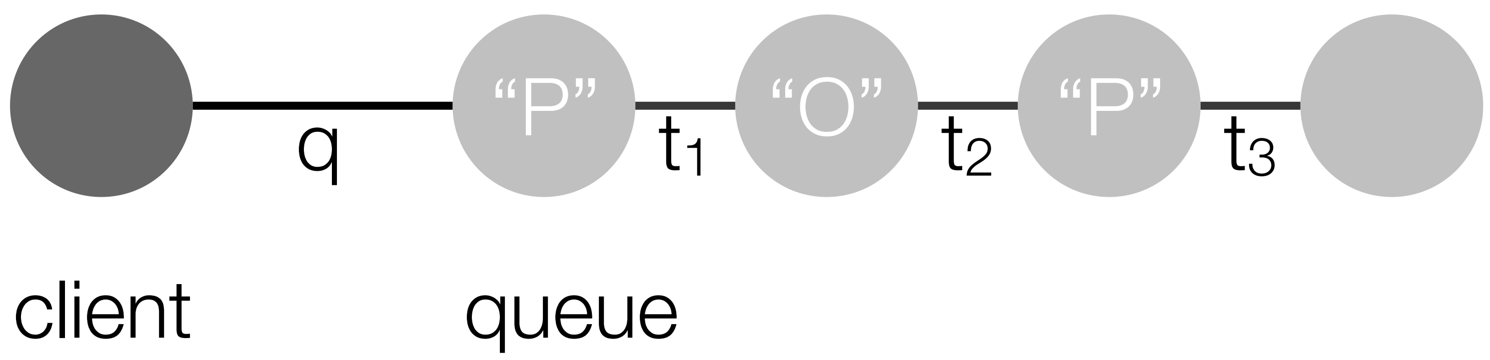


Example: queue



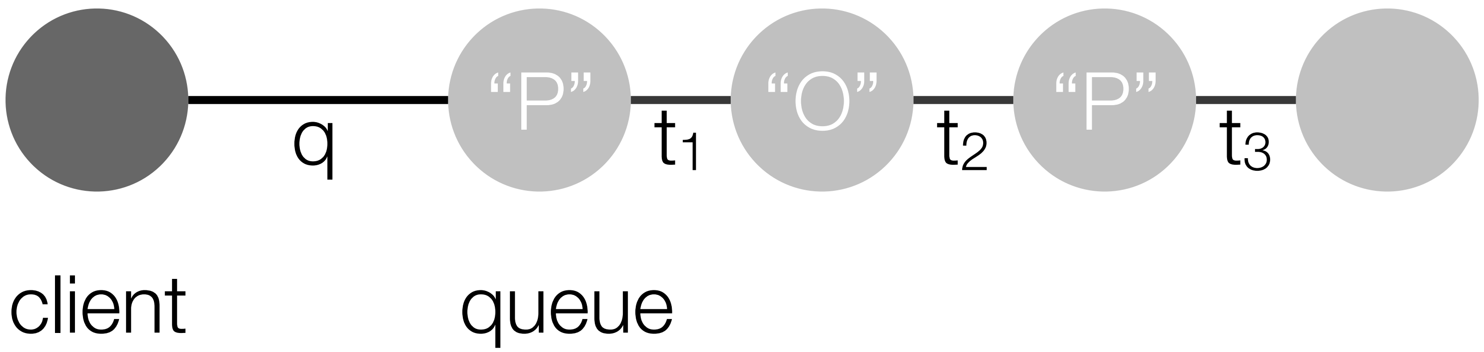
enqueue: client sends "enq" followed by "L" along q

Example: queue

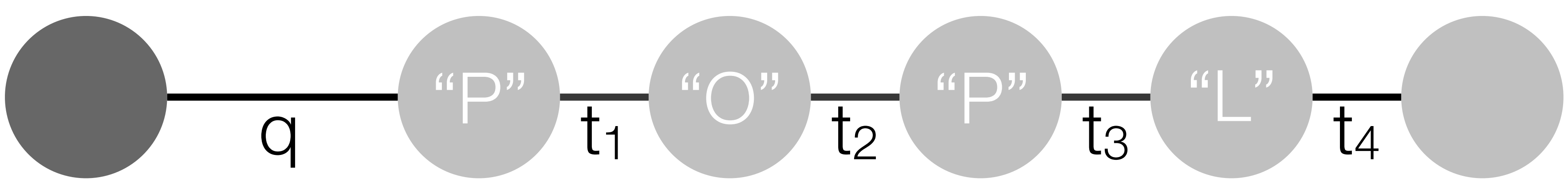


enqueue: client sends “enq” followed by “L” along q

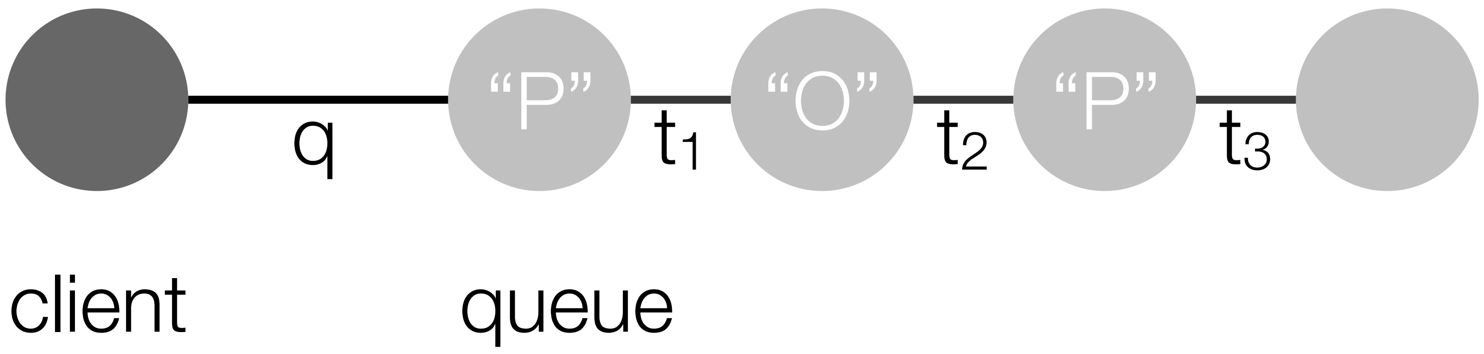
Example: queue



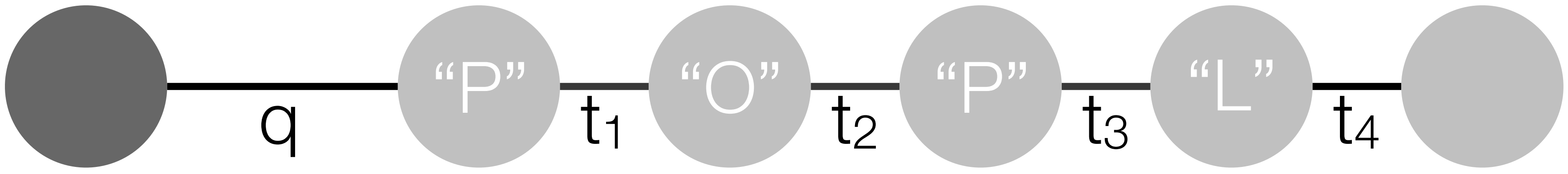
enqueue: client sends "enq" followed by "L" along q



Example: queue

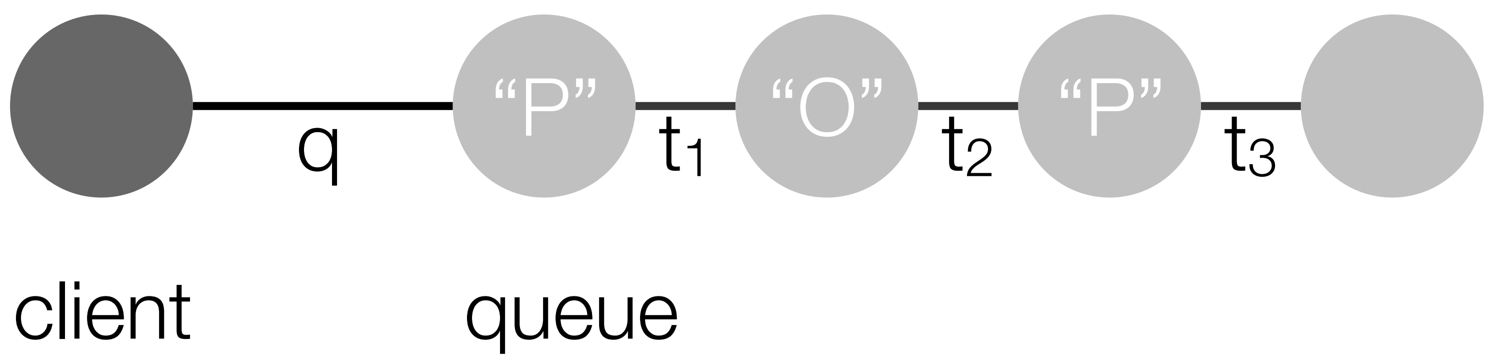


enqueue: client sends "enq" followed by "L" along q

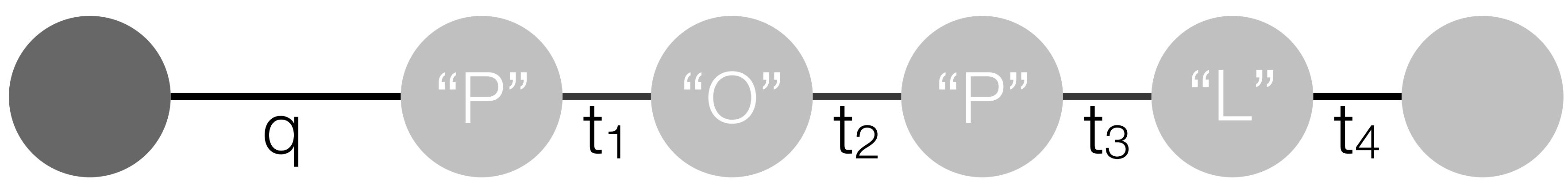


dequeue: client sends "deq", then receives "P"

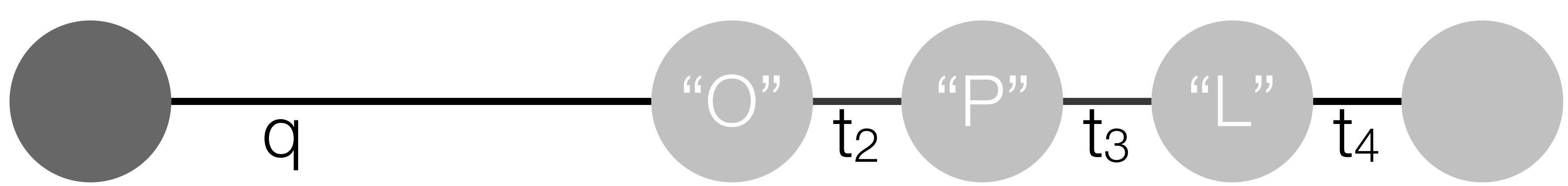
Example: queue



enqueue: client sends “enq” followed by “L” along q



dequeue: client sends “deq”, then receives “P”

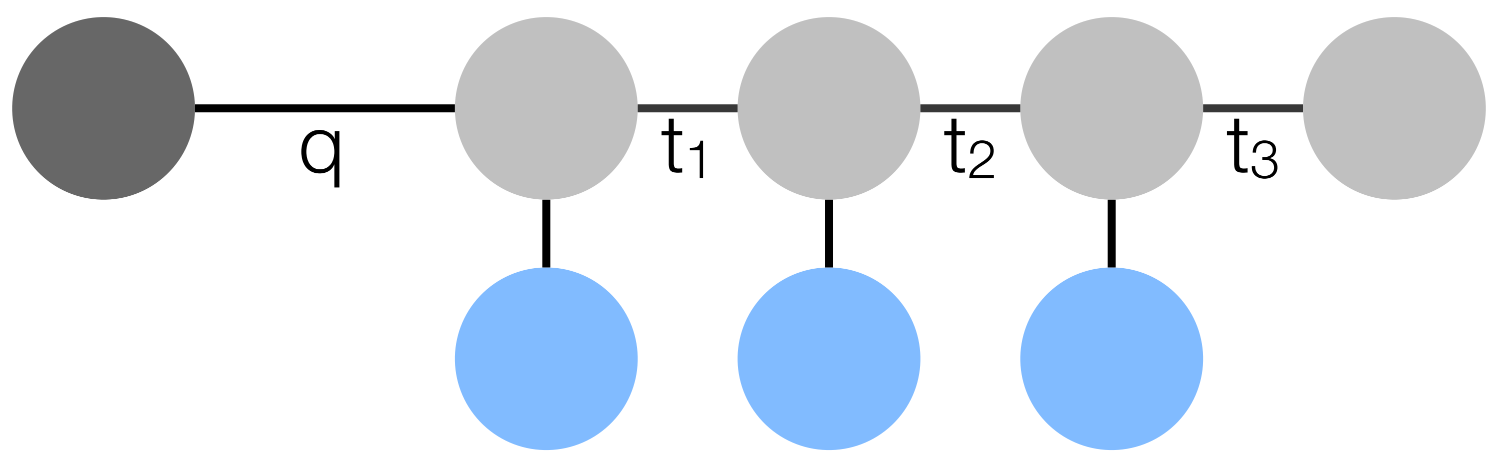


Example: queue

- In our example, we’ve stored characters in the queue
- However, we can also store channel references:

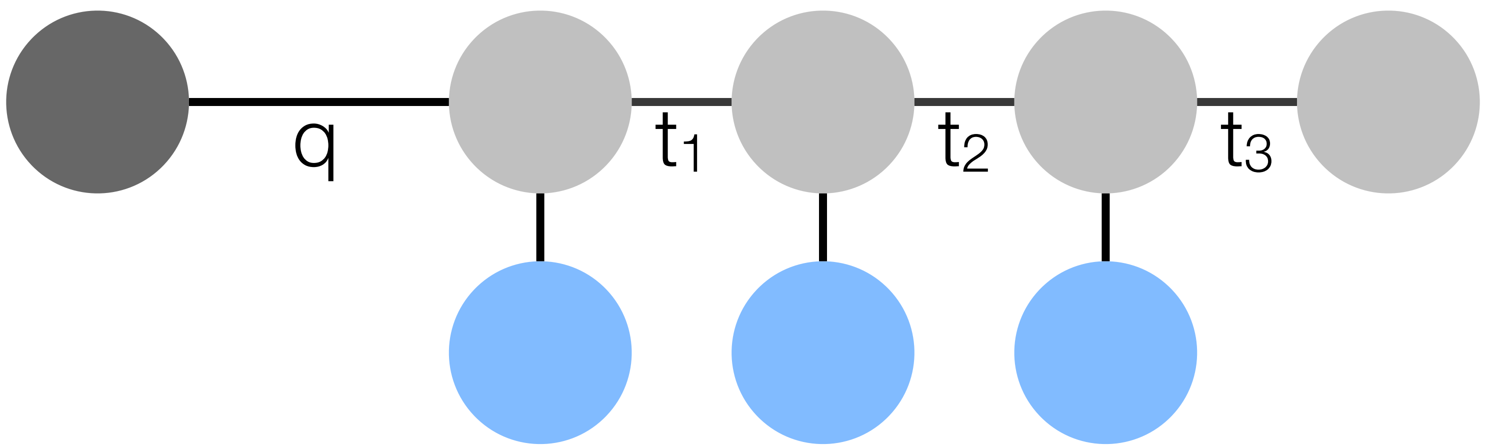
Example: queue

- In our example, we've stored characters in the queue
- However, we can also store channel references:



Example: queue

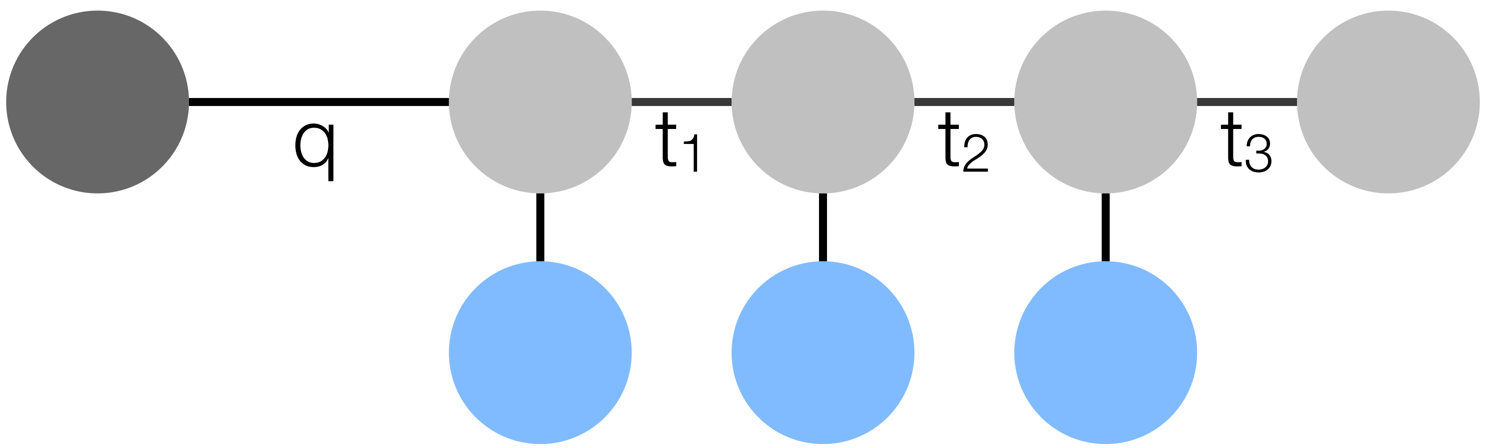
- In our example, we've stored characters in the queue
- However, we can also store channel references:



"higher-order" channels (session types)

Example: queue

- In our example, we've stored characters in the queue
- However, we can also store channel references:



→ "higher-order" channels (session types)

→ "mobility" (pi-calculus)

Types for protocols of message exchange

Types for protocols of message exchange

Session types [Honda 1993]

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

Types for protocols of message exchange

Session types [Honda 1993]

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

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

Types for protocols of message exchange

Session types [Honda 1993]

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

Types for protocols of message exchange

Session types [Honda 1993]

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

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

Types for protocols of message exchange

Session types [Honda 1993]

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

Types for protocols of message exchange

Session types [Honda 1993]

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

external choice: receive label l_i, continue as type A_i

Types for protocols of message exchange

Session types [Honda 1993]

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

Types for protocols of message exchange

Session types [Honda 1993]

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

internal choice: send label l_i, continue as type A_i

Types for protocols of message exchange

Session types [Honda 1993]

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

Types for protocols of message exchange

Session types [Honda 1993]

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

termination: close session and terminate

Types for protocols of message exchange

Session types [Honda 1993]

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

Types for protocols of message exchange

Session types [Honda 1993]

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

recursive session types

Types for protocols of message exchange

Session types [Honda 1993]

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

Types for protocols of message exchange

Session types [Honda 1993]

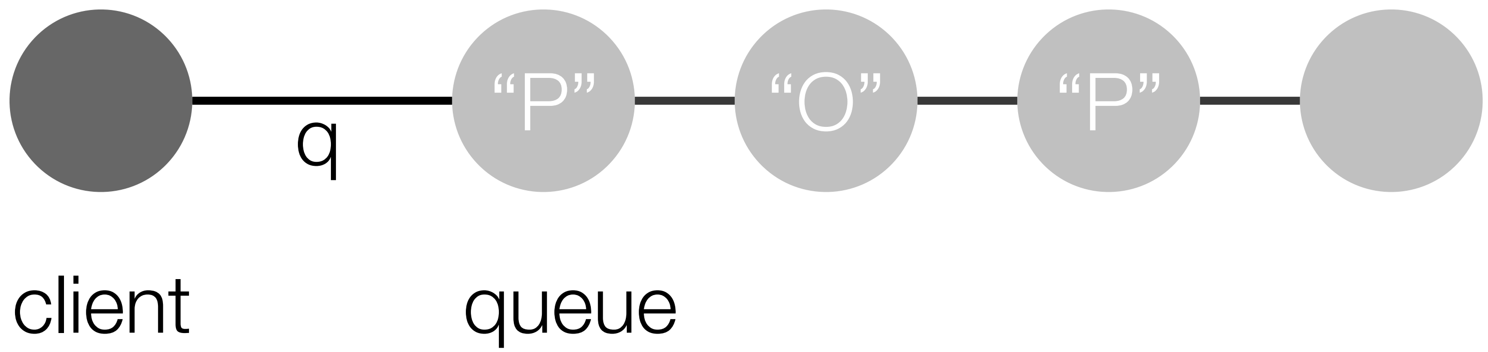
$$
\begin{array}{lcl}
A & \triangleq & ?[T].A' \mid ![T].A' \mid \\
 & & \&\{l_1 : A_1, \ldots, l_n : A_n\} \mid \oplus\{l_1 : A_1, \ldots, l_n : A_n\} \mid \\
 & & \mathsf{end} \mid X \mid \mu X.A' \\
T & \triangleq & A \mid \mathsf{int} \mid \ldots
\end{array}
$$

Example:

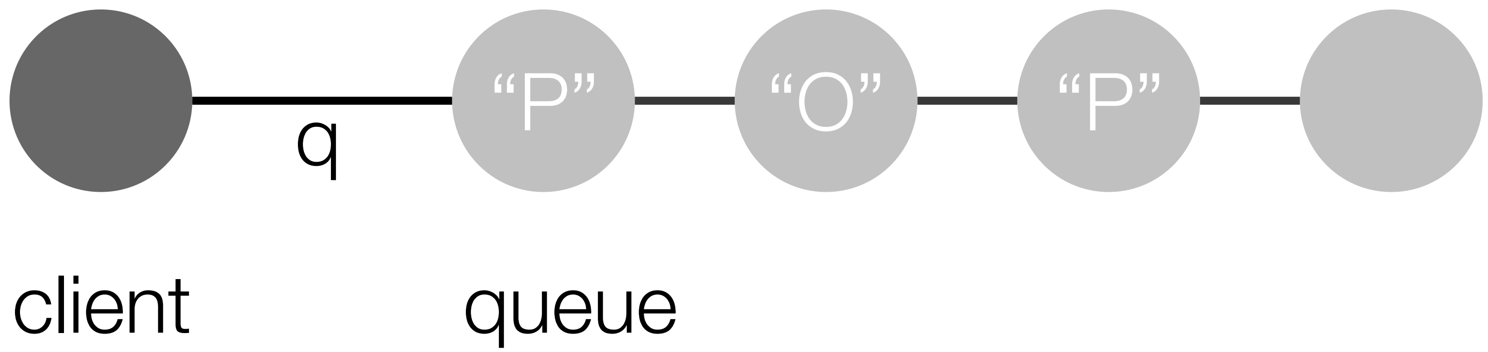
$$
\begin{aligned}
\mathsf{queue} = \&\{&\mathsf{enq} : ?[\mathsf{char}].\mathsf{queue}, \\
&\mathsf{deq} : \oplus\{\mathsf{none} : \mathsf{end}, \mathsf{some} : ![\mathsf{char}].\mathsf{queue}\}\}
\end{aligned}
$$

Session types change with protocol

Session types change with protocol

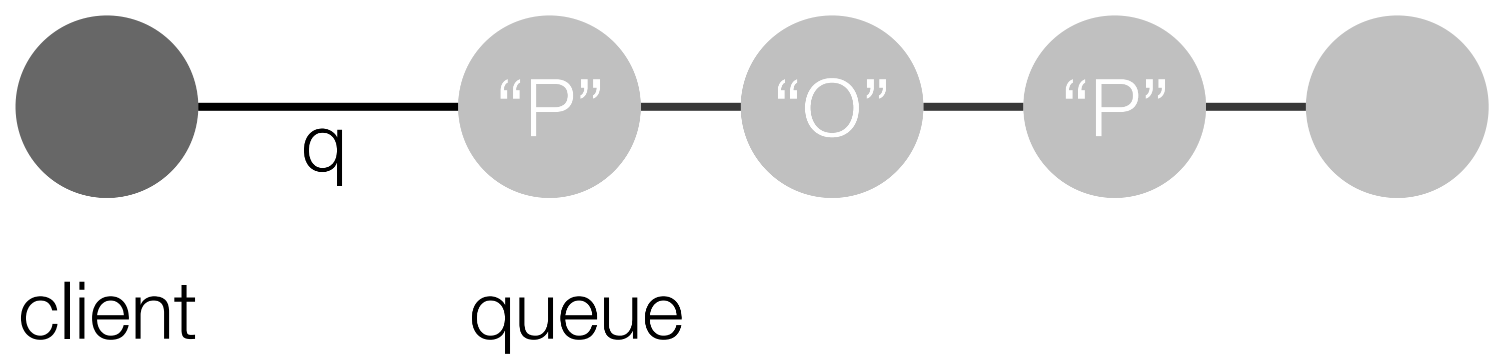


Session types change with protocol



What is the type of channel q?

Session types change with protocol



What is the type of channel q?

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

Session types change with protocol

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

Session types change with protocol

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

Action:

Type:

q: queue

Session types change with protocol

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

Action:	Type:
	q: queue
send “enq” along q	q:

Session types change with protocol

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

Action:	Type:
	q: queue
send "enq" along q	q: $?[\mathsf{char}].\mathsf{queue}$

Session types change with protocol

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

Action:	Type:
	q: queue
send "enq" along q	q: $?[\mathsf{char}].\mathsf{queue}$
send "L" along q	q:

Session types change with protocol

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

Action:	Type:
	q: queue
send “enq” along q	q: $?[\mathsf{char}].\mathsf{queue}$
send “L” along q	q: queue

Session types change with protocol

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

Action:	Type:
	q: queue
send “enq” along q	q: $?[\mathsf{char}].\mathsf{queue}$
send “L” along q	q: queue
send “deq” along q	q:

Session types change with protocol

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

Action:	Type:
	q: queue
send "enq" along q	q: $?[\mathsf{char}].\mathsf{queue}$
send "L" along q	q: queue
send "deq" along q	q: $\oplus\{\mathsf{none} : \mathsf{end}, \mathsf{some} : ![\mathsf{char}].\mathsf{queue}\}$

Session types change with protocol

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

Action:	Type:
	q: queue
send “enq” along q	q: $?[\mathsf{char}].\mathsf{queue}$
send “L” along q	q: queue
send “deq” along q	q: $\oplus\{\mathsf{none} : \mathsf{end}, \mathsf{some} : ![\mathsf{char}].\mathsf{queue}\}$
receive “some” along q	q:

Session types change with protocol

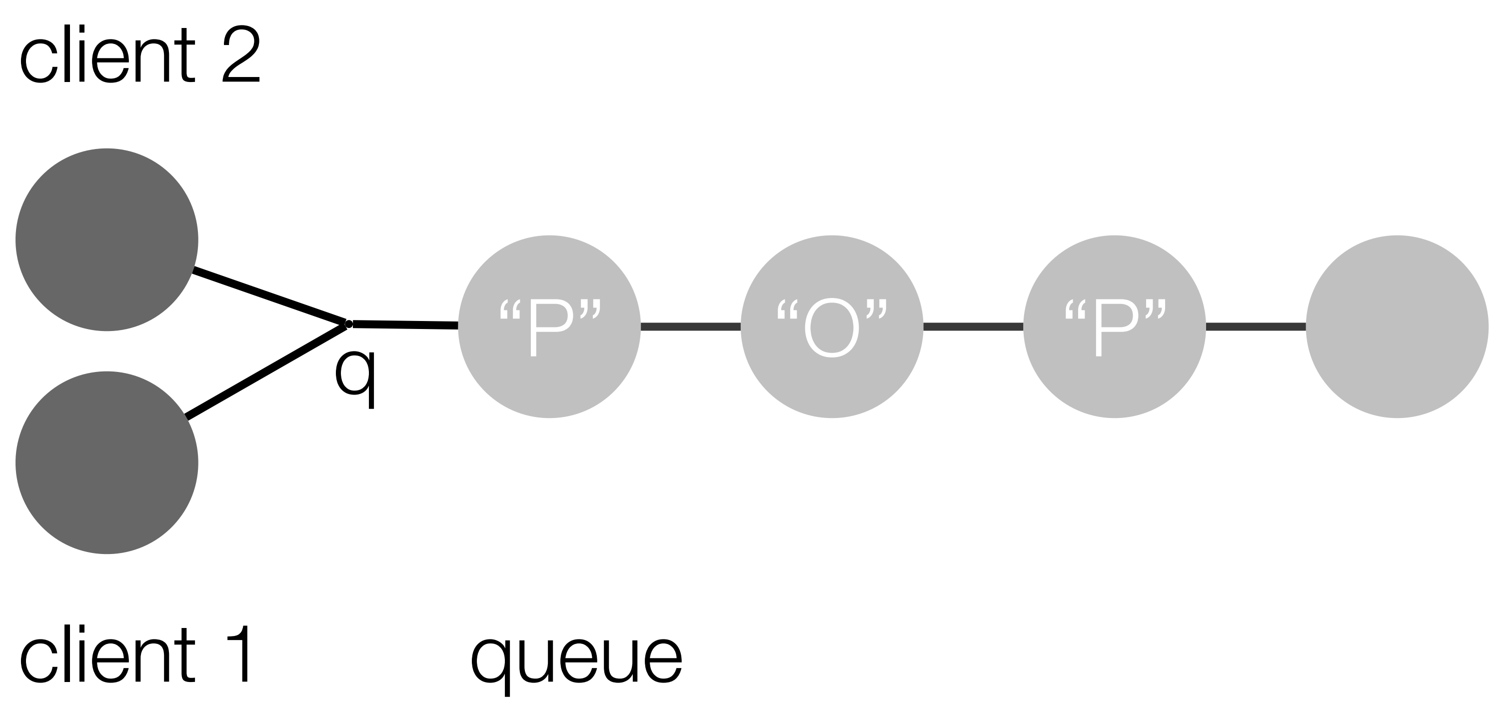
$$\begin{aligned} \mathsf{queue} = \&\{ &\mathsf{enq} : ?[\mathsf{char}].\mathsf{queue}, \\ &\mathsf{deq} : \oplus\{\mathsf{none} : \mathsf{end}, \mathsf{some} : ![\mathsf{char}].\mathsf{queue}\}\} \end{aligned}$$

Action:	Type:
	q: queue
send "enq" along q	q: $?[\mathsf{char}].\mathsf{queue}$
send "L" along q	q: queue
send "deq" along q	q: $\oplus\{\mathsf{none} : \mathsf{end}, \mathsf{some} : ![\mathsf{char}].\mathsf{queue}\}$
receive "some" along q	q: $![\mathsf{char}].\mathsf{queue}$

Session types change with protocol

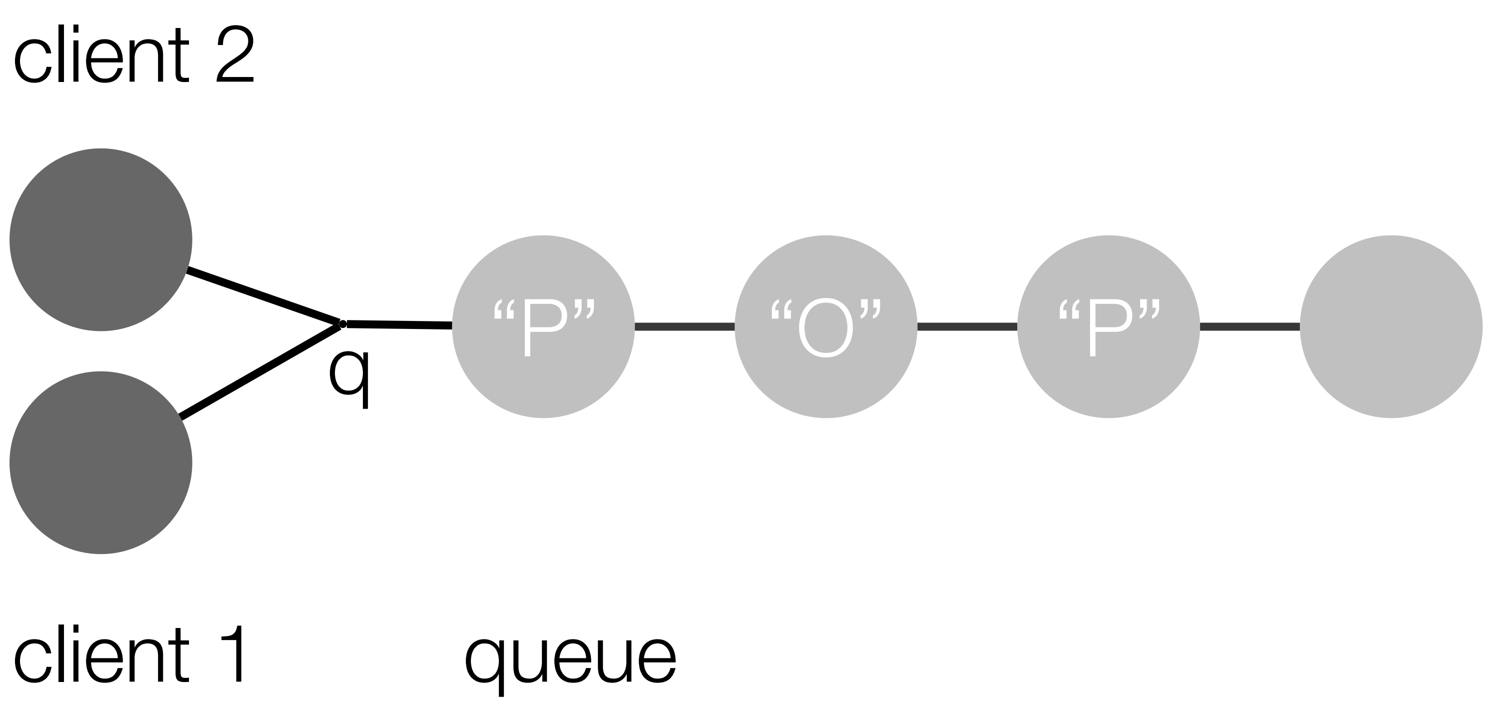
Session types change with protocol

What if?



Session types change with protocol

What if?



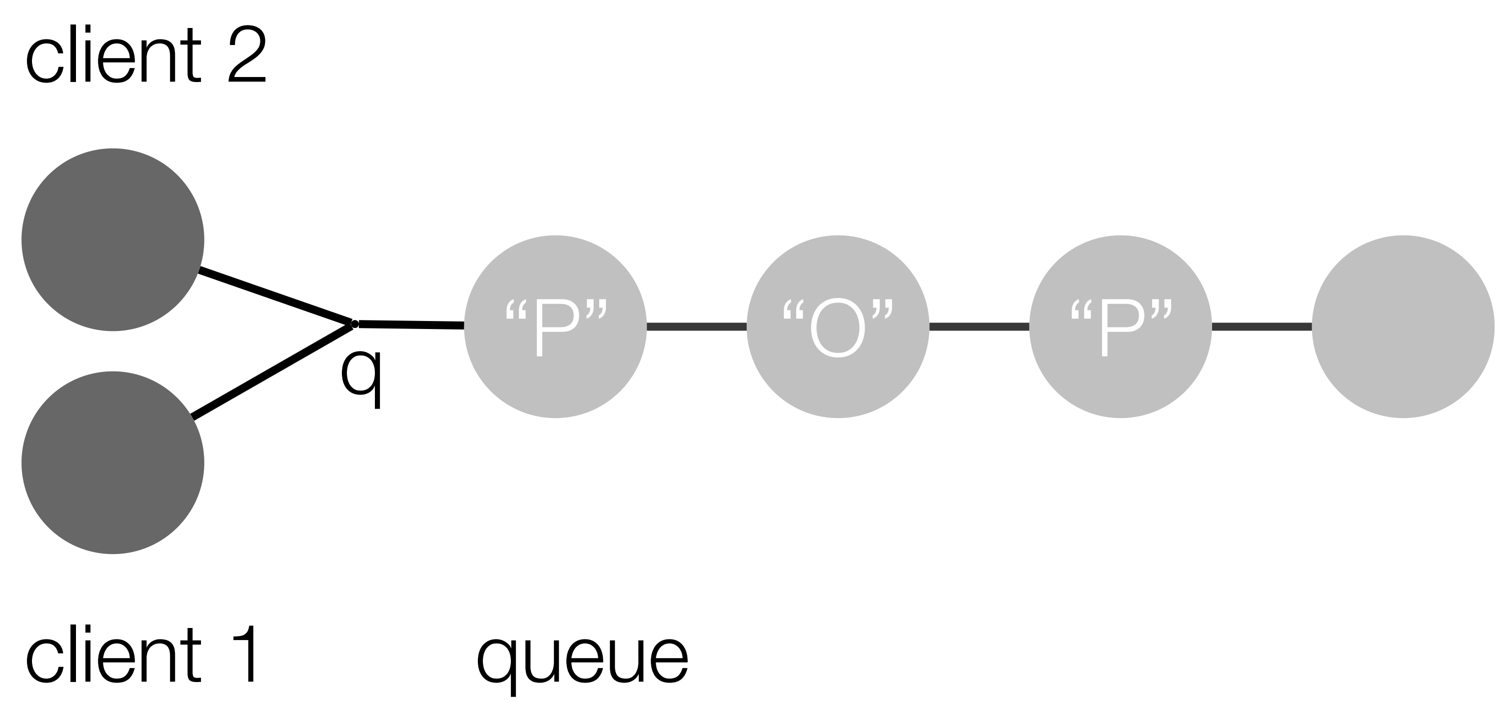
Action:

Type:

q: queue

Session types change with protocol

What if?



Action:

Type:

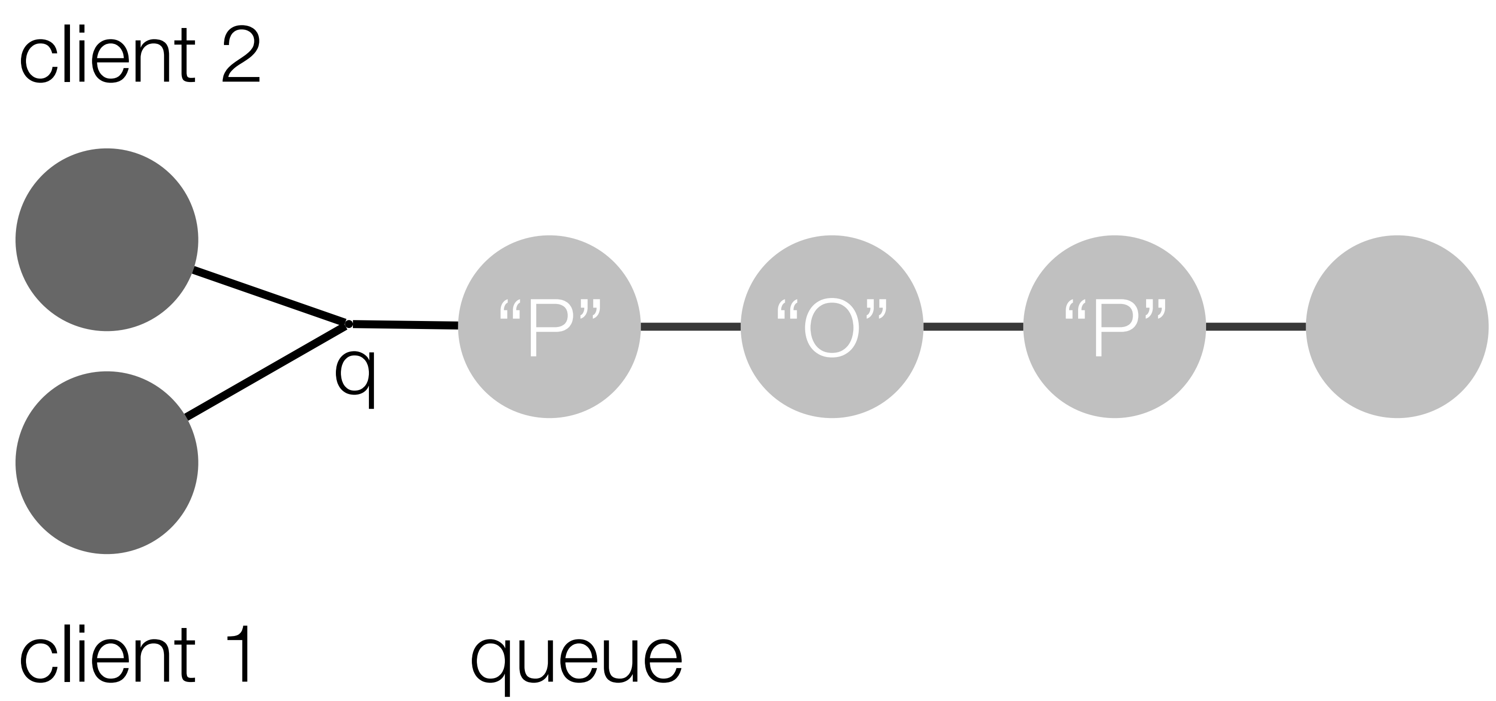
q: $\texttt{queue}$

client 1 sends “enq” along q

q:

Session types change with protocol

What if?



Action:

Type:

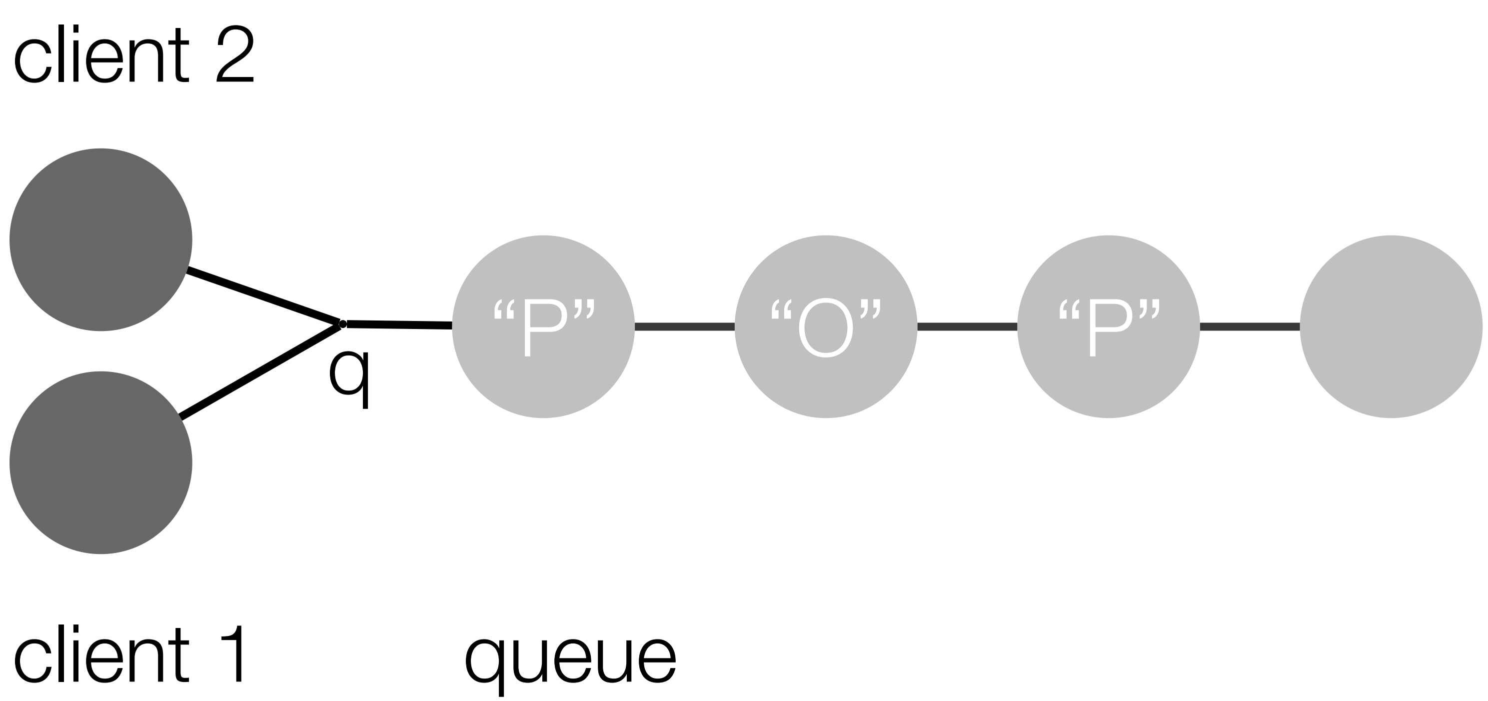
q: queue

client 1 sends "enq" along q

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

Session types change with protocol

What if?



Action:

Type:

q: queue

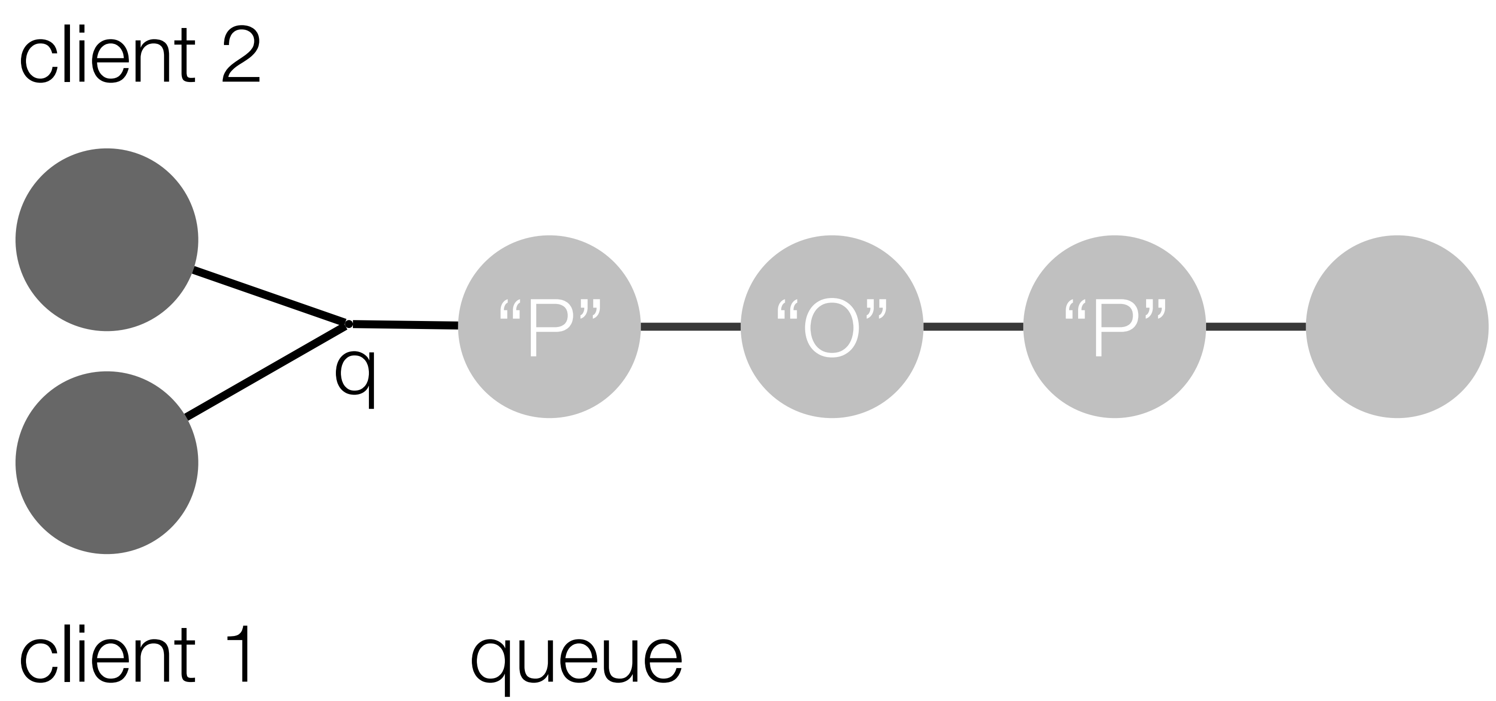
client 1 sends “enq” along q

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

client 2 sends “enq” along q

Session types change with protocol

What if?



Action:

Type:

q: queue

client 1 sends “enq” along q q: $?[\mathsf{char}].\mathsf{queue}$

client 2 sends “enq” along q q is not at expected type!

Preservation (session fidelity)

- Expectations of client and provider match, if they do initially.
- How can we recover preservation?

Preservation (session fidelity)

- Expectations of client and provider match, if they do initially.
- How can we recover preservation?

→ Use linear logic as a foundation for session types

Preservation (session fidelity)

- Expectations of client and provider match, if they do initially.
- How can we recover preservation?

→ Use linear logic as a foundation for session types

→ Let's view session types as linear propositions

Preservation (session fidelity)

- Expectations of client and provider match, if they do initially.
- How can we recover preservation?

Use linear logic as a foundation for session types

Let's view session types as linear propositions

Linear logic allows us to treat channels as "resources"

Preservation (session fidelity)

- Expectations of client and provider match, if they do initially.
- How can we recover preservation?

→ Use linear logic as a foundation for session types

→ Let’s view session types as linear propositions

→ Linear logic allows us to treat channels as “resources”

→ What does that mean?

Linear session types

Linear logic

Rejects the following two structural rules:

$$\frac{\Gamma \vdash C}{\Gamma, A \vdash C}\ \textsf{weaken} \qquad \frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}\ \textsf{contract}$$

"drop resource" "duplicate resource"

Linear logic

Rejects the following two structural rules:

$$\frac{\Gamma \vdash C}{\Gamma, A \vdash C}\ \textsf{weaken} \qquad \frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}\ \textsf{contract}$$

"drop resource" "duplicate resource"

Presented work based on intuitionistic linear logic

Linear logic

Rejects the following two structural rules:

$$\frac{\Gamma \vdash C}{\Gamma, A \vdash C}\ \mathsf{weaken} \qquad\qquad \frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}\ \mathsf{contract}$$

“drop resource” “duplicate resource”

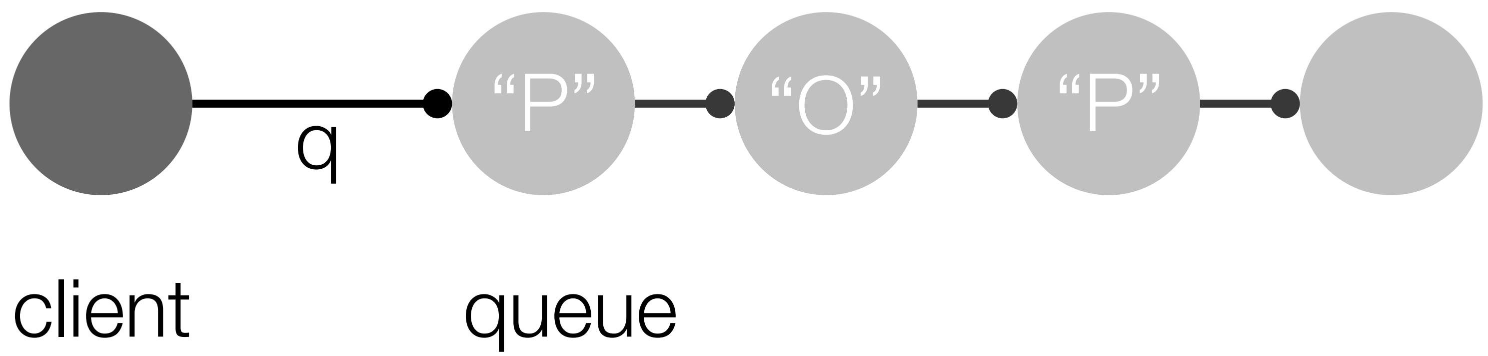
- Presented work based on intuitionistic linear logic
- Distinction of provider (right) from clients (left) of turnstile

Weakening

$$\frac{\Gamma \vdash C}{\Gamma, A \vdash C}\ \mathsf{weaken}$$

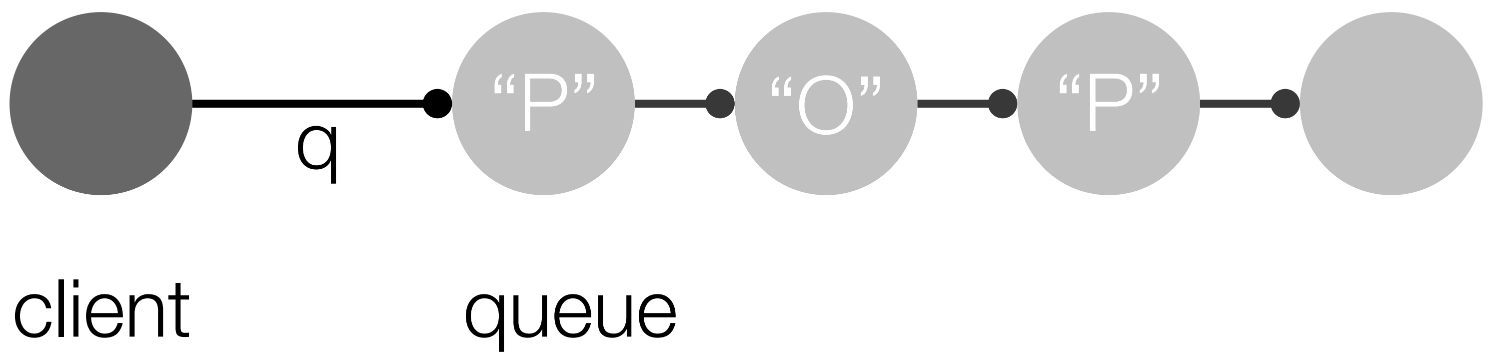
Weakening

$$\frac{\Gamma \vdash C}{\Gamma, A \vdash C}\ \mathsf{weaken}$$



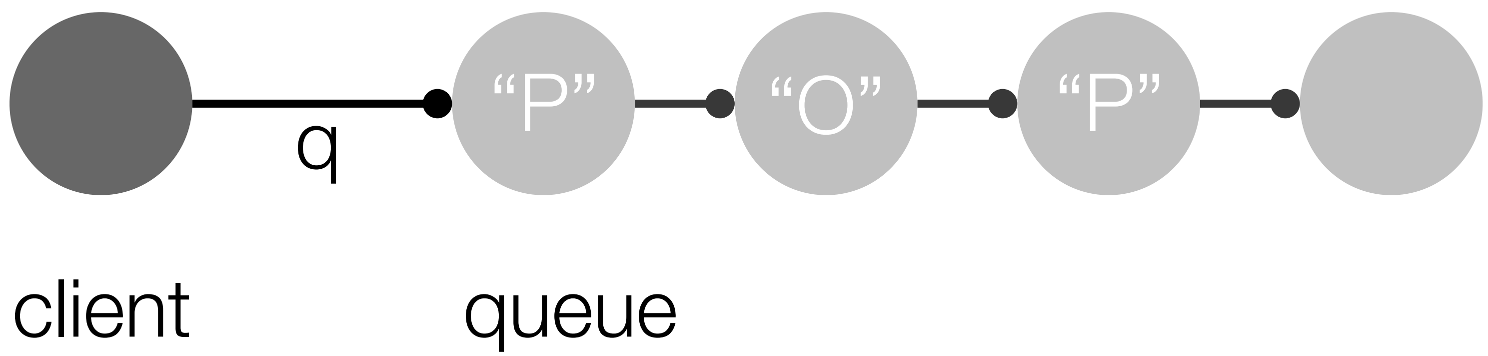
Weakening

$$\frac{\Gamma \vdash C}{\Gamma, A \vdash C}\ \mathsf{weaken}$$



Weakening

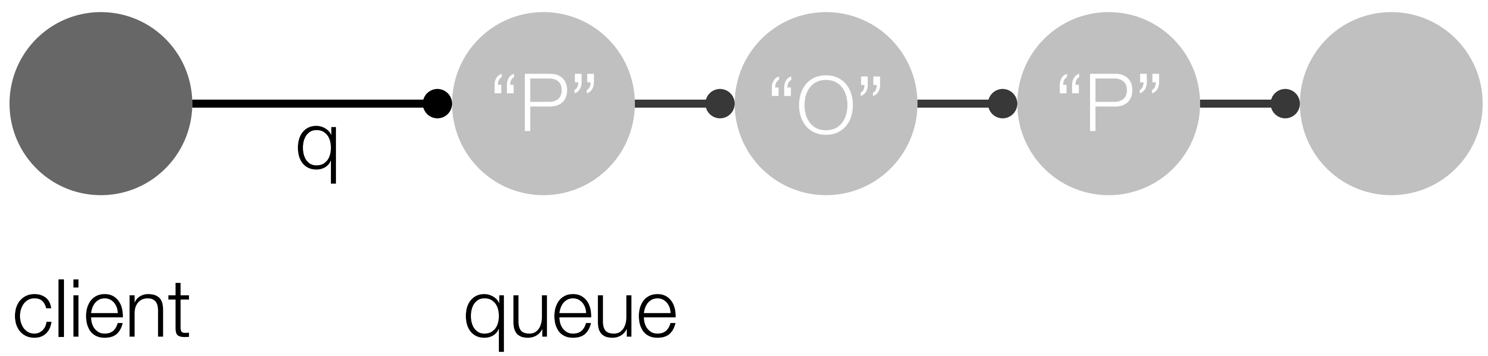
$$\frac{\Gamma \vdash C}{\Gamma, A \vdash C}\ \mathsf{weaken}$$



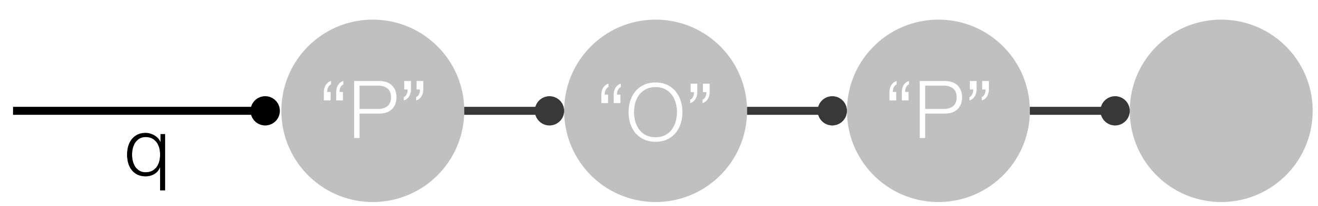
Terminating client without terminating or passing on queue

Weakening

$$\frac{\Gamma \vdash C}{\Gamma, A \vdash C} \text{ weaken}$$

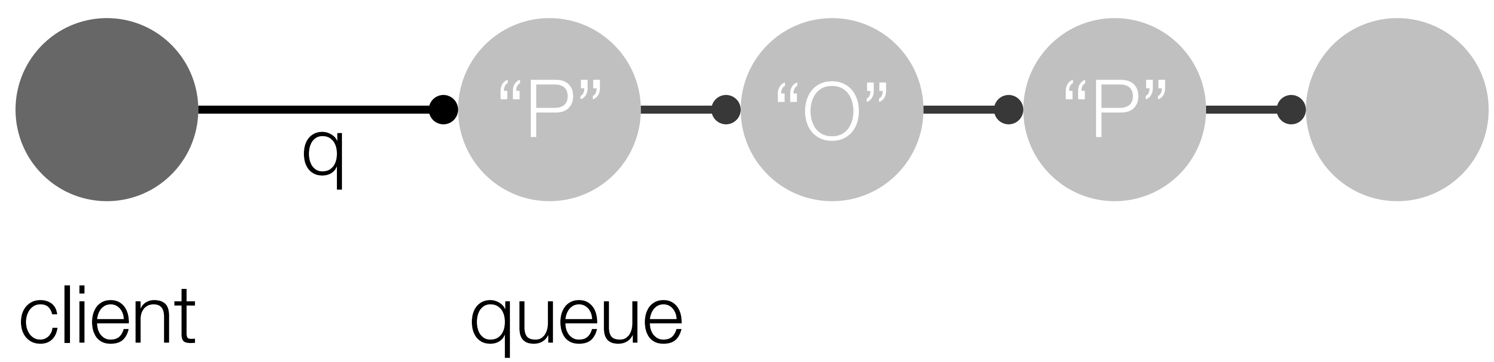


Terminating client without terminating or passing on queue

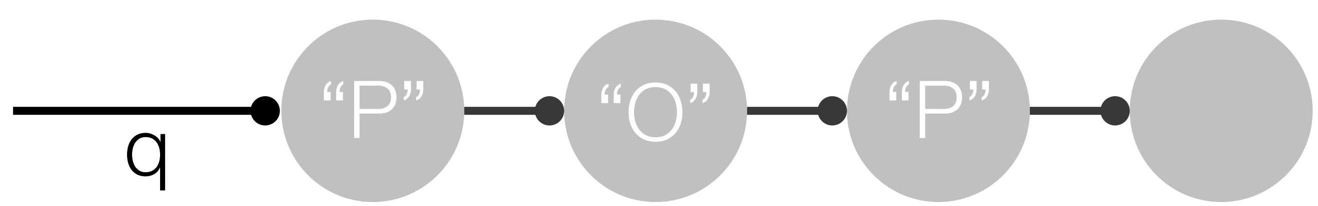


Weakening

$$\frac{\Gamma \vdash C}{\Gamma, A \vdash C}\ \text{weaken}$$



Terminating client without terminating or passing on queue



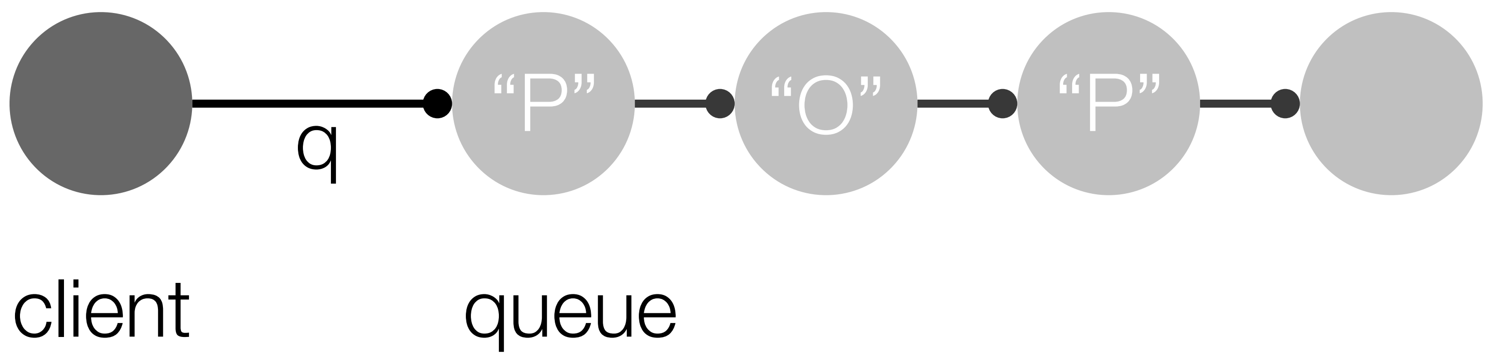
Providing process without a client

Contraction

$$\frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}\ \textsf{contract}$$

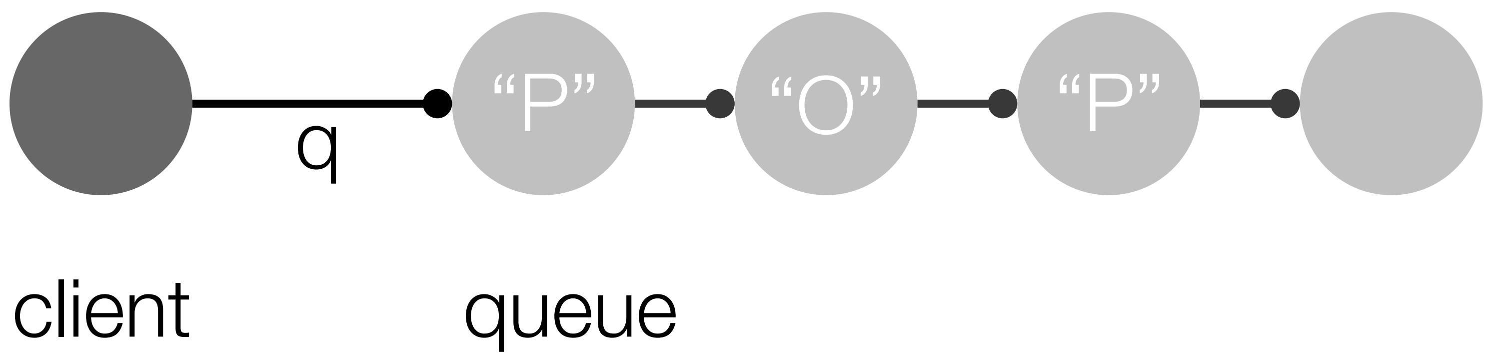
Contraction

$$\frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}\ \textsf{contract}$$



Contraction

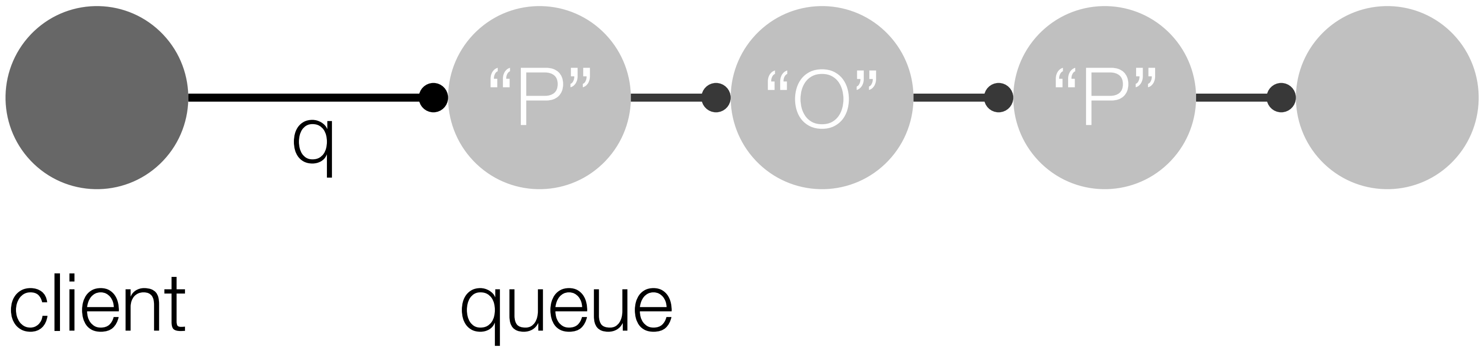
$$\frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C} \text{ contract}$$



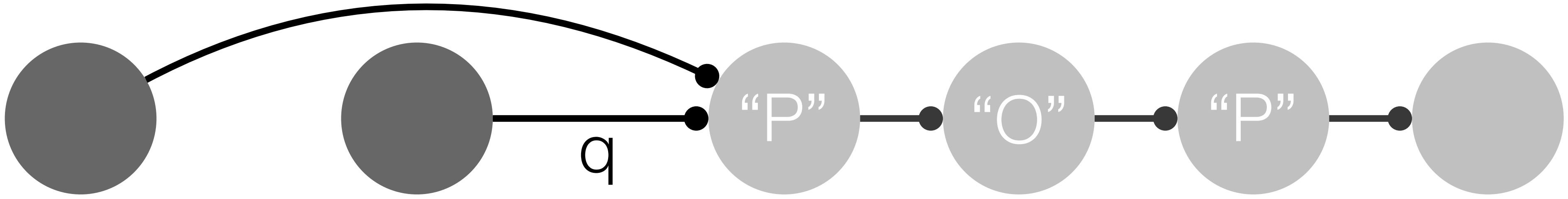
Passing on channel reference and keeping it

Contraction

$$\frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}\ \textsf{contract}$$

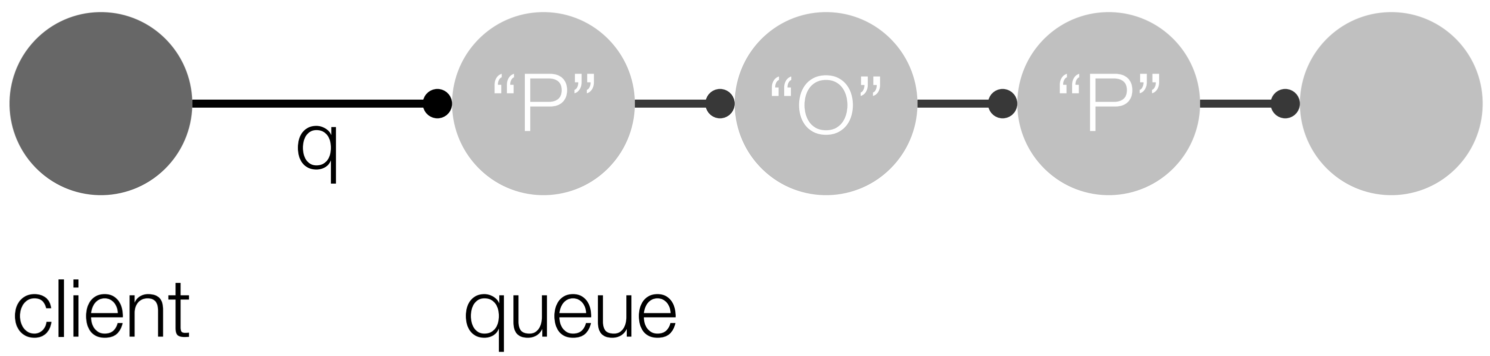


Passing on channel reference and keeping it

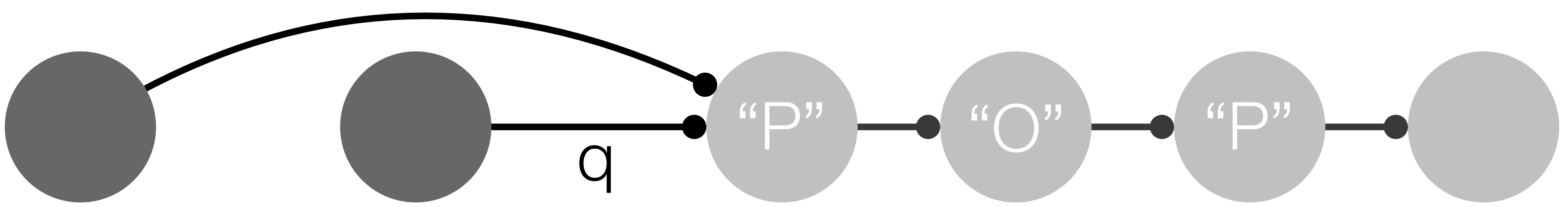


Contraction

$$\frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C} \text{ contract}$$



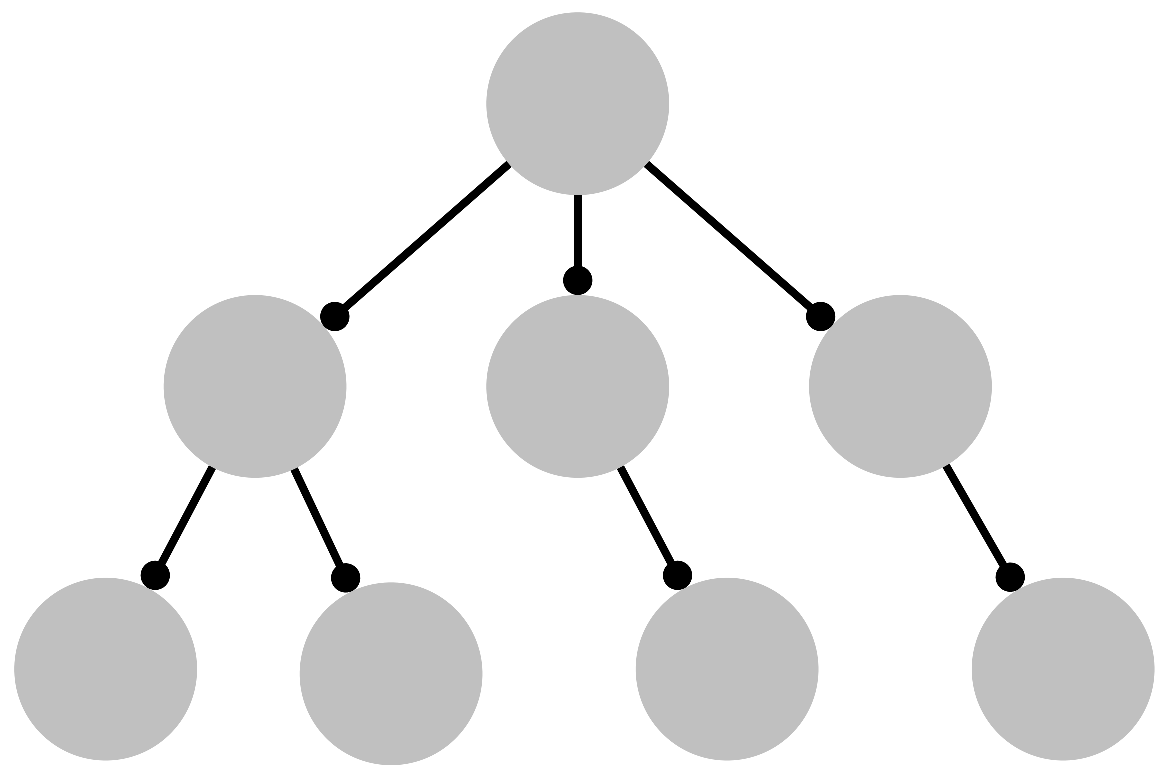
Passing on channel reference and keeping it



Providing process has multiple clients

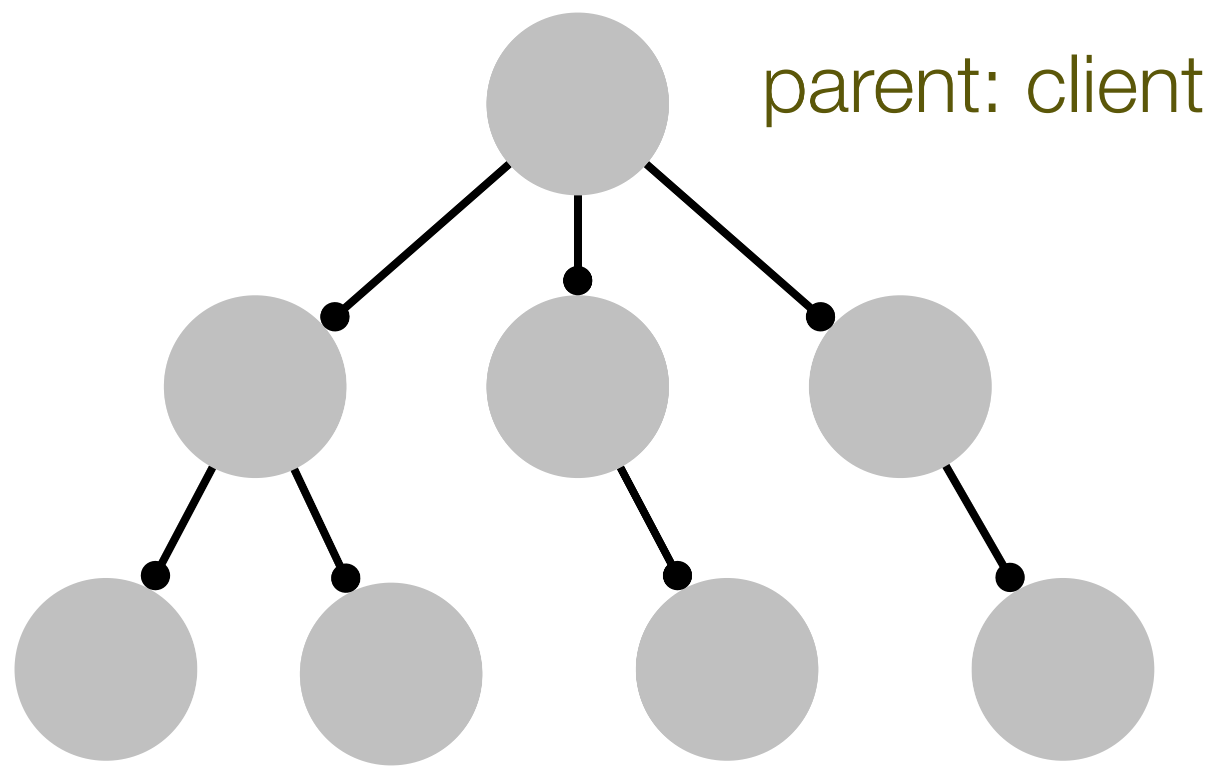
Tree structure

Without weakening and contraction, process graph forms a tree.



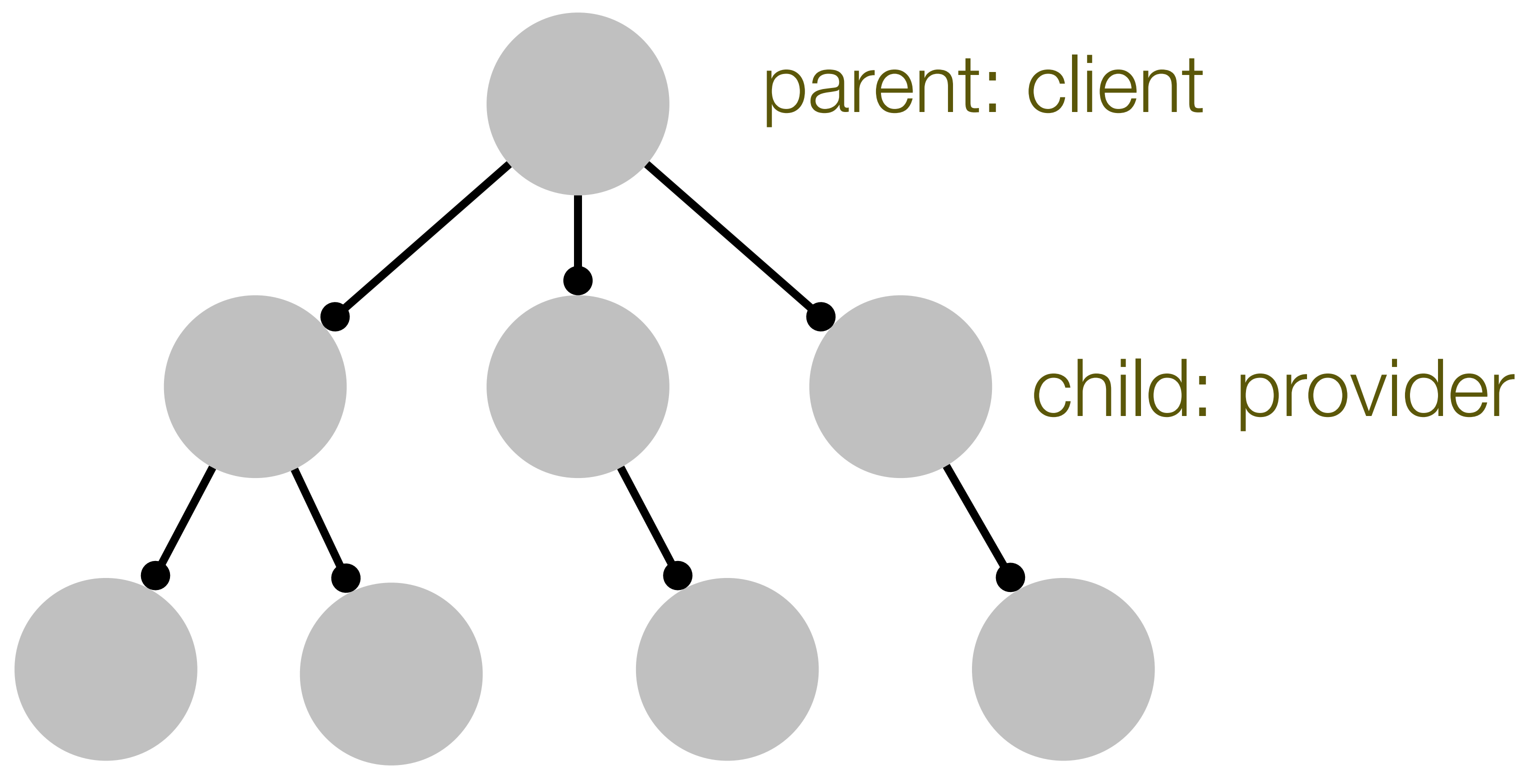
Tree structure

Without weakening and contraction, process graph forms a tree.



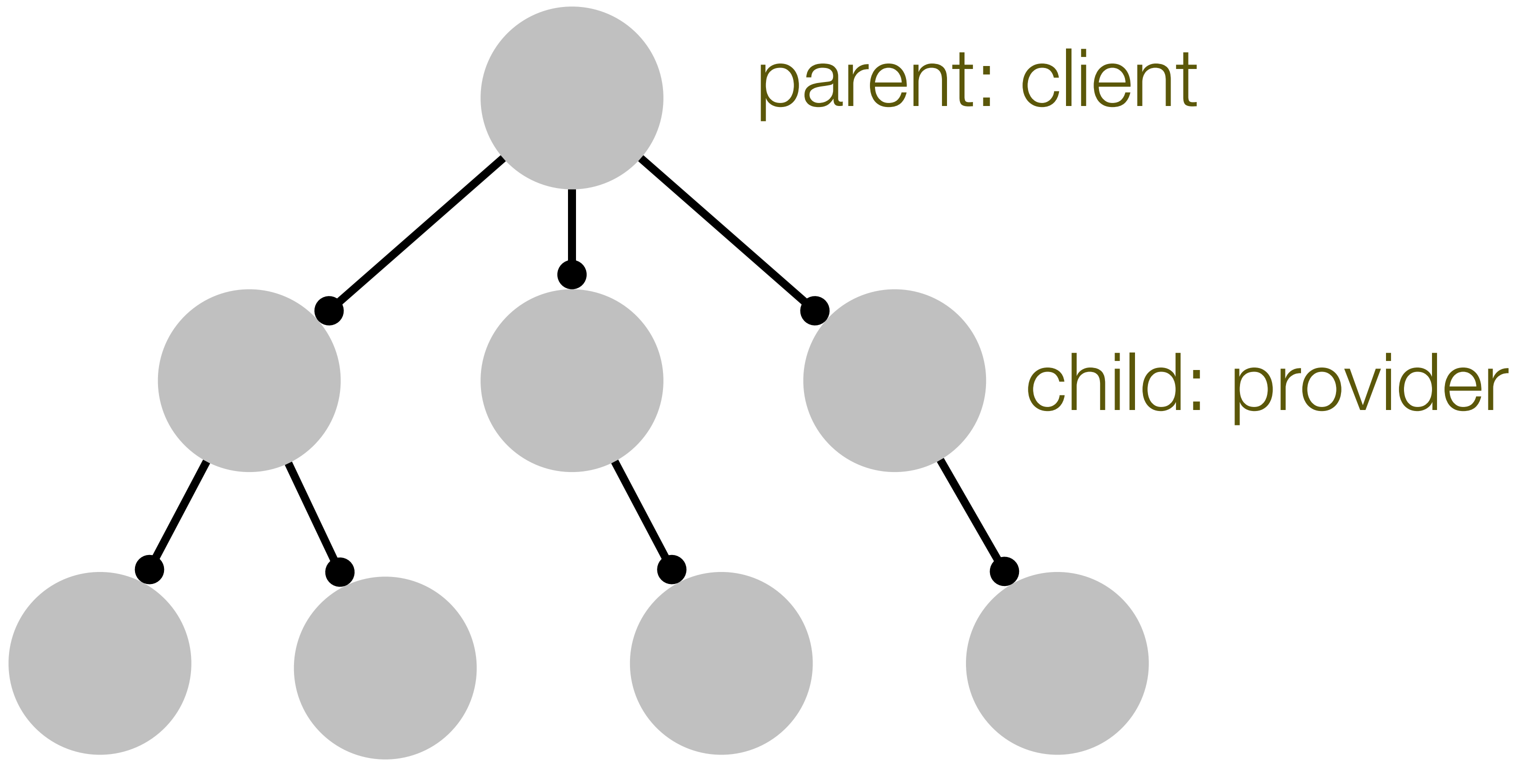
Tree structure

Without weakening and contraction, process graph forms a tree.



Tree structure

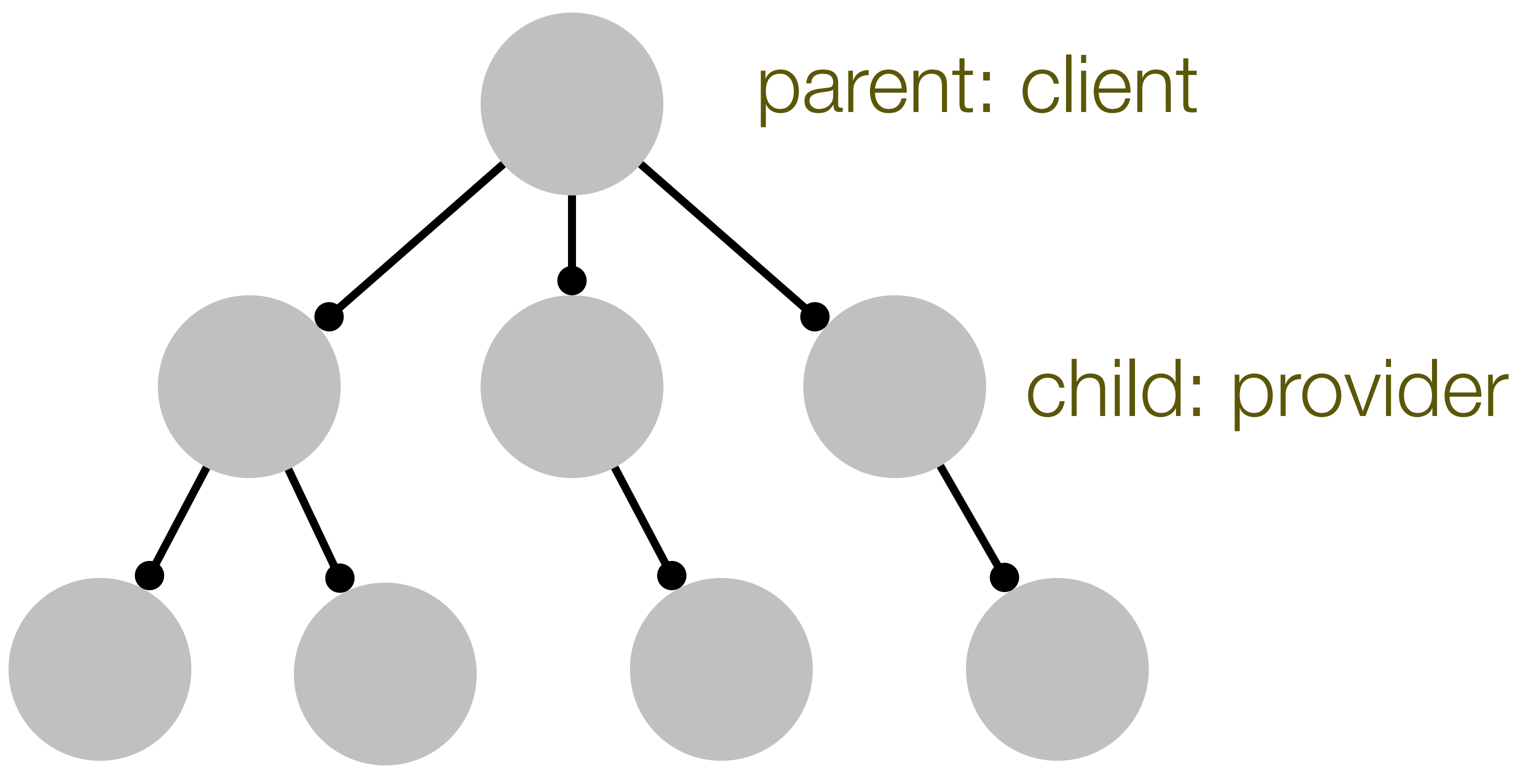
Without weakening and contraction, process graph forms a tree.



Every providing process has exactly one client.

Tree structure

Without weakening and contraction, process graph forms a tree.



Every providing process has exactly one client.

Session fidelity restored.

Reconstructing session types from linear propositions [Caires & Pfenning 2010, Wadler 2012]

Curry-Howard correspondence: intuitionistic linear logic - session-typed pi-calculus [Caires & Pfenning 2010]

Logic:	Type theory:
Linear propositions	Session types
Proofs	Programs
Cut reduction	Communication

Reconstructing session types from linear propositions [Caires & Pfenning 2010, Wadler 2012]

Curry-Howard correspondence: intuitionistic linear logic - session-typed pi-calculus [Caires & Pfenning 2010]

Logic:	Type theory:
Linear propositions	Session types
Proofs	Programs
Cut reduction	Communication

Let’s discover this correspondence together!

Intuitionistic linear logic

Connectives:

$$
\begin{array}{lcll}
A, B & \triangleq & A \otimes B & \text{multiplicative conjunction} \\
 & & A \multimap B & \text{multiplicative implication} \\
 & & A \& B & \text{additive conjunction} \\
 & & A \oplus B & \text{additive disjunction} \\
 & & !A & \text{"of course", persistent truth}
\end{array}
$$

Intuitionistic linear logic

Connectives:

A, B	$\triangleq$	$A \otimes B$	multiplicative conjunction
		$A \multimap B$	multiplicative implication
		$A \& B$	additive conjunction
		$A \oplus B$	additive disjunction
		$!A$	"of course", persistent truth

Judgment:

$$A_1, \ldots, \quad A_n \vdash \quad A$$

Intuitionistic linear logic

Connectives:

A, B	$\triangleq$	$A \otimes B$	multiplicative conjunction
		$A \multimap B$	multiplicative implication
		$A \& B$	additive conjunction
		$A \oplus B$	additive disjunction
		$!A$	"of course", persistent truth

Judgment:

$$A_1, \ldots, \qquad A_n \vdash P :: \qquad A$$

Intuitionistic linear logic

Connectives:

$$
\begin{array}{lll}
A, B & \triangleq & A \otimes B & \text{multiplicative conjunction} \\
 & & A \multimap B & \text{multiplicative implication} \\
 & & A \& B & \text{additive conjunction} \\
 & & A \oplus B & \text{additive disjunction} \\
 & & !A & \text{"of course", persistent truth}
\end{array}
$$

Judgment: $A_1, \ldots, \qquad A_n \vdash P :: (x : A)$

Intuitionistic linear logic

Connectives:

A, B	$\triangleq$	$A \otimes B$	multiplicative conjunction
		$A \multimap B$	multiplicative implication
		$A \& B$	additive conjunction
		$A \oplus B$	additive disjunction
		$!A$	"of course", persistent truth

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

Intuitionistic linear logic

Connectives:

$$
\begin{array}{lcll}
A, B & \triangleq & A \otimes B & \text{multiplicative conjunction} \\
 & & A \multimap B & \text{multiplicative implication} \\
 & & A \& B & \text{additive conjunction} \\
 & & A \oplus B & \text{additive disjunction} \\
 & & !A & \text{"of course", persistent truth}
\end{array}
$$

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

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

Intuitionistic linear logic

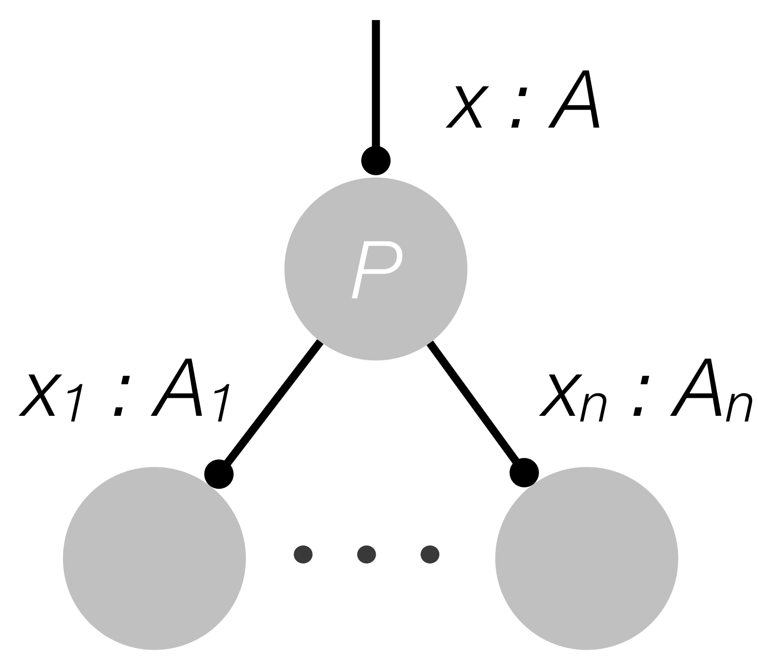
Judgment: $x_1 : A_1, \ldots, x_n : A_n \vdash P :: (x : A)$

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

Intuitionistic linear logic

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

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



Additive conjunction

$$\frac{\Delta \vdash \quad :: (\quad : A) \qquad \Delta \vdash \quad :: (\quad : B)}{\Delta \vdash \qquad\qquad :: (\quad : A \& B)} \;(\text{T-}\&_{\text{R}})$$

Additive conjunction

$$\frac{\Delta \vdash \qquad :: (\quad : A) \qquad \Delta \vdash \qquad :: (\quad : B)}{\Delta \vdash \qquad\qquad\qquad\qquad :: (x : A \& B)} \; (\text{T-}\&_{\mathrm{R}})$$

Additive conjunction

$$\frac{\Delta \vdash \qquad :: (x : A) \qquad \Delta \vdash \qquad :: (x : B)}{\Delta \vdash \qquad\qquad\qquad\qquad :: (x : A \& B)}\ (\text{T-}\&_{\mathrm{R}})$$

Additive conjunction

$$\frac{\Delta \vdash P_1 :: (x : A) \qquad \Delta \vdash P_2 :: (x : B)}{\Delta \vdash \qquad\qquad\qquad :: (x : A \& B)} \ (\text{T-}\&_{\mathrm{R}})$$

Additive conjunction

$$\frac{\Delta \vdash P_1 :: (x : A) \qquad \Delta \vdash P_2 :: (x : B)}{\Delta \vdash \textsf{case}\, x\, \textsf{of}\, (P_1, P_2) :: (x : A \& B)} \;(\text{T-}\&_{\text{R}})$$

Additive conjunction

$$\frac{\Delta \vdash P_1 :: (x : A) \qquad \Delta \vdash P_2 :: (x : B)}{\Delta \vdash \mathsf{case}\, x\, \mathsf{of}\, (P_1, P_2) :: (x : A \& B)} \; (\text{T-}\&_{\mathrm{R}})$$

$$\frac{\Delta, \qquad A \vdash \qquad :: (\quad : C)}{\Delta, \qquad A \& B \vdash \qquad\qquad :: (\quad : C)} \; (\text{T-}\&_{\mathrm{L}_1})$$

Additive conjunction

$$\frac{\Delta \vdash P_1 :: (x : A) \qquad \Delta \vdash P_2 :: (x : B)}{\Delta \vdash \mathsf{case}\, x\, \mathsf{of}\, (P_1, P_2) :: (x : A \& B)} \;(\text{T-}\&_{\text{R}})$$

$$\frac{\Delta, \quad A \vdash \quad :: (z : C)}{\Delta, \quad A \& B \vdash \qquad :: (z : C)} \;(\text{T-}\&_{\text{L}_1})$$

Additive conjunction

$$\frac{\Delta \vdash P_1 :: (x : A) \qquad \Delta \vdash P_2 :: (x : B)}{\Delta \vdash \mathsf{case}\, x\, \mathsf{of}\, (P_1, P_2) :: (x : A \& B)}\ (\text{T-}\&_{\mathrm{R}})$$

$$\frac{\Delta, x : A \vdash \qquad :: (z : C)}{\Delta, x : A \& B \vdash \qquad\qquad :: (z : C)}\ (\text{T-}\&_{\mathrm{L}_1})$$

Additive conjunction

$$\frac{\Delta \vdash P_1 :: (x : A) \qquad \Delta \vdash P_2 :: (x : B)}{\Delta \vdash \mathsf{case}\, x\, \mathsf{of}\, (P_1, P_2) :: (x : A \& B)}\ (\text{T-}\&_{\mathrm{R}})$$

$$\frac{\Delta, x : A \vdash \qquad :: (z : C)}{\Delta, x : A \& B \vdash x.\mathsf{inl}; Q :: (z : C)}\ (\text{T-}\&_{\mathrm{L}_1})$$

Additive conjunction

$$\frac{\Delta \vdash P_1 :: (x : A) \qquad \Delta \vdash P_2 :: (x : B)}{\Delta \vdash \mathsf{case}\, x \,\mathsf{of}\, (P_1, P_2) :: (x : A \& B)} \; (\text{T-}\&_{\text{R}})$$

$$\frac{\Delta, x : A \vdash Q :: (z : C)}{\Delta, x : A \& B \vdash x.\mathsf{inl}; Q :: (z : C)} \; (\text{T-}\&_{\text{L}_1})$$

Additive conjunction

$$\frac{\Delta \vdash P_1 :: (x : A) \qquad \Delta \vdash P_2 :: (x : B)}{\Delta \vdash \mathsf{case}\, x\, \mathsf{of}\, (P_1, P_2) :: (x : A \& B)}\ (\text{T-}\&_{\text{R}})$$

$$\frac{\Delta, x : A \vdash Q :: (z : C)}{\Delta, x : A \& B \vdash x.\mathsf{inl}; Q :: (z : C)}\ (\text{T-}\&_{\text{L}_1})$$

$$\frac{\Delta, x : B \vdash Q :: (z : C)}{\Delta, x : A \& B \vdash x.\mathsf{inr}; Q :: (z : C)}\ (\text{T-}\&_{\text{L}_2})$$

Additive conjunction

$$\frac{\Delta \vdash P_1 :: (x : A) \qquad \Delta \vdash P_2 :: (x : B)}{\Delta \vdash \mathsf{case}\, x\, \mathsf{of}\, (P_1, P_2) :: (x : A \& B)} \; (\text{T-}\&_{\mathrm{R}})$$

$$\frac{\Delta, x : A \vdash Q :: (z : C)}{\Delta, x : A \& B \vdash x.\mathsf{inl}; Q :: (z : C)} \; (\text{T-}\&_{\mathrm{L}_1})$$

$$\frac{\Delta, x : B \vdash Q :: (z : C)}{\Delta, x : A \& B \vdash x.\mathsf{inr}; Q :: (z : C)} \; (\text{T-}\&_{\mathrm{L}_2})$$

External choice: client chooses. Generalizes to $\&\{\overline{l : A}\}$.

Additive disjunction

$$\frac{\Delta \vdash P :: (x : A)}{\Delta \vdash x.\mathsf{inl}; P :: (x : A \oplus B)} \ (\text{T-}\oplus_{\text{R}_1})$$

$$\frac{\Delta \vdash P :: (x : B)}{\Delta \vdash x.\mathsf{inr}; P :: (x : A \oplus B)} \ (\text{T-}\oplus_{\text{R}_2})$$

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

Internal choice: provider chooses. Generalizes to $\oplus\{\overline{l : A}\}$.

Multiplicative implication

$$\frac{\Delta, \qquad A \vdash \qquad :: (\quad : B)}{\Delta \vdash \qquad\qquad\qquad :: (\quad : A \multimap B)} \ (\text{T-}\multimap_{\text{R}})$$

Multiplicative implication

$$\frac{\Delta, \quad\quad A \vdash \quad\quad :: (\quad : B)}{\Delta \vdash \quad\quad\quad\quad\quad\quad\quad\quad :: (x : A \multimap B)} \; (\mathrm{T}\text{-}\multimap_{\mathrm{R}})$$

Multiplicative implication

$$\frac{\Delta, \quad A \vdash \quad :: (x : B)}{\Delta \vdash \quad :: (x : A \multimap B)} \; (\mathrm{T}\text{-}\!\multimap_{\mathrm{R}})$$

Multiplicative implication

$$\frac{\Delta, y : A \vdash \qquad :: (x : B)}{\Delta \vdash \qquad\qquad\qquad :: (x : A \multimap B)} \; (\mathrm{T}\text{-}{\multimap}_{\mathrm{R}})$$

Multiplicative implication

$$\frac{\Delta, y : A \vdash \quad :: (x : B)}{\Delta \vdash y \leftarrow \mathsf{recv}\ x; P :: (x : A \multimap B)}\ (\mathrm{T}\text{-}{\multimap}_{\mathrm{R}})$$

Multiplicative implication

$$\frac{\Delta, y : A \vdash P :: (x : B)}{\Delta \vdash y \leftarrow \mathsf{recv}\ x; P :: (x : A \multimap B)} \ (\text{T-}\multimap_{\mathrm{R}})$$

Multiplicative implication

$$\frac{\Delta, y : A \vdash \ P :: (x : B)}{\Delta \ \vdash \ y \leftarrow \mathsf{recv}\ x; P :: (x : A \multimap B)}\ (\mathrm{T}\text{-}\multimap_{\mathrm{R}})$$

$$\frac{\Delta \ \vdash \qquad :: (\quad : A) \qquad \Delta', \qquad B \ \vdash \qquad :: (\quad : C)}{\Delta, \Delta', \qquad A \multimap B \ \vdash \qquad\qquad\qquad :: (\quad : C)}\ (\mathrm{T}\text{-}\multimap_{\mathrm{L}})$$

Channel input: provider receives a channel.

Multiplicative implication

$$\frac{\Delta, y : A \vdash P :: (x : B)}{\Delta \vdash y \leftarrow \mathsf{recv}\ x; P :: (x : A \multimap B)} \ (\mathrm{T}\text{-}{\multimap_{\mathrm{R}}})$$

$$\frac{\Delta \vdash \quad :: (\quad : A) \qquad \Delta', \quad B \vdash \quad :: (z : C)}{\Delta, \Delta', \quad A \multimap B \vdash \qquad :: (z : C)} \ (\mathrm{T}\text{-}{\multimap_{\mathrm{L}}})$$

Channel input: provider receives a channel.

Multiplicative implication

$$\frac{\Delta, y : A \vdash P :: (x : B)}{\Delta \vdash y \leftarrow \textsf{recv}\ x; P :: (x : A \multimap B)}\ (\text{T-}\multimap_{\text{R}})$$

$$\frac{\Delta \vdash \quad :: (\quad : A) \qquad \Delta', x : B \vdash \quad :: (z : C)}{\Delta, \Delta', x : A \multimap B \vdash \qquad\qquad :: (z : C)}\ (\text{T-}\multimap_{\text{L}})$$

Channel input: provider receives a channel.

Multiplicative implication

$$\frac{\Delta, y : A \vdash P :: (x : B)}{\Delta \vdash y \leftarrow \mathsf{recv}\ x; P :: (x : A \multimap B)}\ (\mathrm{T}\text{-}\multimap_{\mathrm{R}})$$

$$\frac{\Delta \vdash \quad :: (\quad : A) \qquad \Delta', x : B \vdash \quad :: (z : C)}{\Delta, \Delta', x : A \multimap B \vdash \mathsf{send}\ x\ (y \leftarrow Q); Q' :: (z : C)}\ (\mathrm{T}\text{-}\multimap_{\mathrm{L}})$$

Channel input: provider receives a channel.

Multiplicative implication

$$\frac{\Delta, y : A \vdash P :: (x : B)}{\Delta \vdash y \leftarrow \textsf{recv}\ x; P :: (x : A \multimap B)} \ (\mathrm{T}\text{-}\multimap_{\mathrm{R}})$$

$$\frac{\Delta \vdash \quad :: (\quad : A) \qquad \Delta', x : B \vdash Q' :: (z : C)}{\Delta, \Delta', x : A \multimap B \vdash \textsf{send}\ x\ (y \leftarrow Q); Q' :: (z : C)} \ (\mathrm{T}\text{-}\multimap_{\mathrm{L}})$$

Channel input: provider receives a channel.

Multiplicative implication

$$\frac{\Delta, y : A \vdash \; P :: (x : B)}{\Delta \vdash \; y \leftarrow \textsf{recv}\; x; P :: (x : A \multimap B)} \; (\text{T-}{\multimap}_{\text{R}})$$

$$\frac{\Delta \vdash \qquad :: (y : A) \qquad \Delta', x : B \vdash \; Q' :: (z : C)}{\Delta, \Delta', x : A \multimap B \vdash \; \textsf{send}\; x\; (y \leftarrow Q); Q' :: (z : C)} \; (\text{T-}{\multimap}_{\text{L}})$$

Channel input: provider receives a channel.

Multiplicative implication

$$\frac{\Delta, y : A \vdash P :: (x : B)}{\Delta \vdash y \leftarrow \textsf{recv}\ x; P :: (x : A \multimap B)} \ (\text{T-}\multimap_{\text{R}})$$

$$\frac{\Delta \vdash Q :: (y : A) \qquad \Delta', x : B \vdash Q' :: (z : C)}{\Delta, \Delta', x : A \multimap B \vdash \textsf{send}\ x\ (y \leftarrow Q); Q' :: (z : C)} \ (\text{T-}\multimap_{\text{L}})$$

Channel input: provider receives a channel.

Multiplicative conjunction

$$\frac{\Delta \vdash Q :: (y : A) \qquad \Delta' \vdash P :: (x : B)}{\Delta, \Delta' \vdash \mathsf{send}\ x\ (y \leftarrow Q); P :: (x : A \otimes B)}\ (\text{T-}\otimes_{\text{R}})$$

$$\frac{\Delta, x : B, y : A \vdash Q' :: (z : C)}{\Delta, x : A \otimes B \vdash y \leftarrow \mathsf{recv}\ x; Q' :: (z : C)}\ (\text{T-}\otimes_{\text{L}})$$

Channel output: provider sends a channel.

Unit of multiplicative conjunction

$$\frac{}{\cdot \vdash \qquad\qquad\qquad :: (\quad : \mathbf{1})}\ (\text{T-}\mathbf{1}_{\mathrm{R}})$$

Unit of multiplicative conjunction

$$\frac{}{\cdot \vdash \qquad\qquad :: (x : \mathbf{1})} \text{(T-}\mathbf{1}_{\mathrm{R}}\text{)}$$

Unit of multiplicative conjunction

$$\frac{}{\cdot \vdash \mathsf{close}\ x :: (x : \mathbf{1})}\ (\text{T-}\mathbf{1}_{\text{R}})$$

Unit of multiplicative conjunction

$$\frac{}{\cdot \vdash \mathsf{close}\ x :: (x : \mathbf{1})}\ (\text{T-}\mathbf{1}_{\text{R}})$$

$$\frac{\Delta \vdash \quad :: (\quad : C)}{\Delta, \quad : \mathbf{1} \vdash \qquad\qquad :: (\quad : C)}\ (\text{T-}\mathbf{1}_{\text{L}})$$

Unit of multiplicative conjunction

$$\frac{}{\cdot \vdash \mathsf{close}\ x :: (x : \mathbf{1})}\ (\text{T-}\mathbf{1}_{\text{R}})$$

$$\frac{\Delta \vdash \quad :: (z : C)}{\Delta, \quad : \mathbf{1} \vdash \qquad :: (z : C)}\ (\text{T-}\mathbf{1}_{\text{L}})$$

Unit of multiplicative conjunction

$$\frac{}{\cdot \vdash \mathsf{close}\ x :: (x : \mathbf{1})}\ (\text{T-}\mathbf{1}_{\text{R}})$$

$$\frac{\Delta \vdash \qquad :: (z : C)}{\Delta, x : \mathbf{1} \vdash \qquad\qquad :: (z : C)}\ (\text{T-}\mathbf{1}_{\text{L}})$$

Unit of multiplicative conjunction

$$\frac{}{\cdot \vdash \mathsf{close}\ x :: (x : \mathbf{1})}\ (\text{T-}\mathbf{1}_{\text{R}})$$

$$\frac{\Delta \vdash \qquad :: (z : C)}{\Delta, x : \mathbf{1} \vdash \mathsf{wait}\ x; Q :: (z : C)}\ (\text{T-}\mathbf{1}_{\text{L}})$$

Unit of multiplicative conjunction

$$\frac{}{\cdot \vdash \ \mathsf{close}\ x :: (x : \mathbf{1})}\ (\mathrm{T}\text{-}\mathbf{1}_{\mathrm{R}})$$

$$\frac{\Delta \vdash Q :: (z : C)}{\Delta, x : \mathbf{1} \vdash \ \mathsf{wait}\ x; Q :: (z : C)}\ (\mathrm{T}\text{-}\mathbf{1}_{\mathrm{L}})$$

Unit of multiplicative conjunction

$$\frac{}{\cdot \vdash \mathsf{close}\ x :: (x : \mathbf{1})}\ (\text{T-}\mathbf{1}_\text{R})$$

$$\frac{\Delta \vdash Q :: (z : C)}{\Delta, x : \mathbf{1} \vdash \mathsf{wait}\ x; Q :: (z : C)}\ (\text{T-}\mathbf{1}_\text{L})$$

→ Termination

Unit of multiplicative conjunction

$$\frac{}{\cdot \vdash \mathsf{close}\ x :: (x : \mathbf{1})}\ (\text{T-}\mathbf{1}_{\text{R}})$$

$$\frac{\Delta \vdash Q :: (z : C)}{\Delta, x : \mathbf{1} \vdash \mathsf{wait}\ x; Q :: (z : C)}\ (\text{T-}\mathbf{1}_{\text{L}})$$

- Termination
- $\&$ and $\oplus$ must consist of at least one label.

Recap

Recap

Connectives:

A, B	$\triangleq$	$A \otimes B$	channel output
		$A \multimap B$	channel input
		$\oplus\{\overline{l : A}\}$	internal choice
		$\&\{\overline{l : A}\}$	external choice
		$\mathbf{1}$	termination

Recap

Connectives:

$$
\begin{array}{lcll}
A, B & \triangleq & A \otimes B & \text{channel output} \\
 & & A \multimap B & \text{channel input} \\
 & & \oplus\{\overline{l : A}\} & \text{internal choice} \\
 & & \&\{\overline{l : A}\} & \text{external choice} \\
 & & \mathbf{1} & \text{termination}
\end{array}
$$

Example:

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

Judgmental rules

Judgmental rules

$$\frac{\Delta \vdash P(x : A) \qquad \Delta', x : A \vdash Q :: (z : C)}{\Delta, \Delta' \vdash x \leftarrow P; Q :: (z : C)} \; (\textsc{T-cut})$$

Judgmental rules

$$\frac{\Delta \vdash P(x : A) \qquad \Delta', x : A \vdash Q :: (z : C)}{\Delta, \Delta' \vdash x \leftarrow P; Q :: (z : C)} \; (\text{T-}\textsc{cut})$$

Parallel composition (spawning new process)

Judgmental rules

$$\frac{\Delta \vdash P(x : A) \qquad \Delta', x : A \vdash Q :: (z : C)}{\Delta, \Delta' \vdash x \leftarrow P; Q :: (z : C)} \; (\text{T-}\textsc{cut})$$

Parallel composition (spawning new process)

$$\frac{}{y : A \vdash \mathsf{fwd}\ x\ y :: (x : A)} \; (\text{T-}\textsc{id})$$

Judgmental rules

$$\frac{\Delta \vdash P(x : A) \qquad \Delta', x : A \vdash Q :: (z : C)}{\Delta, \Delta' \vdash x \leftarrow P; Q :: (z : C)} \; (\text{T-}\textsc{cut})$$

Parallel composition (spawning new process)

$$\frac{}{y : A \vdash \mathsf{fwd}\; x\; y :: (x : A)} \; (\text{T-}\textsc{id})$$

Forward: process offering along x terminates, client henceforth interacts with process offering along y.

Let's program in Concurrent C0!

C0 [Pfenning 2010, Arnold 2010]

- safe subset of C supporting contracts
- teaching language developed at CMU
- http://c0.typesafety.net

Concurrent C0 [Willsey & Prabhu & Pfenning 2016]

- extends C0 with session types

Installing Concurrent C0

- see http://www.cs.cmu.edu/~balzers/popl_tutorial_2019

What about type safety?

What about type safety?

Preservation (aka session fidelity)

What about type safety?

Preservation (aka session fidelity)

- no weakening and contraction: process graph forms a tree

What about type safety?

Preservation (aka session fidelity)

- no weakening and contraction: process graph forms a tree
- every provider has a unique client

What about type safety?

Preservation (aka session fidelity) ✓

- no weakening and contraction: process graph forms a tree
- every provider has a unique client

What about type safety?

Preservation (aka session fidelity) ✓

- no weakening and contraction: process graph forms a tree
- every provider has a unique client

Progress

What about type safety?

Preservation (aka session fidelity) ✓

- no weakening and contraction: process graph forms a tree
- every provider has a unique client

Progress

- What are the dangers to progress?

What about type safety?

Preservation (aka session fidelity) ✓

- no weakening and contraction: process graph forms a tree
- every provider has a unique client

Progress

- What are the dangers to progress?
- 2 scenarios:

What about type safety?

Preservation (aka session fidelity) ✓

- no weakening and contraction: process graph forms a tree
- every provider has a unique client

Progress

- What are the dangers to progress?
- 2 scenarios:
  - provider ready to synchronize, client not

What about type safety?

Preservation (aka session fidelity) ✓

- no weakening and contraction: process graph forms a tree
- every provider has a unique client

Progress

- What are the dangers to progress?
- 2 scenarios:
  - provider ready to synchronize, client not
  - client ready to synchronize, provider not

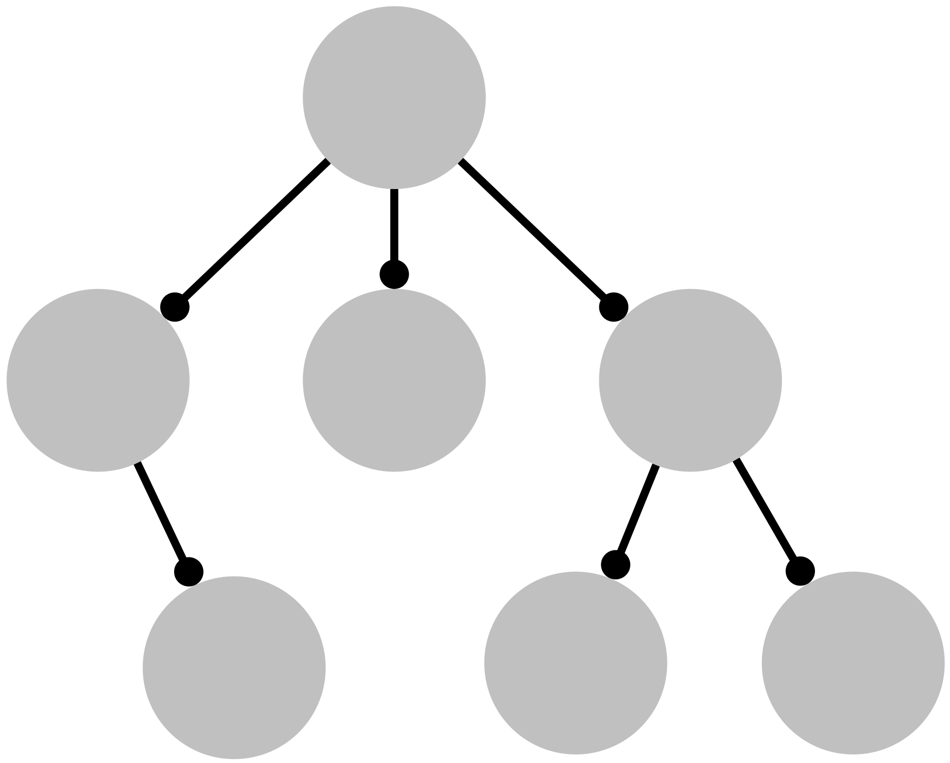
What about type safety?

Preservation (aka session fidelity) ✓

- no weakening and contraction: process graph forms a tree
- every provider has a unique client

Progress

- What are the dangers to progress?
- 2 scenarios:
  - provider ready to synchronize, client not
  - client ready to synchronize, provider not



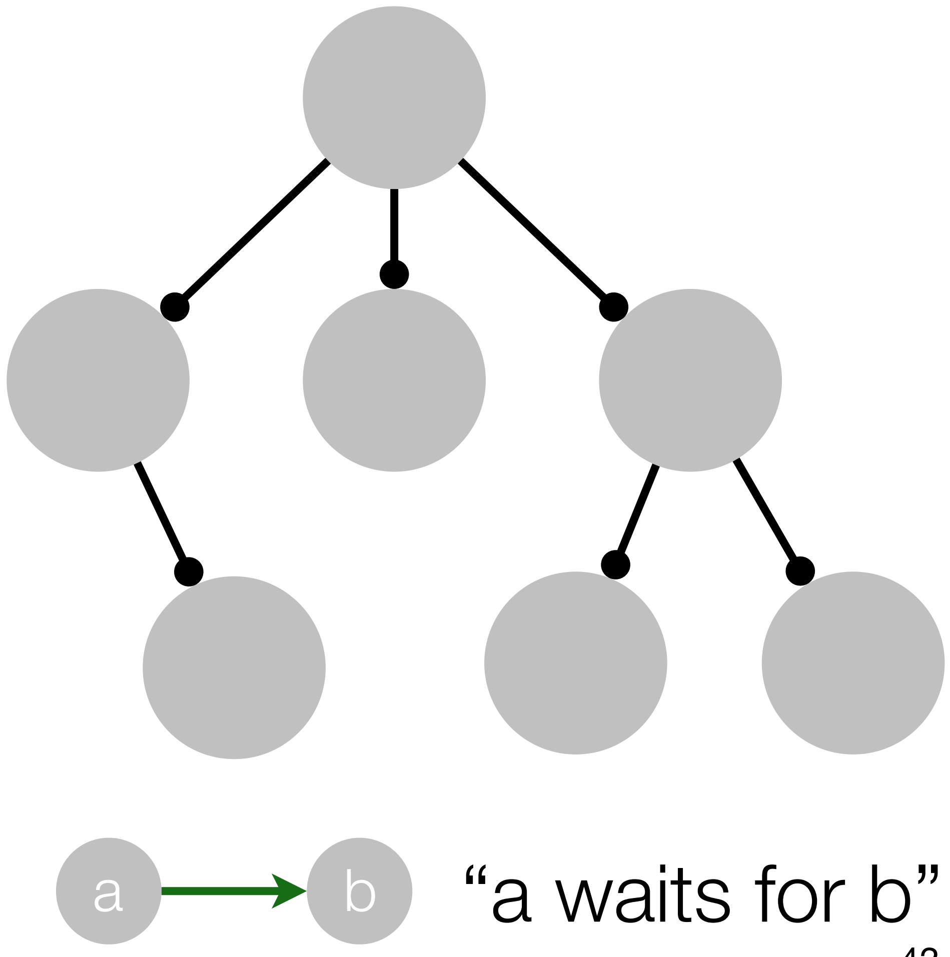
What about type safety?

Preservation (aka session fidelity) ✓

- no weakening and contraction: process graph forms a tree
- every provider has a unique client

Progress

- What are the dangers to progress?
- 2 scenarios:
  - provider ready to synchronize, client not
  - client ready to synchronize, provider not



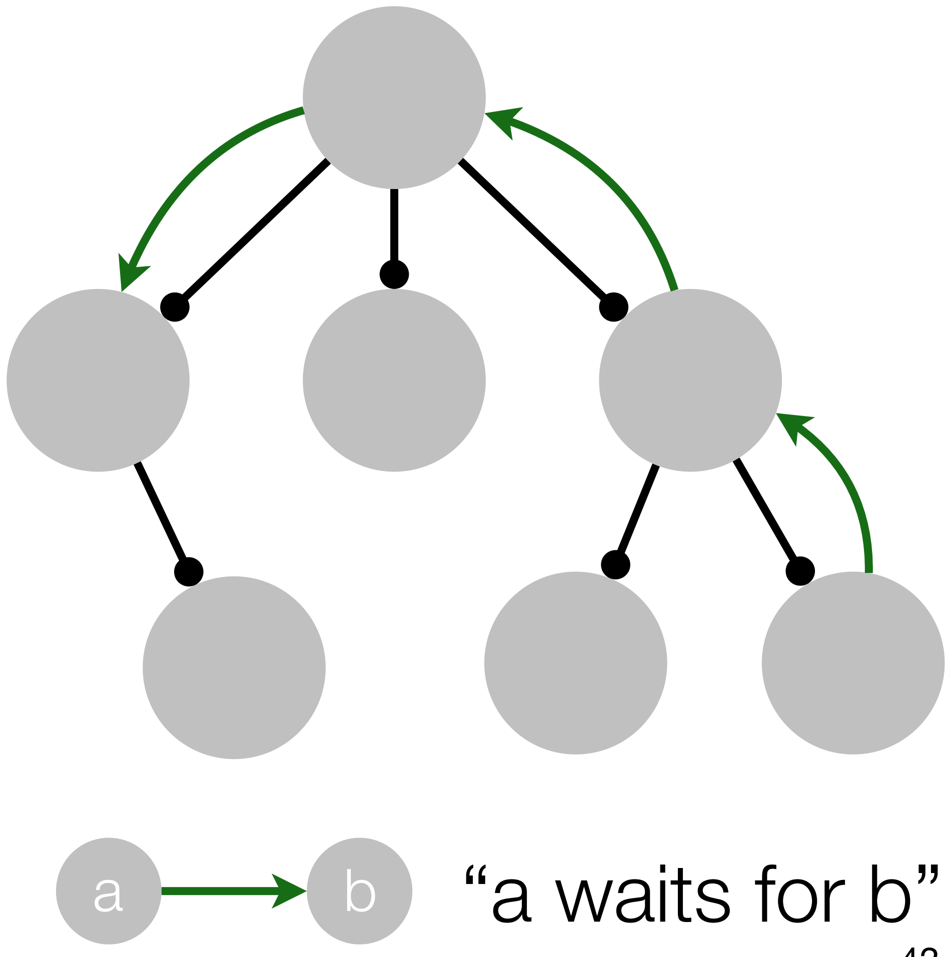
What about type safety?

Preservation (aka session fidelity) ✓

- no weakening and contraction: process graph forms a tree
- every provider has a unique client

Progress

- What are the dangers to progress?
- 2 scenarios:
  - provider ready to synchronize, client not
  - client ready to synchronize, provider not



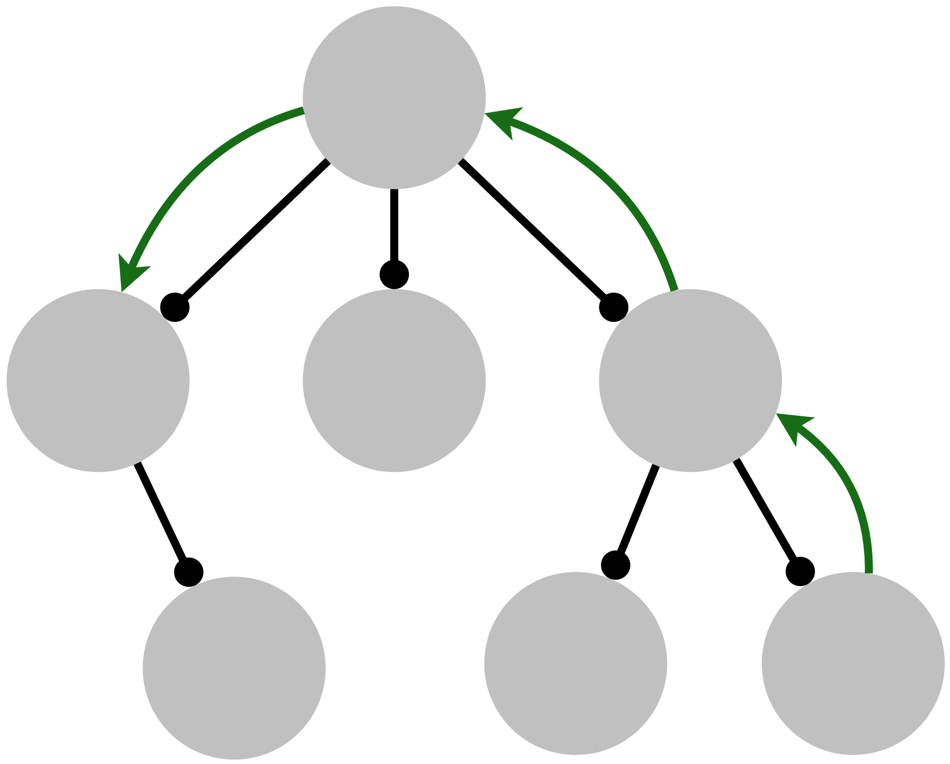
What about type safety?

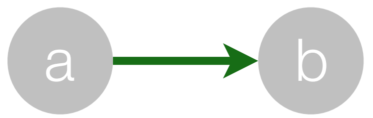
Preservation (aka session fidelity) ✓

- no weakening and contraction: process graph forms a tree
- every provider has a unique client

Progress

- What are the dangers to progress?
- 2 scenarios:
  - provider ready to synchronize, client not
  - client ready to synchronize, provider not
- green arrows can only go along edges, thus there are no cycles





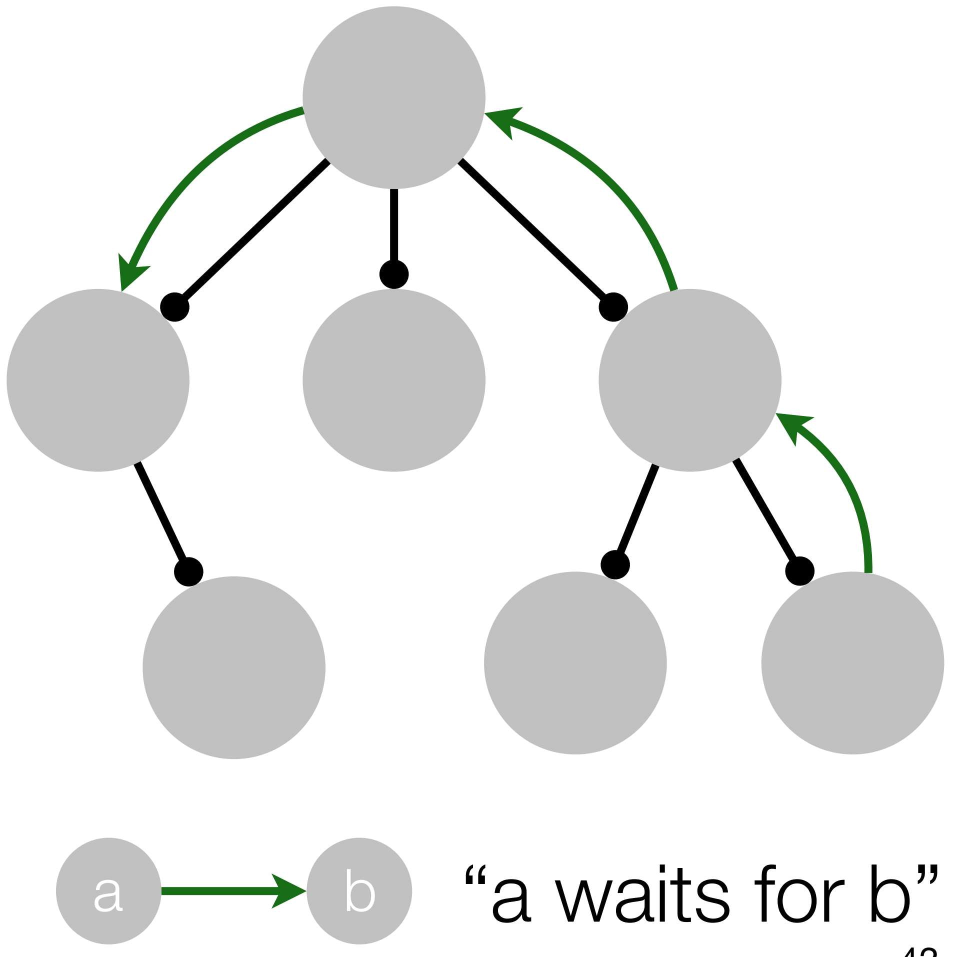
What about type safety?

Preservation (aka session fidelity) ✓

- no weakening and contraction: process graph forms a tree
- every provider has a unique client

Progress ✓

- What are the dangers to progress?
- 2 scenarios:
  - provider ready to synchronize, client not
  - client ready to synchronize, provider not
- green arrows can only go along edges, thus there are no cycles



Of course!

Of course!

Connectives

$$
\begin{array}{llll}
A, B & \triangleq & A \otimes B & \text{channel output} \\
 & & A \multimap B & \text{channel input} \\
 & & \oplus\{\overline{l : A}\} & \text{internal choice} \\
 & & \&\{\overline{l : A}\} & \text{external choice} \\
 & & \mathbf{1} & \text{termination} \\
 & & !A & \text{persisten truth}
\end{array}
$$

Of course!

Connectives

A, B	$\triangleq$	$A \otimes B$	channel output
		$A \multimap B$	channel input
		$\oplus\{\overline{l : A}\}$	internal choice
		$\&\{\overline{l : A}\}$	external choice
		$\mathbf{1}$	termination
		$!A$	persisten truth

Of course!

Connectives

$$
\begin{array}{lcll}
A, B & \triangleq & A \otimes B & \text{channel output} \\
 & & A \multimap B & \text{channel input} \\
 & & \oplus\{\overline{l : A}\} & \text{internal choice} \\
 & & \&\{\overline{l : A}\} & \text{external choice} \\
 & & \mathbf{1} & \text{termination} \\
 & & !A & \text{persisten truth}
\end{array}
$$

➡ Unrestricted proposition, can be used arbitrarily often

Of course!

Connectives

$$
\begin{array}{llll}
A, B & \triangleq & A \otimes B & \text{channel output} \\
 & & A \multimap B & \text{channel input} \\
 & & \oplus\{\overline{l : A}\} & \text{internal choice} \\
 & & \&\{\overline{l : A}\} & \text{external choice} \\
 & & \mathbf{1} & \text{termination} \\
 & & !A & \text{persisten truth}
\end{array}
$$

Unrestricted proposition, can be used arbitrarily often

$$\Psi; \Delta \vdash P :: (x : A)$$

Of course!

Connectives

A, B	$\triangleq$	$A \otimes B$	channel output
		$A \multimap B$	channel input
		$\oplus\{\overline{l : A}\}$	internal choice
		$\&\{\overline{l : A}\}$	external choice
		$\mathbf{1}$	termination
		$!A$	persisten truth

Unrestricted proposition, can be used arbitrarily often

$$\Psi; \Delta \vdash P :: (x : A)$$

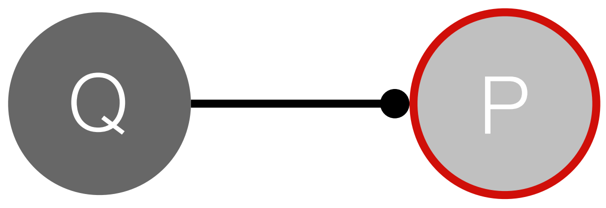
Ψ structural context (weakening and contraction)

Of course!

What is the computational meaning of “of course”?

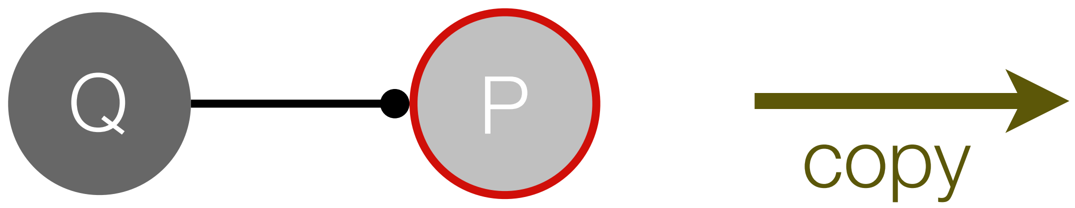
Of course!

What is the computational meaning of “of course”?



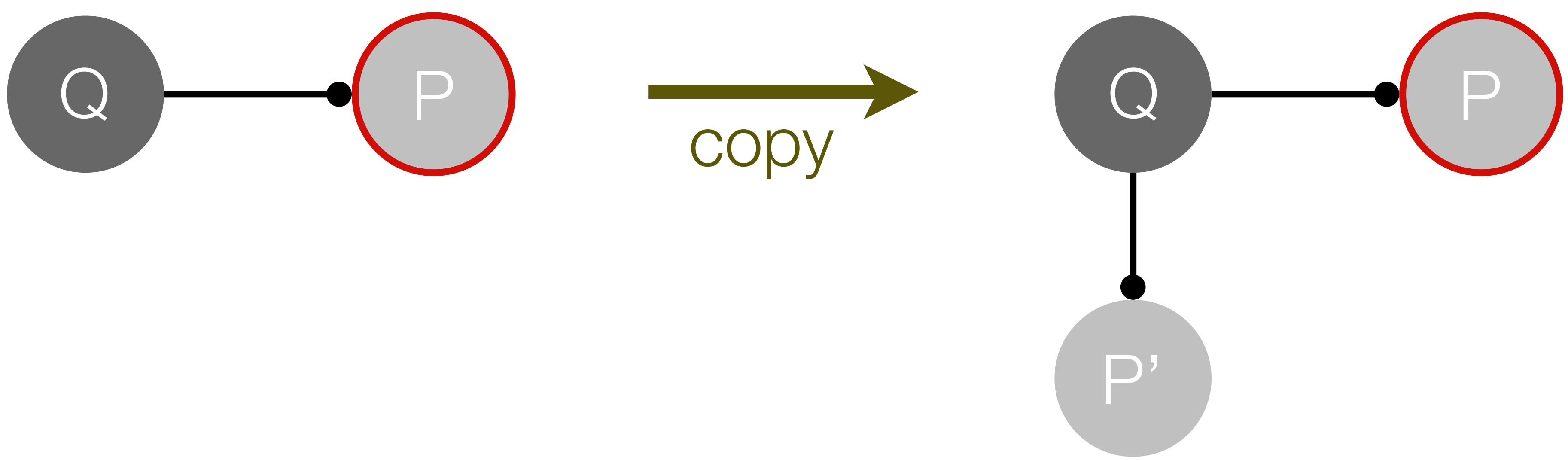
Of course!

What is the computational meaning of “of course”?



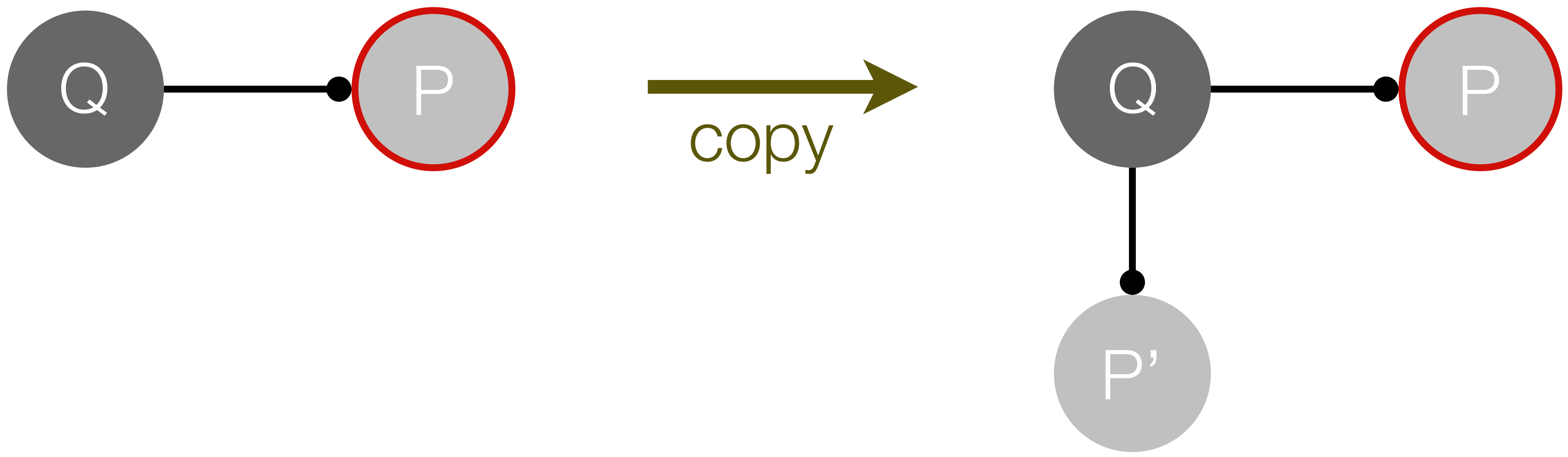
Of course!

What is the computational meaning of “of course”?



Of course!

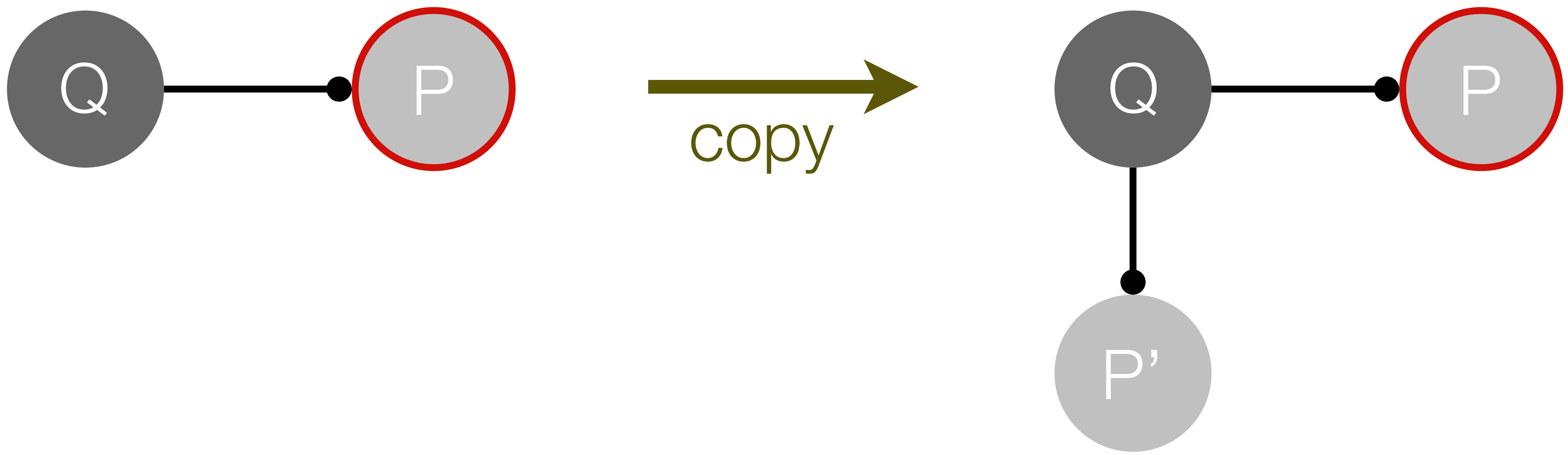
What is the computational meaning of “of course”?



Copy: obtain linear copy from linear server process

Of course!

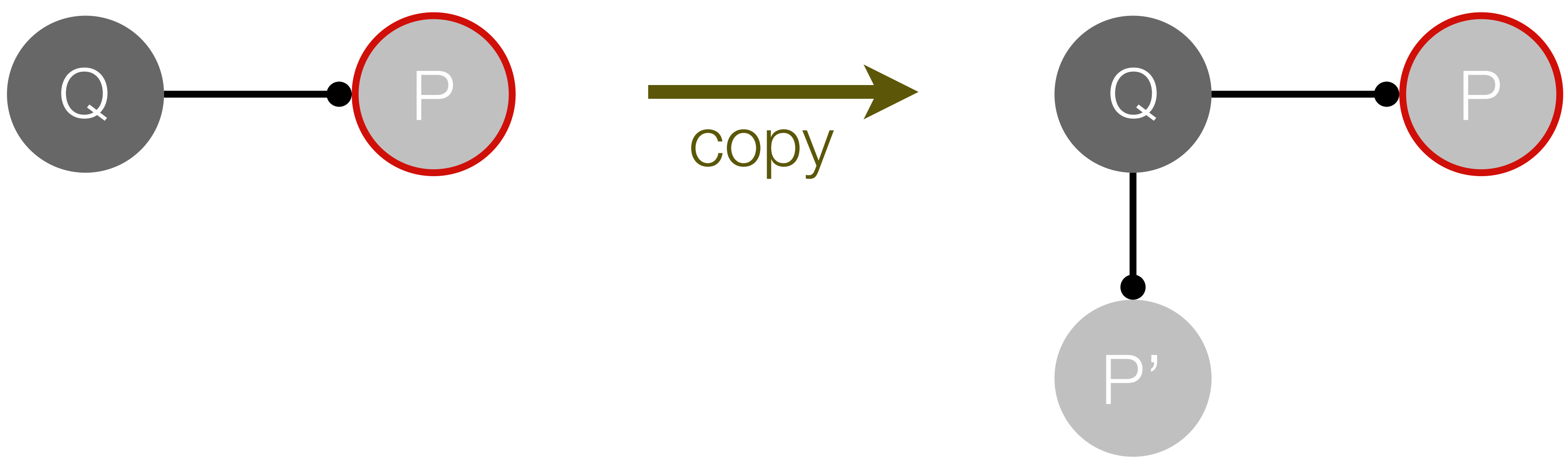
What is the computational meaning of “of course”?



- Copy: obtain linear copy from linear server process
- Corresponds to replication in the pi-calculus

Of course!

What is the computational meaning of “of course”?



Copy: obtain linear copy from linear server process

Corresponds to replication in the pi-calculus

For some applications, a copying semantics is sufficient. For other applications, sharing is necessary.

Manifest sharing

Manifest sharing - key ideas [Balzer & Pfenning 2017]

Manifest sharing - key ideas [Balzer & Pfenning 2017]

Acquire-release

- Multiple aliases (shared channels) to a process permitted, but communication requires exclusive access.

Manifest sharing - key ideas [Balzer & Pfenning 2017]

Acquire-release

- Multiple aliases (shared channels) to a process permitted, but communication requires exclusive access.

Manifestation in type structure

- Enrich type structure with adjoint modalities to prescribe acquire and release points.

Manifest sharing - key ideas [Balzer & Pfenning 2017]

Acquire-release

- Multiple aliases (shared channels) to a process permitted, but communication requires exclusive access.

Manifestation in type structure

- Enrich type structure with adjoint modalities to prescribe acquire and release points.

Equi-synchronizing session type

- Invariant guaranteeing that a process is released to the same type at which it was previously acquired, should it be released at all.
- Avoids run-time type checking upon acquire.

Manifest sharing - key ideas [Balzer & Pfenning 2017]

Acquire-release

- Multiple aliases (shared channels) to a process permitted, but communication requires exclusive access.

Manifestation in type structure

- Enrich type structure with adjoint modalities to prescribe acquire and release points.

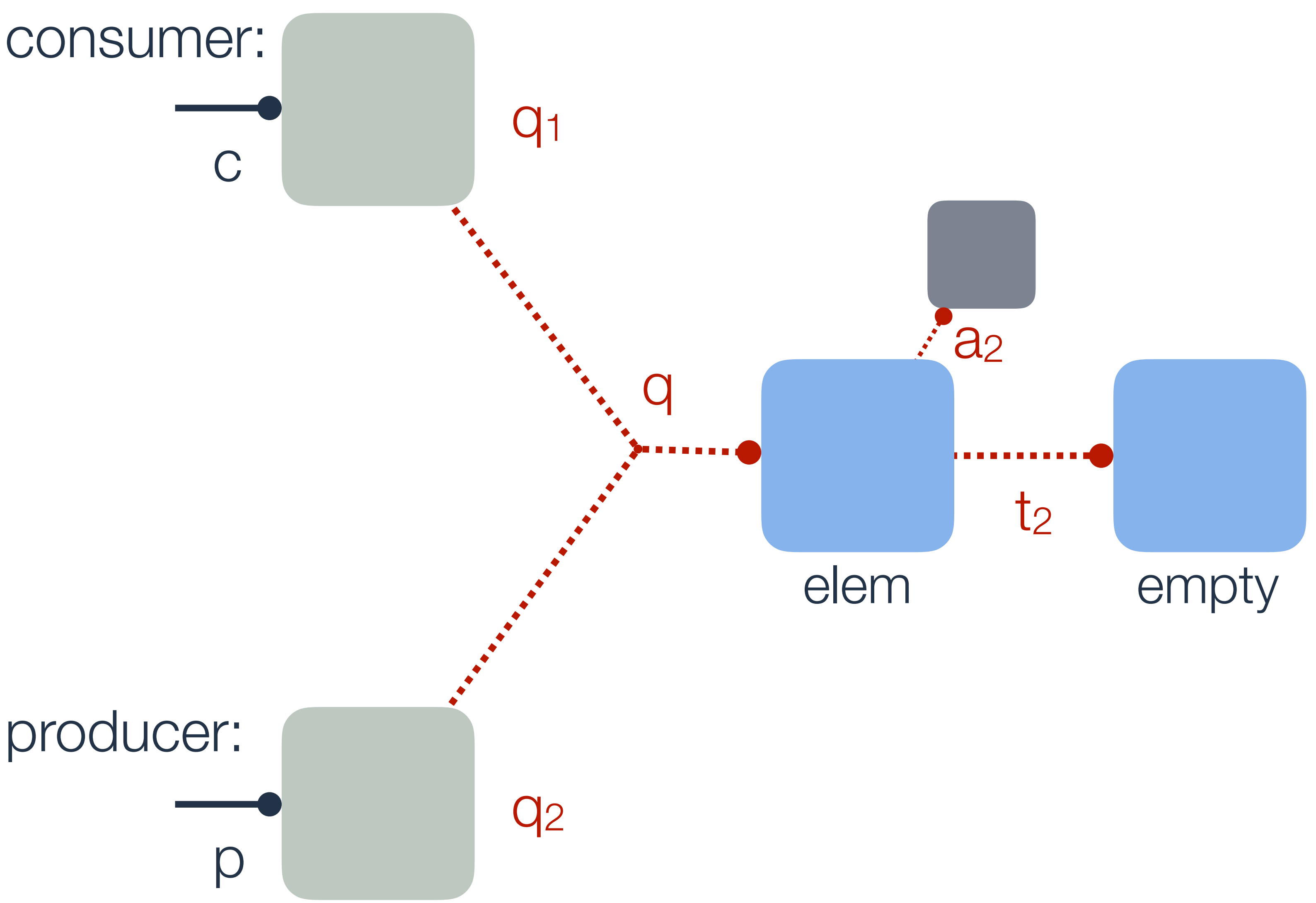
Equi-synchronizing session type

- Invariant guaranteeing that a process is released to the same type at which it was previously acquired, should it be released at all.
- Avoids run-time type checking upon acquire.

→ let's look at each in turn

Programmatic working of acquire-release

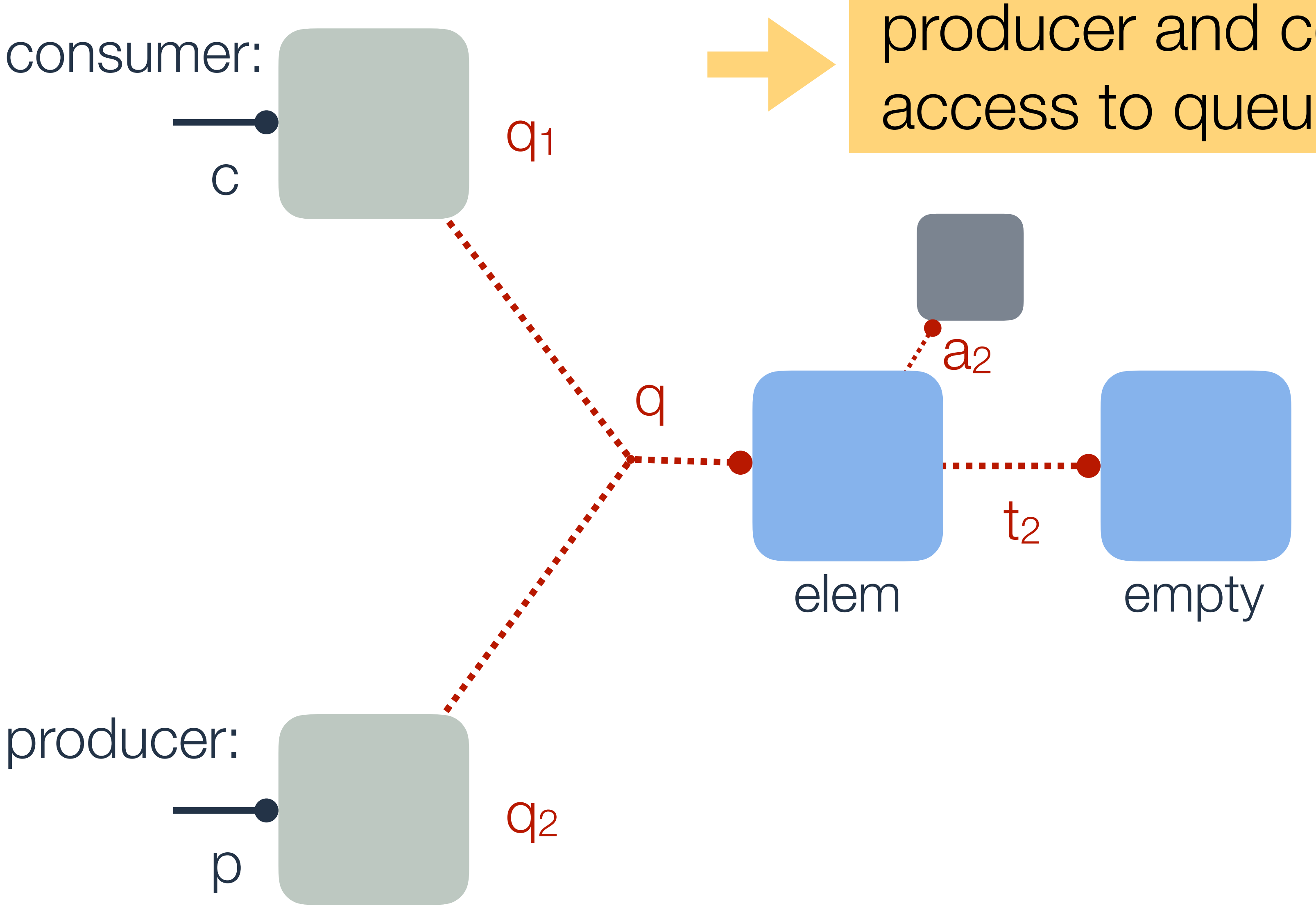
Programmatic working of acquire-release



Legend: linear channel x — shared channel y

Programmatic working of acquire-release

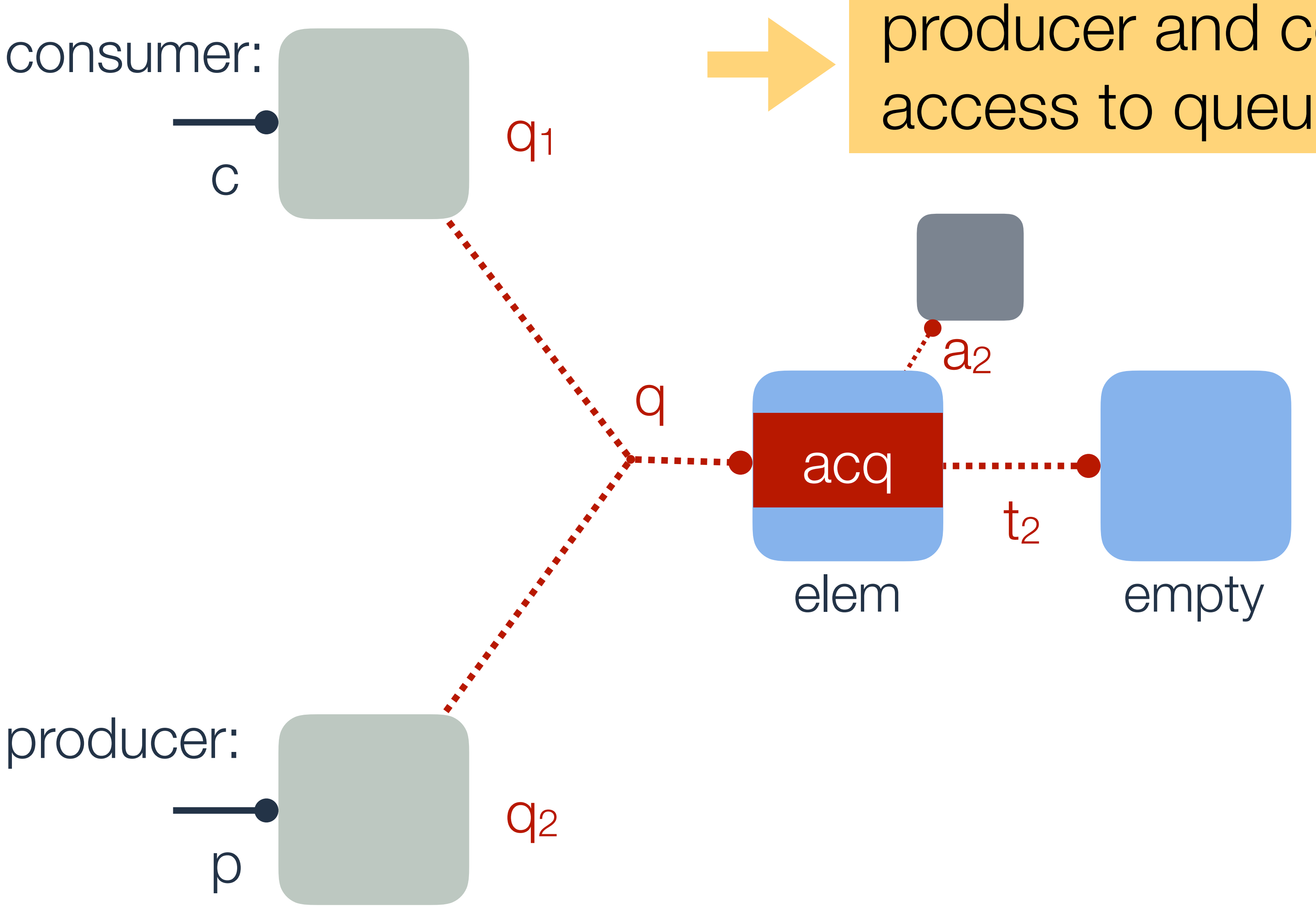
producer and consumer contend for access to queue



Legend: x linear channel x y shared channel y

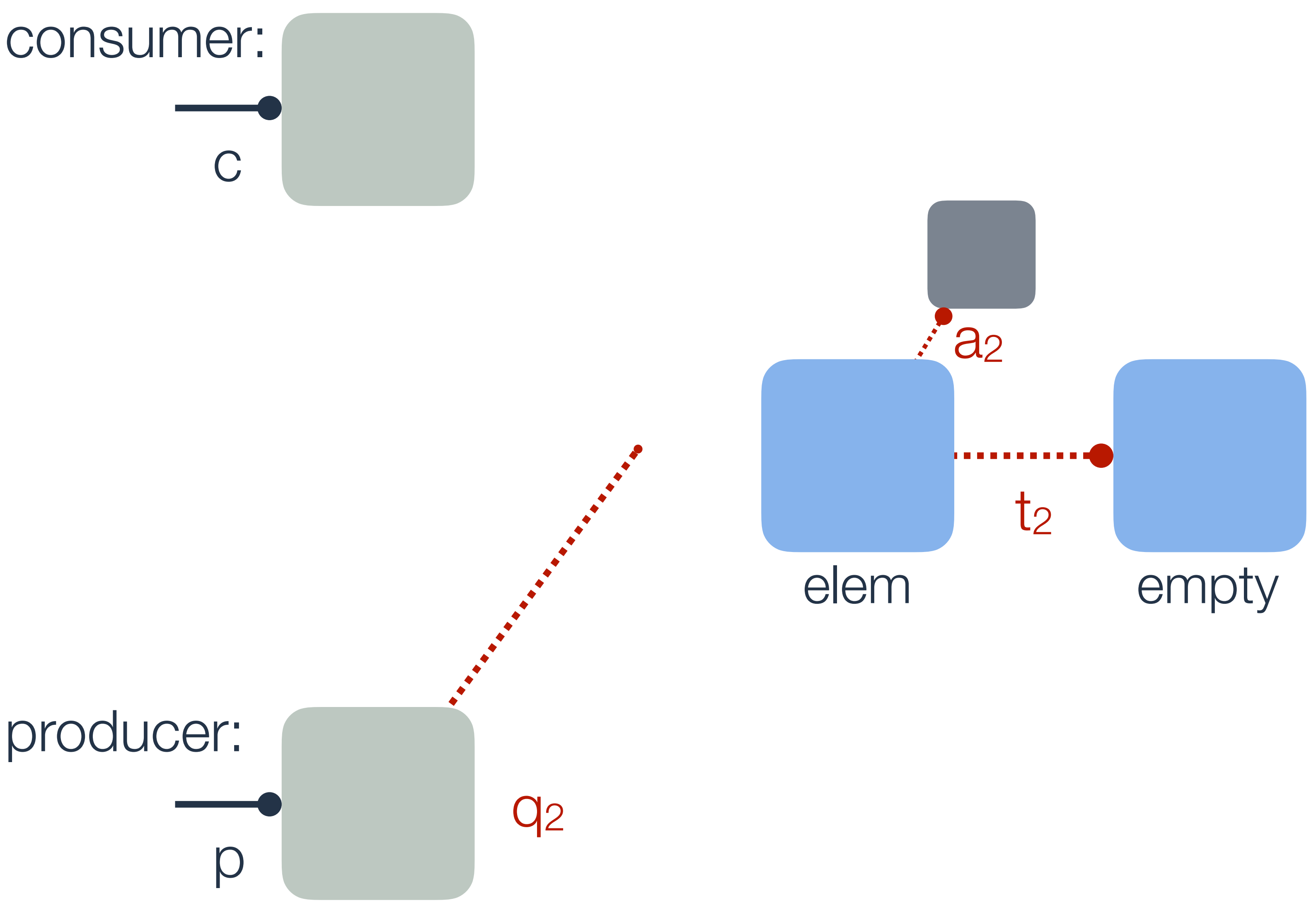
Programmatic working of acquire-release

producer and consumer contend for access to queue



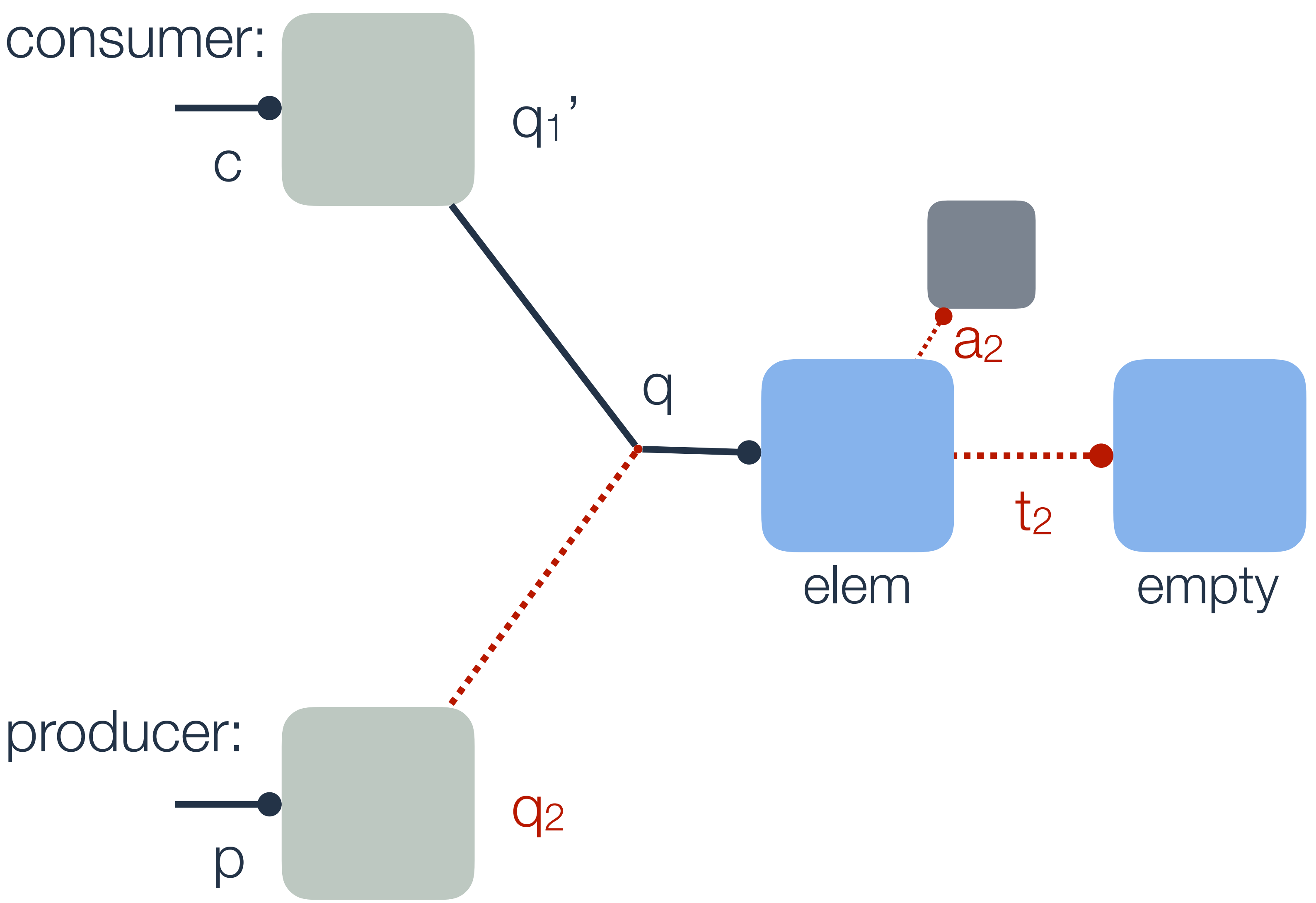
Legend: linear channel x — shared channel y

Programmatic working of acquire-release



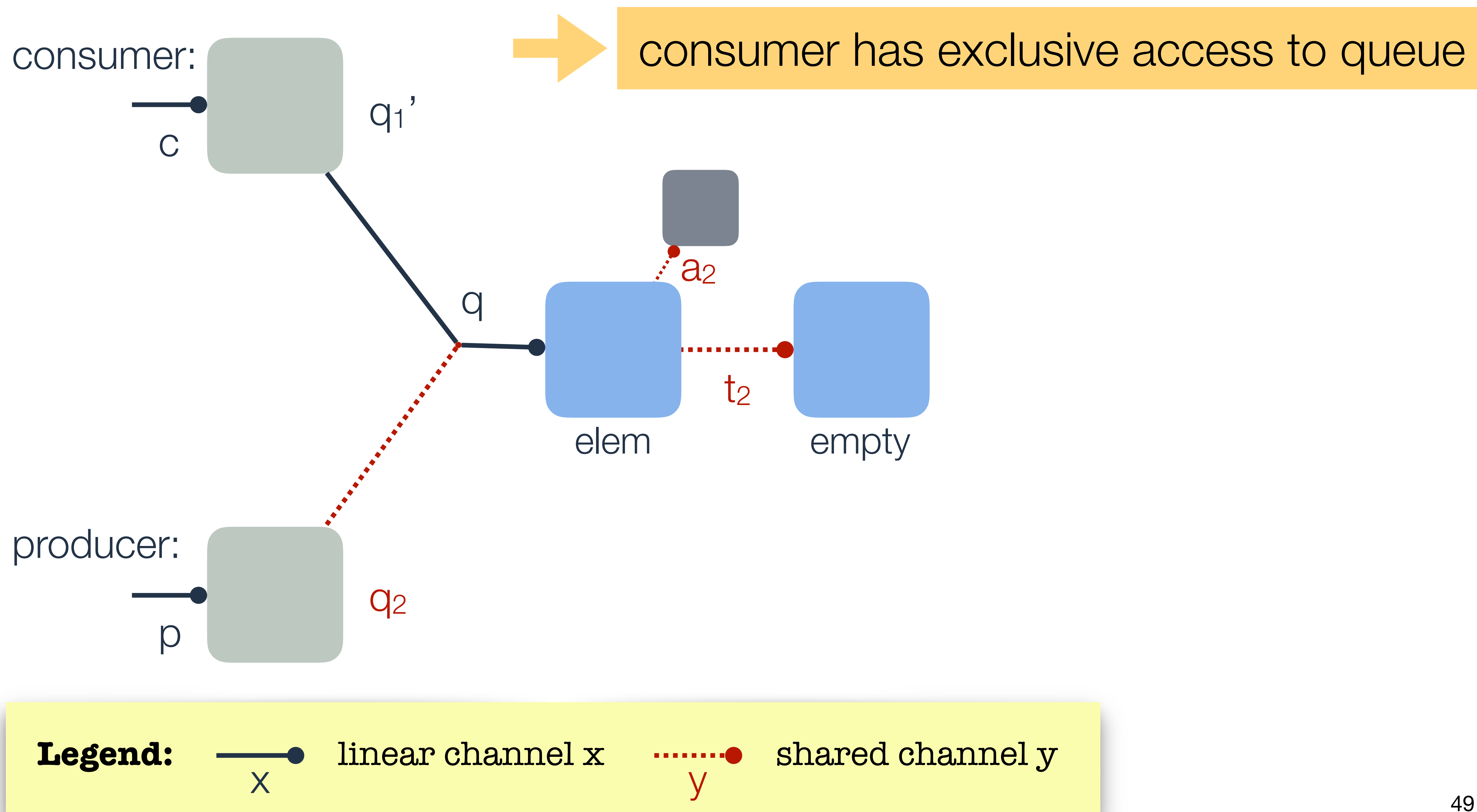
Legend: x linear channel x y shared channel y

Programmatic working of acquire-release



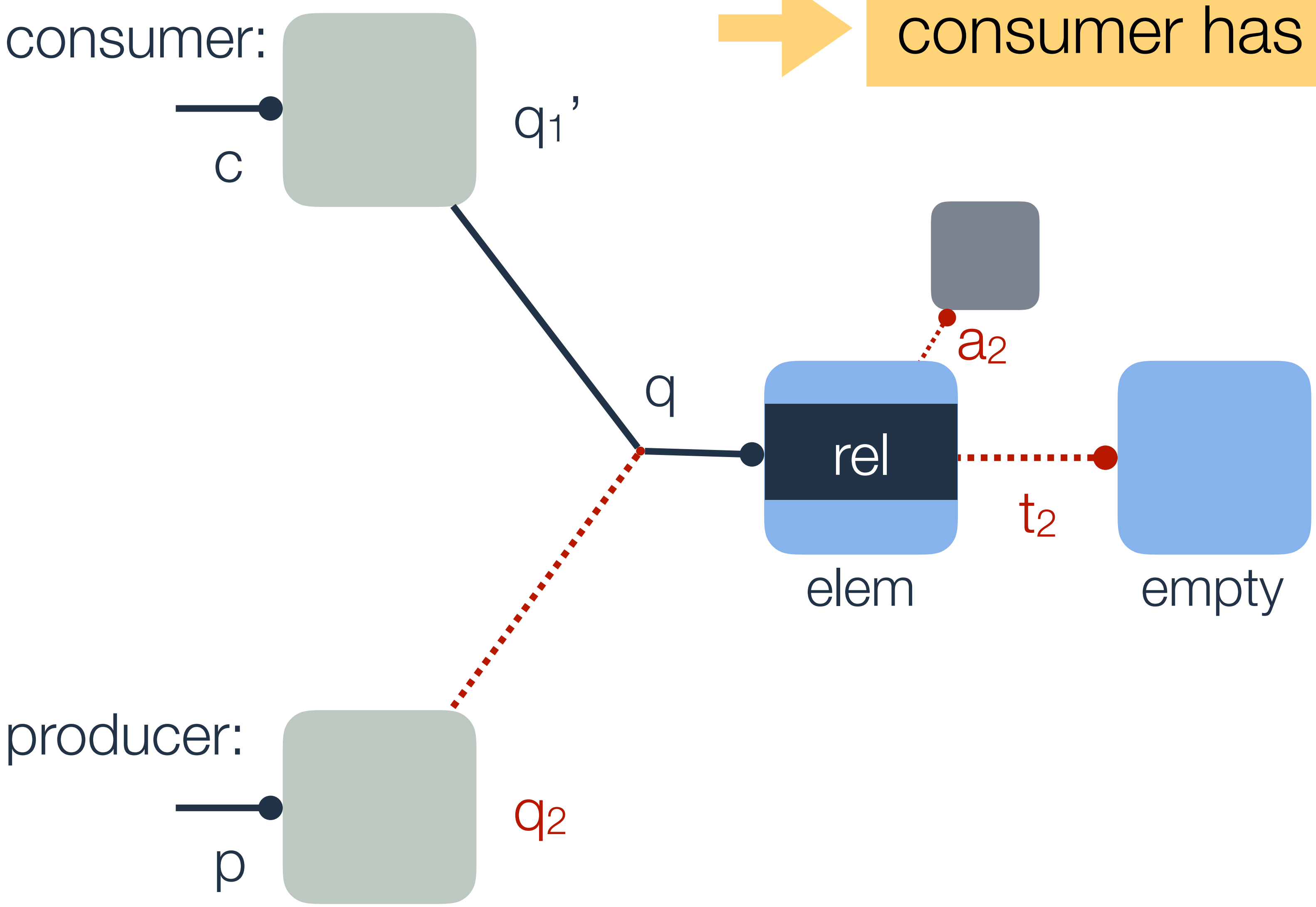
Legend: linear channel x — shared channel y

Programmatic working of acquire-release



Programmatic working of acquire-release

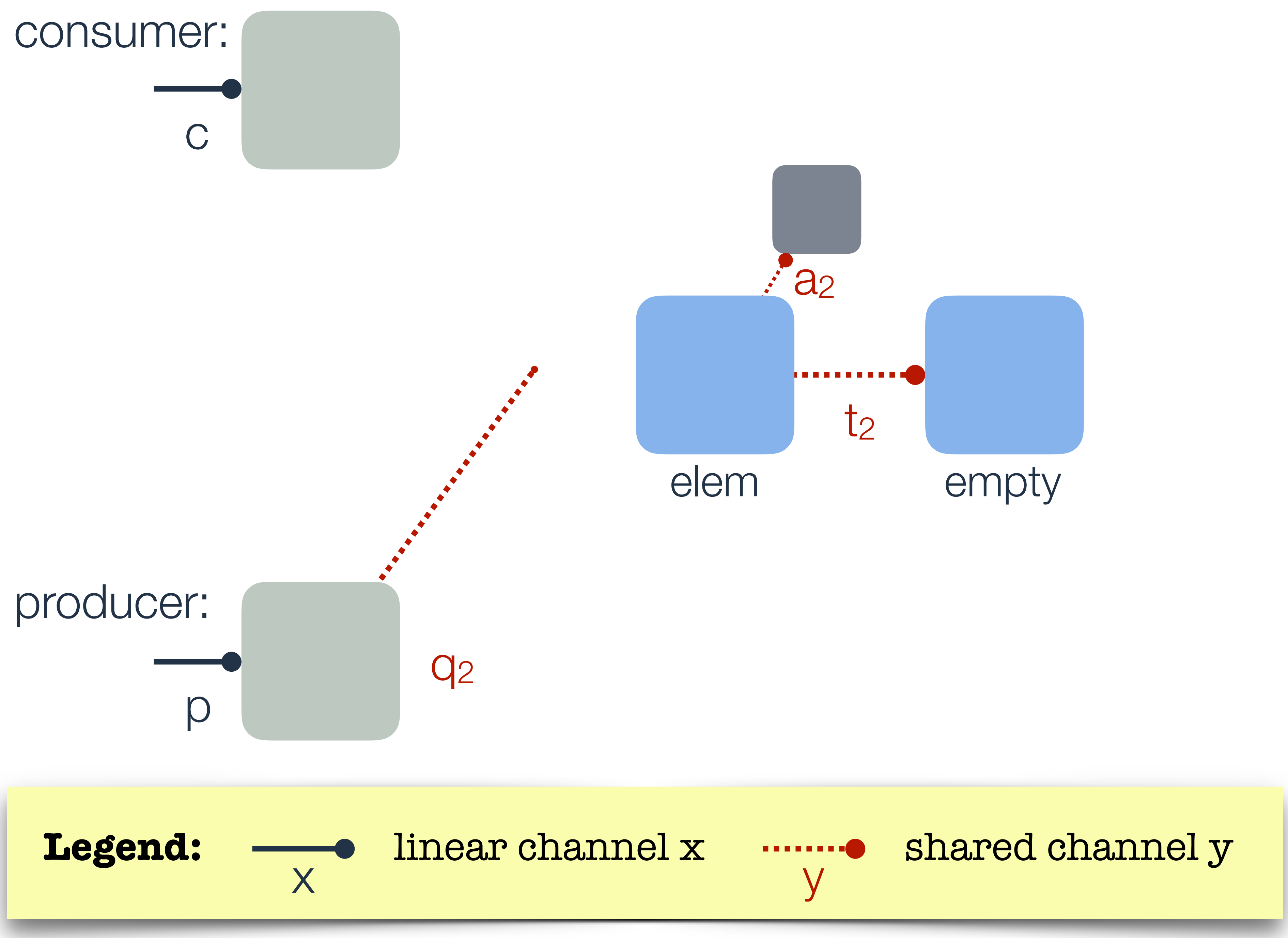
consumer has exclusive access to queue



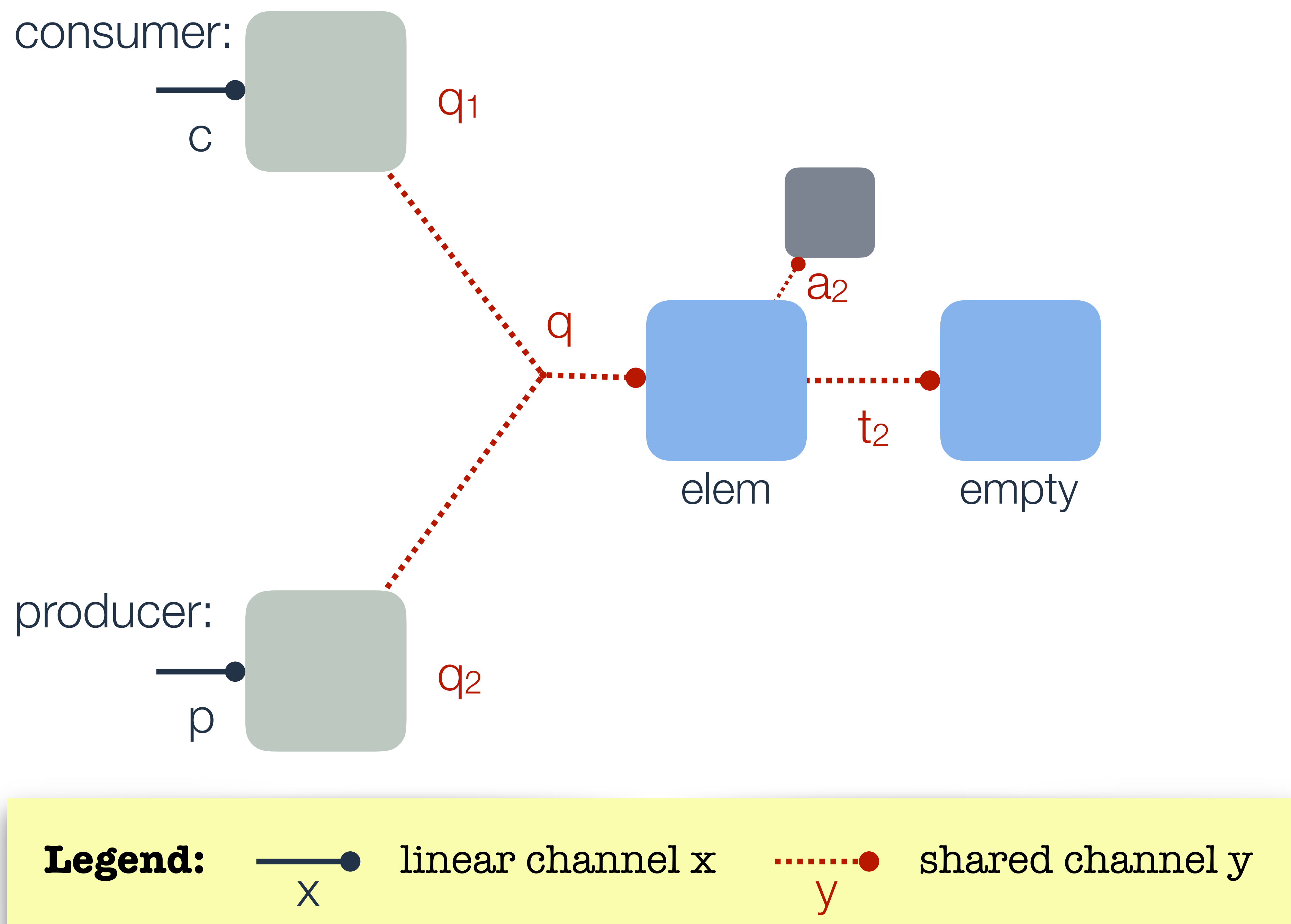
Legend: linear channel x shared channel y

x y

Programmatic working of acquire-release

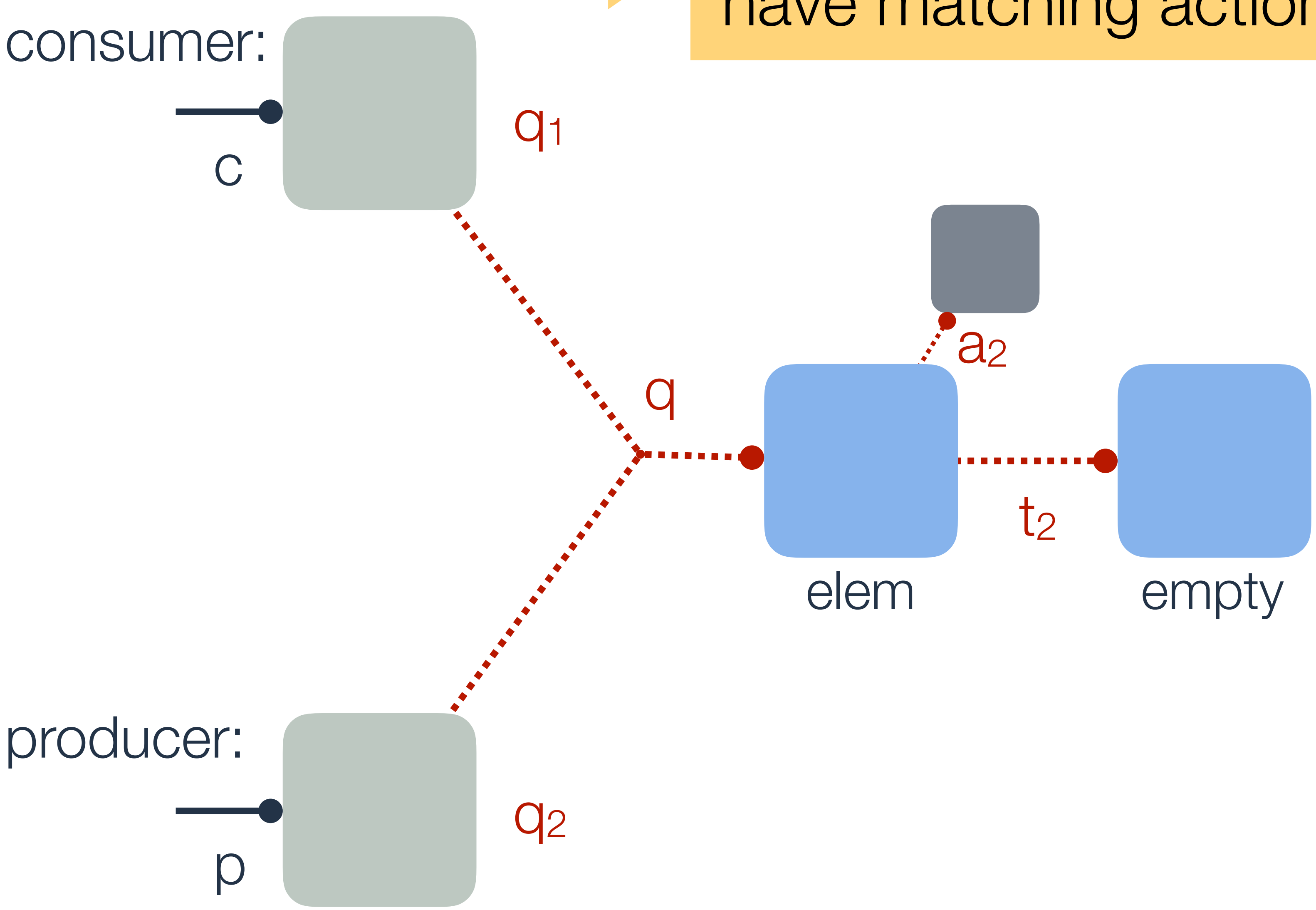


Programmatic working of acquire-release



Programmatic working of acquire-release

Note: acquire and release messages by client have matching actions by provider

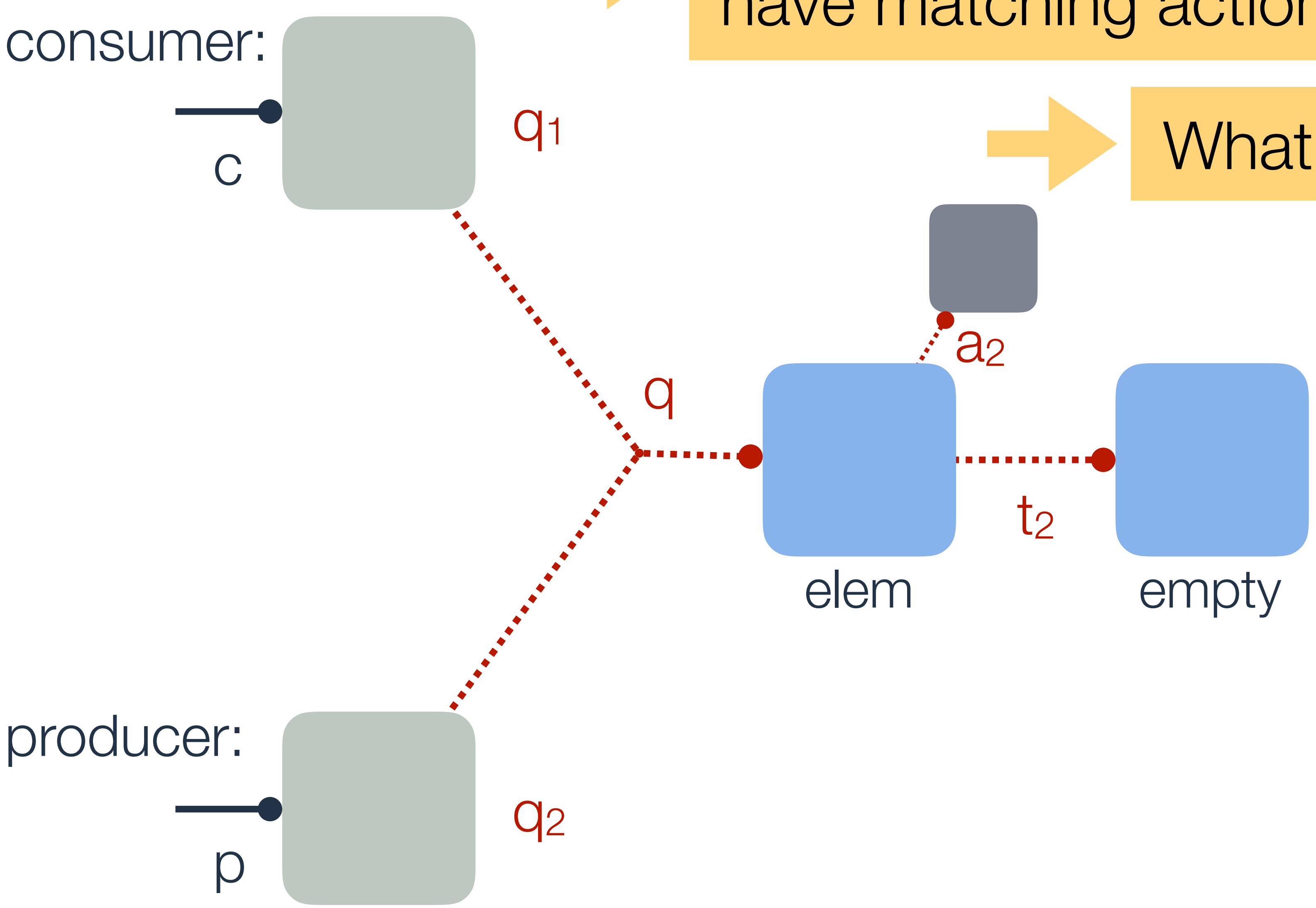


Legend: x linear channel x y shared channel y

Programmatic working of acquire-release

Note: acquire and release messages by client have matching actions by provider

What should be the type of q?



Legend: x linear channel x y shared channel y

Manifestation of acquire-release in typing

Manifestation of acquire-release in typing

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

Manifestation of acquire-release in typing

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

Manifestation of acquire-release in typing

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

Manifestation of acquire-release in typing

acquire

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

Manifestation of acquire-release in typing

acquire release

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

release release

Manifestation of acquire-release in typing

acquire release

$$\mathsf{queue}\ A = \&\{\mathsf{enq} : A \multimap \mathsf{queue}\ A,\ \mathsf{deq} : \oplus\{\mathsf{none} : \mathsf{queue}\ A,\ \mathsf{some} : A \otimes \mathsf{queue}\ A\}\}$$

release release

Legend: process in linear phase process in shared phase

Adjoint stratification of session types[1]

1 Generalization of [Benton 1994; Reed 2009; Pfenning and Griffith 2015]

Adjoint stratification of session types[1]

stratify session types into a linear and shared layer, s.t. S > L

1 Generalization of [Benton 1994; Reed 2009; Pfenning and Griffith 2015]

Adjoint stratification of session types[1]

stratify session types into a linear and shared layer, s.t. S > L

$$A_{\mathsf{S}} \quad \triangleq$$

$$\begin{aligned} A_{\mathsf{L}}, B_{\mathsf{L}} \quad \triangleq \quad & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \\ & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \end{aligned}$$

1 Generalization of [Benton 1994; Reed 2009; Pfenning and Griffith 2015]

Adjoint stratification of session types[1]

stratify session types into a linear and shared layer, s.t. S > L

+

$$A_{\mathsf{S}} \triangleq$$

$$A_{\mathsf{L}}, B_{\mathsf{L}} \triangleq \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \;\&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}}$$

-

1 Generalization of [Benton 1994; Reed 2009; Pfenning and Griffith 2015]

Adjoint stratification of session types[1]

- stratify session types into a linear and shared layer, s.t. S > L
- connect layers with modalities going back and forth

+

$$
\begin{aligned}
A_{\mathsf{S}} &\triangleq \\
A_{\mathsf{L}}, B_{\mathsf{L}} &\triangleq \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \\
&\quad\ \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}}
\end{aligned}
$$

-

1 Generalization of [Benton 1994; Reed 2009; Pfenning and Griffith 2015]

Adjoint stratification of session types[1]

stratify session types into a linear and shared layer, s.t. S > L

connect layers with modalities going back and forth

+

$$
\begin{aligned}
A_{\mathsf{S}} \quad &\triangleq \quad {\uparrow}_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} \\
A_{\mathsf{L}}, B_{\mathsf{L}} \quad &\triangleq \quad \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \\
&\qquad \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid {\downarrow}_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}}
\end{aligned}
$$

-

1 Generalization of [Benton 1994; Reed 2009; Pfenning and Griffith 2015]

Adjoint stratification of session types[1]

- stratify session types into a linear and shared layer, s.t. S > L
- connect layers with modalities going back and forth

$$
\begin{array}{lll}
+ & & \\
A_{\mathsf{S}} & \triangleq & \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} \\
A_{\mathsf{L}}, B_{\mathsf{L}} & \triangleq & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \\
 & & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \\
- & &
\end{array}
$$

- support of communication of shared channels, using Π for receive and $\exists$ for send[2]

1 Generalization of [Benton 1994; Reed 2009; Pfenning and Griffith 2015]

2 Based on [Cervesato et al. 2002; Watkins et al. 2001]

Adjoint stratification of session types[1]

stratify session types into a linear and shared layer, s.t. S > L

connect layers with modalities going back and forth

+

$$A_{\mathsf{S}} \triangleq \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$$

$$\begin{aligned} A_{\mathsf{L}}, B_{\mathsf{L}} \triangleq\ & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\ & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \end{aligned}$$

-

support of communication of shared channels, using Π for receive and $\exists$ for send[2]

1 Generalization of [Benton 1994; Reed 2009; Pfenning and Griffith 2015]

2 Based on [Cervesato et al. 2002; Watkins et al. 2001]

Adjoint stratification of session types[1]

+

$$A_{\mathsf{S}} \triangleq \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$$

$$\begin{aligned} A_{\mathsf{L}}, B_{\mathsf{L}} \triangleq\ & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\ & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \end{aligned}$$

-

Shared queue

+

$$A_{\mathsf{S}} \triangleq \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$$

$$A_{\mathsf{L}}, B_{\mathsf{L}} \triangleq \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid$$
$$\&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}}$$

-

Shared queue

+

$$
\begin{array}{lcl}
A_{\mathsf{S}} & \triangleq & \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} \\
A_{\mathsf{L}}, B_{\mathsf{L}} & \triangleq & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\
 & & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}}
\end{array}
$$

-

$$
\begin{array}{lll}
\mathsf{queue}\ A_{\mathsf{S}} = & \&\{\mathsf{enq} : \Pi x{:}A_{\mathsf{S}}. & \mathsf{queue}\ A_{\mathsf{S}}, \\
 & \quad \mathsf{deq} : \oplus\{\mathsf{none} : & \mathsf{queue}\ A_{\mathsf{S}},\ \mathsf{some} : \exists x{:}A_{\mathsf{S}}.\quad \mathsf{queue}\ A_{\mathsf{S}}\}\}
\end{array}
$$

Shared queue

+

$$
\begin{array}{lcl}
A_{\mathsf{S}} & \triangleq & \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} \\
A_{\mathsf{L}}, B_{\mathsf{L}} & \triangleq & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\
 & & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}}
\end{array}
$$

-

$$
\begin{array}{lll}
\mathsf{queue}\ A_{\mathsf{S}} = & \&\{\mathsf{enq} : \Pi x{:}A_{\mathsf{S}}. & \mathsf{queue}\ A_{\mathsf{S}}, \\
 & \quad \mathsf{deq} : \oplus\{\mathsf{none} : & \mathsf{queue}\ A_{\mathsf{S}},\ \mathsf{some} : \exists x{:}A_{\mathsf{S}}.\ \ \mathsf{queue}\ A_{\mathsf{S}}\}\}
\end{array}
$$

Shared queue

+

$$
\begin{array}{lcl}
A_{\mathsf{S}} & \triangleq & \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} \\
A_{\mathsf{L}}, B_{\mathsf{L}} & \triangleq & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\
 & & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}}
\end{array}
$$

-

$$
\begin{array}{l}
\mathsf{queue}\ A_{\mathsf{S}} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \&\{\mathsf{enq} : \Pi x{:}A_{\mathsf{S}}. \quad \mathsf{queue}\ A_{\mathsf{S}}, \\
\qquad\qquad \mathsf{deq} : \oplus\{\mathsf{none} : \quad \mathsf{queue}\ A_{\mathsf{S}},\ \mathsf{some} : \exists x{:}A_{\mathsf{S}}. \quad \mathsf{queue}\ A_{\mathsf{S}}\}\}
\end{array}
$$

Shared queue

+

$$
\begin{array}{lcl}
A_{\mathsf{S}} & \triangleq & \uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}} \\
A_{\mathsf{L}}, B_{\mathsf{L}} & \triangleq & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\
 & & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}}
\end{array}
$$

-

$$
\begin{array}{l}
\mathsf{queue}\ A_{\mathsf{S}} = \uparrow^{\mathsf{S}}_{\mathsf{L}} \&\{\mathsf{enq} : \Pi x{:}A_{\mathsf{S}}.\ \downarrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{queue}\ A_{\mathsf{S}}, \\
\qquad\qquad\qquad \mathsf{deq} : \oplus\{\mathsf{none} : \downarrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{queue}\ A_{\mathsf{S}},\ \mathsf{some} : \exists x{:}A_{\mathsf{S}}.\ \downarrow^{\mathsf{S}}_{\mathsf{L}} \mathsf{queue}\ A_{\mathsf{S}}\}\}
\end{array}
$$

Shared queue

+

-

$$
\begin{array}{lcl}
A_{\mathsf{S}} & \triangleq & \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} \\
A_{\mathsf{L}}, B_{\mathsf{L}} & \triangleq & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\
 & & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}}
\end{array}
$$

$$
\begin{array}{l}
\mathsf{queue}\ A_{\mathsf{S}} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \&\{\mathsf{enq} : \Pi x{:}A_{\mathsf{S}}.\ \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{queue}\ A_{\mathsf{S}}, \\
\qquad \mathsf{deq} : \oplus\{\mathsf{none} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{queue}\ A_{\mathsf{S}},\ \mathsf{some} : \exists x{:}A_{\mathsf{S}}.\ \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{queue}\ A_{\mathsf{S}}\}\}
\end{array}
$$

$\uparrow_{\mathsf{L}}^{\mathsf{S}}$ and $\downarrow_{\mathsf{L}}^{\mathsf{S}}$ prescribe acquire-release points, resp.

Equi-synchronizing session type

Equi-synchronizing session type

Invariant guaranteeing that a process is released to the same type at which it was previously acquired, should it be released at all.

Equi-synchronizing session type

Invariant guaranteeing that a process is released to the same type at which it was previously acquired, should it be released at all.

➡ Guarantees session fidelity w/o run-time checking upon acquire

Equi-synchronizing session type

Invariant guaranteeing that a process is released to the same type at which it was previously acquired, should it be released at all.

Guarantees session fidelity w/o run-time checking upon acquire

Examples of equi-synchronizing session types:

$$\begin{aligned}\mathsf{queue}\ A_{\mathsf{S}} = {\uparrow}^{\mathsf{S}}_{\mathsf{L}}\ \&\{&\mathsf{enq} : \Pi x{:}A_{\mathsf{S}}.\ {\downarrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{queue}\ A_{\mathsf{S}},\\ &\mathsf{deq} : \oplus\{\mathsf{none} : {\downarrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{queue}\ A_{\mathsf{S}},\ \mathsf{some} : \exists x{:}A_{\mathsf{S}}.\ {\downarrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathsf{queue}\ A_{\mathsf{S}}\}\}\end{aligned}$$

Equi-synchronizing session type

Invariant guaranteeing that a process is released to the same type at which it was previously acquired, should it be released at all.

Guarantees session fidelity w/o run-time checking upon acquire

Examples of equi-synchronizing session types:

$$\begin{aligned}\mathsf{queue}\ A_{\mathsf{S}} &= \uparrow_{\mathsf{L}}^{\mathsf{S}} \&\{\mathsf{enq} : \Pi x{:}A_{\mathsf{S}}.\ \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{queue}\ A_{\mathsf{S}}, \\ &\qquad\qquad \mathsf{deq} : \oplus\{\mathsf{none} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{queue}\ A_{\mathsf{S}},\ \mathsf{some} : \exists x{:}A_{\mathsf{S}}.\ \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{queue}\ A_{\mathsf{S}}\}\}\end{aligned}$$

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

Equi-synchronizing session type

Invariant guaranteeing that a process is released to the same type at which it was previously acquired, should it be released at all.

Guarantees session fidelity w/o run-time checking upon acquire

Examples of equi-synchronizing session types:

$$\begin{aligned}\mathsf{queue}\ A_{\mathsf{S}} = {\uparrow}_{\mathsf{L}}^{\mathsf{S}}\, \&\{&\mathsf{enq} : \Pi x{:}A_{\mathsf{S}}.\ {\downarrow}_{\mathsf{L}}^{\mathsf{S}}\,\mathsf{queue}\ A_{\mathsf{S}},\\ &\mathsf{deq} : \oplus\{\mathsf{none} : {\downarrow}_{\mathsf{L}}^{\mathsf{S}}\,\mathsf{queue}\ A_{\mathsf{S}},\ \mathsf{some} : \exists x{:}A_{\mathsf{S}}.\ {\downarrow}_{\mathsf{L}}^{\mathsf{S}}\,\mathsf{queue}\ A_{\mathsf{S}}\}\}\end{aligned}$$

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

Non-equi-synchronizing:

$$\begin{aligned}\mathsf{queue}\ A_{\mathsf{S}} = {\uparrow}_{\mathsf{L}}^{\mathsf{S}}\, \&\{&\mathsf{enq} : \Pi x{:}A_{\mathsf{S}}.\ {\downarrow}_{\mathsf{L}}^{\mathsf{S}}\,\mathsf{queue}\ A_{\mathsf{S}},\\ &\mathsf{deq} : \oplus\{\mathsf{none} : {\downarrow}_{\mathsf{L}}^{\mathsf{S}}{\uparrow}_{\mathsf{L}}^{\mathsf{S}}\,\mathbf{1},\ \mathsf{some} : \exists x{:}A_{\mathsf{S}}.\ {\downarrow}_{\mathsf{L}}^{\mathsf{S}}\,\mathsf{queue}\ A_{\mathsf{S}}\}\}\end{aligned}$$

Typing judgments

Typing judgments

$$
\begin{array}{lcl}
A_{\mathsf{S}} & \triangleq & \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} \\
A_{\mathsf{L}}, B_{\mathsf{L}} & \triangleq & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\
 & & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}}
\end{array}
$$

+

-

Typing judgments

+

-

$$
\begin{array}{lll}
A_{\mathsf{S}} & \triangleq & \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} \\
A_{\mathsf{L}}, B_{\mathsf{L}} & \triangleq & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\
 & & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}}
\end{array}
$$

$\Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$	shared process P, providing session of type A_{S} along x_{S}, using channels in Γ
$\Gamma;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$	linear process P, providing session of type A_{L} along x_{L}, using channels in Γ and Δ
Γ	shared (structural) context
Δ	linear context

Typing judgments

+

$$
\begin{array}{lcl}
A_{\mathsf{S}} & \triangleq & \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}} \\
A_{\mathsf{L}}, B_{\mathsf{L}} & \triangleq & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\
 & & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}}
\end{array}
$$

-

$\Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$ — shared process P, providing session of type A_{S} along x_{S}, using channels in Γ

$\Gamma;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$ — linear process P, providing session of type A_{L} along x_{L}, using channels in Γ and Δ

Γ — shared (structural) context

Δ — linear context

Typing judgments

+

$$A_{\mathsf{S}} \triangleq {\uparrow}_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$$

$$\begin{aligned} A_{\mathsf{L}}, B_{\mathsf{L}} \triangleq\ & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\ & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid {\downarrow}_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \end{aligned}$$

-

$\Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$	shared process P, providing session of type A_{S} along x_{S}, using channels in Γ
$\Gamma;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$	linear process P, providing session of type A_{L} along x_{L}, using channels in Γ and Δ
Γ	shared (structural) context
Δ	linear context

Typing judgments

+	A_{S}	$\triangleq$	$\uparrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{L}}$
	$A_{\mathsf{L}}, B_{\mathsf{L}}$	$\triangleq$	$\oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid$
-			$\&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}}$

$\Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$	shared process P, providing session of type A_{S} along x_{S}, using channels in Γ
$\Gamma;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$	linear process P, providing session of type A_{L} along x_{L}, using channels in Γ and Δ
Γ	shared (structural) context
Δ	linear context

Typing judgments

+

A_S	$\triangleq$	$\uparrow_L^S A_L$
A_L, B_L	$\triangleq$	$\oplus\{\overline{l : A_L}\} \mid A_L \otimes B_L \mid \mathbf{1} \mid \exists x{:}A_S.\, B_L \mid$
		$\&\{\overline{l : A_L}\} \mid A_L \multimap B_L \mid \downarrow_L^S A_S \mid \Pi x{:}A_S.\, B_L$

-

$\Gamma \vdash_\Sigma P :: (x_S : A_S)$ shared process P, providing session of type A_S along x_S, using channels in Γ

$\Gamma;\ \Delta \vdash_\Sigma P :: (x_L : A_L)$ linear process P, providing session of type A_L along x_L, using channels in Γ and Δ

Γ shared (structural) context

Δ linear context

Typing judgments

+

$$A_{\mathsf{S}} \triangleq \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}$$

$$\begin{aligned} A_{\mathsf{L}}, B_{\mathsf{L}} \triangleq\ & \oplus\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \otimes B_{\mathsf{L}} \mid \mathbf{1} \mid \exists x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \mid \\ & \&\{\overline{l : A_{\mathsf{L}}}\} \mid A_{\mathsf{L}} \multimap B_{\mathsf{L}} \mid \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \mid \Pi x{:}A_{\mathsf{S}}.\, B_{\mathsf{L}} \end{aligned}$$

-

$\Gamma \vdash_{\Sigma} P :: (x_{\mathsf{S}} : A_{\mathsf{S}})$	shared process P, providing session of type A_{S} along x_{S}, using channels in Γ
$\Gamma;\ \Delta \vdash_{\Sigma} P :: (x_{\mathsf{L}} : A_{\mathsf{L}})$	linear process P, providing session of type A_{L} along x_{L}, using channels in Γ and Δ
Γ	shared (structural) context
Δ	linear context

Typing of acquire

Typing of acquire

$$\frac{\Gamma, x_{\mathsf{S}} :\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}};\ \Delta, x_{\mathsf{L}} : A_{\mathsf{L}} \vdash_{\Sigma} Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma, x_{\mathsf{S}} :\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}};\ \Delta \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}}\ ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\mathrm{T}\text{-}{\uparrow_{\mathsf{L}}^{\mathsf{S}}}\mathrm{L})$$

Typing of acquire

$$\frac{\Gamma, \boxed{x_{\mathsf{S}} :\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}};\ \Delta, \boxed{x_{\mathsf{L}} : A_{\mathsf{L}}} \vdash_{\Sigma} Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma, \boxed{x_{\mathsf{S}} :\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}}};\ \Delta \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}}\ ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\text{T-}\uparrow_{\mathsf{L}}^{\mathsf{S}}\text{L})$$

Typing of acquire

$$\frac{\Gamma, x_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}};\ \Delta, x_{\mathsf{L}} : A_{\mathsf{L}} \vdash_{\Sigma} Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma, x_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}};\ \Delta \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}}\ ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\mathrm{T}\text{-}\uparrow_{\mathsf{L}}^{\mathsf{S}}\mathrm{L})$$

$$\frac{\Gamma;\ \cdot \vdash_{\Sigma} P_{x_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})}{\Gamma \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{accept}\ x_{\mathsf{S}}\ ; P_{x_{\mathsf{L}}} :: (x_{\mathsf{S}} : \uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}})}\ (\mathrm{T}\text{-}\uparrow_{\mathsf{L}}^{\mathsf{S}}\mathrm{R})$$

Typing of acquire

$$\frac{\Gamma, x_{\mathsf{S}} :\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}};\ \Delta, x_{\mathsf{L}} : A_{\mathsf{L}} \vdash_{\Sigma} Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma, x_{\mathsf{S}} :\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}};\ \Delta \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{acquire}\ x_{\mathsf{S}}\ ; Q_{x_{\mathsf{L}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\mathrm{T}\text{-}{\uparrow_{\mathsf{L}}^{\mathsf{S}}}\mathrm{L})$$

$$\frac{\Gamma;\ \cdot \vdash_{\Sigma} P_{x_{\mathsf{L}}} :: (x_{\mathsf{L}} : A_{\mathsf{L}})}{\Gamma \vdash_{\Sigma} x_{\mathsf{L}} \leftarrow \mathsf{accept}\ x_{\mathsf{S}}\ ; P_{x_{\mathsf{L}}} :: (x_{\mathsf{S}} :\uparrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{L}})}\ (\mathrm{T}\text{-}{\uparrow_{\mathsf{L}}^{\mathsf{S}}}\mathrm{R})$$

Typing of release

Typing of release

$$\frac{\Gamma, x_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_{\Sigma} Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : \downarrow^{\mathsf{S}}_{\mathsf{L}} A_{\mathsf{S}} \vdash_{\Sigma} x_{\mathsf{S}} \leftarrow \mathsf{release}\ x_{\mathsf{L}}\ ; Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\mathrm{T}\text{-}\downarrow^{\mathsf{S}}_{\mathsf{L}}\mathrm{L})$$

Typing of release

$$\frac{\Gamma, x_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_{\Sigma} Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \vdash_{\Sigma} x_{\mathsf{S}} \leftarrow \mathsf{release}\ x_{\mathsf{L}}\ ; Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\mathrm{T}\text{-}\downarrow_{\mathsf{L}}^{\mathsf{S}}\mathrm{L})$$

Typing of release

$$\frac{\Gamma, x_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_{\Sigma} Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \vdash_{\Sigma} x_{\mathsf{S}} \leftarrow \mathsf{release}\ x_{\mathsf{L}}\ ; Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\mathrm{T}\text{-}{\downarrow_{\mathsf{L}}^{\mathsf{S}}}\mathrm{L})$$

$$\frac{\Gamma \vdash_{\Sigma} P_{x_{\mathsf{S}}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})}{\Gamma;\ \cdot \vdash_{\Sigma} x_{\mathsf{S}} \leftarrow \mathsf{detach}\ x_{\mathsf{L}}\ ; P_{x_{\mathsf{S}}} :: (x_{\mathsf{L}} : \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}})}\ (\mathrm{T}\text{-}{\downarrow_{\mathsf{L}}^{\mathsf{S}}}\mathrm{R})$$

Typing of release

$$\frac{\Gamma, x_{\mathsf{S}} : A_{\mathsf{S}};\ \Delta \vdash_{\Sigma} Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}{\Gamma;\ \Delta, x_{\mathsf{L}} : \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}} \vdash_{\Sigma} x_{\mathsf{S}} \leftarrow \mathsf{release}\ x_{\mathsf{L}}\ ; Q_{x_{\mathsf{S}}} :: (z_{\mathsf{L}} : C_{\mathsf{L}})}\ (\mathrm{T}\text{-}{\downarrow_{\mathsf{L}}^{\mathsf{S}}}\mathrm{L})$$

$$\frac{\Gamma \vdash_{\Sigma} P_{x_{\mathsf{S}}} :: (x_{\mathsf{S}} : A_{\mathsf{S}})}{\Gamma;\ \cdot \vdash_{\Sigma} x_{\mathsf{S}} \leftarrow \mathsf{detach}\ x_{\mathsf{L}}\ ; P_{x_{\mathsf{S}}} :: (x_{\mathsf{L}} : \downarrow_{\mathsf{L}}^{\mathsf{S}} A_{\mathsf{S}})}\ (\mathrm{T}\text{-}{\downarrow_{\mathsf{L}}^{\mathsf{S}}}\mathrm{R})$$

Let’s program in Concurrent C0!

What about type safety?

What about type safety?

Preservation (aka session fidelity)

What about type safety?

Preservation (aka session fidelity)

- acquire-release and equi-synchronizing invariant guarantee that every linear process has a unique client

What about type safety?

Preservation (aka session fidelity)

- acquire-release and equi-synchronizing invariant guarantee that every linear process has a unique client
- state-altering exchange only along linear channels

What about type safety?

Preservation (aka session fidelity) ✓

- acquire-release and equi-synchronizing invariant guarantee that every linear process has a unique client
- state-altering exchange only along linear channels

What about type safety?

Preservation (aka session fidelity) ✓

- acquire-release and equi-synchronizing invariant guarantee that every linear process has a unique client
- state-altering exchange only along linear channels

Progress

What about type safety?

Preservation (aka session fidelity) ✓

- acquire-release and equi-synchronizing invariant guarantee that every linear process has a unique client
- state-altering exchange only along linear channels

Progress

- acquire-release amounts to “locking”

What about type safety?

Preservation (aka session fidelity) ✓

- acquire-release and equi-synchronizing invariant guarantee that every linear process has a unique client
- state-altering exchange only along linear channels

Progress

- acquire-release amounts to “locking”
- possibility of cyclic dependencies

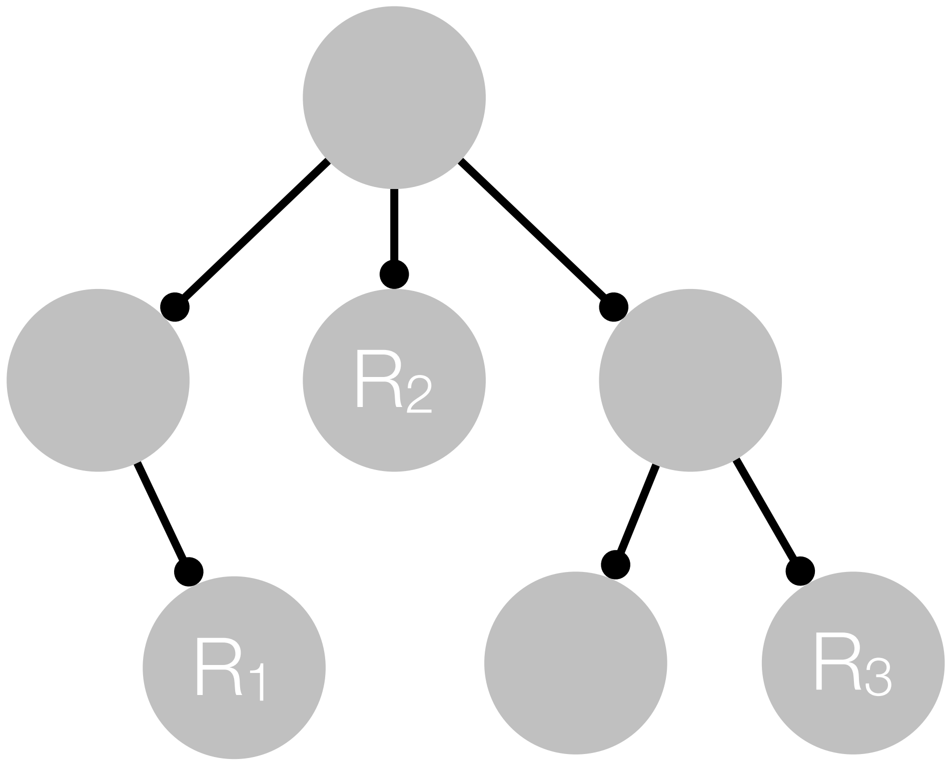
What about type safety?

Preservation (aka session fidelity) ✓

- acquire-release and equi-synchronizing invariant guarantee that every linear process has a unique client
- state-altering exchange only along linear channels

Progress

- acquire-release amounts to “locking”
- possibility of cyclic dependencies



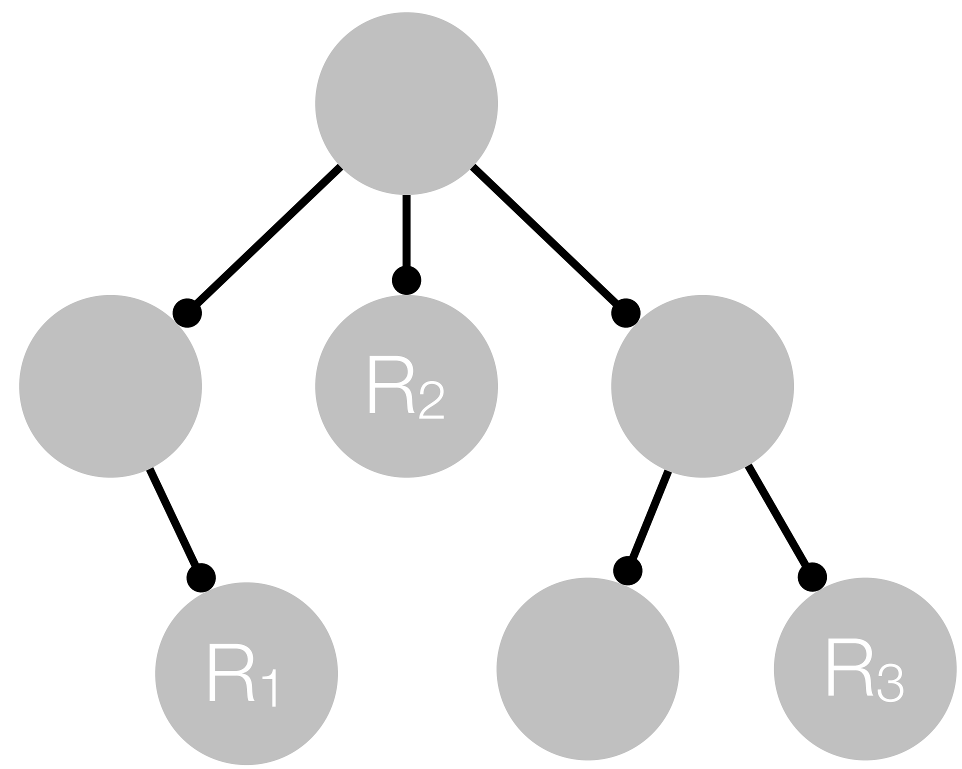
What about type safety?

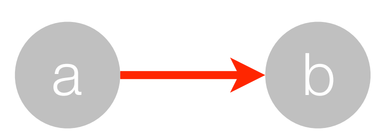
Preservation (aka session fidelity) ✓

- acquire-release and equi-synchronizing invariant guarantee that every linear process has a unique client
- state-altering exchange only along linear channels

Progress

- acquire-release amounts to “locking”
- possibility of cyclic dependencies





“a waits for b to release resource”

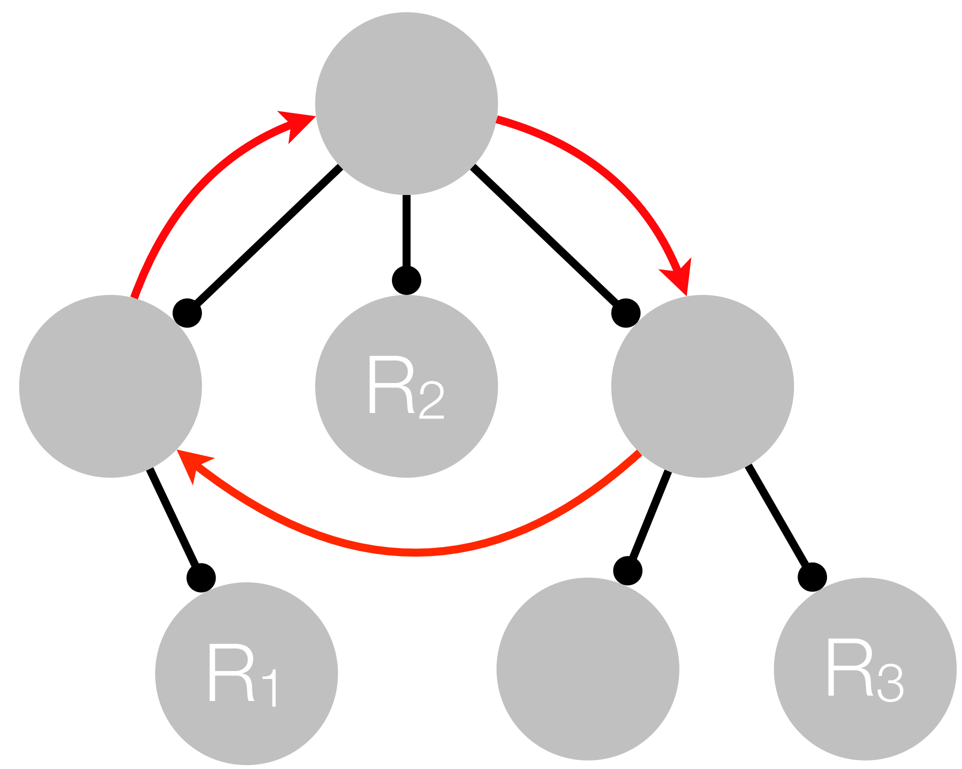
What about type safety?

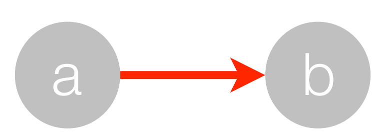
Preservation (aka session fidelity) ✓

- acquire-release and equi-synchronizing invariant guarantee that every linear process has a unique client
- state-altering exchange only along linear channels

Progress

- acquire-release amounts to “locking”
- possibility of cyclic dependencies





“a waits for b to release resource”

What about type safety?

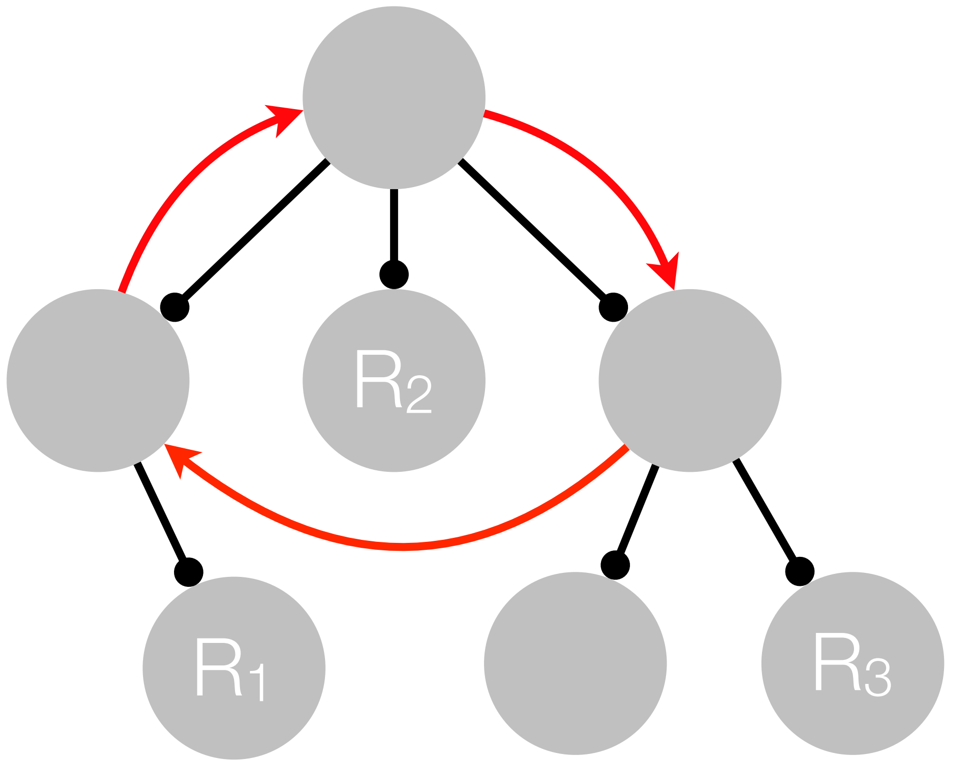
Preservation (aka session fidelity) ✓

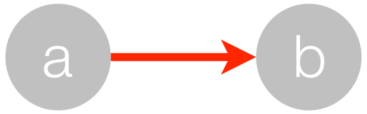
- acquire-release and equi-synchronizing invariant guarantee that every linear process has a unique client
- state-altering exchange only along linear channels

Progress

- acquire-release amounts to "locking"
- possibility of cyclic dependencies

progress must permit deadlock



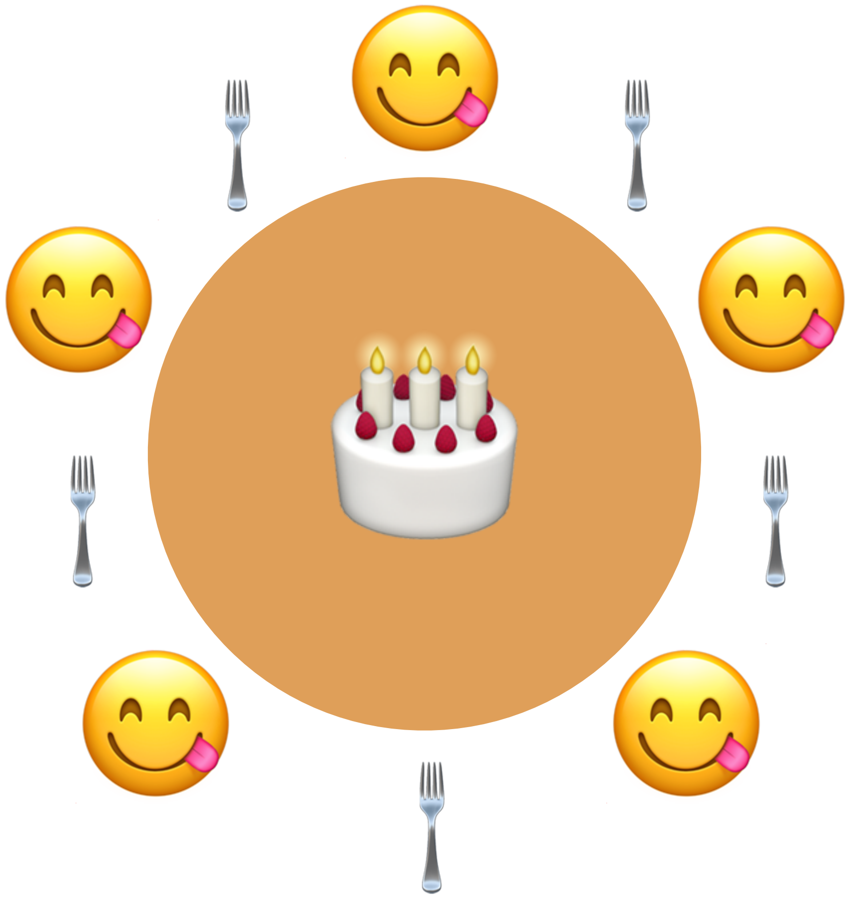


"a waits for b to release resource"

Progress must permit deadlock

Progress must permit deadlock

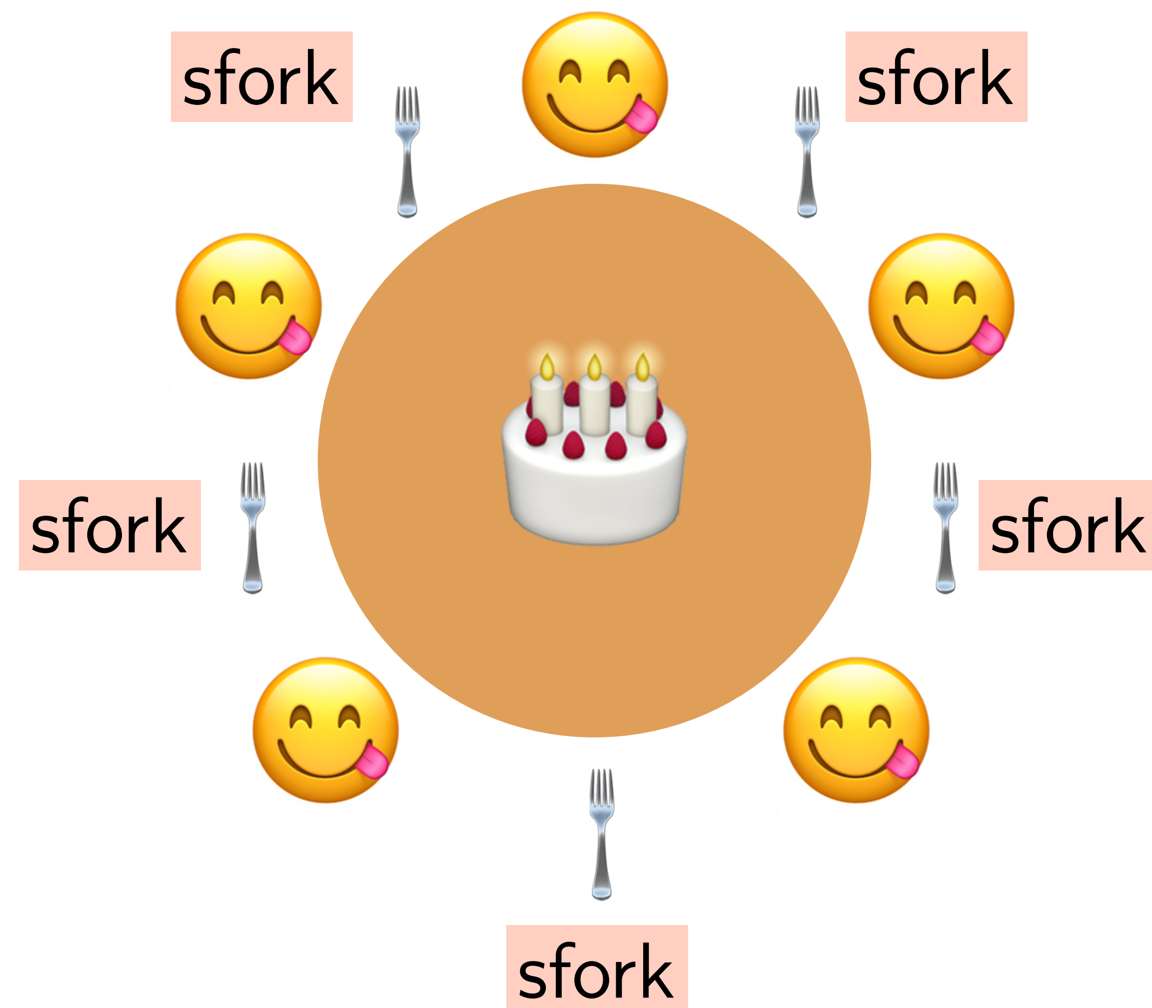
For example, dining philosophers:



Progress must permit deadlock

For example, dining philosophers:

$\mathsf{sfork} = {\uparrow}^{S}_{L}{\downarrow}^{S}_{L}\ \mathsf{sfork}$

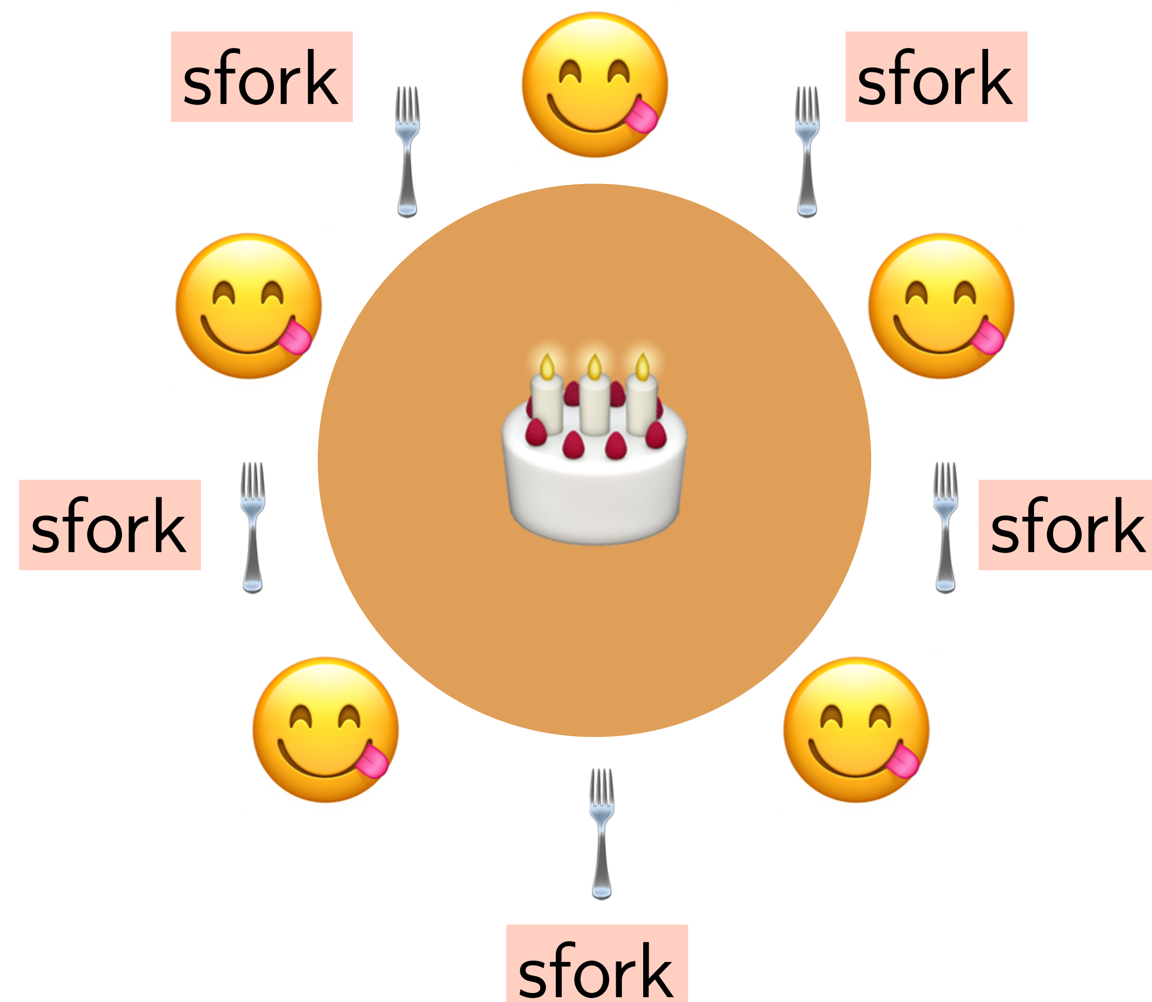


Progress must permit deadlock

For example, dining philosophers:

$\text{sfork} = \uparrow^{S}_{L} \downarrow^{S}_{L} \text{sfork}$

Both potentially deadlocking and non-deadlocking version (Dijkstra's solution) encodable in SILL$_S$

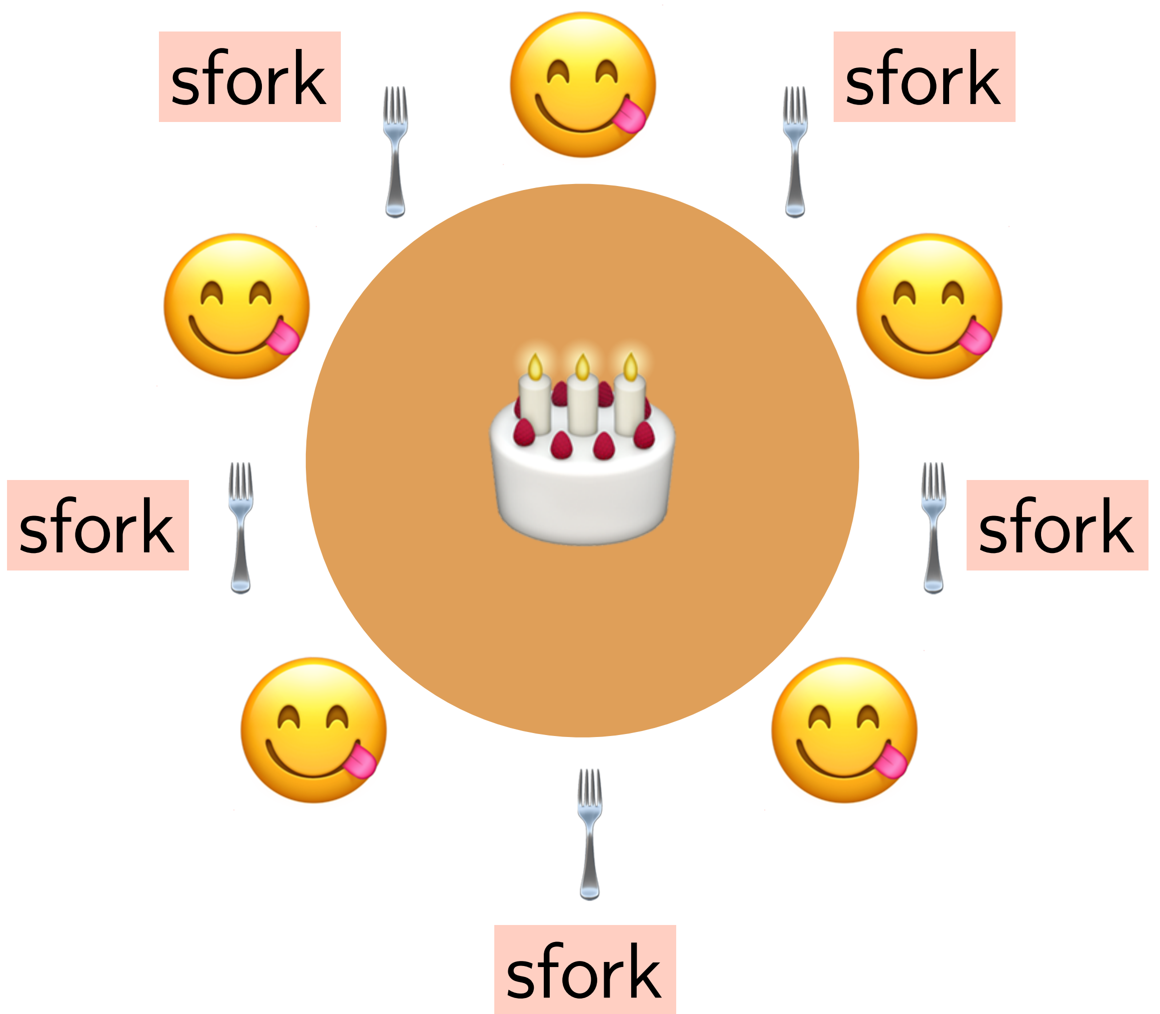


Progress must permit deadlock

For example, dining philosophers:

$\mathsf{sfork} = {\uparrow}^{S}_{L}{\downarrow}^{S}_{L}\,\mathsf{sfork}$

Both potentially deadlocking and non-deadlocking version (Dijkstra's solution) encodable in SILLs



Progress theorem:

blocked process: linear process attempting to acquire

configuration is stuck only if all processes are blocked

Curry-Howard isomorphism revisited

Curry-Howard isomorphism revisited

Linear session types w/o manifest sharing:

- linear propositions — session types
- proofs — processes
- cut reduction — communication

Curry-Howard isomorphism revisited

Linear session types w/o manifest sharing:

- linear propositions — session types
- proofs — processes
- cut reduction — communication

Session types with manifest sharing:

- linear propositions — session types
- proofs — processes
- interleaving of:
  - proof construction —acquire
  - proof reduction — communication
  - proof deconstruction — release

Curry-Howard isomorphism revisited

Linear session types w/o manifest sharing:

- linear propositions — session types
- proofs — processes
- cut reduction — communication

Session types with manifest sharing:

- linear propositions — session types
- proofs — processes
- interleaving of:
  - proof construction —acquire
  - proof reduction — communication
  - proof deconstruction — release

→ deadlock: failure of proof construction

Recovering expressiveness of pi-calculus for session-typed concurrency

Expressiveness of linear session types

Expressiveness of linear session types

- In the simply-typed λ-calculus, the addition of a recursive type $\mathcal{U} = \mathcal{U} \to \mathcal{U}$ recovers full expressiveness of the untyped λ-calculus.

Expressiveness of linear session types

- In the simply-typed λ-calculus, the addition of a recursive type $\mathcal{U} = \mathcal{U} \to \mathcal{U}$ recovers full expressiveness of the untyped λ-calculus.
- Unfortunately, recursion alone is insufficient to recover the expressiveness of the untyped π-calculus for linear session-typed π-calculus.

Expressiveness of linear session types

- In the simply-typed λ-calculus, the addition of a recursive type $\mathcal{U} = \mathcal{U} \to \mathcal{U}$ recovers full expressiveness of the untyped λ-calculus.
- Unfortunately, recursion alone is insufficient to recover the expressiveness of the untyped π-calculus for linear session-typed π-calculus.

→ both recursion and sharing are necessary
[Balzer & Pfenning & Toninho 2018]

Expressiveness of linear session types

- In the simply-typed λ-calculus, the addition of a recursive type $\mathcal{U} = \mathcal{U} \to \mathcal{U}$ recovers full expressiveness of the untyped λ-calculus.
- Unfortunately, recursion alone is insufficient to recover the expressiveness of the untyped π-calculus for linear session-typed π-calculus.

→ both recursion and sharing are necessary
[Balzer & Pfenning & Toninho 2018]

→ key: sharing of channels and acquire-release discipline

Expressiveness of linear session types

- In the simply-typed λ-calculus, the addition of a recursive type $\mathcal{U} = \mathcal{U} \to \mathcal{U}$ recovers full expressiveness of the untyped λ-calculus.
- Unfortunately, recursion alone is insufficient to recover the expressiveness of the untyped π-calculus for linear session-typed π-calculus.

→ both recursion and sharing are necessary
[Balzer & Pfenning & Toninho 2018]

→ key: sharing of channels and acquire-release discipline

→ encoding of untyped asynchronous π-calculus into SILL_S and proof of its operational and observational correspondence

Acquire-release introduces non-determinism

Acquire-release introduces non-determinism

$$\mathsf{coin} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \oplus\{\mathsf{head} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{coin}, \ \mathsf{tail} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{coin}\}\}$$

Acquire-release introduces non-determinism

$$\mathsf{coin} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \oplus\{\mathsf{head} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{coin}, \ \mathsf{tail} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{coin}\}\}$$

Processes:

head: coin	sends "head" and recurs as tail process
tail: coin	sends "tail" and recurs as head process
filpper: $\mathbf{1} \leftarrow \mathsf{coin}$	reads ("flips") given coin once and terminates
ndchoice: $\oplus\{\mathsf{yes} : \mathbf{1}, \mathsf{no} : \mathbf{1}\}$	spawns new coin and flipper processes and then reads ("flips") the coin and terminates

Acquire-release introduces non-determinism

$$\mathsf{coin} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \oplus\{\mathsf{head} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{coin}, \ \mathsf{tail} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathsf{coin}\}$$

Processes:

head: coin	sends “head” and recurs as tail process
tail: coin	sends “tail” and recurs as head process
filpper: $\mathbf{1} \leftarrow \mathsf{coin}$	reads (“flips”) given coin once and terminates
ndchoice: $\oplus\{\mathsf{yes} : \mathbf{1}, \mathsf{no} : \mathbf{1}\}$	spawns new coin and flipper processes and then reads (“flips”) the coin and terminates

→ coin flipping requires prior acquire

Acquire-release introduces non-determinism

$$\text{coin} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \oplus\{\text{head} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \text{coin}, \ \text{tail} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \text{coin}\}\}$$

Processes:

head: coin	sends "head" and recurs as tail process
tail: coin	sends "tail" and recurs as head process
filpper: $\mathbf{1} \leftarrow \text{coin}$	reads ("flips") given coin once and terminates
ndchoice: $\oplus\{\text{yes} : \mathbf{1}, \text{no} : \mathbf{1}\}$	spawns new coin and flipper processes and then reads ("flips") the coin and terminates

- coin flipping requires prior acquire
- value read by ndchoice depends on order of two acquires

Untyped, asynchronous π-calculus encoding

Untyped, asynchronous π-calculus encoding

Characteristics of untyped, asynchronous π-calculus:

- Arity: a π-calculus channel may connect arbitrarily many processes
- Non-determinsm: e.g., $c(x).P \mid \overline{c}\langle a\rangle \mid c(y).Q$
- Untyped: π-calculus channels are untyped

Untyped, asynchronous π-calculus encoding

Characteristics of untyped, asynchronous π-calculus:

- Arity: a π-calculus channel may connect arbitrarily many processes
- Non-determinsm: e.g., $c(x).P \mid \overline{c}\langle a\rangle \mid c(y).Q$
- Untyped: π-calculus channels are untyped

Basic idea:

- π-calculus process → linear SILL_{S} process of type $\mathbf{1}$
- π-calculus channel → shared SILL_{S} process of universal type $\mathcal{U}_{\text{S}}$

Untyped, asynchronous π-calculus encoding

Characteristics of untyped, asynchronous π-calculus:

- Arity: a π-calculus channel may connect arbitrarily many processes
- Non-determinsm: e.g., $c(x).P \mid \overline{c}\langle a\rangle \mid c(y).Q$
- Untyped: π-calculus channels are untyped

Basic idea:

- π-calculus process → linear SILL$_\mathsf{S}$ process of type $\mathbf{1}$
- π-calculus channel → shared SILL$_\mathsf{S}$ process of universal type $\mathcal{U}_\mathsf{S}$

$$\begin{aligned}\mathcal{U}_\mathsf{S} = \uparrow_\mathsf{L}^\mathsf{S} \&\{&\mathsf{ins} : \Pi x{:}\mathcal{U}_\mathsf{S}.\ \downarrow_\mathsf{L}^\mathsf{S} \mathcal{U}_\mathsf{S},\\ &\mathsf{del} : \oplus\{\mathsf{none} : \downarrow_\mathsf{L}^\mathsf{S} \mathcal{U}_\mathsf{S},\\ &\qquad\qquad \mathsf{some} : \exists x{:}\mathcal{U}_\mathsf{S}.\ \downarrow_\mathsf{L}^\mathsf{S} \mathcal{U}_\mathsf{S}\}\}\end{aligned}$$

Untyped, asynchronous π-calculus encoding

Characteristics of untyped, asynchronous π-calculus:

- Arity: a π-calculus channel may connect arbitrarily many processes
- Non-determinsm: e.g., $c(x).P \mid \overline{c}\langle a \rangle \mid c(y).Q$
- Untyped: π-calculus channels are untyped

Basic idea:

- π-calculus process → linear SILL$_\mathsf{S}$ process of type $\mathbf{1}$
- π-calculus channel → shared SILL$_\mathsf{S}$ process of universal type $\mathcal{U}_\mathsf{S}$

$$\begin{aligned}\mathcal{U}_\mathsf{S} = \uparrow_\mathsf{L}^\mathsf{S} \&\{&\mathsf{ins} : \Pi x{:}\mathcal{U}_\mathsf{S}.\ \downarrow_\mathsf{L}^\mathsf{S} \mathcal{U}_\mathsf{S},\\ &\mathsf{del} : \oplus\{\mathsf{none} : \downarrow_\mathsf{L}^\mathsf{S} \mathcal{U}_\mathsf{S},\\ &\qquad\qquad \mathsf{some} : \exists x{:}\mathcal{U}_\mathsf{S}.\ \downarrow_\mathsf{L}^\mathsf{S} \mathcal{U}_\mathsf{S}\}\}\end{aligned}$$

unordered “buffer”

Untyped, asynchronous π-calculus encoding

$$
\begin{aligned}
\mathcal{U}_{\mathsf{S}} = {\uparrow}^{\mathsf{S}}_{\mathsf{L}}\ \&\{&\mathsf{ins} : \Pi x{:}\mathcal{U}_{\mathsf{S}}.\ {\downarrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathcal{U}_{\mathsf{S}},\\
&\mathsf{del} : \oplus\{\mathsf{none} : {\downarrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathcal{U}_{\mathsf{S}},\\
&\qquad\qquad \mathsf{some} : \exists x{:}\mathcal{U}_{\mathsf{S}}.\ {\downarrow}^{\mathsf{S}}_{\mathsf{L}}\ \mathcal{U}_{\mathsf{S}}\}\}
\end{aligned}
$$

Untyped, asynchronous π-calculus encoding

$$\mathcal{U}_{\mathsf{S}} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \&\{\mathsf{ins} : \Pi x{:}\mathcal{U}_{\mathsf{S}}.\ \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathcal{U}_{\mathsf{S}},\ \mathsf{del} : \oplus\{\mathsf{none} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathcal{U}_{\mathsf{S}},\ \mathsf{some} : \exists x{:}\mathcal{U}_{\mathsf{S}}.\ \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathcal{U}_{\mathsf{S}}\}\}$$

Processes:

empty: $\mathcal{U}_{\mathsf{S}}$	empty buffer
elem: $\mathcal{U}_{\mathsf{S}}$	non-deterministically deletes channel from arbitrary point in buffer using ndchoice
send: $\Pi x{:}\mathcal{U}_{\mathsf{S}}.\,\mathbf{1} \leftarrow \mathcal{U}_{\mathsf{S}}$	attempts to acquire given buffer and inserts received channel
poll_receive: $\exists x{:}\mathcal{U}_{\mathsf{S}}.\,\mathbf{1} \leftarrow \mathcal{U}_{\mathsf{S}}$	perpetually attempts to acquire given buffer for deletion until it succeeds

Untyped, asynchronous π-calculus encoding

$$\mathcal{U}_{\mathsf{S}} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \&\{\mathsf{ins} : \Pi x{:}\mathcal{U}_{\mathsf{S}}.\ \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathcal{U}_{\mathsf{S}}, \quad \mathsf{del} : \oplus\{\mathsf{none} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathcal{U}_{\mathsf{S}}, \quad \mathsf{some} : \exists x{:}\mathcal{U}_{\mathsf{S}}.\ \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathcal{U}_{\mathsf{S}}\}\}$$

Processes:

empty: $\mathcal{U}_{\mathsf{S}}$	empty buffer
elem: $\mathcal{U}_{\mathsf{S}}$	non-deterministically deletes channel from arbitrary point in buffer using ndchoice
send: $\Pi x{:}\mathcal{U}_{\mathsf{S}}.\,\mathbf{1} \leftarrow \mathcal{U}_{\mathsf{S}}$	attempts to acquire given buffer and inserts received channel
poll_receive: $\exists x{:}\mathcal{U}_{\mathsf{S}}.\,\mathbf{1} \leftarrow \mathcal{U}_{\mathsf{S}}$	perpetually attempts to acquire given buffer for deletion until it succeeds

Untyped, asynchronous π-calculus:

$$P \triangleq 0 \mid \overline{x}\langle y\rangle \mid x(y).P \mid \nu x\, P \mid P_1 \mid P_2 \mid\ !P$$

Untyped, asynchronous π-calculus encoding

$$\mathcal{U}_{\mathsf{S}} = \uparrow_{\mathsf{L}}^{\mathsf{S}} \&\{\mathsf{ins} : \Pi x{:}\mathcal{U}_{\mathsf{S}}.\ \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathcal{U}_{\mathsf{S}},\ \mathsf{del} : \oplus\{\mathsf{none} : \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathcal{U}_{\mathsf{S}},\ \mathsf{some} : \exists x{:}\mathcal{U}_{\mathsf{S}}.\ \downarrow_{\mathsf{L}}^{\mathsf{S}} \mathcal{U}_{\mathsf{S}}\}\}$$

Processes:

empty: $\mathcal{U}_{\mathsf{S}}$	empty buffer
elem: $\mathcal{U}_{\mathsf{S}}$	non-deterministically deletes channel from arbitrary point in buffer using ndchoice
send: $\Pi x{:}\mathcal{U}_{\mathsf{S}}.\,\mathbf{1} \leftarrow \mathcal{U}_{\mathsf{S}}$	attempts to acquire given buffer and inserts received channel
poll_receive: $\exists x{:}\mathcal{U}_{\mathsf{S}}.\,\mathbf{1} \leftarrow \mathcal{U}_{\mathsf{S}}$	perpetually attempts ... for deletion until it su...

translate $\mathbf{P}$ in terms of processes

Untyped, asynchronous π-calculus:

$$P \triangleq 0 \mid \overline{x}\langle y\rangle \mid x(y).P \mid \nu x\, P \mid P_1 \mid P_2 \mid\ !P$$

Ongoing and future work

Deadlock-freedom [Balzer et al. 2019]

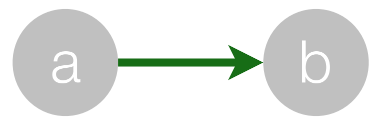
Deadlock-freedom [Balzer et al. 2019]

Two forms of waiting:

- waiting to synchronize:
- waiting to release:

Deadlock-freedom [Balzer et al. 2019]

Two forms of waiting:

- waiting to synchronize: 

“a waits for b to synchronize”
- waiting to release:

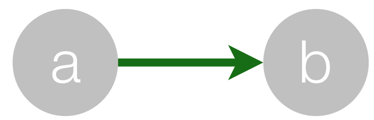
Deadlock-freedom [Balzer et al. 2019]

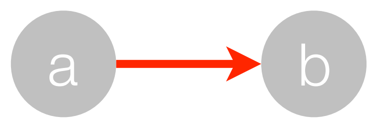
Two forms of waiting:

- waiting to synchronize: $a \to b$ "a waits for b to synchronize"
- waiting to release: $a \to b$ "a waits for b to release resource"

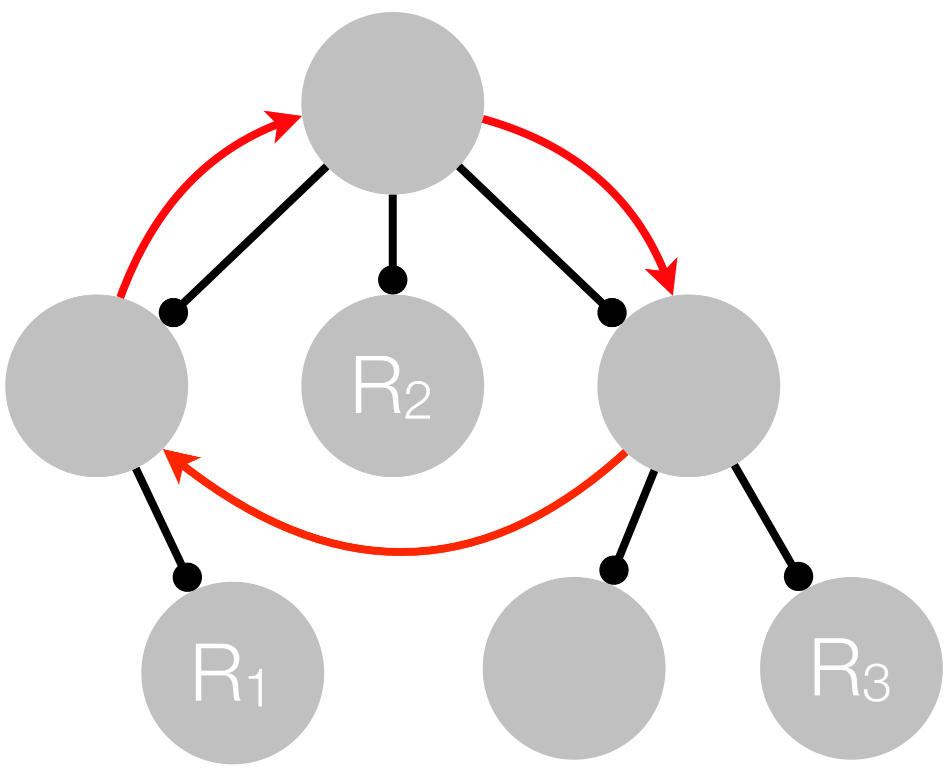
Deadlock-freedom [Balzer et al. 2019]

Two forms of waiting:

- waiting to synchronize: 

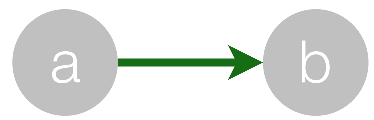
“a waits for b to synchronize”
- waiting to release: 

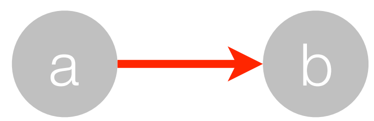
“a waits for b to release resource”



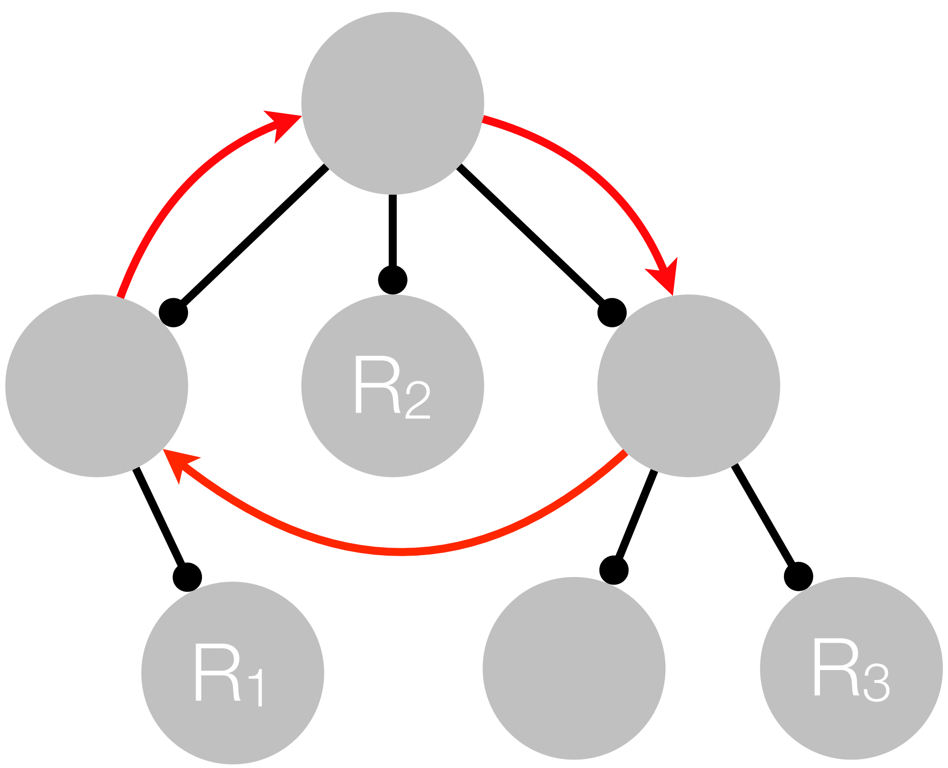
Deadlock-freedom [Balzer et al. 2019]

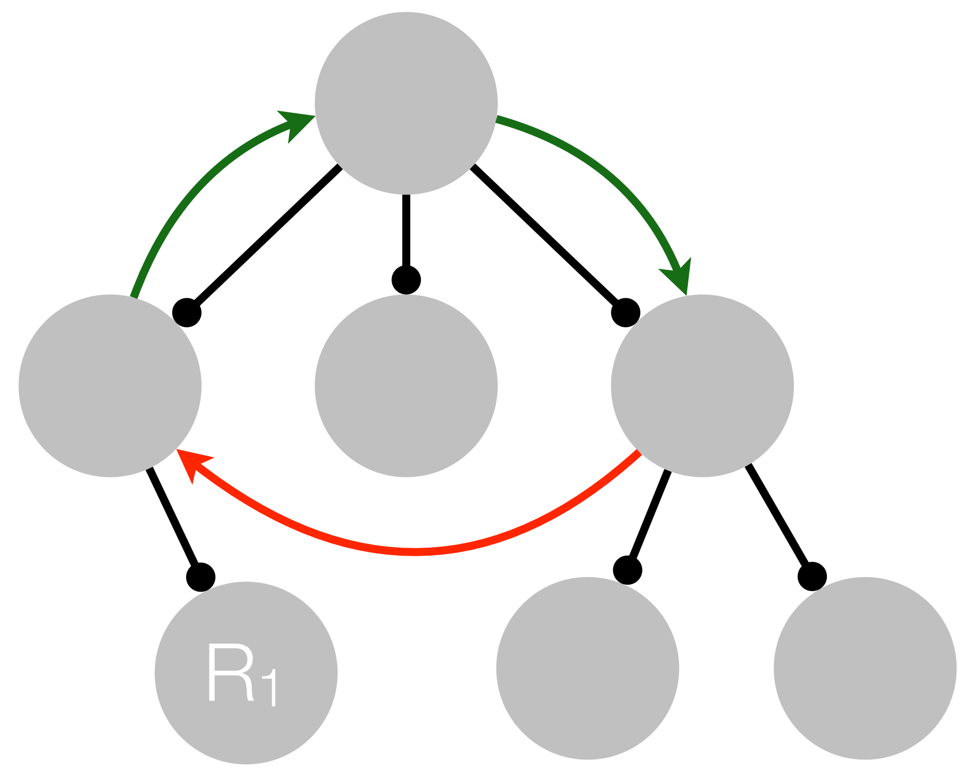
Two forms of waiting:

- waiting to synchronize: 

“a waits for b to synchronize”
- waiting to release: 

“a waits for b to release resource”

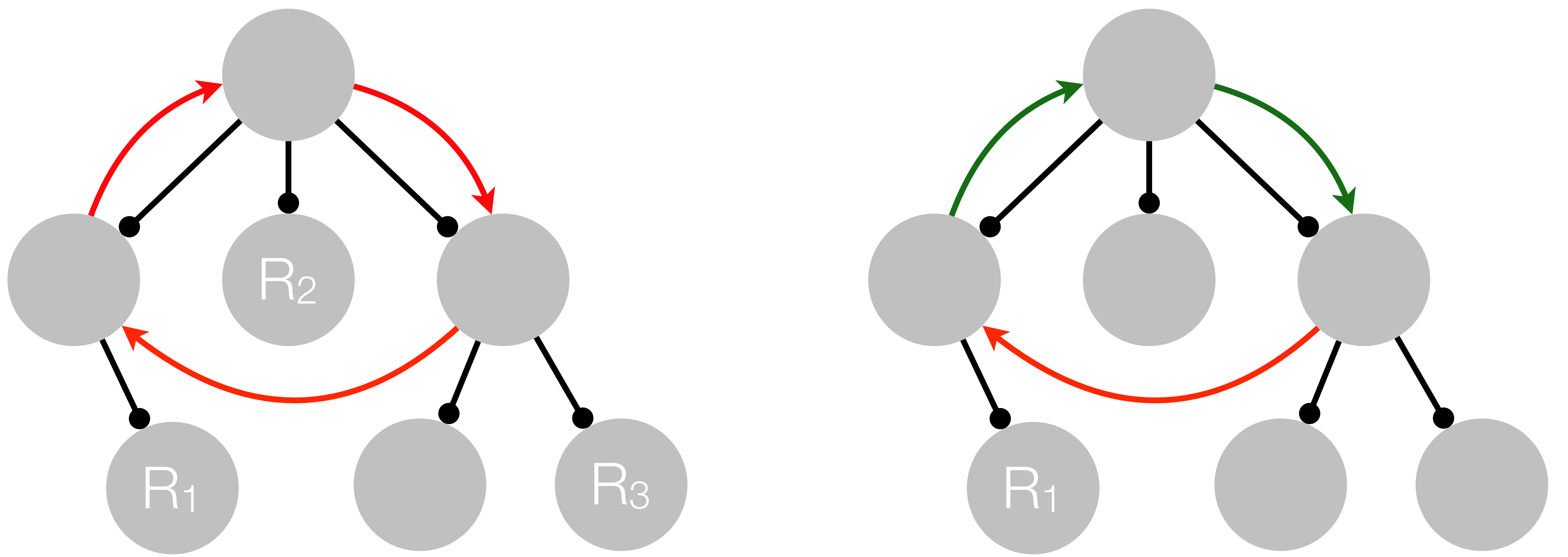




Deadlock-freedom [Balzer et al. 2019]

Two forms of waiting:

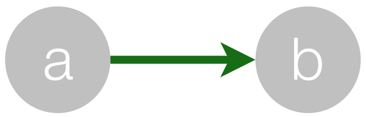
- waiting to synchronize: $a \rightarrow b$ "a waits for b to synchronize"
- waiting to release: $a \rightarrow b$ "a waits for b to release resource"

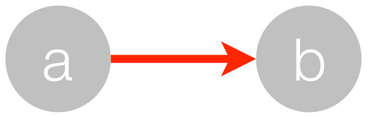


→ plain "locking-up" is no longer sufficient

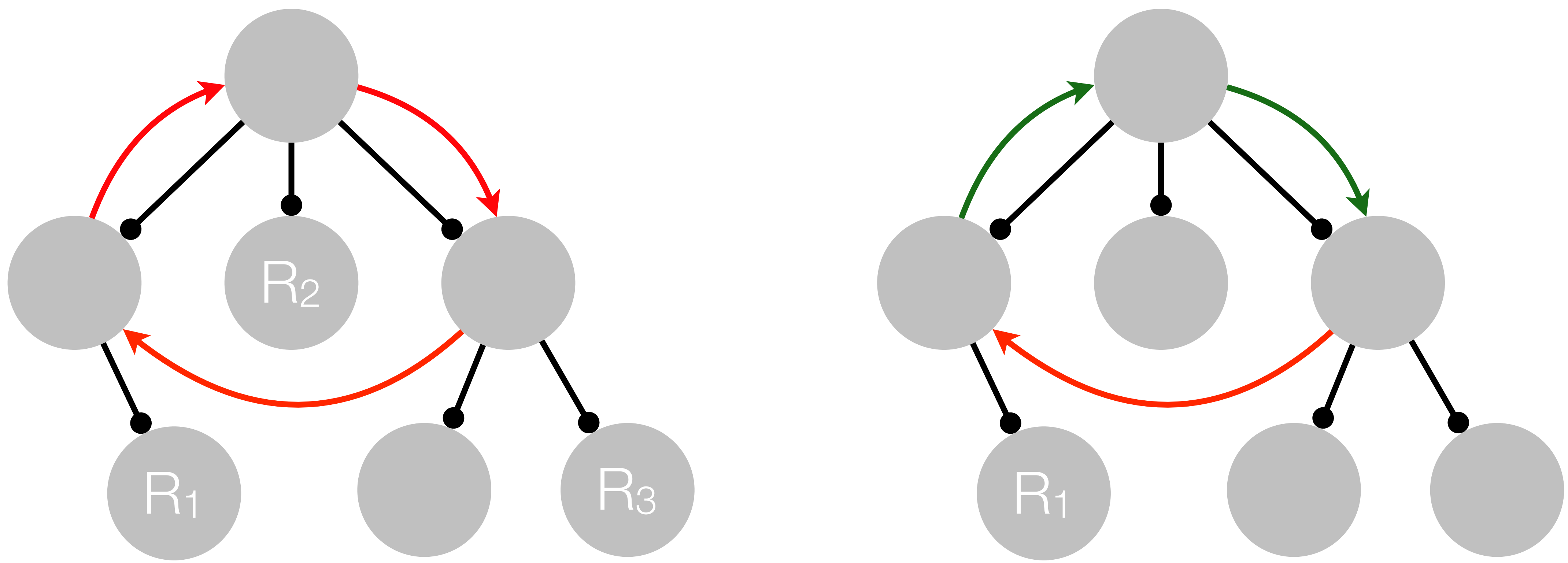
Deadlock-freedom [Balzer et al. 2019]

Two forms of waiting:

- waiting to synchronize: 

“a waits for b to synchronize”
- waiting to release: 

“a waits for b to release resource”



plain “locking-up” is no longer sufficient

see “manifest deadlock-freedom” [Balzer et al. 2019]

Session types for Rust (with Mozilla Research)

- Development of a library for Manifest Sharing in Rust
- Explore application to Servo

Digital contracts (with Das & Hoffmann & Pfenning)

- Development of Nomos, a new digital contract language
- Static guarantees:
  - protocol enforcement (shared session types)
  - control over resource usage (resource analysis)
  - tracking of assets (linear type system)

Summary

- Message-passing concurrent programming
- Session types are a natural fit for typing such programs
- Session types have logical foundation via Curry-Howard correspondence
  - linear session types
  - shared session types
- Manifest sharing recovers the expressiveness of the untyped asynchronous pi-calculus

Summary

- Message-passing concurrent programming
- Session types are a natural fit for typing such programs
- Session types have logical foundation via Curry-Howard correspondence
  - linear session types
  - shared session types
- Manifest sharing recovers the expressiveness of the untyped asynchronous pi-calculus

Thank you for your attention!