Asynchronous Session Types: Exceptions and Multiparty Interactions

Marco Carbone

IT University of Copenhagen

4 June 2009 - Bertinoro

PEOPLE INVOLVED

- **Kohei Honda & Nobuko Yoshida**

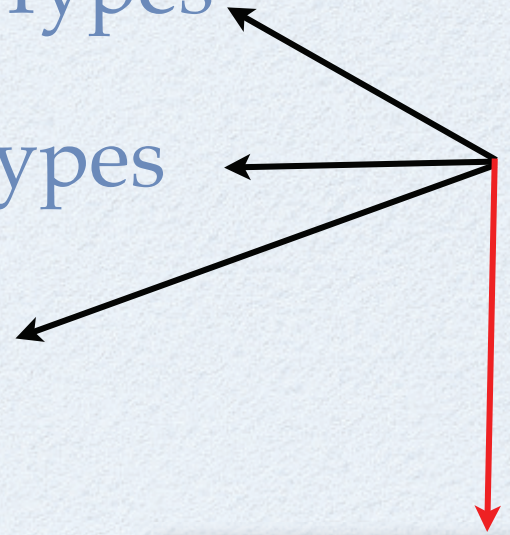
- Gary Brown
- Steve Ross-Talbot
- Andi Bejleri
- Dimitris Mostrous
- Ray Hu
- Joshua Guttman (security)
- ...and many more (including W3C and Robin Milner)

LECTURE PLAN

1. Asynchrony and Session Types
2. Exceptions and Session Types
3. Multiparty Session Types

LECTURE PLAN

1. Asynchrony and Session Types
2. Exceptions and Session Types
3. Multiparty Session Types



- Motivations
- Syntax and Semantics
- Typing Rules

Why types?

Types are about language primitives [...], they get associated directly with the languages.

[...]basic idea of types should be very weak: for example, if we have session-based primitives and we do not have session types, then they can have a constructor error. Same with Java types, same with ML types.

But this is not only about safety, this is about how we conceive the basic dynamics of programming languages.

-- Kohei Honda

Notation

Term	**Symbol**	**Note**
Public channels (or *shared* or *service* channels)	$a, b, \ldots$	
Session channels	$k, s, t, r \ldots$	
Polarised Session channels	κ, λ	$\kappa \in \{s^+, s^-\}$
Variables	x, y, z	
Process Term Variables	X, Y	
Public channel or Variable	$u, u', \ldots$	

SYNTAX FOR SESSIONS

P :: =	**!**a(s)**.** P	(service)
	a(s)**.** P	(request)
	s**?**(x)**.** P	(input)
	s**!**<e>**.** P	(output)
	s**?** **((**x**))**. P	(session rec)
	s**!**<<s'>>**.** P	(delegation)
	s ▷ { l_i : P_i }	(branch)
	s ◁ l**.** P	(select)
	(υ a) P / (υ s) P	(res)
	P | Q	(par)
	if e **then** P **else** P	(cond)
	0	(inact)
	μX**.** P	(rec)
	X	(recVar)

SYNTAX FOR SESSIONS

P :: =	**!**a(s)**.** P	(service)
	a(s)**.** P	(request)
	s**?**(x)**.** P	(input)
	s**!**<e>**.** P	(output)
	s**?** **((**x**))**. P	(session rec)
	s**!**<<s'>>**.** P	(delegation)
	s ▷ { l_i : P_i }	(branch)
	s ◁ l**.** P	(select)
	(υ a) P / (υ s) P	(res)
	P | Q	(par)
	if e **then** P **else** P	(cond)
	0	(inact)
	μX**.** P	(rec)
	X	(recVar)

SYNTAX FOR SESSIONS

P :: =	**!**a(s)**.** P	(service)
	a(s)**.** P	(request)
	s**?**(x)**.** P	(input)
	s**!**<e>**.** P	(output)
	s**?** **((**x**))**. P	(session rec)
	s**!**<<s'>>**.** P	(delegation)
	s ▷ { l_i : P_i }	(branch)
	s ◁ l**.** P	(select)
	(υ a) P / (υ s) P	(res)
	P | Q	(par)
	if e **then** P **else** P	(cond)
	0	(inact)
	μX**.** P	(rec)
	X	(recVar)

SYNTAX FOR SESSIONS

P :: =	**!**a(s)**.** P	(service)
	a(s)**.** P	(request)
	s**?**(x)**.** P	(input)
	s**!**<e>**.** P	(output)
	s**?** ((x))**.** P	(session rec)
	s**!**<<s'>>**.** P	(delegation)
	s $\triangleright$ { l_i : P_i }	(branch)
	s $\triangleleft$ l**.** P	(select)
	(υ a) P / (υ s) P	(res)
	P | Q	(par)
	if e **then** P **else** P	(cond)
	0	(inact)
	μX**.** P	(rec)
	X	(recVar)

SYNTAX FOR SESSIONS

P :: =	**!**a(s)**.** P	(service)
	a(s)**.** P	(request)
	s**?**(x)**.** P	(input)
	s**!**<e>**.** P	(output)
	s**?** **((**x**))**. P	(session rec)
	s**!**<<s'>>**.** P	(delegation)
	s $\triangleright$ { l_i : P_i }	(branch)
	s $\triangleleft$ l**.** P	(select)
	(υ a) P / (υ s) P	(res)
	P | Q	(par)
	if e **then** P **else** P	(cond)
	0	(inact)
	μX**.** P	(rec)
	X	(recVar)

ASYNCHRONY&SESSIONS

Why asynchronous?

ASYNCHRONY&SESSIONS

Why asynchronous?

Reality is mostly asynchronous !!!

ASYNCHRONY&SESSIONS

- First proposed by
  - Neubauer, Thiemann [Unpublished]
- Adopted by several people:
  - Gay&Vasconcelos, Singularity, etc.
- We shall use it today!

Asynchrony in the π-calculus

Synchronous: $s?(x).\, s!\langle e\rangle.\, t!\langle e'\rangle.0$

Asynchrony in the π-calculus

Synchronous: $s?(x).\, s!\langle e\rangle.\, t!\langle e'\rangle.0$

Asynchrony in the π-calculus

Synchronous: $s?(x).\, s!\langle e\rangle.\, t!\langle e'\rangle.0$

Asynchrony in the π-calculus

Synchronous: $s?(x).\, s!\langle e\rangle.\, t!\langle e'\rangle.0$

Asynchronous: $s?(x).\, (s!\langle e\rangle.\, 0 \;\mid\; t!\langle e'\rangle.\, 0)$

Asynchrony in the π-calculus

Synchronous: $s?(x).\, s!\langle e\rangle.\, t!\langle e'\rangle.0$

Asynchronous: $s?(x).\, (s!\langle e\rangle.\, 0 \mid t!\langle e'\rangle.\, 0)$

Asynchrony in the π-calculus

Synchronous: $s?(x).\, s!\langle e\rangle.\, t!\langle e'\rangle.0$

Asynchronous: $s?(x).\, (s!\langle e\rangle.\, 0 \mid t!\langle e'\rangle.\, 0)$

Outputs **don't** have successors (no prefixing)

What about session (types)?

Synchronous Case.

Type : $\downarrow(int).\ \downarrow(bool).\ \uparrow(string)$

Processes :

$$a(s).\, s?(x).\, s?(y).\, s!\langle x+1\rangle$$

$$\mid$$

$$\overline{a}(s).\, s!\langle 5\rangle.\, s!\langle \texttt{true}\rangle.\, s?(z)$$

What about session (types)?

Synchronous Case.

Type : $\downarrow(int).\ \downarrow(bool).\ \uparrow(string)$

Processes : $a(s).\ s?(x).\ s?(y).\ s!\langle x+1\rangle$

$(\nu s)\ |$

$\overline{a}(s).\ s!\langle 5\rangle.\ s!\langle \texttt{true}\rangle.\ s?(z)$

What about session (types)?

Synchronous Case.

Type : $\downarrow(int).\ \downarrow(bool).\ \uparrow(string)$

Processes :

$$a(s).\ s?(x).\ s?(y).\ s!\langle 5+1\rangle$$
$$(\nu s)\ |$$
$$\overline{a}(s).\ s!\langle 5\rangle.\ s!\langle \texttt{true}\rangle.\ s?(z)$$

What about session (types)?

Synchronous Case.

Type : $\downarrow(int).\ \downarrow(bool).\ \uparrow(string)$

Processes :

$$(\nu s)\ \Big|\ \begin{array}{l} a(s).\, s?(x).\, s?(y).\, s!\langle 5+1\rangle \\ \overline{a}(s).\, s!\langle 5\rangle.\, s!\langle \texttt{true}\rangle.\, s?(z) \end{array}$$

What about session types?

Asynchronous Case.

Type : $\downarrow(int).\ \downarrow(bool).\ \uparrow(string)$

Processes :

$$a(s).\, s?(x).\, s?(y).\, s!\langle x+1\rangle$$

$$\mid$$

$$\overline{a}(s).\,(\ s!\langle 5\rangle \ \mid\ s!\langle \texttt{true}\rangle \ \mid\ s?(z)\ )$$

What about session types?

Asynchronous Case.

Type : $\downarrow(int).\ \downarrow(bool).\ \uparrow(string)$

Processes : $a(s).\, s?(x).\, s?(y).\, s!\langle x+1\rangle$

$(\nu s)\ |$

$\overline{a}(s).\,(\ s!\langle 5\rangle\ \mid\ s!\langle \texttt{true}\rangle\ \mid\ s?(z)\ )$

What about session types?

Asynchronous Case.

Type : $\downarrow(int).\ \downarrow(bool).\ \uparrow(string)$

Processes : $a(s).\, s?(x).\, s?(y).\, s!\langle \texttt{true} + 1 \rangle$

$(\nu s)\ |$

$\overline{a}(s).\, (\ s!\langle 5 \rangle\ \mid\ s!\langle \texttt{true} \rangle\ \mid\ s?(z)\ )$

What about session types?

Asynchronous Case.

Type : $\downarrow(int).\ \downarrow(bool).\ \uparrow(string)$

Processes : $a(s).\, s?(x).\, s?(y).\, s!\langle \texttt{true} + 1 \rangle$

$$(\nu s)\ |$$

$$\overline{a}(s).\,(\ s!\langle 5 \rangle\ \mid\ s!\langle \texttt{true} \rangle\ \mid\ s?(z)\ )$$

What about session types?

Asynchronous Case.

Type : $\downarrow(int).\ \downarrow(bool).\ \uparrow(string)$

Processes : $a(s).\, s?(x).\, s?(y).\, s!\langle \texttt{true} + 1 \rangle$

$(\nu s)\ |$

$\overline{a}(s).\,(\ s!\langle 5 \rangle\ \mid\ s!\langle \texttt{true} \rangle\ \mid\ s?(z)\ )$

!!! Need to preserve ordering !!!

Queues: Syntax

$$L ::= \epsilon \mid h :: L$$

(message list)

Queues: Syntax

$$L ::= \epsilon \mid h :: L$$

(message list)

in side $\rightarrow$ $h_1 :: h_2 :: \ldots :: h_{n-1} :: h_n$ $\rightarrow$ *out side*

QUEUES: SYNTAX

$L ::= \epsilon \mid h :: L$ (message list)

in side → $h_1 :: h_2 :: \ldots :: h_{n-1} :: h_n$ → *out side*

option 1: $s : L$ (simple channel queue)

QUEUES: SYNTAX

$L ::= \epsilon \mid h :: L$ (message list)

in side $\rightarrow$ $h_1 :: h_2 :: \ldots :: h_{n-1} :: h_n$ $\rightarrow$ *out side*

option 1: $s : L$ (simple channel queue)

option 2: $\kappa \hookrightarrow \overline{\kappa} : L$ (polarised queue)

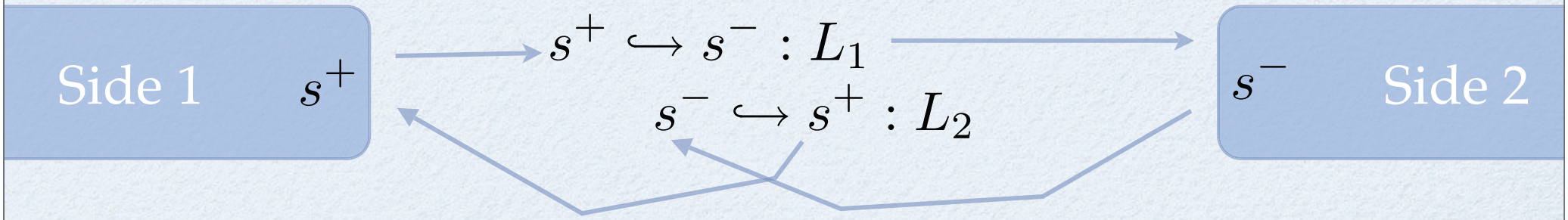
QUEUES: SYNTAX

$L ::= \epsilon \mid h :: L$ (message list)

in side → $h_1 :: h_2 :: \ldots :: h_{n-1} :: h_n$ → *out side*

option 1: $s : L$ (simple channel queue)

option 2: $\kappa \hookrightarrow \overline{\kappa} : L$ (polarised queue)



QUEUES: SEMANTICS

$$\kappa!\langle e\rangle.\,P \;\mid\; \kappa \hookrightarrow \overline{\kappa} : L \;\longrightarrow\; P \;\mid\; \kappa \hookrightarrow \overline{\kappa} : (v :: L) \qquad e \Downarrow v$$

QUEUES: SEMANTICS

$$\kappa!\langle e\rangle.\,P \mid \kappa \hookrightarrow \overline{\kappa} : L \longrightarrow P \mid \kappa \hookrightarrow \overline{\kappa} : (v :: L) \qquad e \Downarrow v$$

$$\overline{\kappa} \hookrightarrow \kappa : (v :: L) \mid \kappa(x).\,P \longrightarrow \overline{\kappa} \hookrightarrow \kappa : L \mid P\{v/x\}$$

QUEUES: SEMANTICS

$$\kappa!\langle e\rangle.\,P \mid \kappa \hookrightarrow \overline{\kappa} : L \longrightarrow P \mid \kappa \hookrightarrow \overline{\kappa} : (v :: L) \qquad e \Downarrow v$$

$$\overline{\kappa} \hookrightarrow \kappa : (v :: L) \mid \kappa(x).\,P \longrightarrow \overline{\kappa} \hookrightarrow \kappa : L \mid P\{v/x\}$$

$$a(s).\,P \mid \overline{a}(s).\,Q \longrightarrow$$

$$(\nu s)(\,P \mid Q \mid s^- \hookrightarrow s^+ : L \mid s^+ \hookrightarrow s^- : L\,)$$

BACK TO THE EXAMPLE

Type : $\downarrow(int).\ \downarrow(bool).\ \uparrow(string)$

Processes : $a(s).\, s?(x).\, s?(y).\, s!\langle x+1\rangle$

$\mid$

$\overline{a}(s).\, s!\langle 5\rangle.\, s!\langle \texttt{true}\rangle.\, s?(z)$

BACK TO THE EXAMPLE

Type : $\downarrow(int).\ \downarrow(bool).\ \uparrow(string)$

$$(\nu s)\Big(\quad s^-?(x).\, s^-(y).\, s^-!\langle x+1\rangle \quad \Big)$$

$$s^+!\langle 5\rangle.\, s^+\langle \texttt{true}\rangle.\, s^+(z)$$

$$\mid\ s^- \hookrightarrow s^+ : \epsilon \ \mid\ s^+ \hookrightarrow s^- : \epsilon$$

BACK TO THE EXAMPLE

Type : $\downarrow(int).\ \downarrow(bool).\ \uparrow(string)$

$$(\nu s)\Big(\quad s^-?(x).\, s^-(y).\, s^-!\langle x+1\rangle \quad \Big)$$

$$s^+!\langle 5\rangle.\, s^+\langle \texttt{true}\rangle.\, s^+(z)$$

$$\mid\ s^- \hookrightarrow s^+ : \epsilon \ \mid\ s^+ \hookrightarrow s^- : 5$$

BACK TO THE EXAMPLE

Type : $\downarrow(int).\ \downarrow(bool).\ \uparrow(string)$

$$(\nu s)\Big(s^-?(x).\, s^-(y).\, s^-!\langle x+1\rangle \;\Big|\; s^+!\langle 5\rangle.\, s^+\langle \texttt{true}\rangle.\, s^+(z) \;\Big|\; s^- \hookrightarrow s^+ : \epsilon \;\Big|\; s^+ \hookrightarrow s^- : \texttt{true} :: 5 \Big)$$

BACK TO THE EXAMPLE

Type : $\downarrow(int).\ \downarrow(bool).\ \uparrow(string)$

$$(\nu s)\Big(s^-?(x).\, s^-(y).\, s^-!\langle 5+1\rangle \;\Big|\; s^+!\langle 5\rangle.\, s^+\langle \texttt{true}\rangle.\, s^+(z) \;\Big|\; s^- \hookrightarrow s^+ : \epsilon \;\Big|\; s^+ \hookrightarrow s^- : \texttt{true} :: 5 \Big)$$

EXCEPTIONS

Exceptions, literally...

- "An Exception is a person or thing that is excluded from a general statement or does not follow a rule" (Mac Dictionary)
- "Exception (handling) is a programming language construct or computer hardware mechanism designed to handle the occurrence of some condition that changes the normal flow of execution." (Wikipedia)

Exceptions, literally...

- "An Exception is a person or thing that is excluded from a general statement or does not follow a rule" (Mac Dictionary)
- "Exception (handling) is a programming language construct or computer hardware mechanism designed to handle the occurrence of some condition that changes the normal flow of execution." (Wikipedia)

EXCEPTIONS, IN GENERAL

try { /* Default Code */ }

catch { /* Handler Code */ }

If an exception is thrown by the default code then the handler is executed.

Exceptions are thrown with a special command **throw**

EXCEPTIONS & SESSIONS

What if apply this idea to *sessions*?

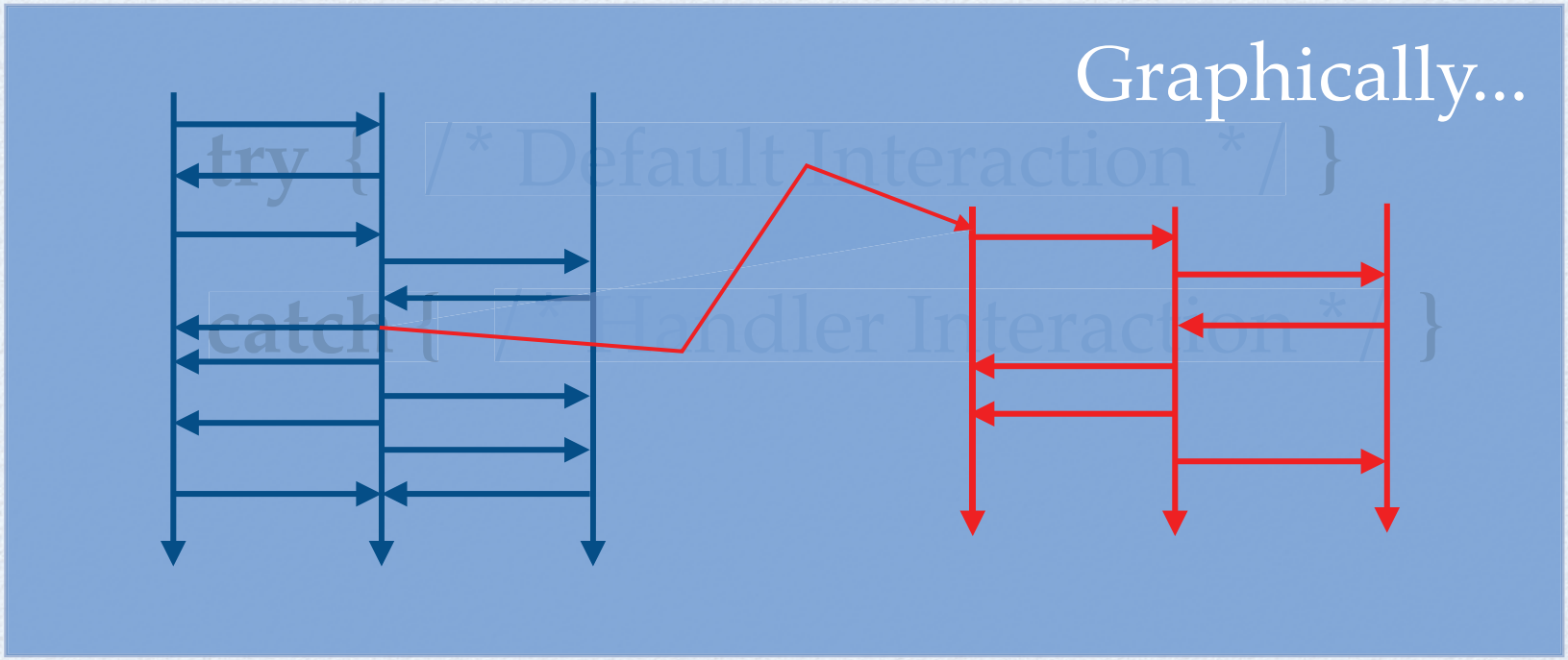
try { /* Default Interaction */ }

catch { /* Handler Interaction */ }

EXCEPTIONS & SESSIONS

What if apply this idea to *sessions*?

Graphically...



EXAMPLE: ASYNCH ESCAPE

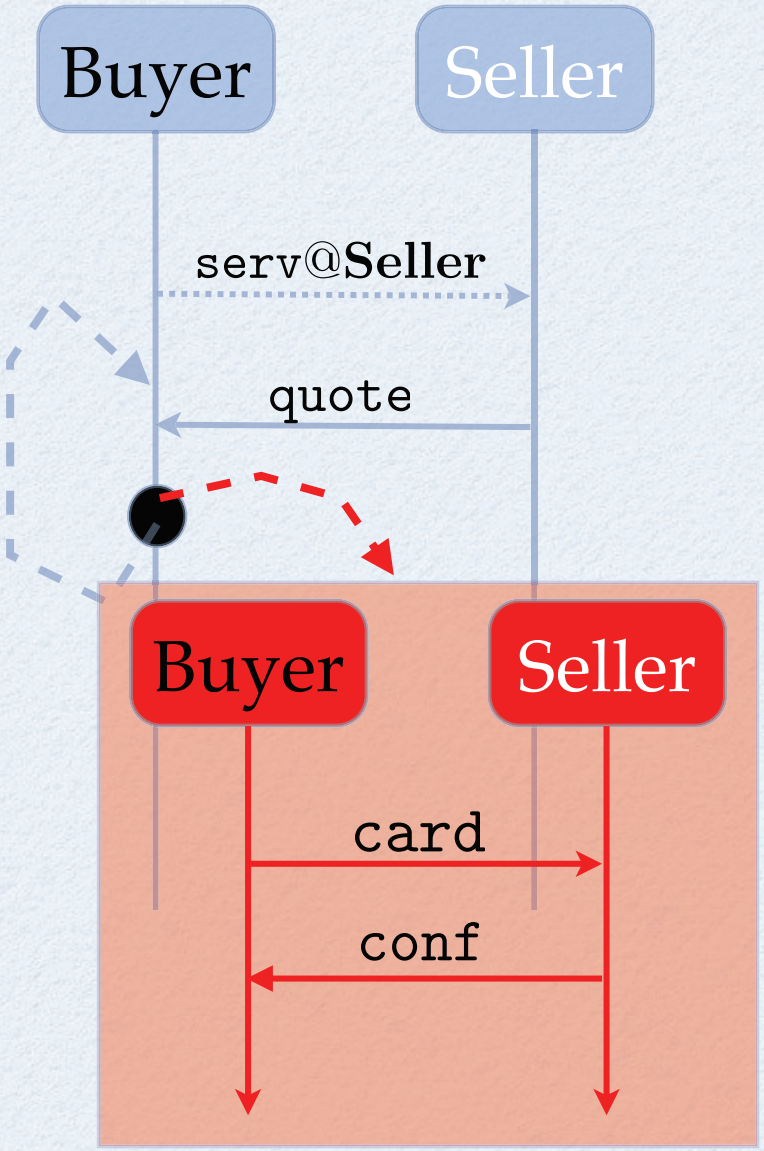
Buyer

invoke serv@Seller(s);

try {
  $\mu X.$
  $s?(y).$
  if $(ok(y))$ **then throw else** X
}

catch $\{s!\langle \texttt{card} \rangle.\ s?(y)\ \}$

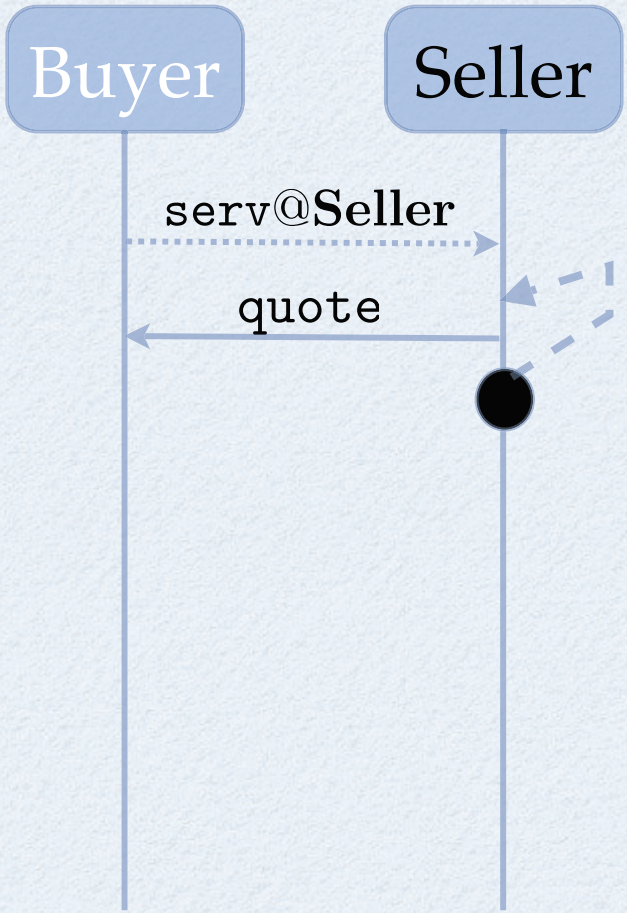
EXAMPLE: ASYNCH ESCAPE



Buyer

$$\textbf{invoke}\ \texttt{serv@Seller}(s);$$
$$\textbf{try}\ \{$$
$$\quad \mu X.$$
$$\quad s?(y).$$
$$\quad \textbf{if}\ (ok(y))\ \textbf{then throw else}\ X$$
$$\}$$
$$\textbf{catch}\ \{s!\langle \texttt{card} \rangle.\ s?(y)\ \}$$

EXAMPLE: ASYNCH ESCAPE



Seller

$$\textbf{service}\ \texttt{serv}(s)\ \{$$

$$\quad \textbf{try}\ \{$$

$$\quad\quad \mu X.$$

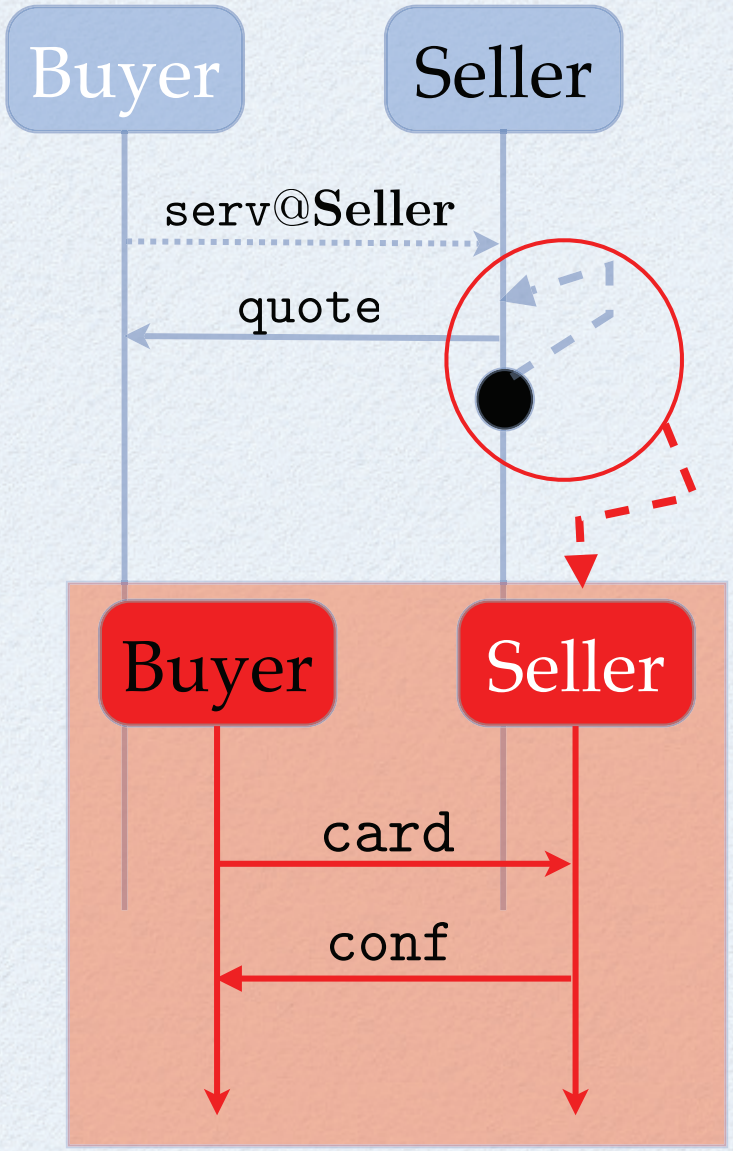
$$\quad\quad s!\langle \texttt{quote} \rangle.\ X$$

$$\quad \}$$

$$\quad \textbf{catch}\ \{s?(x).s!\langle \texttt{conf} \rangle\ \}$$

$$\}$$

EXAMPLE: ASYNCH ESCAPE



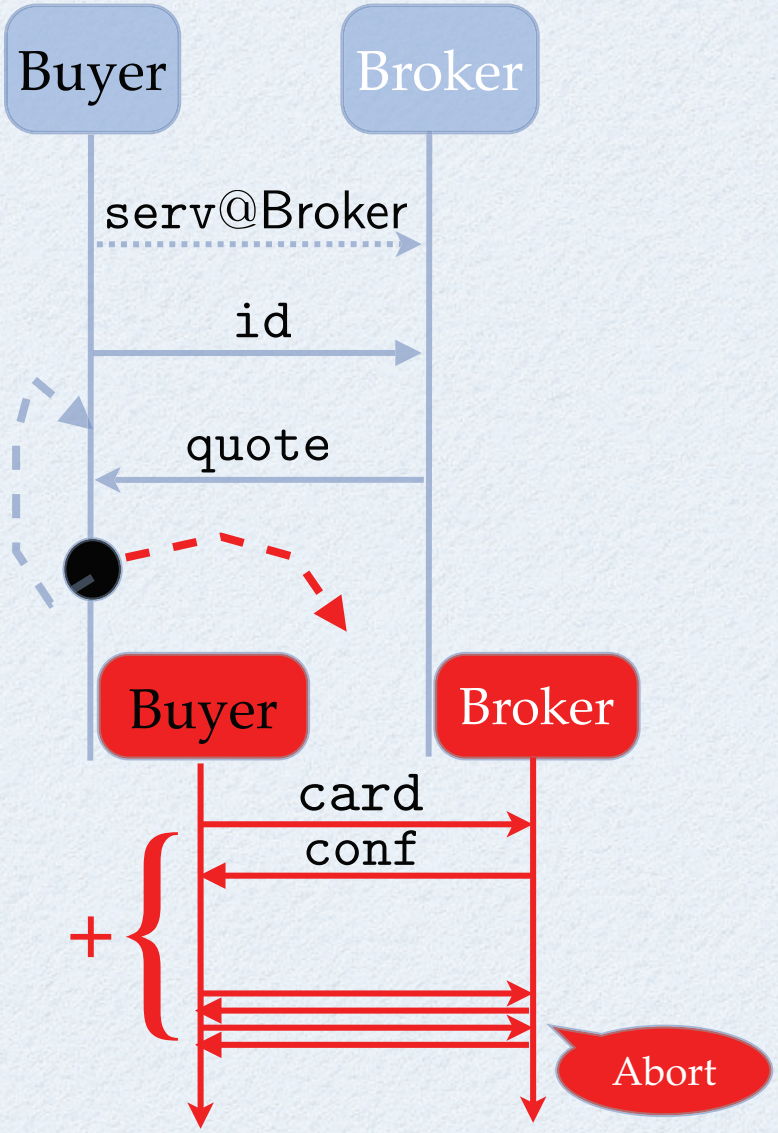
Seller

$$\textbf{service}\ \texttt{serv}(s)\ \{$$
$$\quad \textbf{try}\ \{$$
$$\quad\quad \mu X.$$
$$\quad\quad s!\langle \texttt{quote} \rangle.\ X$$
$$\quad \}$$
$$\quad \textbf{catch}\ \{ s?(x).s!\langle \texttt{conf} \rangle\ \}$$
$$\}$$

INTERACTIONAL EXCEPTIONS

- The mechanism above is called *Interactional Exception*
- It is very hard to implement with branching/ selection (especially having asynchrony)
- Is it useful? => appears in many financial protocols!

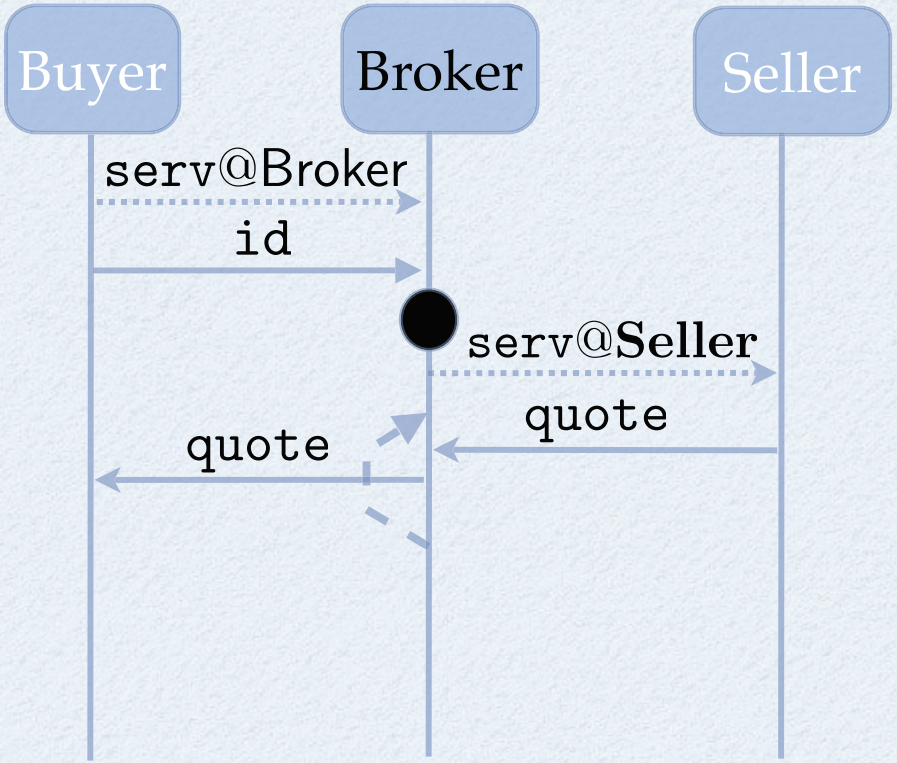
EXAMPLE: NESTED ESCAPE



Buyer

invoke $\texttt{serv@Broker}(t)$;
try {
$t!\langle \texttt{id} \rangle.$
$\mu X.$
$t?(y).$
if $(ok(y))$ **then throw else** X
}
catch {
$l_1 : \; t!\langle \texttt{card} \rangle.\; t?(y)$
$l_2 : \; P_{\texttt{abort}}$
}

EXAMPLE: NESTED ESCAPE



Broker

service $\mathtt{serv}(t)$ {

try (t) {

$t?(x).$

if $\mathtt{bad}(x)$ **then throw else**

invoke $\mathtt{serv@Seller}(s)$;

try (t, s) {

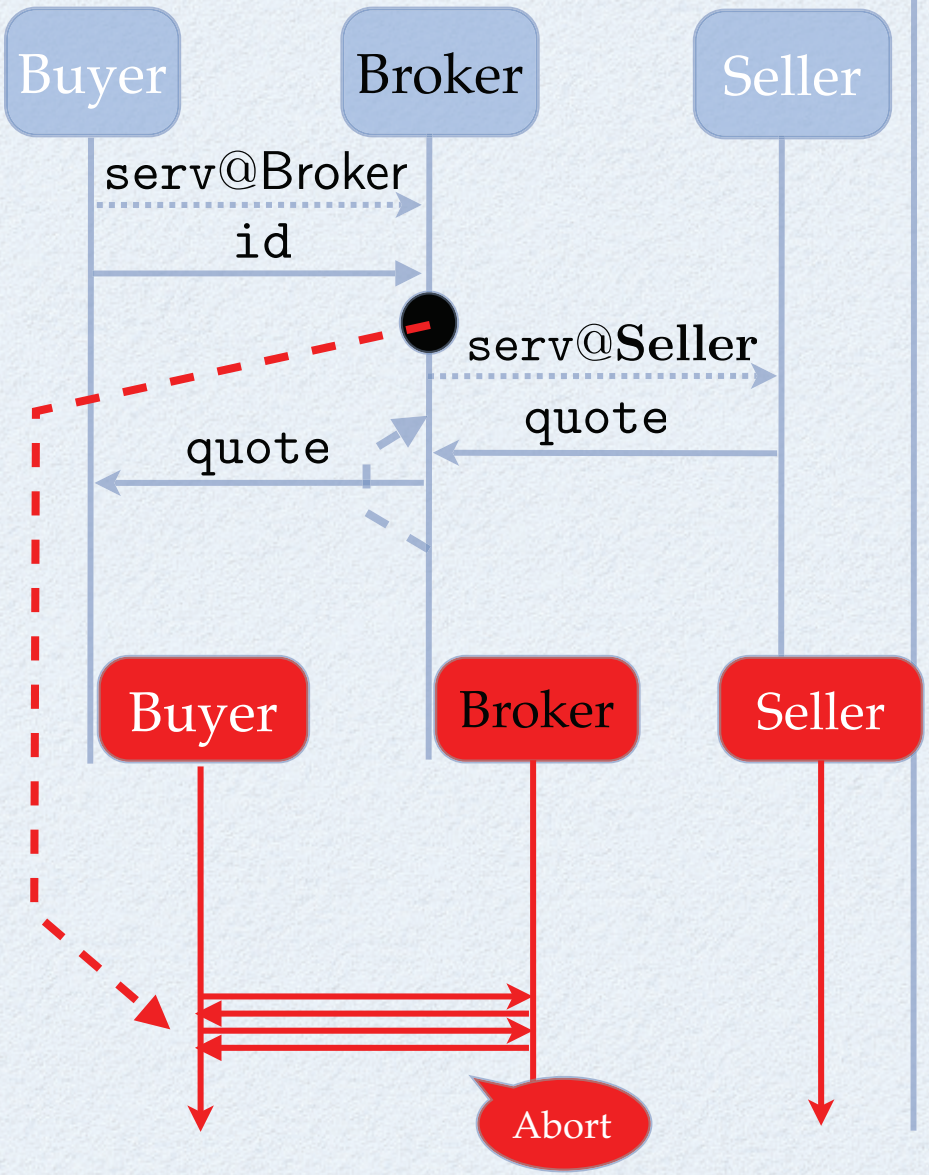
$\mu X.\ s?(x).\ t!\langle x + 10\% \rangle.\ X$

catch { $t \triangleright l_1.\ \ldots \mathrm{fwd} \ldots$ }

catch { $t \triangleright l_2.\ P'_{\mathtt{abort}}$ }

}

EXAMPLE: NESTED ESCAPE



Broker

service $\mathtt{serv}(t)$ {

try (t) {

$t?(x).$

if $\mathtt{bad}(x)$ **then throw else**

invoke $\mathtt{serv@Seller}(s);$

try (t, s) {

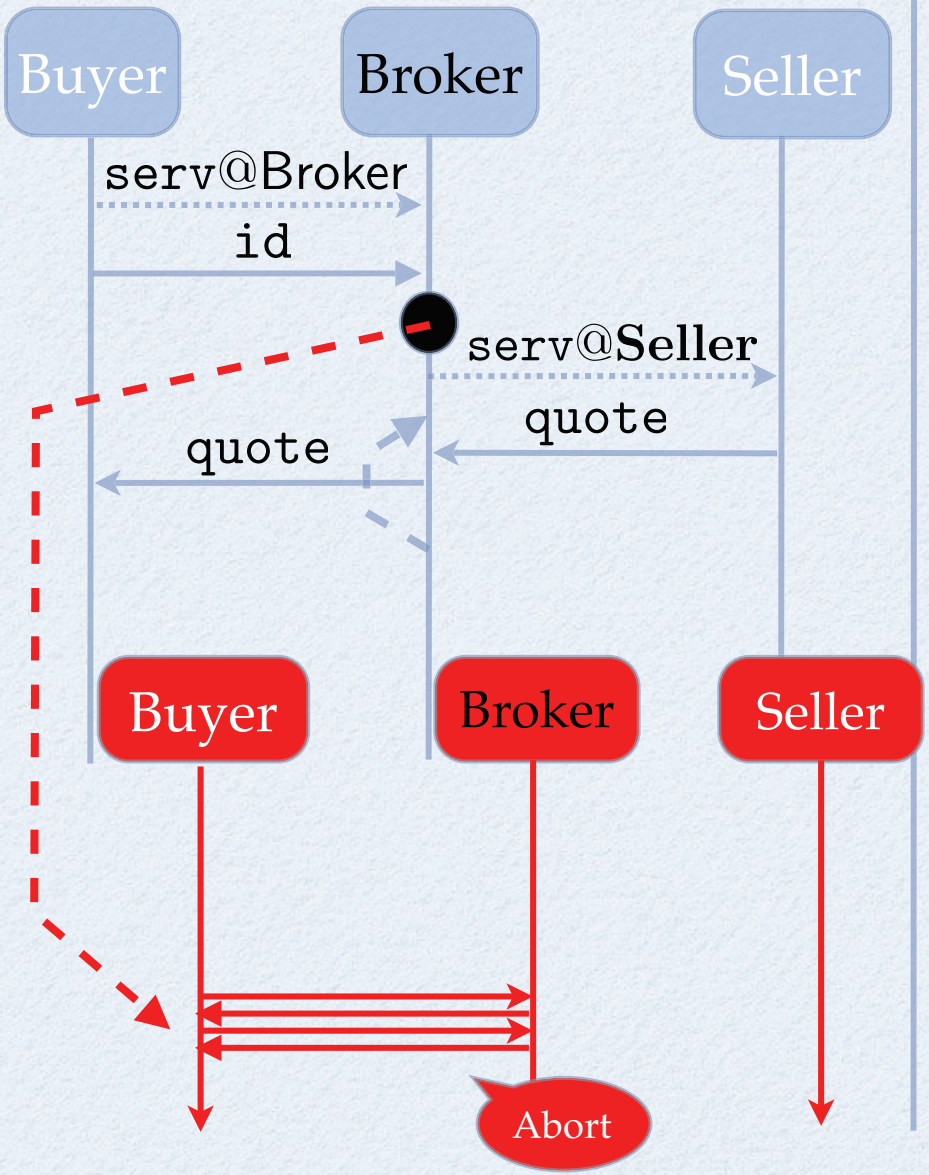
$\mu X.\ s?(x).\ t!\langle x + 10\% \rangle.\ X$

catch { $t \triangleright l_1.\ \ldots \mathrm{fwd} \ldots$ }

catch { $t \triangleright l_2.\ P'_{\mathsf{abort}}$ }

}

EXAMPLE: NESTED ESCAPE



Broker

service $\mathtt{serv}(t)$ {

try (t) {

$t?(x).$

if $\mathtt{bad}(x)$ **then throw else**

invoke $\mathtt{serv@Seller}(s);$

try (t, s) {

$\mu X.\ s?(x).\ t!\langle x + 10\% \rangle.\ X$

}

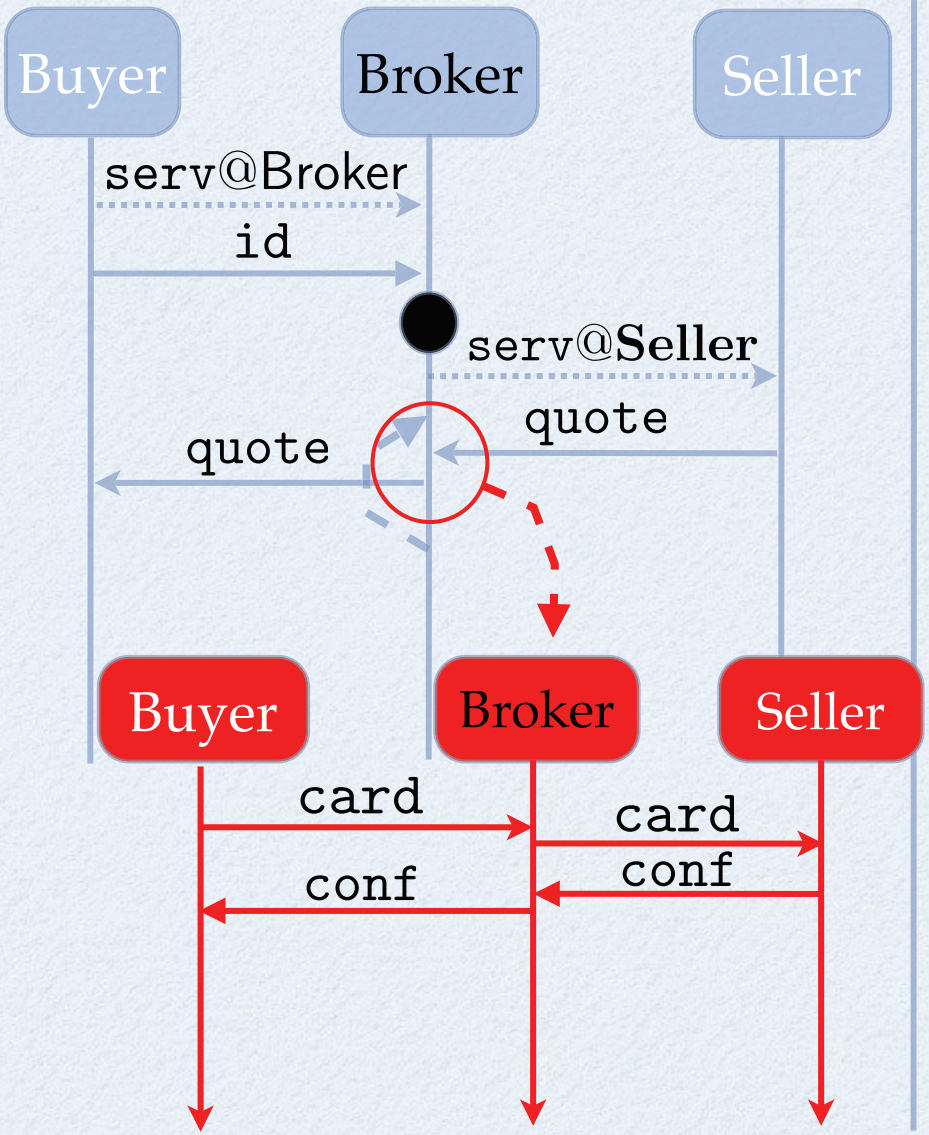
catch { $t \triangleright l_1.\ \ldots \mathrm{fwd} \ldots$ }

}

catch { $t \triangleright l_2.\ P'_{\mathtt{abort}}$ }

}

EXAMPLE: NESTED ESCAPE



Broker

service $\mathtt{serv}(t)$ {

try (t) {

$t?(x).$

if $\mathtt{bad}(x)$ **then throw else**

invoke $\mathtt{serv@Seller}(s)$;

try (t, s) {

$\mu X.\ s?(x).\ t!\langle x + 10\% \rangle.\ X$

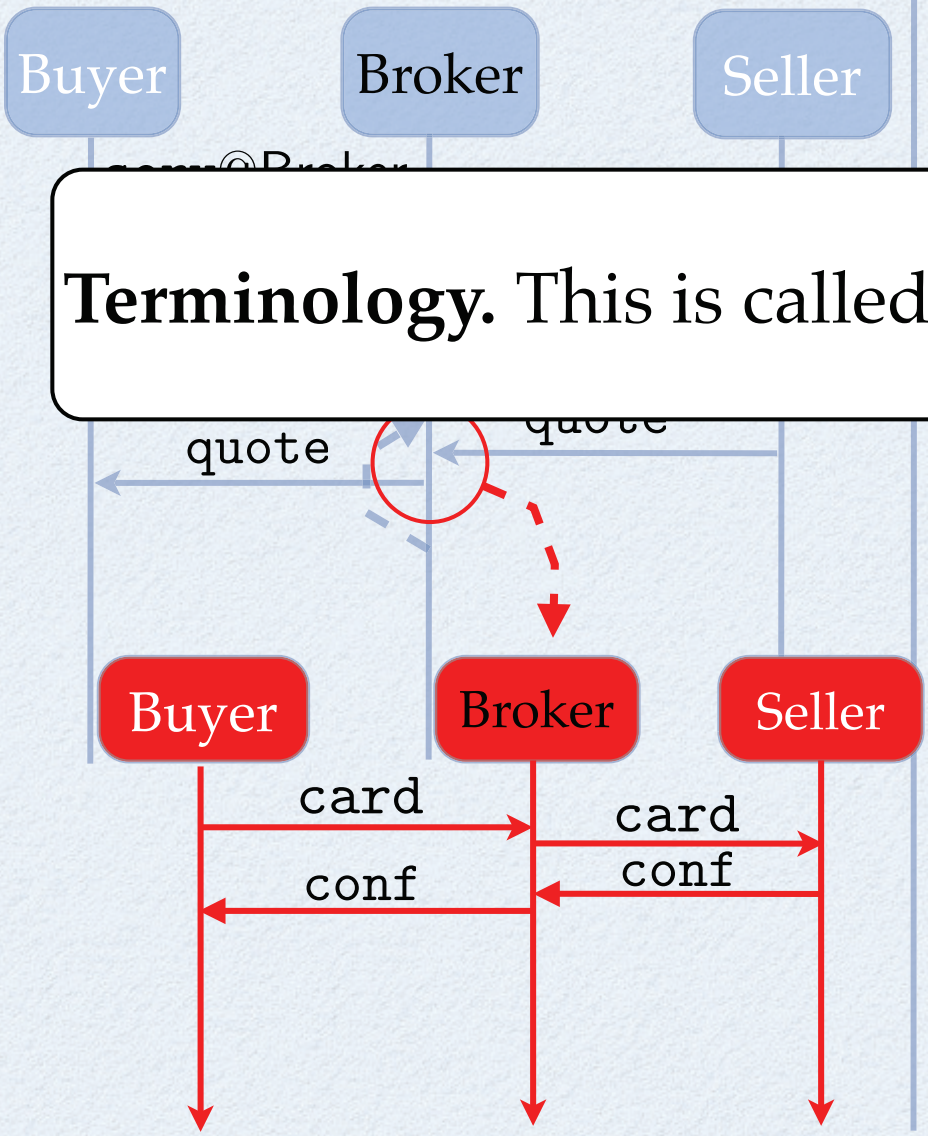
}

catch { $t \triangleright l_1.\ \ldots \mathrm{fwd} \ldots$ }

catch { $t \triangleright l_2.\ P'_{\mathtt{abort}}$ }

}

EXAMPLE: NESTED ESCAPE



Terminology. This is called *refinement.*

Broker

service $\texttt{serv}(t)$ {

if $\texttt{bad}(x)$ **then throw else**

invoke $\texttt{serv@Seller}(s)$;

try (t, s) {

$\mu X.\ s?(x).\ t!\langle x + 10\% \rangle.\ X$

catch { $t \triangleright l_1.\ \ldots \texttt{fwd} \ldots$ }

~~**catch** { $t \triangleright l_2.\ P'_{\texttt{abort}}$ }~~

}

π-CALCULUS WITH SESSIONS

P :: =	$*a(\kappa).\ P$	(service)
	$a(\kappa).\ P$	(request)
	$\kappa?(x).\ P$	(input)
	$\kappa!\langle e\rangle.\ P$	(output)
	$\kappa \triangleright \{ l_i : P_i \}$	(branch)
	$\kappa \triangleleft l.\ P$	(select)
	$(\upsilon\ a)\ P$ / $(\upsilon\ \kappa)\ P$	(res)
	$P \mid Q$	(par)
	if e **then** P **else** P	(cond)
	0	(inact)
	μX. P	(rec)
	X	(recVar)

π-CALCULUS WITH SESSIONS

P :: =	~~$*a(\kappa).\ P$~~	~~(service)~~
	~~$a(\kappa).\ P$~~	~~(request)~~
	$\kappa?(x).\ P$	(input)
	$\kappa!\langle e \rangle.\ P$	(output)
	$\kappa \triangleright \{ l_i : P_i \}$	(branch)
	$\kappa \triangleleft l.\ P$	(select)
	$(\nu\ a)\ P$ / $(\nu\ \kappa)\ P$	(res)
	$P \mid Q$	(par)
	if e **then** P **else** P	(cond)
	0	(inact)
	$\mu X.\ P$	(rec)
	X	(recVar)

π-CALCULUS WITH SESSIONS

P :: =

~~$*a(\kappa).\ P$~~	~~(service)~~
~~$a(\kappa).\ P$~~	~~(request)~~
$!a(\kappa)[\ P, Q\ ]$	(service)
$a(\kappa')[\ \tilde{\kappa}, P, Q\ ]$	(request)
throw	(throw)
(υ a**)** P / **(υ** κ**)** P	(res)
P | Q	(par)
if e **then** P **else** P	(cond)
0	(inact)
μX. P	(rec)
X	(recVar)

Refinement and other assumptions

Syntactic Assumptions. In order to have consistent operational semantics, we stipulate the following syntactic constraints:

1. *(Consistent Refinement)* given $\overline{c}(\lambda)[\tilde{\kappa}, P, Q]$, for each $\overline{c'}(\lambda')[\tilde{\kappa}', P', Q']$ occurring in P and any $\kappa_i \in \tilde{\kappa}$, we have $\kappa_i \in \tilde{\kappa}'$ implies $\tilde{\kappa} \subseteq \tilde{\kappa}'$ (for consistent refinement). Further, such a refinement never occurs inside a handler (otherwise we have ambiguity when launching a handler);

2. recursions is *guarded*, i.e. P in $\mu X.\ P$ is prefixed by an input, output, branch, select or conditional; moreover, a free term variable never occurs free in $\overline{c}(\lambda)[\tilde{\kappa}, P, Q]$;

3. the term (accept) never occurs under an input/output/recursion prefix nor inside a default process or handler thus protecting its availability from exceptions;

4. **throw** never occurs inside a handler hence preventing a handler from throwing a further exception in the same session.

Refinement and other assumptions

Syntactic Assumptions. In order to have consistent operational semantics, we stipulate the following syntactic constraints:

1. *(Consistent Refinement)* given $\overline{c}(\lambda)[\tilde{\kappa}, P, Q]$, for each $\overline{c'}(\lambda')[\tilde{\kappa}', P', Q']$ occurring in P and any $\kappa_i \in \tilde{\kappa}$, we have $\kappa_i \in \tilde{\kappa}'$ implies $\tilde{\kappa} \subseteq \tilde{\kappa}'$ (for consistent refinement). Further, such a refinement never occurs inside a handler (otherwise we have ambiguity when launching a handler);

2. recursions is *guarded*, i.e. P in $\mu X.\ P$ is prefixed by an input, output, branch, select or conditional; moreover, a free term variable never occurs free in $\overline{c}(\lambda)[\tilde{\kappa}, P, Q]$;

3. the term (accept) never occurs under an input/output/recursion prefix nor inside a default process or handler thus protecting its availability from exceptions;

4. **throw** never occurs inside a handler hence preventing a handler from throwing a further exception in the same session.

Refinement and other assumptions

Syntactic Assumptions. In order to have consistent operational semantics, we stipulate the following syntactic constraints:

1. *(Consistent Refinement)* given $\overline{c}(\lambda)[\tilde{\kappa}, P, Q]$, for each $\overline{c'}(\lambda')[\tilde{\kappa}', P', Q']$ occurring in P and any $\kappa_i \in \tilde{\kappa}$, we have $\kappa_i \in \tilde{\kappa}'$ implies $\tilde{\kappa} \subseteq \tilde{\kappa}'$ (for consistent refinement). Further, such a refinement never occurs inside a handler (otherwise we have ambiguity when launching a handler);

2. recursions is *guarded*, i.e. P in $\mu X.\, P$ is prefixed by an input, output, branch, select or conditional; moreover, a free term variable never occurs free in $\overline{c}(\lambda)[\tilde{\kappa}, P, Q]$;

3. the term (accept) never occurs under an input/output/recursion prefix nor inside a default process or handler thus protecting its availability from exceptions;

4. **throw** never occurs inside a handler hence preventing a handler from throwing a further exception in the same session.

Refinement and other assumptions

Syntactic Assumptions. In order to have consistent operational semantics, we stipulate the following syntactic constraints:

1. *(Consistent Refinement)* given $\overline{c}(\lambda)[\tilde{\kappa}, P, Q]$, for each $\overline{c'}(\lambda')[\tilde{\kappa}', P', Q']$ occurring in P and any $\kappa_i \in \tilde{\kappa}$, we have $\kappa_i \in \tilde{\kappa}'$ implies $\tilde{\kappa} \subseteq \tilde{\kappa}'$ (for consistent refinement). Further, such a refinement never occurs inside a handler (otherwise we have ambiguity when launching a handler);

2. recursions is *guarded*, i.e. P in $\mu X.\ P$ is prefixed by an input, output, branch, select or conditional; moreover, a free term variable never occurs free in $\overline{c}(\lambda)[\tilde{\kappa}, P, Q]$;

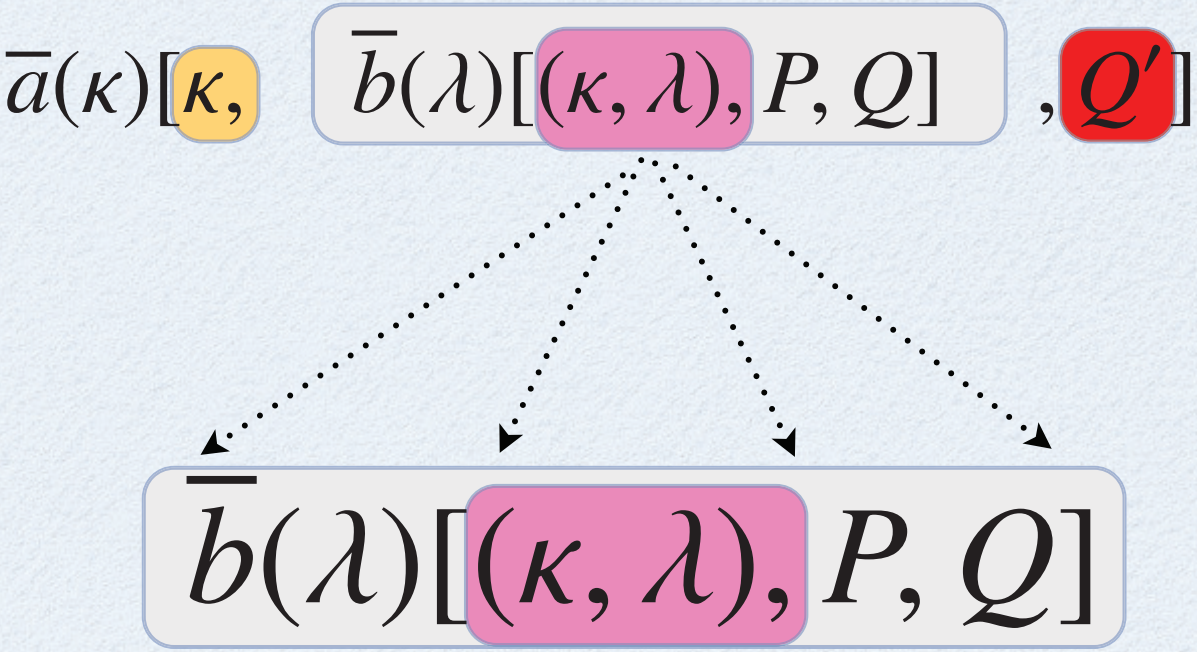
3. the term (accept) never occurs under an input/output/recursion prefix nor inside a default process or handler thus protecting its availability from exceptions;

4. **throw** never occurs inside a handler hence preventing a handler from throwing a further exception in the same session.

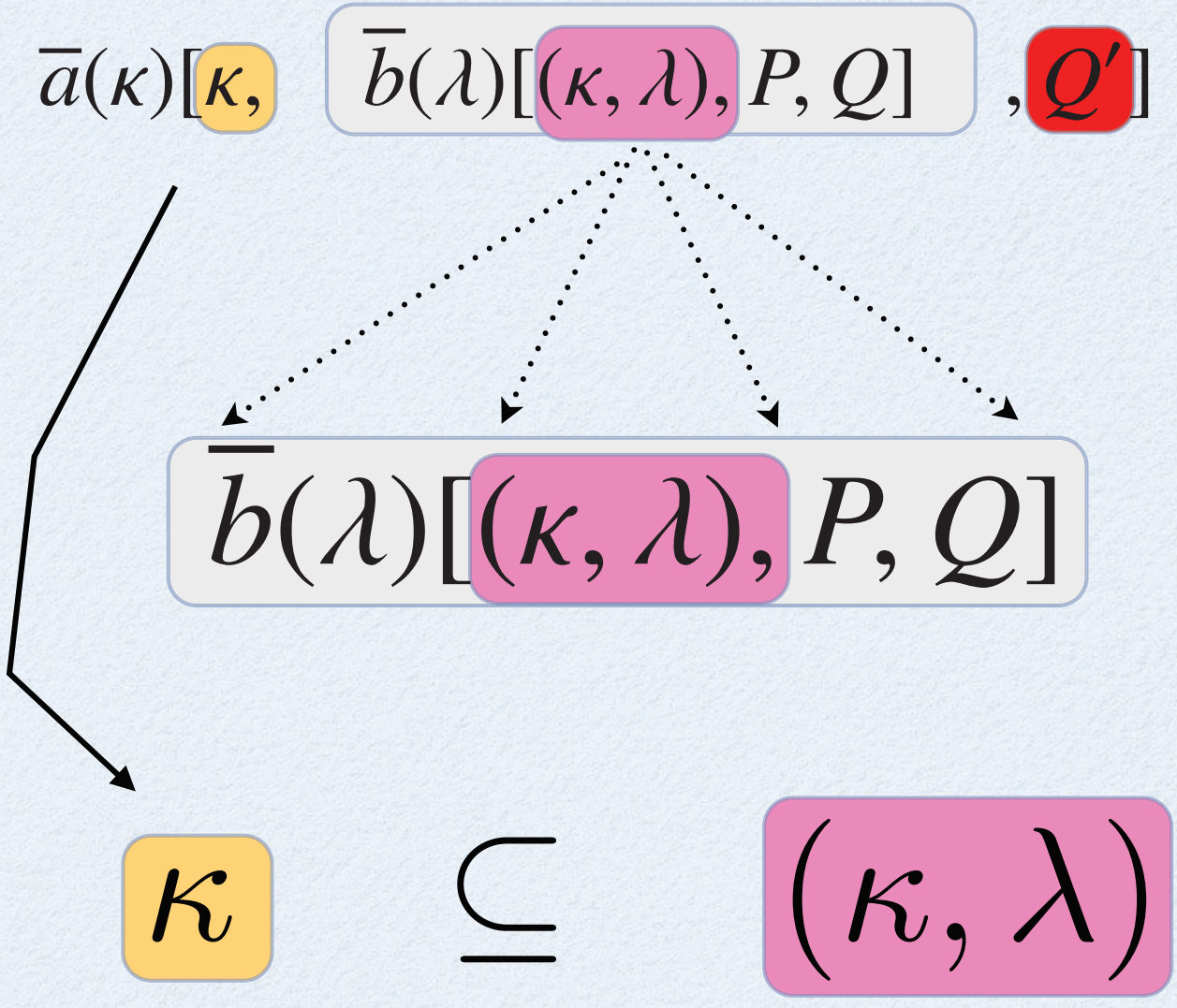
REFINEMENT

$$\overline{a}(\kappa)[\kappa,\ \overline{b}(\lambda)[(\kappa,\lambda), P, Q]\ , Q']$$

REFINEMENT


$\overline{a}(\kappa)[\kappa,$
$\overline{b}(\lambda)[(\kappa,\lambda), P, Q]$
$, Q']$
$\overline{b}(\lambda)[(\kappa,\lambda), P, Q]$

REFINEMENT


$\overline{a}(\kappa)[\kappa,$
$\overline{b}(\lambda)[(\kappa,\lambda), P, Q]$
$, Q']$
$\overline{b}(\lambda)[(\kappa,\lambda), P, Q]$
κ
$\subseteq$
(κ,λ)

REFINEMENT (2)

$$\overline{a}(\kappa)[(\kappa, \kappa'), \quad \overline{b}(\lambda)[(\lambda, \kappa), P, Q], \quad Q']$$

REFINEMENT (2)

$$\overline{a}(\kappa)[(\kappa, \kappa'), \quad \overline{b}(\lambda)[(\lambda, \kappa), P, Q], \quad Q']$$

$$(\kappa, \kappa') \not\subseteq (\lambda, \kappa)$$

SEMANTICS: RUN-TIME SYNTAX

Technically challenging - hard to preserve session structure

We wish to have asynchronous ordered communication!

We need to have a *run-time* syntax which extends the static one as follows:

$$\mathbf{try}\{P\}\ \mathbf{catch}\{\tilde{\kappa} : Q\} \qquad \text{(try-catch)}$$

$$\tilde{\kappa}\{\!\{Q\}\!\} \qquad \text{(wrap)}$$

$$\kappa \hookrightarrow_{\phi} \overline{\kappa} : L \qquad \text{(queue)}$$

$$(\nu s)P \qquad \text{(res)}$$

SEMANTICS: SESSION INIT

$$!a(s)[P, Q] \quad | \quad C[\,\overline{a}(s)[\tilde{\kappa}, P', Q']\,]$$

SEMANTICS: SESSION INIT

$$!a(s)[P,Q] \quad | \quad C[\,\overline{a}(s)[\tilde{\kappa},P',Q']\,]$$

$$\longrightarrow$$

$$!a(s)[P.Q] \;|$$

SEMANTICS: SESSION INIT

$$!a(s)[P,Q] \quad | \quad C[\,\overline{a}(s)[\tilde{\kappa},P',Q']\,]$$

$$\longrightarrow$$

$$!a(s)[P.Q] \mid (\nu s)\Big(\qquad\qquad\Big)$$

SEMANTICS: SESSION INIT

$$!a(s)[P,Q] \quad | \quad C[\,\overline{a}(s)[\tilde{\kappa},P',Q']\,]$$

$$\longrightarrow$$

$$!a(s)[P.Q] \;|\; (\nu s)\left(\begin{array}{l}\mathbf{try}\{P\}\;\mathbf{catch}\{s^{-}:Q\}\;|\\ \\ \end{array}\right)$$

SEMANTICS: SESSION INIT

$$!a(s)[P, Q] \quad | \quad C[\ \overline{a}(s)[\tilde{\kappa}, P', Q']\ ]$$

$$\longrightarrow$$

$$!a(s)[P.Q] \ | \ (\nu s)\left(\begin{array}{l} \mathbf{try}\{P\}\ \mathbf{catch}\{s^- : Q\} \ | \\ C[\ \mathbf{try}\{P'\}\ \mathbf{catch}\{\tilde{\kappa} : Q'\}\ ] \ | \end{array}\right)$$

SEMANTICS: SESSION INIT

$$!a(s)[P,Q] \quad | \quad C[\ \overline{a}(s)[\tilde{\kappa},P',Q']\ ]$$

$$\longrightarrow$$

$$!a(s)[P.Q] \ | \ (\nu s)\left(\begin{array}{l}\mathbf{try}\{P\}\ \mathbf{catch}\{s^- : Q\} \ | \\ C[\ \mathbf{try}\{P'\}\ \mathbf{catch}\{\tilde{\kappa} : Q'\}\ ] \ | \\ s^- \hookrightarrow_0 s^+ : \epsilon \ |\end{array}\right)$$

SEMANTICS: SESSION INIT

$$!a(s)[P,Q] \quad | \quad C[\,\overline{a}(s)[\tilde{\kappa},P',Q']\,]$$

$$\longrightarrow$$

$$!a(s)[P.Q] \;|\; (\nu s)\left(\begin{array}{l}\mathbf{try}\{P\}\;\mathbf{catch}\{s^- : Q\} \;|\\ C[\,\mathbf{try}\{P'\}\;\mathbf{catch}\{\tilde{\kappa} : Q'\}\,] \;|\\ s^- \hookrightarrow_0 s^+ : \epsilon \;|\\ s^+ \hookrightarrow_0 s^- : \epsilon\end{array}\right)$$

WHY WRAPS?

$$\mathbf{try}\{\ \ P' \mid \mathbf{try}\{\ P\ \}\ \mathbf{catch}\ \{\ \lambda :\ Q\ \}\ \ \}\ \ \mathbf{catch}\ \ \{\ \kappa :\ Q'\ \}$$

WHY WRAPS?

$$\mathbf{try}\{\quad P' \quad|\quad \mathbf{try}\{\ P\ \}\ \mathbf{catch}\ \{\ \lambda:\ Q\ \}\quad \}\quad \mathbf{catch}\quad \{\ \kappa:\ Q'\ \}$$

- P throws an exception

$$\mathbf{try}\{\quad P' \quad|\quad Q \quad \}\quad \mathbf{catch}\quad \{\ \kappa:\ Q'\ \}$$

WHY WRAPS?

$$\mathbf{try}\{\ \ P' \mid \mathbf{try}\{\ P\ \}\ \mathbf{catch}\ \{\ \lambda :\ Q\ \}\ \ \}\ \ \mathbf{catch}\ \ \{\ \kappa :\ Q'\ \}$$

- P throws an exception

$$\mathbf{try}\{\ \ P' \mid Q\ \ \}\ \ \mathbf{catch}\ \ \{\ \kappa :\ Q'\ \}$$

- P' throws an exception

- *What happens to Q?*

WHY WRAPS?

$$\mathbf{try}\{\quad P' \quad|\quad \mathbf{try}\{\ P\ \}\ \mathbf{catch}\ \{\ \lambda:\ Q\ \}\quad \}\quad \mathbf{catch}\quad \{\ \kappa:\ Q'\ \}$$

- P throws an exception

$$\mathbf{try}\{\quad P' \quad|\quad \lambda\{\{Q\}\}\quad \}\quad \mathbf{catch}\quad \{\ \kappa:\ Q'\ \}$$

META REDUCTION

When a process must be suddenly terminated, *Meta Reduction will* take care of:

i) propagation (partly)
ii) wrapped processes

Meta reduction is a relation on processes:

$$P \searrow (P', S)$$

where

i) *S* is a set of session names (keep track of propagation)
ii) *P'* contains wraps only (in parallel and/or nested)

Meta Reduction (2)

Some rules:

$$P \searrow (P', S)$$

META REDUCTION (2)

Some rules:

$$\frac{P \searrow (P', S)}{\mathbf{try}\{P\}\ \mathbf{catch}\{\tilde{\kappa} : Q\} \ \searrow \ (\tilde{\kappa}\{\!\{Q\}\!\} \mid P' \ , \ S \cup \tilde{\kappa})}$$

META REDUCTION (2)

Some rules:

$$\frac{P \searrow (P', S)}{\mathbf{try}\{P\}\ \mathbf{catch}\{\tilde{\kappa} : Q\} \ \searrow\ (\tilde{\kappa}\{\!\{Q\}\!\} \mid P'\ ,\ S \cup \tilde{\kappa})}$$

$$\tilde{\kappa}\{\!\{Q\}\!\} \ \searrow\ (\tilde{\kappa}\{\!\{Q\}\!\}\ ,\ \emptyset)$$

EXAMPLE OF META REDUCTION

$$\mathbf{try}\{\quad \mathbf{throw} \;\mid\; \mathbf{try}\{\, P \,\}\, \mathbf{catch}\, \{\, \lambda : \; Q \,\} \quad \}\quad \mathbf{catch}\, \{\, \kappa : \; Q' \,\}$$

EXAMPLE OF META REDUCTION

$$\mathbf{try}\{\quad \mathbf{throw} \quad|\quad \mathbf{try}\{\ P\ \}\ \mathbf{catch}\ \{\ \lambda :\ Q\ \}\quad \}\quad \mathbf{catch}\ \{\ \kappa :\ Q'\ \}$$

- if **throw** is exec then we must deal with

EXAMPLE OF META REDUCTION

$$\mathbf{try}\{\ \ \mathbf{throw}\ \mid\ \mathbf{try}\{\ P\ \}\ \mathbf{catch}\ \{\ \lambda:\ Q\ \}\ \ \}\ \ \mathbf{catch}\ \{\ \kappa:\ Q'\ \}$$

- if **throw** is exec then we must deal with

$$\mathbf{throw}\ \mid\ \mathbf{try}\{\ P\ \}\ \mathbf{catch}\ \{\ \lambda:\ Q\ \}$$

EXAMPLE OF META REDUCTION

$$\mathbf{try}\{\quad \mathbf{throw} \;\mid\; \mathbf{try}\{\, P \,\}\, \mathbf{catch}\, \{\, \lambda :\; Q \,\} \quad \}\quad \mathbf{catch}\, \{\, \kappa :\; Q' \,\}$$

- if **throw** is exec then we must deal with

$$\mathbf{throw} \;\mid\; \mathbf{try}\{\, P \,\}\, \mathbf{catch}\, \{\, \lambda :\; Q \,\}$$

- we get (i) something from P and $\lambda\{\{Q\}\}$
- and (ii) we must propagate over λ

$$(P'' \;\mid\; \lambda\{\{Q\}\},\quad \{\lambda\})$$

SEMANTICS (3): THROW

$$\mathbf{try}\{\ \mathbf{throw} \mid P\ \}\ \mathbf{catch}\ \{\kappa, \kappa' : Q\} \mid \kappa \hookrightarrow_0 \overline{\kappa} : L \mid \kappa' \hookrightarrow_0 \overline{\kappa}' : L'$$

SEMANTICS (3): THROW

$$\mathbf{try}\{\ P\ \}\ \mathbf{catch}\ \{\kappa, \kappa' : Q\}\ \searrow\ (R, S)$$

$$\mathbf{try}\{\ \mathbf{throw} \mid P\ \}\ \mathbf{catch}\ \{\kappa, \kappa' : Q\}\ |\ \kappa \hookrightarrow_0 \overline{\kappa} : L \mid \kappa' \hookrightarrow_0 \overline{\kappa}' : L'$$

SEMANTICS (3): THROW

$$\mathbf{try}\{\ P\ \}\ \mathbf{catch}\ \{\kappa, \kappa' : Q\}\ \searrow\ (R, S)$$

$$\mathbf{try}\{\ \mathbf{throw} \mid P\ \}\ \mathbf{catch}\ \{\kappa, \kappa' : Q\} \mid \kappa \hookrightarrow_0 \overline{\kappa} : L \mid \kappa' \hookrightarrow_0 \overline{\kappa}' : L'$$

$$\longrightarrow$$

$$R \mid \kappa \hookrightarrow_0 \overline{\kappa} : (\dagger :: L) \mid \kappa' \hookrightarrow_0 \overline{\kappa}' : (\dagger :: L')$$

SEMANTICS (3): THROW

$$\mathbf{try}\{\ P\ \}\ \mathbf{catch}\ \{\kappa, \kappa' : Q\} \ \searrow\ (R, S)$$

$$\mathbf{try}\{\ \mathbf{throw} \mid P\ \}\ \mathbf{catch}\ \{\kappa, \kappa' : Q\} \mid \kappa \hookrightarrow_0 \overline{\kappa} : L \mid \kappa' \hookrightarrow_0 \overline{\kappa}' : L'$$

$$\kappa_1 \hookrightarrow \overline{\kappa_1} : L_1 \mid \ \ldots\ \mid \kappa_n \hookrightarrow \overline{\kappa_n} : L_n$$

$$\longrightarrow$$

for all $\kappa_i \in S$

$$R \mid \kappa \hookrightarrow_0 \overline{\kappa} : (\dagger :: L) \mid \kappa' \hookrightarrow_0 \overline{\kappa}' : (\dagger :: L')$$

$$\kappa_1 \hookrightarrow \overline{\kappa_1} : (\dagger :: L_1) \mid \ \ldots\ \mid \kappa_n \hookrightarrow \overline{\kappa_n} : (\dagger :: L_n)$$

Process P may contain other try catch blocks!

SEMANTICS (4): RTHROW

$$\mathbf{try}\{P\}\ \mathbf{catch}\{\kappa, \kappa' : Q\} \mid \overline{\kappa} \hookrightarrow_0 \kappa : (L'' :: \dagger) \mid \kappa \hookrightarrow_0 \overline{\kappa} : L \mid \kappa' \hookrightarrow_0 \overline{\kappa}' : L'$$

SEMANTICS (4): RTHROW

$$\mathbf{try}\{P\}\ \mathbf{catch}\ \{\kappa, \kappa' : Q\} \searrow (R, S)$$

$$\mathbf{try}\{P\}\ \mathbf{catch}\{\kappa, \kappa' : Q\} \mid \overline{\kappa} \hookrightarrow_0 \kappa : (L'' :: \dagger) \mid \kappa \hookrightarrow_0 \overline{\kappa} : L \mid \kappa' \hookrightarrow_0 \overline{\kappa}' : L'$$

SEMANTICS (4): RTHROW

$$\mathbf{try}\{P\}\ \mathbf{catch}\ \{\kappa, \kappa' : Q\} \searrow (R, S)$$

$$\mathbf{try}\{P\}\ \mathbf{catch}\{\kappa, \kappa' : Q\} \mid \overline{\kappa} \hookrightarrow_0 \kappa : (L'' :: \dagger) \mid \kappa \hookrightarrow_0 \overline{\kappa} : L \mid \kappa' \hookrightarrow_0 \overline{\kappa}' : L'$$

$$\longrightarrow$$

$$R \mid \overline{\kappa} \hookrightarrow_1 \kappa : L'' \mid \kappa \hookrightarrow_0 \overline{\kappa} : (\dagger :: L) \mid \kappa' \hookrightarrow_0 \overline{\kappa}' : (\dagger :: L')$$

SEMANTICS (4): RTHROW

$$\mathbf{try}\{P\}\ \mathbf{catch}\ \{\kappa, \kappa' : Q\} \searrow (R, S)$$

$$\mathbf{try}\{P\}\ \mathbf{catch}\{\kappa, \kappa' : Q\} \mid \overline{\kappa} \hookrightarrow_0 \kappa : (L'' :: \dagger) \mid \kappa \hookrightarrow_0 \overline{\kappa} : L \mid \kappa' \hookrightarrow_0 \overline{\kappa}' : L'$$

$$\kappa_1 \hookrightarrow \overline{\kappa_1} : L_1 \mid \ldots \mid \kappa_n \hookrightarrow \overline{\kappa_n} : L_n$$

$$\longrightarrow$$

for all $\kappa_i \in S$

$$R \mid \overline{\kappa} \hookrightarrow_1 \kappa : L'' \mid \kappa \hookrightarrow_0 \overline{\kappa} : (\dagger :: L) \mid \kappa' \hookrightarrow_0 \overline{\kappa}' : (\dagger :: L')$$

$$\kappa_1 \hookrightarrow \overline{\kappa_1} : (\dagger :: L_1) \mid \ldots \mid \kappa_n \hookrightarrow \overline{\kappa_n} : (\dagger :: L_n)$$

STRUCTURAL CONGR.

$$(\lambda \in \tilde{\kappa} \Rightarrow \dagger \notin L)$$

$$\textbf{try}\{\, P \mid \lambda \hookrightarrow_{\phi} \overline{\lambda} : L \,\}\ \textbf{catch}\ \{\, \tilde{\kappa} : Q \,\} \ \equiv\ \textbf{try}\{\, P \,\}\ \textbf{catch}\ \{\, \tilde{\kappa} : Q \,\} \mid \lambda \hookrightarrow_{\phi} \overline{\lambda} : L$$

$$\tilde{\kappa}\{\!|P \mid \overline{\lambda} \hookrightarrow_{\phi} \lambda : L|\!\} \ \equiv\ \tilde{\kappa}\{\!|P|\!\} \mid \overline{\lambda} \hookrightarrow_{\phi} \lambda : L \qquad (\lambda \notin \tilde{\kappa})$$

$$\tilde{\kappa}\{\!|P|\!\} \mid \overline{\kappa}_i \hookrightarrow_{\phi} \kappa_i : L \ \equiv\ \tilde{\kappa}\{\!|P \mid \overline{\kappa}_i \hookrightarrow_{\phi-1} \kappa_i : L|\!\}$$

$$\textbf{try}\{\, (\nu a)\, P \,\}\ \textbf{catch}\ \{\, \tilde{\kappa} : Q \,\} \ \equiv\ (\nu a)\ \textbf{try}\{\, P \,\}\ \textbf{catch}\ \{\, \tilde{\kappa} : Q \,\} \qquad (a \notin fn(Q))$$

$$\tilde{\kappa}\{\!|(\nu a)\, P|\!\} \ \equiv\ (\nu a)\, \tilde{\kappa}\{\!|P|\!\}$$

EXAMPLE

$$\mathbf{try}\{\ \mathbf{throw} \mid \kappa!\langle 5\rangle\ \}\ \mathbf{catch}\ \{\kappa : \kappa!\langle \mathtt{tt}\rangle\} \mid \kappa \hookrightarrow_0 \overline{\kappa} : \epsilon \mid$$
$$\mathbf{try}\{\ \mathbf{throw} \mid \overline{\kappa}?(x)\ \}\ \mathbf{catch}\ \{\overline{\kappa} : \overline{\kappa}?(x)\} \mid \overline{\kappa} \hookrightarrow_0 \kappa : \epsilon$$

$$\longrightarrow$$

$$\kappa\{\!|\kappa!\langle \mathtt{tt}\rangle|\!\} \mid \kappa \hookrightarrow_0 \overline{\kappa} : \dagger \mid$$
$$\mathbf{try}\{\ \mathbf{throw} \mid \overline{\kappa}?(x)\ \}\ \mathbf{catch}\ \{\overline{\kappa} : \overline{\kappa}?(x)\} \mid \overline{\kappa} \hookrightarrow_0 \kappa : \epsilon$$

$$\longrightarrow$$

$$\kappa\{\!|\mathbf{0}|\!\} \mid \kappa \hookrightarrow_0 \overline{\kappa} : (\mathtt{tt} :: \dagger) \mid$$
$$\mathbf{try}\{\ \mathbf{throw} \mid \overline{\kappa}?(x)\ \}\ \mathbf{catch}\ \{\overline{\kappa} : \overline{\kappa}?(x)\} \mid \overline{\kappa} \hookrightarrow_0 \kappa : \epsilon$$

EXAMPLE

$$\kappa\{[\![\mathbf{0}]\!] \mid \kappa \hookrightarrow_0 \overline{\kappa} : (\mathtt{tt} :: \dagger) \mid \mathbf{try}\{\, \mathbf{throw} \mid \overline{\kappa}?(x) \,\}\, \mathbf{catch}\, \{\, \overline{\kappa} : \overline{\kappa}?(x) \,\} \mid \overline{\kappa} \hookrightarrow_0 \kappa : \epsilon$$

$$\longrightarrow$$

$$\kappa\{[\![\mathbf{0}]\!] \mid \kappa \hookrightarrow_1 \overline{\kappa} : \mathtt{tt} \mid \overline{\kappa}\{[\![\overline{\kappa}?(x)]\!]\} \mid \overline{\kappa} \hookrightarrow_0 \kappa : \dagger$$

$$\longrightarrow$$

$$\kappa\{[\![\mathbf{0}]\!] \mid \overline{\kappa} \hookrightarrow_1 \kappa : \epsilon \mid \kappa \hookrightarrow_1 \overline{\kappa} : \mathtt{tt} \mid \overline{\kappa}\{[\![\overline{\kappa}?(x)]\!]\}$$

OTHER RULES

$$\tilde{\kappa}\{\!\{Q\}\!\} \mid \overline{\kappa}_i \hookrightarrow_0 \kappa_i : (L :: v) \longrightarrow \tilde{\kappa}\{\!\{Q\}\!\} \mid \overline{\kappa}_i \hookrightarrow_0 \kappa_i : L$$

$$\tilde{\kappa}\{\!\{Q\}\!\} \mid \overline{\kappa}_i \hookrightarrow_0 \kappa_i : (L :: \dagger) \longrightarrow \tilde{\kappa}\{\!\{Q\}\!\} \mid \overline{\kappa}_i \hookrightarrow_1 \kappa_i : L$$

SESSION TYPES

The grammar extends standard session types:

$$\begin{array}{rcl} \alpha, \beta & ::= & \downarrow(\theta).\,\alpha \mid \uparrow(\theta).\,\alpha \mid \oplus\{l_i : \alpha_i\}_{i \in I} \mid \&\{l_i : \alpha_i\}_{i \in I} \mid \alpha[\![\beta]\!] \mid \mathbf{end} \mid \mu \mathbf{t}.\,\alpha \mid \mathbf{t} \\ \theta & ::= & \langle \alpha[\![\beta]\!] \rangle \mid \mathtt{bool} \mid \ldots \end{array}$$

SESSION TYPES

The grammar extends standard session types:

$$\begin{array}{rcl} \alpha, \beta & ::= & \downarrow(\theta).\,\alpha \mid \uparrow(\theta).\,\alpha \mid \oplus\{l_i : \alpha_i\}_{i \in I} \mid \&\{l_i : \alpha_i\}_{i \in I} \mid \alpha\{\!\![\beta]\!\!\} \mid \mathbf{end} \mid \mu \mathbf{t}.\,\alpha \mid \mathbf{t} \\ \theta & ::= & \langle \alpha\{\!\![\beta]\!\!\} \rangle \mid \mathtt{bool} \mid \ldots \end{array}$$

try-catch type

$$\alpha\{\!\![\beta]\!\!\}$$

$$\mathbf{try}\{P\}\ \mathbf{catch}\{\tilde{\kappa} : Q\}$$

SESSION TYPES (2)

Judgements

$$\Gamma \vdash P \triangleright \Delta$$

where

- Γ contains shared types
- Δ contains session/queue types

SESSION TYPES (2)

Judgements

$$\Gamma \vdash P \triangleright \Delta$$

where

(Session Typing) $\Delta ::= \emptyset \mid \Delta,\ \kappa : \alpha \mid \Delta,\ (\kappa, \overline{\kappa}) : \alpha \mid \Delta,\ (\kappa, \overline{\kappa}) : \bot$

(Service Typing) $\Gamma ::= \emptyset \mid \Gamma,\ c : \langle \alpha \{\!| \beta |\!\} \rangle \mid c : \mathtt{bool} \mid \Gamma,\ X : \Delta$

SESSION TYPES (2)

Judgements

$$\Gamma \vdash P \triangleright \Delta$$

where

(Session Typing) $\Delta ::= \emptyset \mid \Delta,\ \kappa : \alpha \mid \Delta,\ (\kappa, \overline{\kappa}) : \alpha \mid \Delta,\ (\kappa, \overline{\kappa}) : \bot$

(Service Typing) $\Gamma ::= \emptyset \mid \Gamma,\ c : \langle \alpha \{\!| \beta |\!\} \rangle \mid c : \mathtt{bool} \mid \Gamma,\ X : \Delta$

$$\Gamma \vdash \mathbf{try}\{P\}\ \mathbf{catch}\{\kappa : Q\} \triangleright \Delta \cdot \kappa :_p \alpha\{\{\beta\}\}$$

SESSION TYPES (2)

Judgements

$$\Gamma \vdash P \triangleright \Delta$$

where

(Session Typing) $\quad \Delta ::= \emptyset \mid \Delta,\ \kappa : \alpha \mid \Delta,\ (\kappa, \overline{\kappa}) : \alpha \mid \Delta,\ (\kappa, \overline{\kappa}) : \bot$

(Service Typing) $\quad \Gamma ::= \emptyset \mid \Gamma,\ c : \langle \alpha \{\!| \beta |\!\} \rangle \mid c : \mathtt{bool} \mid \Gamma,\ X : \Delta$

$$\Gamma \vdash Q \triangleright \kappa :_u \beta$$

$$\Gamma \vdash \mathbf{try}\{P\}\ \mathbf{catch}\{\kappa : Q\} \triangleright \Delta \cdot \kappa :_p \alpha\{\{\beta\}\}$$

SESSION TYPES (2)

Judgements

$$\Gamma \vdash P \triangleright \Delta$$

where

(Session Typing) $\quad \Delta ::= \emptyset \mid \Delta,\ \kappa : \alpha \mid \Delta,\ (\kappa, \overline{\kappa}) : \alpha \mid \Delta,\ (\kappa, \overline{\kappa}) : \bot$

(Service Typing) $\quad \Gamma ::= \emptyset \mid \Gamma,\ c : \langle \alpha \{\!\![\beta]\!\!\} \rangle \mid c : \mathtt{bool} \mid \Gamma,\ X : \Delta$

$$\frac{\Gamma \vdash Q \triangleright \kappa :_u \beta \qquad \Gamma \vdash P \triangleright \Delta \cdot \kappa :_\rho \alpha\{\{\beta\}\}}{\Gamma \vdash \mathbf{try}\{P\}\ \mathbf{catch}\{\kappa : Q\} \triangleright \Delta \cdot \kappa :_p \alpha\{\{\beta\}\}}$$

all protected ("p") in Δ - $\varrho \in \{p,u\}$

TYPING RULES

$$\frac{\mathsf{fv}(\Gamma) = \emptyset \quad \alpha_i \in \{\mathsf{end}, \mathsf{end}\{\![\beta_i]\!\}\}}{\Gamma \vdash \mathbf{0} \triangleright \prod_i \kappa_i : \alpha_i}$$

$$\frac{\Gamma, a : \langle \alpha\{\![\beta]\!\} \rangle \vdash P \triangleright \Delta}{\Gamma \vdash (\nu a)\, P \triangleright \Delta}$$

TYPING RULES: INIT

$$\frac{\begin{array}{l}\Gamma \vdash P \triangleright \prod_i \kappa_i : \overline{\alpha}_i\{\![\overline{\beta}_i]\!\} \\ \Gamma' \vdash Q \triangleright \prod_i \kappa_i : \overline{\beta}_i \qquad s^+ = \kappa_j \\ \Gamma \vdash c : \langle \alpha_j\{\![\beta_j]\!\}\rangle \qquad \Gamma' \subseteq \Gamma,\ \mathsf{fv}(\Gamma') = \emptyset\end{array}}{\Gamma \vdash \overline{c}(s^+)[\tilde{\kappa}, P, Q] \triangleright \prod_{i \neq j} \kappa_i : \overline{\alpha}_i\{\![\overline{\beta}_i]\!\}}$$

$$\frac{\begin{array}{l}\Gamma \vdash P \triangleright s^- : \alpha\{\![\beta]\!\} \\ \Gamma \vdash Q \triangleright s^- : \beta \qquad \mathsf{fv}(\Gamma) = \emptyset\end{array}}{\Gamma,\ a : \langle \alpha\{\![\beta]\!\}\rangle \vdash *a(s^-)[P, Q] \triangleright \emptyset}$$

Typing Rules: In/Out

$$\frac{\Gamma, x : \theta \vdash P \triangleright \Delta \cdot \kappa : \alpha}{\Gamma \vdash \kappa?(x).\, P \triangleright \Delta \cdot \kappa : \downarrow(\theta).\, \alpha}$$

$$\frac{\Gamma \vdash e : \theta \qquad \Gamma \vdash P \triangleright \Delta \cdot \kappa : \alpha}{\Gamma \vdash \kappa!\langle e \rangle.\, P \triangleright \Delta \cdot \kappa : \uparrow(\theta).\, \alpha}$$

Typing Rules: Branching/ Selection

$$\frac{\Gamma \vdash P_i \triangleright \Delta \cdot \kappa : \alpha_i \quad \forall i \in I}{\Gamma \vdash \kappa \rhd \{l_i : P_i\}_{i \in I} \triangleright \Delta \cdot \kappa : \&\{l_i : \alpha_i\}_{i \in I}}$$

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot \kappa : \alpha_j}{\Gamma \vdash \kappa \lhd l_j . P \triangleright \Delta \cdot \kappa : \oplus\{l_i : \alpha_i\}_{i \in I}}$$

TYPING EXAMPLE

Let's show that:

$$\Gamma \vdash \begin{array}{l} *\texttt{chSeller}(s^-)[\\ \quad s^-!\langle \texttt{quote} \rangle, \\ \quad s^-?(y_2).\, s^-!\langle \texttt{time} \rangle\ ] \end{array} \triangleright \emptyset$$

where:

$$\Gamma = \mathsf{chSeller} : \uparrow (int).\, \texttt{end} \quad \{[\downarrow (cardt).\ \uparrow (time).\, \texttt{end}]\}$$

TYPING EXAMPLE (2)

$$\Gamma \vdash \begin{array}{l} *\texttt{chSeller}(s^-)[\\ \quad s^-!\langle \texttt{quote} \rangle, \\ \quad s^-?(y_2).\, s^-!\langle \texttt{time} \rangle\ ] \end{array} \triangleright \emptyset$$

$$\frac{\mathsf{fv}(\Gamma) = \emptyset \quad \alpha_i \in \{\mathsf{end}, \mathsf{end}\{\![\beta_i]\!\}\}}{\Gamma \vdash \mathbf{0} \triangleright \prod_i \kappa_i : \alpha_i}$$

$$\Gamma \vdash \mathbf{0} \triangleright s^- : \mathsf{end}$$

TYPING EXAMPLE (2)

$$\Gamma \vdash \begin{array}{l} *\texttt{chSeller}(s^-)[\\ \quad s^-!\langle \texttt{quote} \rangle, \\ \quad s^-?(y_2).\, s^-!\langle \texttt{time} \rangle\ ] \end{array} \triangleright \emptyset$$

$$\frac{\Gamma \vdash e : \theta \qquad \Gamma \vdash P \triangleright \Delta \cdot \kappa : \alpha}{\Gamma \vdash \kappa!\langle e \rangle.\, P \triangleright \Delta \cdot \kappa : \uparrow(\theta).\, \alpha}$$

$$\Gamma \vdash \mathbf{0} \triangleright s^- : \mathsf{end}$$

$$\Gamma \vdash s^-!\langle \texttt{quote} \rangle.\, \mathbf{0} \triangleright s^- : \uparrow(int).\, \mathsf{end}$$

TYPING EXAMPLE (2)

$$\Gamma \vdash \begin{array}{l} *\texttt{chSeller}(s^-)[\\ \quad s^-!\langle \texttt{quote} \rangle, \\ \quad s^-?(y_2).\, s^-!\langle \texttt{time} \rangle\ ] \end{array} \triangleright \emptyset$$

$$\frac{\Gamma, x : \theta \vdash P \triangleright \Delta \cdot \kappa : \alpha}{\Gamma \vdash \kappa?(x).\, P \triangleright \Delta \cdot \kappa : \downarrow(\theta).\, \alpha}$$

$$\Gamma \vdash \mathbf{0} \triangleright s^- : \mathsf{end}$$

$$\Gamma \vdash s^-!\langle \texttt{quote} \rangle.\, \mathbf{0} \triangleright s^- :\uparrow (int).\, \mathsf{end}$$

$$\Gamma \vdash s^-?(y_2).\, s^-!\langle \texttt{time} \rangle.\, \mathbf{0} \triangleright s^- :\downarrow (cardt).\, \uparrow (time)$$

TYPING EXAMPLE (3)

$$\Gamma \vdash \begin{array}{l} *\texttt{chSeller}(s^-)[\\ \quad s^-!\langle \texttt{quote} \rangle, \\ \quad s^-?(y_2).\, s^-!\langle \texttt{time} \rangle\ ] \end{array} \triangleright \emptyset$$

$$\frac{\Gamma \vdash P \triangleright s^- : \alpha\{\!|\beta|\!\} \qquad \Gamma \vdash Q \triangleright s^- : \beta \qquad \mathsf{fv}(\Gamma) = \emptyset}{\Gamma,\ a : \langle \alpha\{\!|\beta|\!\} \rangle \vdash *a(s^-)[P, Q] \triangleright \emptyset}$$

$$\Gamma \vdash \quad s^-!\langle \texttt{quote} \rangle.\, \mathbf{0} \quad \triangleright \ s^- :\uparrow (int).\, \mathsf{end}$$

$$\Gamma \vdash \ s^-?(y_2).\, s^-!\langle \texttt{time} \rangle.\, \mathbf{0} \ \triangleright \ s^- :\downarrow (cardt).\ \uparrow (time)$$

TYPING EXAMPLE (3)

$$\Gamma \vdash \begin{array}{l} *\texttt{chSeller}(s^-)[\\ \quad s^-!\langle \texttt{quote} \rangle, \\ \quad s^-?(y_2).\, s^-!\langle \texttt{time} \rangle\ ] \end{array} \triangleright \emptyset$$

$$\frac{\Gamma \vdash P \triangleright s^- : \alpha\{\!|\beta|\!\} \qquad \Gamma \vdash Q \triangleright s^- : \beta \qquad \mathsf{fv}(\Gamma) = \emptyset}{\Gamma,\ a : \langle \alpha\{\!|\beta|\!\} \rangle \vdash *a(s^-)[P, Q] \triangleright \emptyset}$$

$$\Gamma \vdash \quad s^-!\langle \texttt{quote} \rangle.\, \mathbf{0} \quad \triangleright \ s^- :\uparrow (int).\, \mathsf{end}$$

$$\Gamma \vdash \ s^-?(y_2).\, s^-!\langle \texttt{time} \rangle.\, \mathbf{0} \ \triangleright \ s^- :\downarrow (cardt).\ \uparrow (time)$$

Now we can apply the rule and finally get:

$$\Gamma \vdash \begin{array}{l} *\texttt{chSeller}(s^-)[\\ \quad s^-!\langle \texttt{quote} \rangle, \\ \quad s^-?(y_2).\, s^-!\langle \texttt{time} \rangle\ ] \end{array} \triangleright \emptyset$$

RESULTS

Theorem (Subject Reduction).

If $\Gamma \vdash P \triangleright \emptyset$ and $P \longrightarrow^* Q$ then $\Gamma \vdash Q \triangleright \emptyset$

Theorem (Communication Safety).

Communication Error Example.

$$\kappa(x).\ P \mid \kappa(x).\ P' \mid \overline{\kappa} \hookrightarrow_0 \kappa : 5$$

RESULTS

Theorem (Subject Reduction).

If $\Gamma \vdash P \triangleright \emptyset$ and $P \longrightarrow^* Q$ then $\Gamma \vdash Q \triangleright \emptyset$

Theorem (Communication Safety).

Communication Safe Example.

$$\mathbf{try}\{\ \mathbf{throw}\ |(\kappa!\langle 5\rangle)\ \}\ \mathbf{catch}\ \{\kappa : (\kappa!\langle \mathtt{tt}\rangle)\}$$
$$\mathbf{try}\{\ \ \ \ \ \ \ \ \ |(\overline{\kappa}?(x))\ \}\ \mathbf{catch}\ \{\overline{\kappa} : (\overline{\kappa}?(x))\}$$

Exceptions: Summary

- Interactional exceptions
- Asynchrony and Exceptions
- Very “wild” mechanism that can be tamed with session types
- Still missing bits e.g. delegation (name passing)

EXCEPTIONS, FUTURE WORK

- (extension) a re-try operation in the handler e.g.

  try{P}**catch** { k: **try**{P'}**catch**{k:Q} }

we would then have nested wrap as

  k{{ k{{Q}} }}

- (extension) multiparty sessions
- (extension) Termination ==> Progress (Concur08)

- integration into WS-CDL ([PLACES08])

- (future work) introduce delegation (very hard):
  - liveness?, termination?, etc.
- (future work) exception kinds (e.g. in Java)

EXERCISES

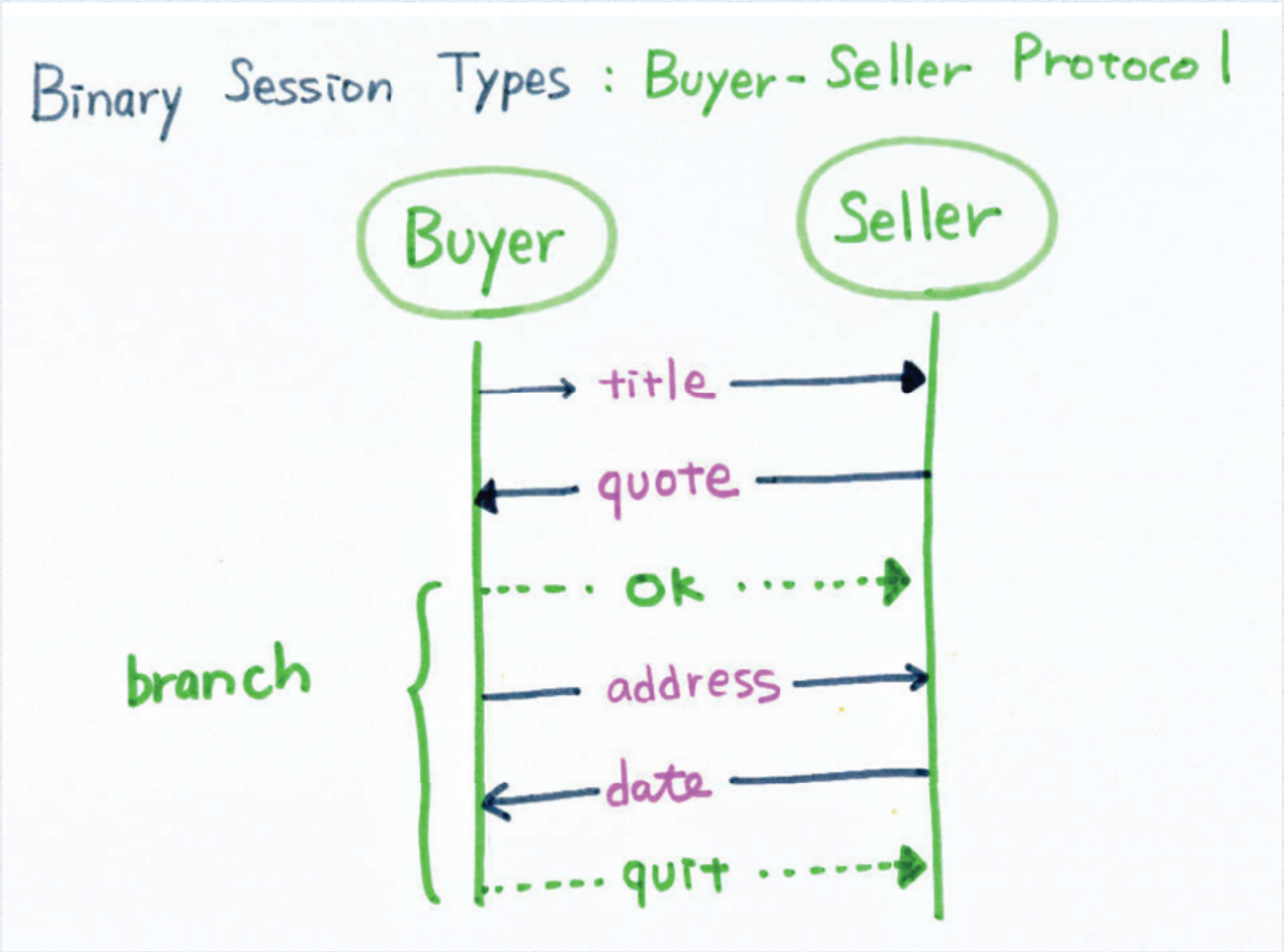
- Apply the typing rules to examples in the note.

- Prove subject reduction ;-)

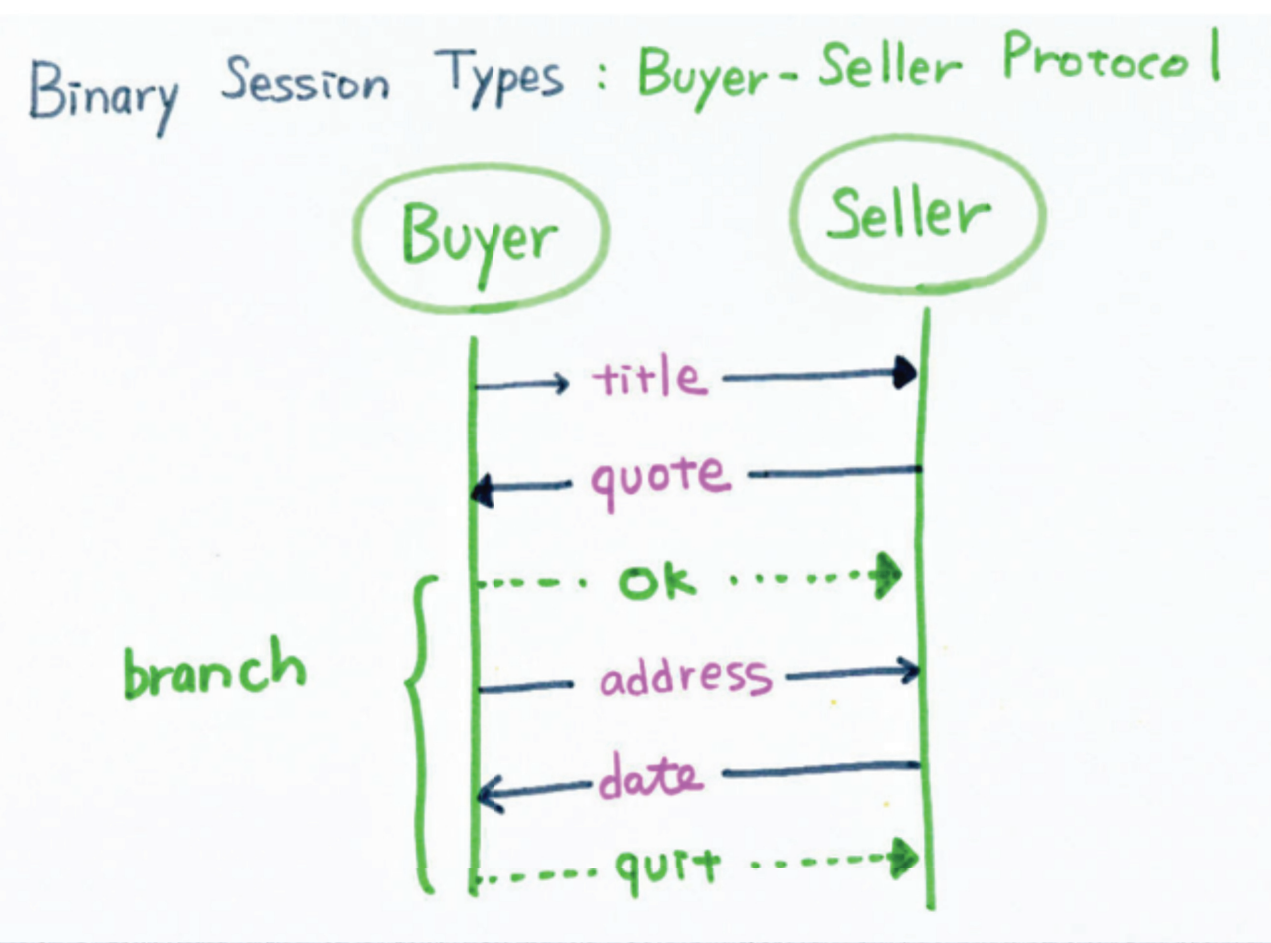
I'm here until Sunday!

Multiparty

IS BINARY ENOUGH?

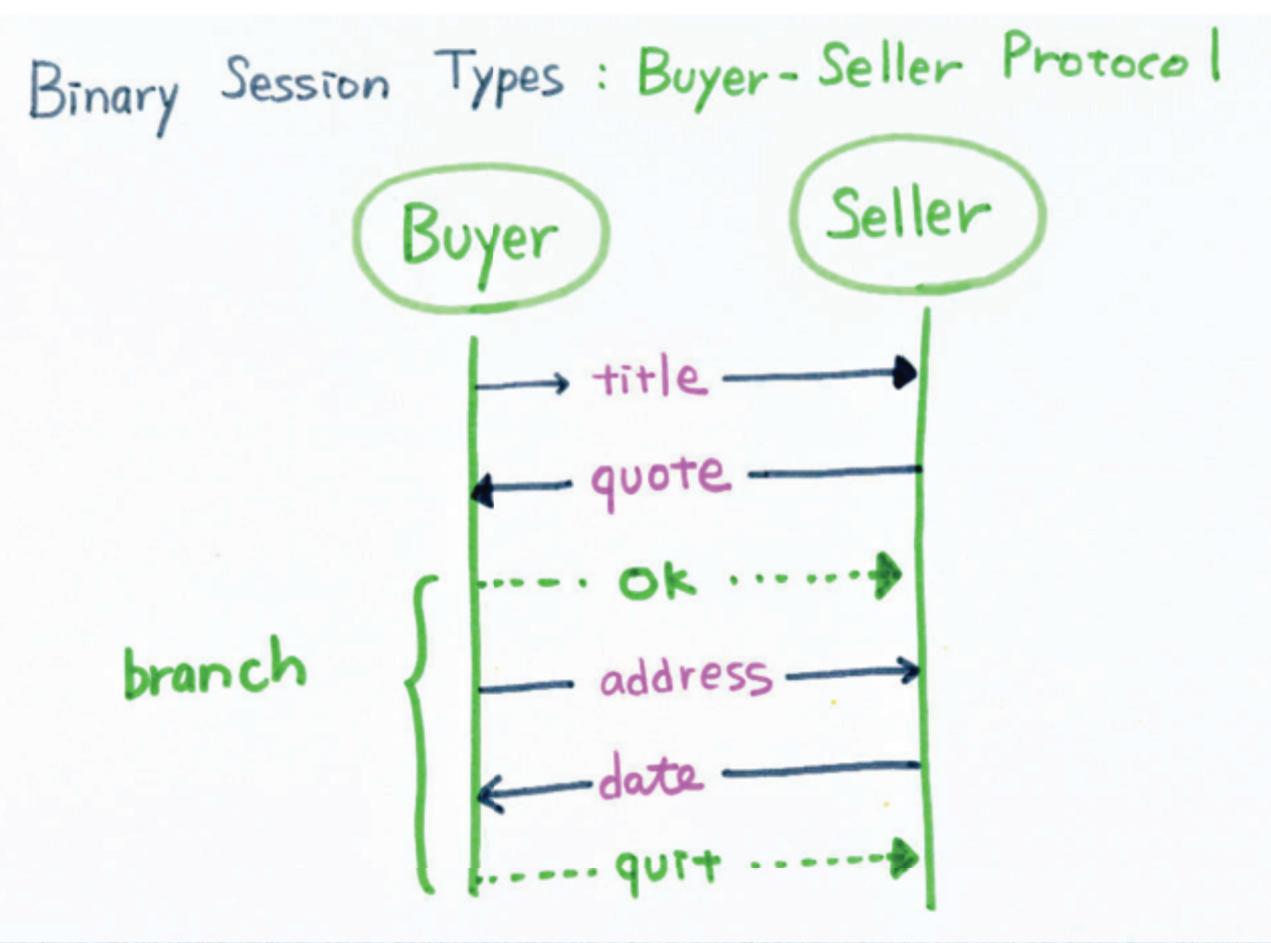


IS BINARY ENOUGH?



$$\uparrow String.\ \downarrow Int.\ \oplus \{\mathsf{ok} :\uparrow String.\ \downarrow Date.\ \mathsf{end};\quad \mathsf{quit} : \mathsf{end}\}$$

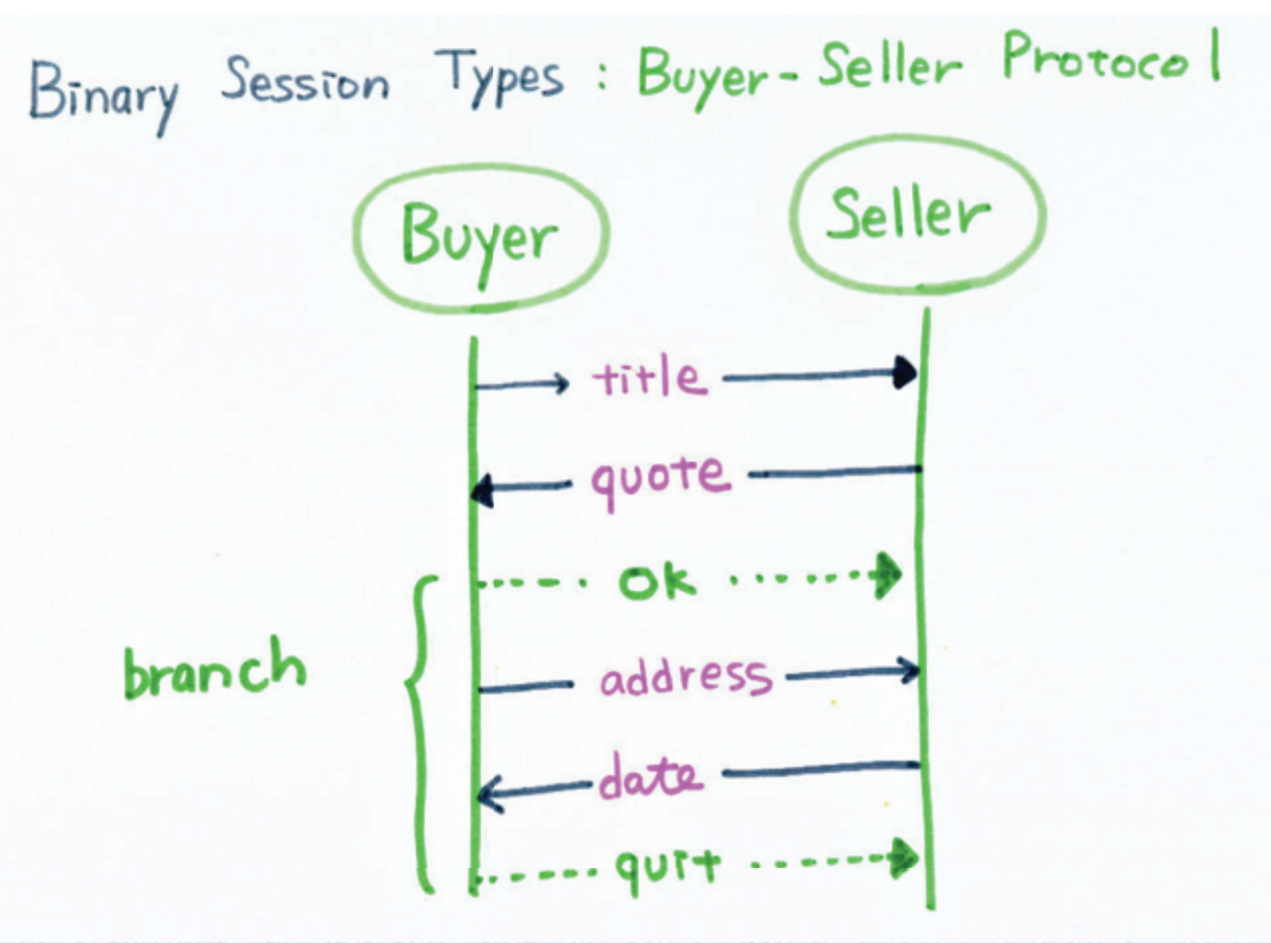
IS BINARY ENOUGH?



$\uparrow String.\ \downarrow Int.\ \oplus \{\mathsf{ok} :\uparrow String.\ \downarrow Date.\ \mathsf{end};\quad \mathsf{quit} : \mathsf{end}\}$

$\downarrow String.\ \uparrow Int.\ \& \{\mathsf{ok} :\downarrow String.\ \uparrow Date.\ \mathsf{end};\quad \mathsf{quit} : \mathsf{end}\}$

IS BINARY ENOUGH?



P has type α

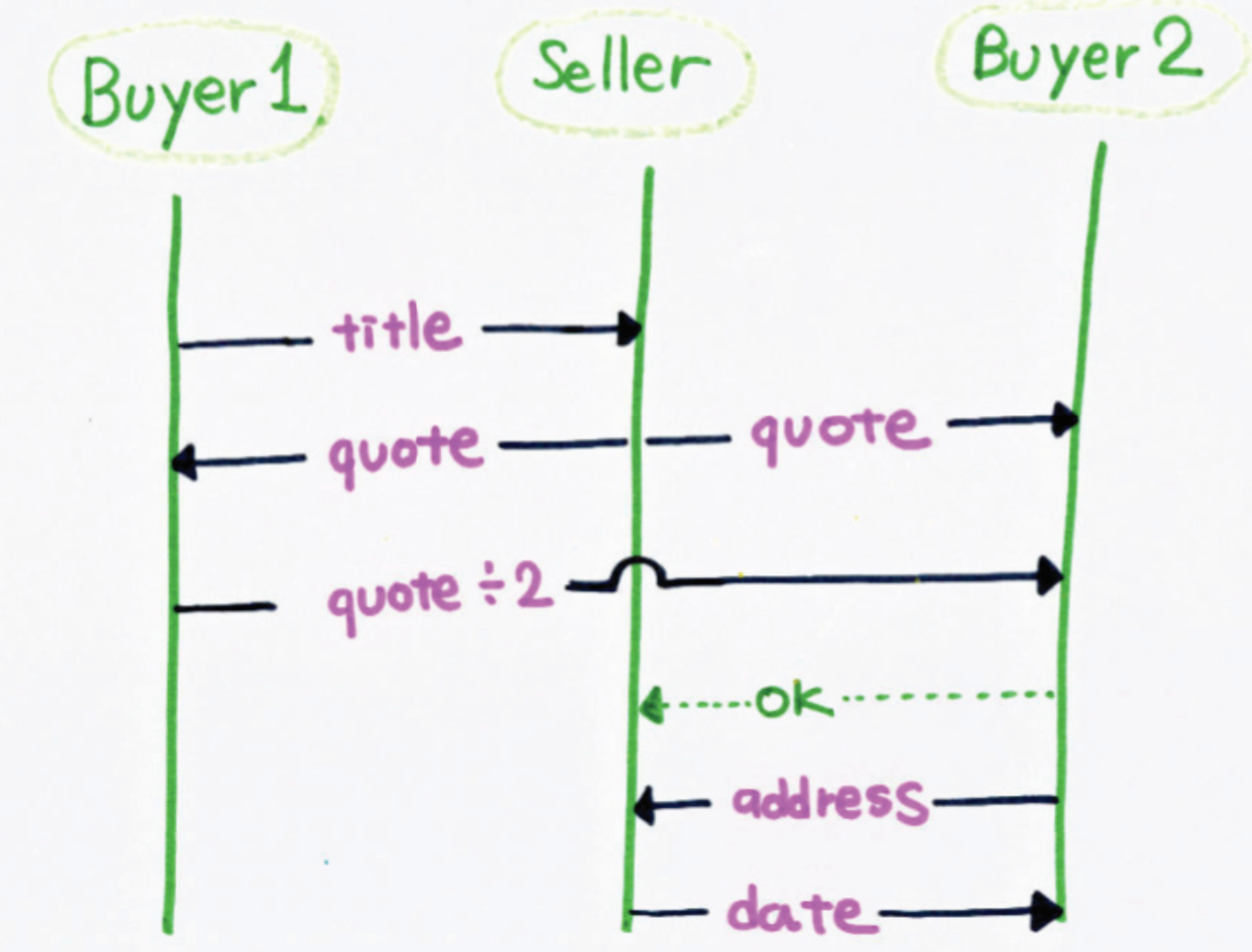
Q has type $\overline{\alpha}$

then $P \mid Q$ is typable

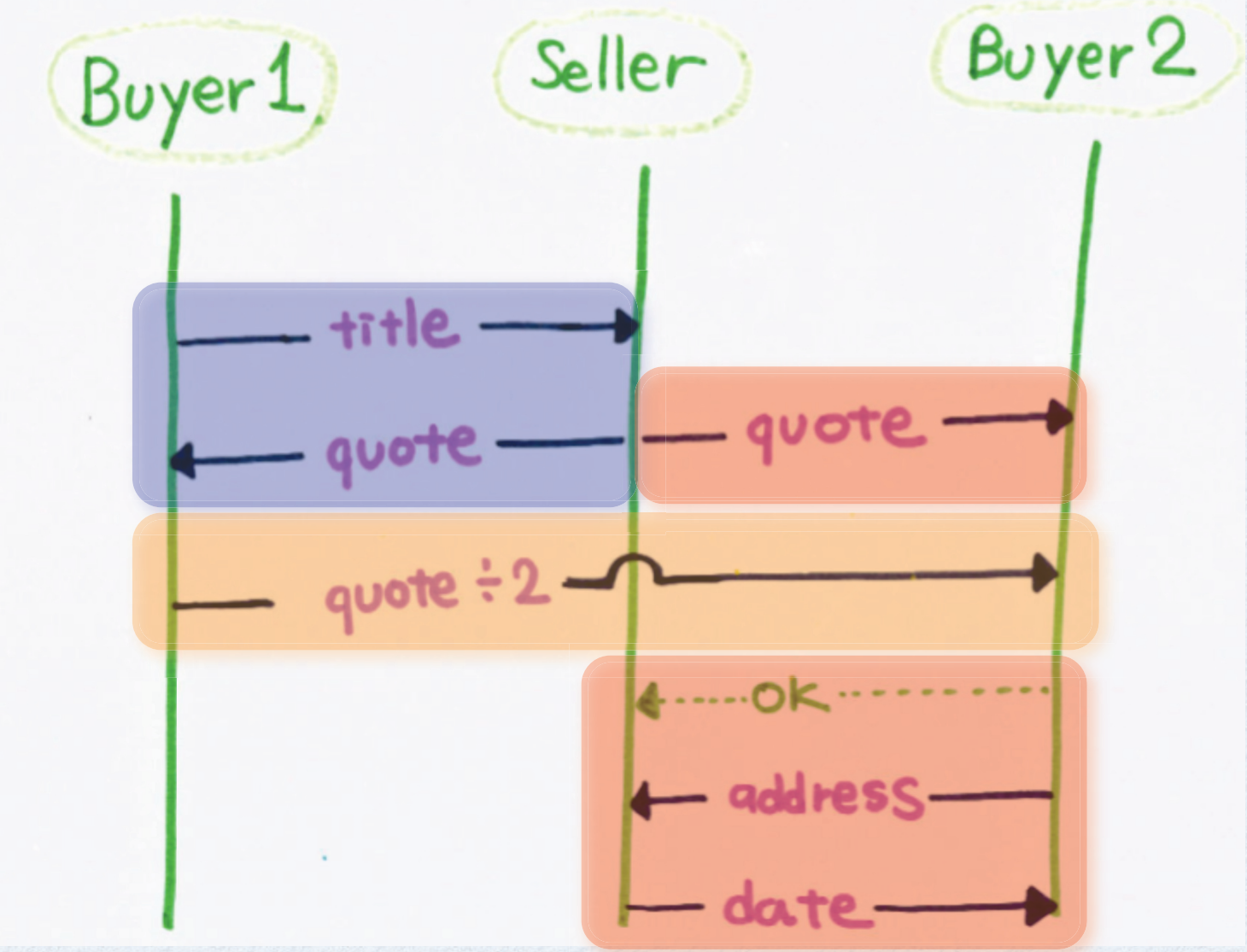
$\uparrow String.\ \downarrow Int.\ \oplus \{\mathsf{ok} :\uparrow String.\ \downarrow Date.\ \mathsf{end};\quad \mathsf{quit} : \mathsf{end}\}$

$\downarrow String.\ \uparrow Int.\ \& \{\mathsf{ok} :\downarrow String.\ \uparrow Date.\ \mathsf{end};\quad \mathsf{quit} : \mathsf{end}\}$

A MULTIPARTY SESSION



A MULTIPARTY SESSION

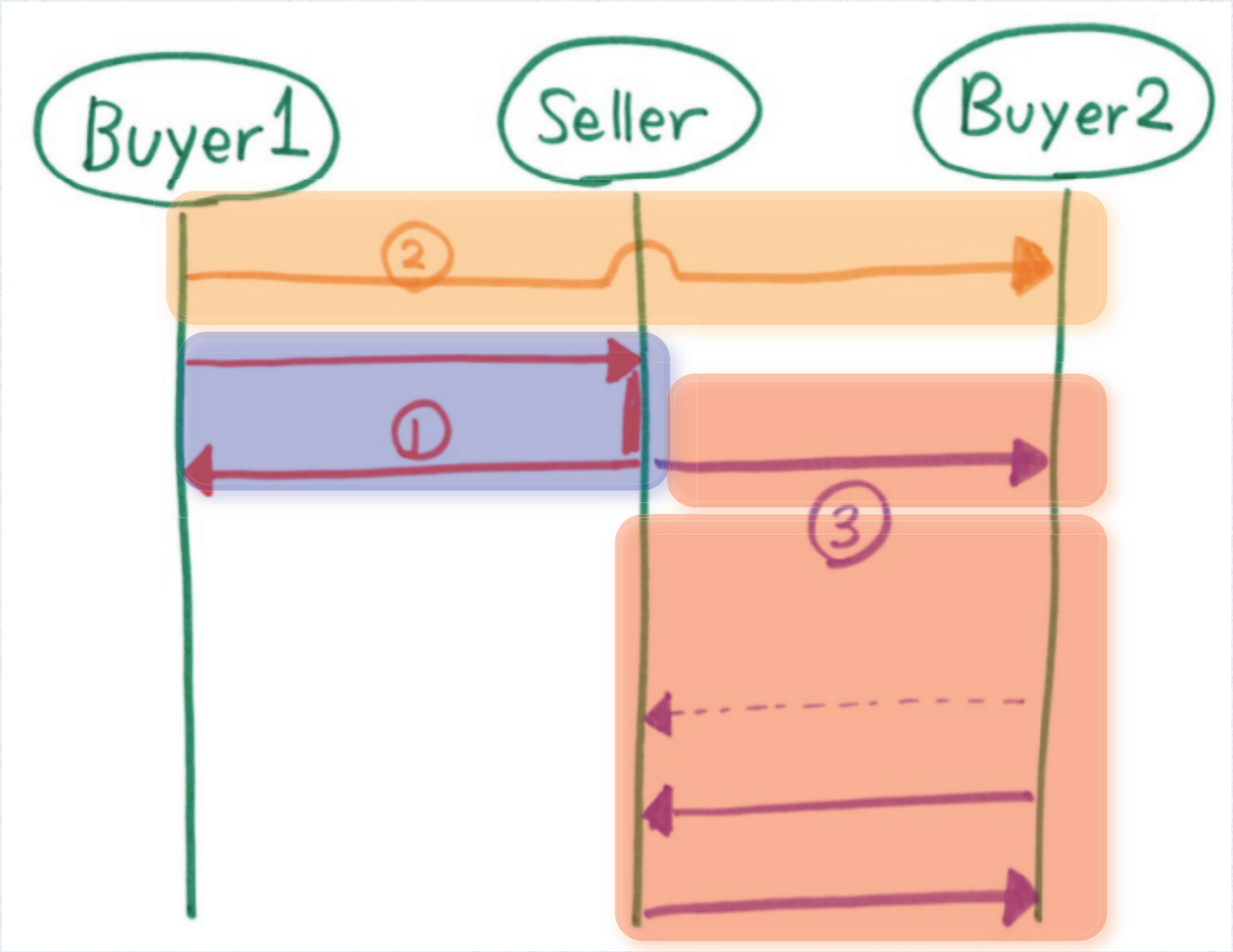


1

2

3

A MULTIPARTY SESSION

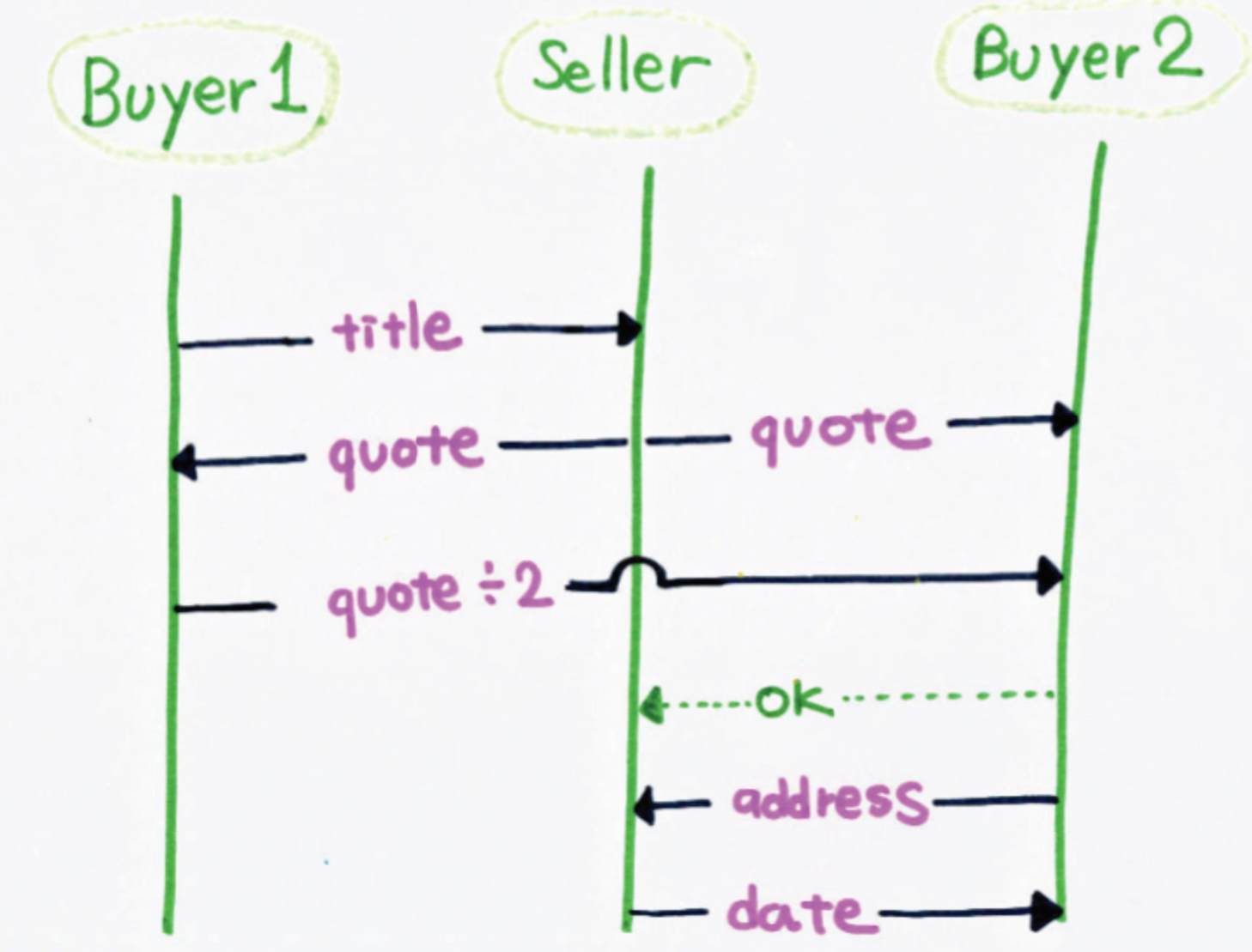


1

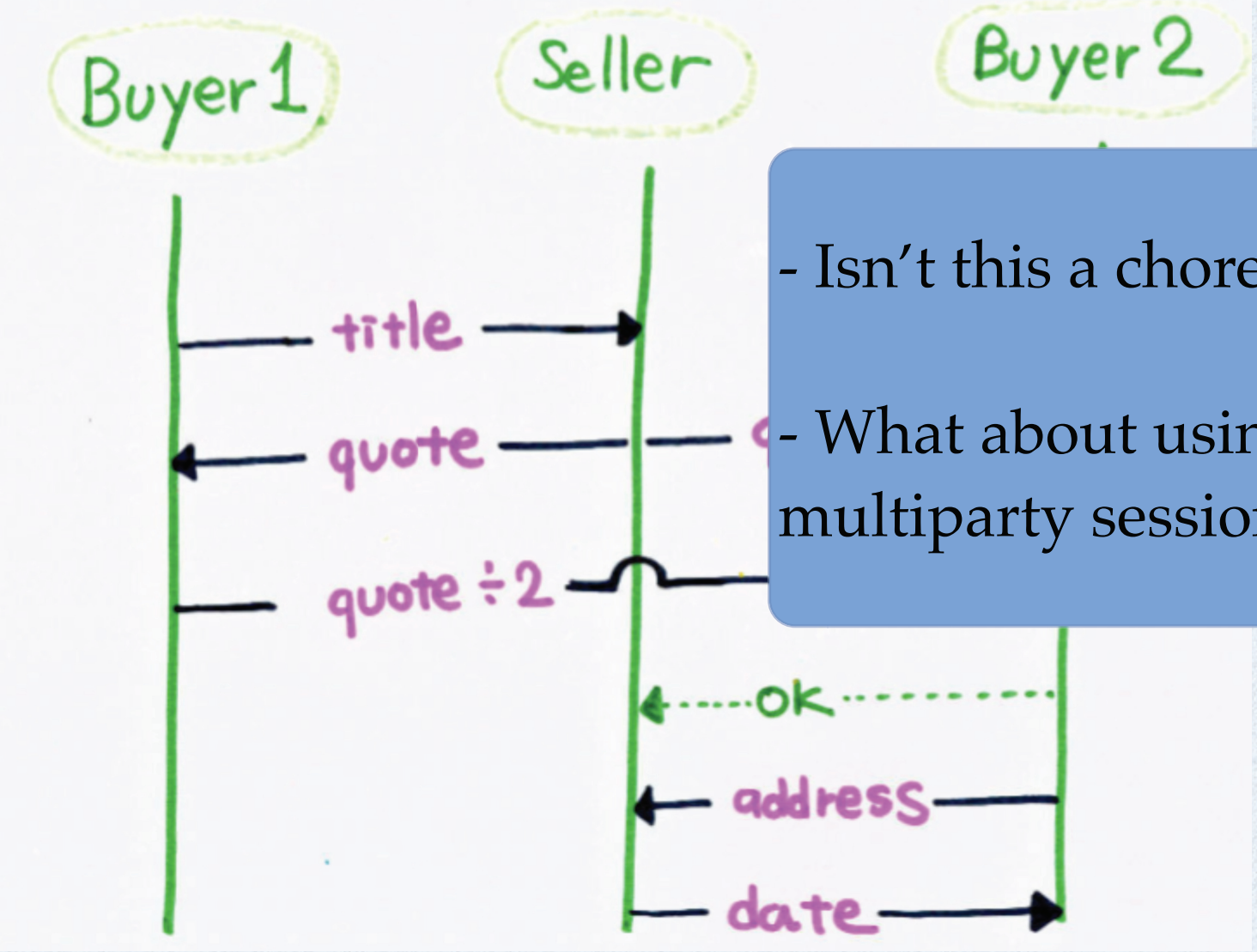
2

3

A MULTIPARTY SESSION



A MULTIPARTY SESSION



- Isn't this a choreography?

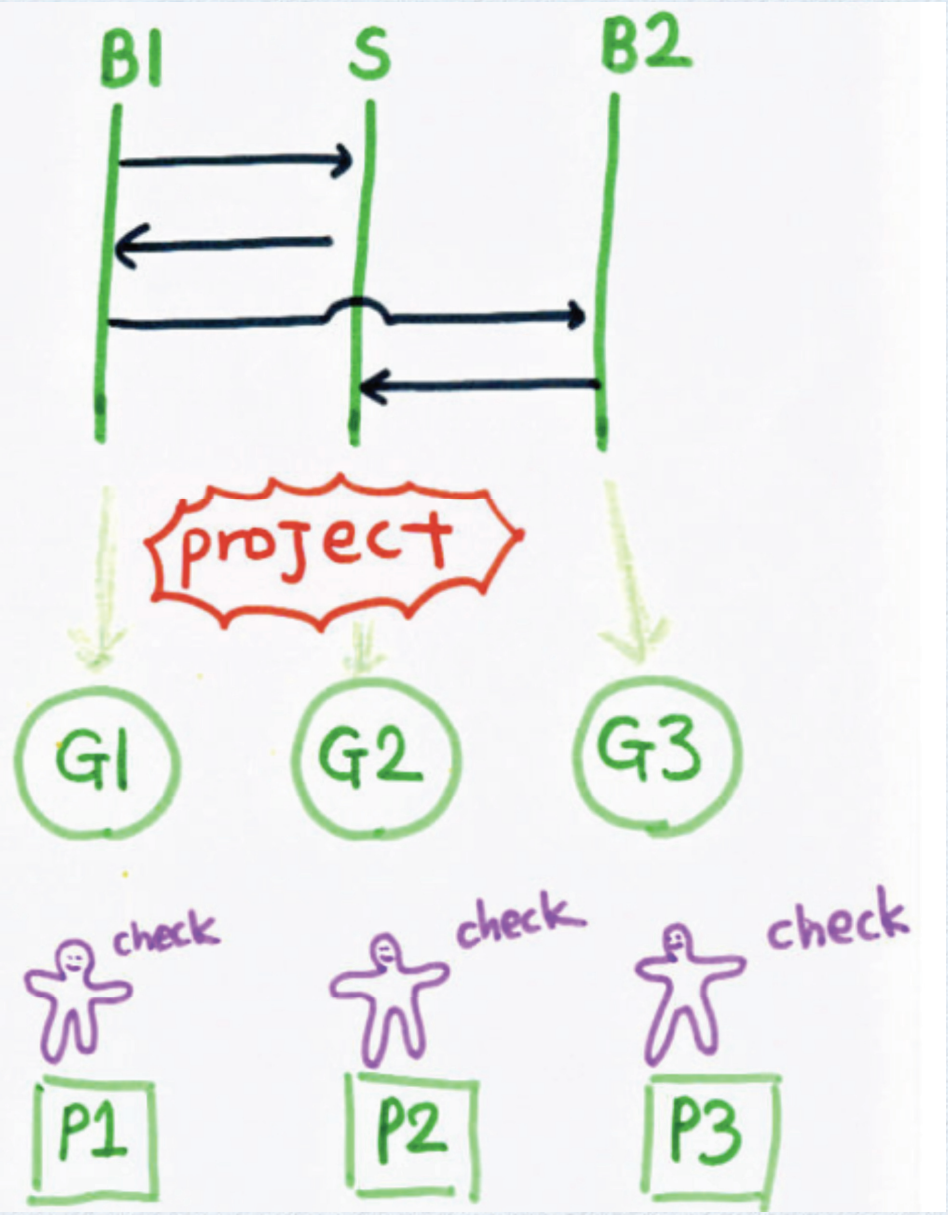
- What about using it as a multiparty session type?

A DESIGN APPROACH

1. Write a Global Type

2. Check Global Type is Good

3. Project and Local Type-Check



π-CALCULUS WITH SESSIONS

P :: =	$*a(\kappa).\ P$	(service)
	$a(\kappa).\ P$	(request)
	$\kappa?(x).\ P$	(input)
	$\kappa!\langle e\rangle.\ P$	(output)
	$\kappa \triangleright \{ l_i : P_i \}$	(branch)
	$\kappa \triangleleft l.\ P$	(select)
	$(\upsilon\ a)\ P$ / $(\upsilon\ \kappa)\ P$	(res)
	$P \mid Q$	(par)
	if e **then** P **else** P	(cond)
	0	(inact)
	μX. P	(rec)
	X	(recVar)

π-CALCULUS WITH SESSIONS

P :: =	~~$*a(\kappa).\ P$~~	~~(service)~~
	~~$a(\kappa).\ P$~~	~~(request)~~
	$\kappa?(x).\ P$	(input)
	$\kappa!\langle e \rangle.\ P$	(output)
	$\kappa \triangleright \{ l_i : P_i \}$	(branch)
	$\kappa \triangleleft l.\ P$	(select)
	$(\nu\ a)\ P$ / $(\nu\ \kappa)\ P$	(res)
	$P \mid Q$	(par)
	if e **then** P **else** P	(cond)
	0	(inact)
	$\mu X.\ P$	(rec)
	X	(recVar)

π-CALCULUS WITH SESSIONS

P :: = ~~$*a(k).\ P$~~ (service)

~~$a(k).\ P$~~ (request)

$a_{[p]}(\tilde{s}).\ P$ (accept)

$\overline{a}_{[2..n]}(\tilde{s}).\ P$ (request)

$s?((\tilde{s})).\ P$ (session reception)

$s!\langle\langle \tilde{s} \rangle\rangle.\ P$ (delegation)

X (recVar)

SEMANTICS

$$\overline{a}[2..n]\,(\tilde{s}).\ P_1 \mid a[2]\,(\tilde{s}).\ P_2 \mid \cdots \mid a[n]\,(\tilde{s}).\ P_n$$

$$\rightarrow\ (\nu\tilde{s})\,(P_1 \mid P_2 \mid \ldots \mid P_n \mid s_1 : \epsilon \mid \cdots \mid s_m : \epsilon)$$

SEMANTICS

$$\overline{a}[2..n]\,(\tilde{s}).\ P_1 \mid a[2]\,(\tilde{s}).\ P_2 \mid \cdots \mid a[n]\,(\tilde{s}).\ P_n$$

$$\rightarrow\ (\nu\tilde{s})\,(P_1 \mid P_2 \mid \ldots \mid P_n \mid s_1 : \epsilon \mid \cdots \mid s_m : \epsilon)$$

In/Out/Branch/Select/etc. are similar to the exception case

$$s!\langle \tilde{e} \rangle.\ P \mid s : L \rightarrow P \mid s : (\tilde{v} :: L)$$
$$s!\langle\langle \tilde{t} \rangle\rangle.\ P \mid s : L \rightarrow P \mid s : (\tilde{t} :: L)$$

Assumption. One session channel per each (ordered) pair of roles.

REDUCTION EXAMPLE

Buyer1 $\stackrel{\text{def}}{=} \overline{a}[2,3](b_1, b_2, b_2', s_1, s_2).\ s_1!\langle\text{“War and Peace”}\rangle.$

$b_1?(quote).\ \ b_2'!\langle quote \text{ div } 2\rangle.\ P_1$

Buyer2 $\stackrel{\text{def}}{=} a[2](b_1, b_2, b_2', s_1, s_2).\ \ b_2?(quote).\ \ b_2'?(contrib).$

$\textbf{if}\ (quote - contrib \leq 99)$

$\textbf{then}\quad s_2 \lhd \text{ok}.\ s_2!\langle address\rangle; b_2?(x).\ P_2$

$\textbf{else}\quad s_2 \lhd \text{quit}.\ \mathbf{0}$

Seller $\stackrel{\text{def}}{=} a[3](b_1, b_2, b_2', s_1, s_2).\ \ s_1?(title).\ \ \ b_1, b_2!\langle quote\rangle.$

$s_2 \rhd \{\text{ok}:\ s_2?(x).\ b_2!\langle date\rangle; Q,\quad \text{quit}:\ \mathbf{0}\}$

REDUCTION EXAMPLE

Buyer1 $\stackrel{\text{def}}{=}$ $s_1!\langle\text{“War and Peace”}\rangle.$

$b_1?(quote).\ \ b_2'!\langle quote \text{ div } 2\rangle.\ P_1$

Buyer2 $\stackrel{\text{def}}{=}$ $b_2?(quote).\ \ b_2'?(contrib).$

$\textbf{if}\ (quote - contrib \leq 99)$

$\textbf{then}\quad s_2 \lhd \text{ok}.\ s_2!\langle address\rangle; b_2?(x).\ P_2$

$\textbf{else}\quad s_2 \lhd \text{quit}.\ \mathbf{0}$

Seller $\stackrel{\text{def}}{=}$ $s_1?(title).\ \ b_1, b_2!\langle quote\rangle.$

$s_2 \rhd \{\text{ok}:\ s_2?(x).\ b_2!\langle date\rangle; Q,\quad \text{quit}:\ \mathbf{0}\}$

$b_1 : \epsilon$

$b_2 : \epsilon$

$b_2' : \epsilon$

$s_1 : \epsilon$

$s_2 : \epsilon$

REDUCTION EXAMPLE

Buyer1 $\stackrel{\text{def}}{=}$

$b_1?(quote).\ b_2'!\langle quote \text{ div } 2\rangle.\ P_1$

Buyer2 $\stackrel{\text{def}}{=}$

$b_2?(quote).\ b_2'?(contrib).$

$\textbf{if}\ (quote - contrib \leq 99)$

$\textbf{then}\quad s_2 \lhd \text{ok}.\ s_2!\langle address\rangle; b_2?(x).\ P_2$

$\textbf{else}\quad s_2 \lhd \text{quit}.\ \mathbf{0}$

Seller $\stackrel{\text{def}}{=}$

$s_1?(title).\ b_1, b_2!\langle quote\rangle.$

$s_2 \rhd \{\text{ok}:\ s_2?(x).\ b_2!\langle date\rangle; Q,\quad \text{quit}:\ \mathbf{0}\}$

$b_1 : \epsilon$

$b_2 : \epsilon$

$b_2' : \epsilon$

$s_1 : w\&p$

$s_2 : \epsilon$

REDUCTION EXAMPLE

Buyer1 $\stackrel{\text{def}}{=}$

$b_1?(quote).\ b_2'!\langle quote \text{ div } 2\rangle.\ P_1$

Buyer2 $\stackrel{\text{def}}{=}$

$b_2?(quote).\ b_2'?(contrib).$

if $(quote - contrib \leq 99)$

then $s_2 \lhd \text{ok}.\ s_2!\langle address\rangle; b_2?(x).\ P_2$

else $s_2 \lhd \text{quit}.\ \mathbf{0}$

Seller $\stackrel{\text{def}}{=}$

$b_1, b_2!\langle quote\rangle.$

$s_2 \rhd \{\text{ok}:\ s_2?(x).\ b_2!\langle date\rangle; Q,\quad \text{quit}:\ \mathbf{0}\}$

$b_1 : \epsilon$

$b_2 : \epsilon$

$b_2' : \epsilon$

$s_1 : \epsilon$

$s_2 : \epsilon$

REDUCTION EXAMPLE

Buyer1 $\stackrel{\text{def}}{=}$

$$b_1?(quote).\ b_2'!\langle quote \text{ div } 2\rangle.\ P_1$$

Buyer2 $\stackrel{\text{def}}{=}$

$$b_2?(quote).\ b_2'?(contrib).$$

$$\textbf{if } (quote - contrib \leq 99)$$

$$\textbf{then} \quad s_2 \lhd \text{ok}.\ s_2!\langle address\rangle; b_2?(x).\ P_2$$

$$\textbf{else} \quad s_2 \lhd \text{quit}.\ \mathbf{0}$$

Seller $\stackrel{\text{def}}{=}$

$$s_2 \rhd \{\text{ok}:\ s_2?(x).\ b_2!\langle date\rangle; Q,\quad \text{quit}:\ \mathbf{0}\}$$

$b_1 : quote$

$b_2 : quote$

$b_2' : \epsilon$

$s_1 : \epsilon$

$s_2 : \epsilon$

REDUCTION EXAMPLE

Buyer1 $\stackrel{\text{def}}{=}$

$b_1?(quote).\ b_2'!\langle quote \text{ div } 2\rangle.\ P_1$

Buyer2 $\stackrel{\text{def}}{=}$

$b_2'?(contrib).$

$\textbf{if}\ (quote - contrib \leq 99)$

$\textbf{then}\quad s_2 \lhd \text{ok}.\ s_2!\langle address\rangle; b_2?(x).\ P_2$

$\textbf{else}\quad s_2 \lhd \text{quit}.\ \mathbf{0}$

Seller $\stackrel{\text{def}}{=}$

$s_2 \rhd \{\text{ok}:\ s_2?(x).\ b_2!\langle date\rangle; Q,\quad \text{quit}:\ \mathbf{0}\}$

$b_1 : \epsilon$

$b_2 : quote$

$b_2' : \epsilon$

$s_1 : \epsilon$

$s_2 : \epsilon$

REDUCTION EXAMPLE

Buyer1 $\stackrel{\text{def}}{=}$

$b_2'!\langle quote \text{ div } 2\rangle.\ P_1$

Buyer2 $\stackrel{\text{def}}{=}$

$b_2'?(contrib).$

$\textbf{if}\ (quote - contrib \leq 99)$

$\textbf{then}\quad s_2 \lhd \text{ok}.\ s_2!\langle address\rangle; b_2?(x).\ P_2$

$\textbf{else}\quad s_2 \lhd \text{quit}.\ \mathbf{0}$

Seller $\stackrel{\text{def}}{=}$

$s_2 \rhd \{\text{ok}:\ s_2?(x).\ b_2!\langle date\rangle; Q,\quad \text{quit}:\ \mathbf{0}\}$

$b_1 : \epsilon$

$b_2 : \epsilon$

$b_2' : \epsilon$

$s_1 : \epsilon$

$s_2 : \epsilon$

REDUCTION EXAMPLE

Buyer1 $\stackrel{\text{def}}{=}$

$$P_1$$

Buyer2 $\stackrel{\text{def}}{=}$ $b_2'?(contrib).$

$$\textbf{if } (quote - contrib \leq 99)$$

$$\textbf{then} \quad s_2 \lhd \mathsf{ok}.\ s_2!\langle address\rangle; b_2?(x).\ P_2$$

$$\textbf{else} \quad s_2 \lhd \mathsf{quit}.\ \mathbf{0}$$

Seller $\stackrel{\text{def}}{=}$

$$s_2 \rhd \{\mathsf{ok}:\ s_2?(x).\ b_2!\langle date\rangle; Q,\quad \mathsf{quit}:\ \mathbf{0}\}$$

$b_1 : \epsilon$

$b_2 : \epsilon$

$b_2' : quote/2$

$s_1 : \epsilon$

$s_2 : \epsilon$

REDUCTION EXAMPLE

Buyer1 $\stackrel{\text{def}}{=}$

P_1

Buyer2 $\stackrel{\text{def}}{=}$

if $(quote - contrib \leq 99)$

then $s_2 \lhd \mathsf{ok}.\ s_2!\langle address\rangle; b_2?(x).\ P_2$

else $s_2 \lhd \mathsf{quit}.\ \mathbf{0}$

Seller $\stackrel{\text{def}}{=}$

$s_2 \rhd \{\mathsf{ok}:\ s_2?(x).\ b_2!\langle date\rangle; Q,\quad \mathsf{quit}:\ \mathbf{0}\}$

$b_1 : \epsilon$

$b_2 : \epsilon$

$b_2' : \epsilon$

$s_1 : \epsilon$

$s_2 : \epsilon$

REDUCTION EXAMPLE

Buyer1 $\stackrel{\text{def}}{=}$

$$P_1$$

Buyer2 $\stackrel{\text{def}}{=}$

$$\textbf{if } (quote - contrib \leq 99)$$

$$\textbf{then } s_2 \lhd \text{ok}.\, s_2!\langle address\rangle;\, b_2?(x).\, P_2$$

$$\textbf{else } s_2 \lhd \text{quit}.\, \mathbf{0}$$

Seller $\stackrel{\text{def}}{=}$

$$s_2 \rhd \{\text{ok}: s_2?(x).\, b_2!\langle date\rangle;\, Q,\quad \text{quit}: \mathbf{0}\}$$

$b_1 : \epsilon$

$b_2 : \epsilon$

$b_2' : \epsilon$

$s_1 : \epsilon$

$s_2 : \epsilon$

GLOBAL TYPES

$$
\begin{array}{rcll}
G & ::= & \mathtt{p} \to \mathtt{p}' : k\,\langle\theta\rangle.G' & \text{(values)} \\
 & | & \mathtt{p} \to \mathtt{p}' : k\,\{l_j : G_j\}_{j\in J} & \text{(branching)} \\
 & | & G, G' & \text{(parallel)} \\
 & | & \mu \mathbf{t}.G & \text{(recursive)} \\
 & | & \mathbf{t} & \text{(variable)} \\
 & | & \mathsf{end} & \text{(end)}
\end{array}
$$

$$
\theta \;::=\; \tilde{S} \;\mid\; \alpha@\mathtt{p} \qquad S \;::=\; \mathsf{bool} \mid \mathsf{nat} \mid \ldots \mid \langle G \rangle
$$

Global Types: Examples

Two-Buyers Protocol

$$\mathtt{B1} \to \mathtt{S} : s_1\langle \mathsf{string}\rangle.\quad \mathtt{S} \to \mathtt{B1} : b_1\langle \mathsf{int}\rangle.\quad \mathtt{S} \to \mathtt{B2} : b_2\langle \mathsf{int}\rangle.\quad \mathtt{B1} \to \mathtt{B2} : b_2'\langle \mathsf{int}\rangle.$$

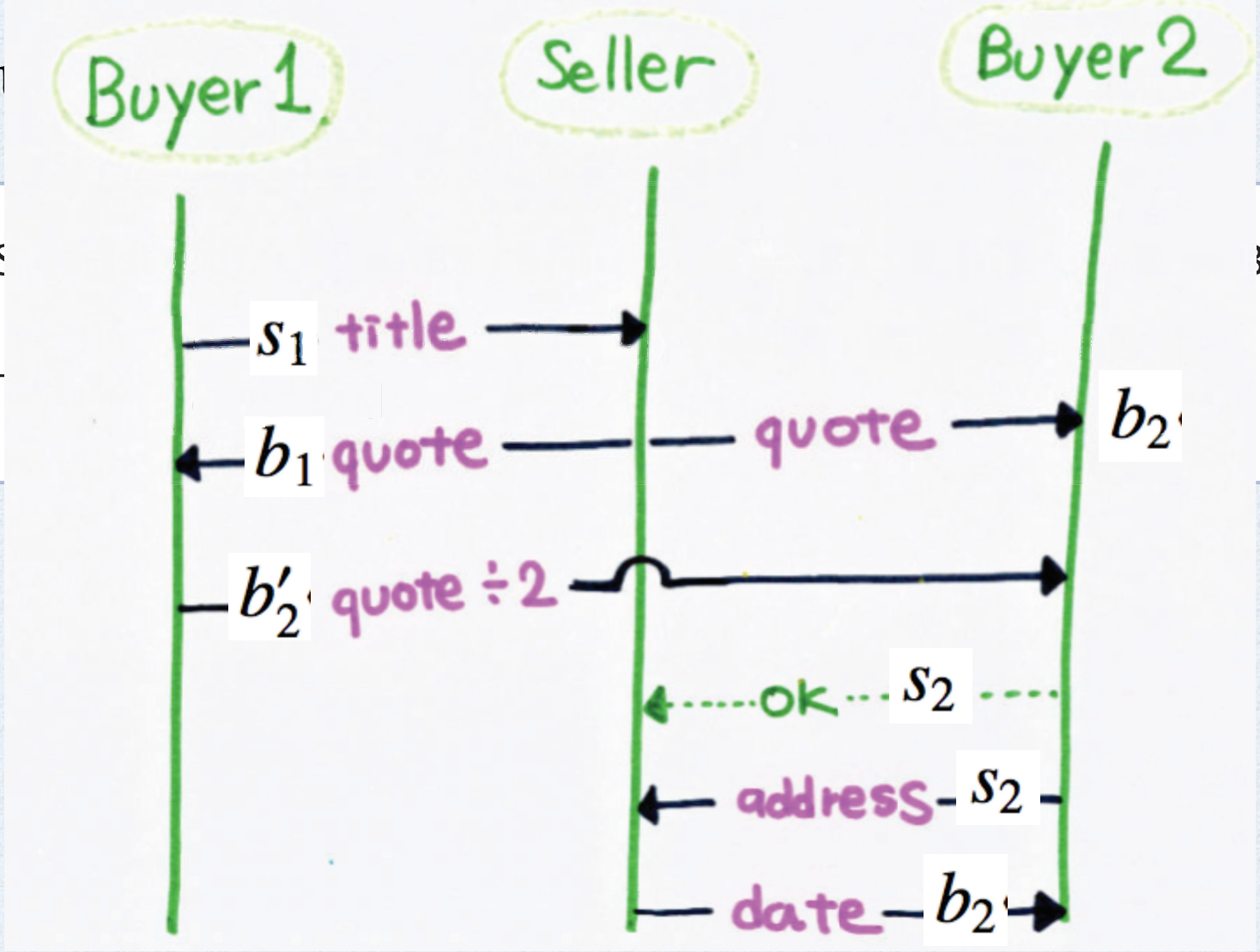
$$\mathtt{B2} \to \mathtt{S} : s_2 \left\{ \begin{array}{l} \mathsf{ok} : \mathtt{B2} \to \mathtt{S} : s_2\langle \mathsf{string}\rangle.\mathtt{S} \to \mathtt{B2} : b_2\langle \mathsf{date}\rangle.\mathsf{end}, \\ \mathsf{quit} : \mathsf{end} \} \end{array} \right\}$$

GLOBAL TYPES: EXAMPLES

Two-Bu

B1 → S

B2 –



$2 : b'_2\langle\text{int}\rangle$.

LOCAL TYPES

$$\alpha ::= k!\langle\theta\rangle.\,\alpha \mid k?\langle\theta\rangle.\,\alpha \mid k \oplus \{l_i : \alpha_i\}_{i\in I}$$

$$\mid k \,\&\, \{l_i : \alpha_i\}_{i\in I} \mid \mu \mathbf{t}.\,\alpha \mid \mathbf{t} \mid \mathsf{end}$$

- (almost) **standard**
- actions labelled with channel to be used.

EXAMPLE

Buyer1 $\stackrel{\text{def}}{=} \overline{a}_{[2,3]}(b_1, b_2, b_2', s_1, s_2).\ \ s_1!\langle\text{“War and Peace”}\rangle.$

$b_1?(quote).\ \ b_2'!\langle quote \text{ div } 2\rangle.\ P_1$

In the two buyer protocol, write Buyer1 as

$\overline{a}_{[2,3]}(b_1, b_2, b_2', s_1, s_2).\ Q_1$ and Buyer2 as $a_{[2]}(b_1, b_2, b_2', s_1, s_2).\ Q_2$. Then

$\Gamma \vdash Q_1 \triangleright \tilde{s} : s_1!\langle\text{string}\rangle.\ b_1?\langle\text{int}\rangle.\ b_2'!\langle\text{int}\rangle@\text{B1}$

$\Gamma \vdash Q_2 \triangleright \tilde{s} : b_2?\langle\text{int}\rangle.\ b_2'?\langle\text{int}\rangle.\ s_2 \oplus \{\text{ok} : s_2!\langle\text{string}\rangle.\ b_2?\langle\text{date}\rangle.\ \text{end},\ \text{quit} : \text{end}\}@\text{B2}$

EXAMPLE

$$\text{Buyer2} \stackrel{\text{def}}{=} a_{[2]}(b_1, b_2, b_2', s_1, s_2).\ \ b_2?(quote).\ \ b_2'?(contrib).$$
$$\textbf{if}\ (quote - contrib \leq 99)$$
$$\textbf{then}\quad s_2 \triangleleft \text{ok}.\ s_2!\langle address\rangle; b_2?(x).\ P_2$$
$$\textbf{else}\quad s_2 \triangleleft \text{quit}.\ \mathbf{0}$$

In the two buyer protocol, write Buyer1 as $\overline{a}_{[2,3]}(b_1, b_2, b_2', s_1, s_2).\ Q_1$ and Buyer2 as $a_{[2]}(b_1, b_2, b_2', s_1, s_2).\ Q_2$. Then

$$\Gamma \vdash Q_1 \triangleright \tilde{s} : s_1!\langle \text{string}\rangle.\ b_1?\langle \text{int}\rangle.\ b_2'!\langle \text{int}\rangle@\text{B1}$$
$$\Gamma \vdash Q_2 \triangleright \tilde{s} : b_2?\langle \text{int}\rangle.\ b_2'?\langle \text{int}\rangle.\ s_2 \oplus \{\text{ok} : s_2!\langle \text{string}\rangle.\ b_2?\langle \text{date}\rangle.\ \text{end},\ \text{quit} : \text{end}\}@\text{B2}$$

PROJECTION

$$(\mathtt{p}_1 \to \mathtt{p}_2 : k\,\langle\theta\rangle.G')\upharpoonright \mathtt{p} = \begin{cases} k!\,\langle\theta\rangle.(G'\upharpoonright \mathtt{p}) & \text{if } \mathtt{p} = \mathtt{p}_1 \neq \mathtt{p}_2 \\ k?\,\langle\theta\rangle.(G'\upharpoonright \mathtt{p}) & \text{if } \mathtt{p} = \mathtt{p}_2 \neq \mathtt{p}_1 \\ (G'\upharpoonright \mathtt{p}) & \text{if } \mathtt{p} \neq \mathtt{p}_2 \wedge \mathtt{p} \neq \mathtt{p}_1 \end{cases}$$

PROJECTION

$$(\mathtt{p}_1 \to \mathtt{p}_2 : k\,\langle\theta\rangle.G') \upharpoonright \mathtt{p} = \begin{cases} k!\,\langle\theta\rangle.(G' \upharpoonright \mathtt{p}) & \text{if } \mathtt{p} = \mathtt{p}_1 \neq \mathtt{p}_2 \\ k?\,\langle\theta\rangle.(G' \upharpoonright \mathtt{p}) & \text{if } \mathtt{p} = \mathtt{p}_2 \neq \mathtt{p}_1 \\ (G' \upharpoonright \mathtt{p}) & \text{if } \mathtt{p} \neq \mathtt{p}_2 \wedge \mathtt{p} \neq \mathtt{p}_1 \end{cases}$$

$$(G_1, G_2) \upharpoonright \mathtt{p} = \begin{cases} G_i \upharpoonright \mathtt{p} & \text{if } \mathtt{p} \in G_i \text{ and } \mathtt{p} \notin G_j,\ i \neq j \in \{1,2\} \\ \mathsf{end} & \text{if } \mathtt{p} \notin G_1 \text{ and } \mathtt{p} \notin G_2 \end{cases}$$

PROJECTION

$$(\mathtt{p}_1 \to \mathtt{p}_2 : k\,\langle\theta\rangle.G')\upharpoonright \mathtt{p} = \begin{cases} k!\,\langle\theta\rangle.(G'\upharpoonright \mathtt{p}) & \text{if } \mathtt{p} = \mathtt{p}_1 \neq \mathtt{p}_2 \\ k?\,\langle\theta\rangle.(G'\upharpoonright \mathtt{p}) & \text{if } \mathtt{p} = \mathtt{p}_2 \neq \mathtt{p}_1 \\ (G'\upharpoonright \mathtt{p}) & \text{if } \mathtt{p} \neq \mathtt{p}_2 \wedge \mathtt{p} \neq \mathtt{p}_1 \end{cases}$$

$$(G_1, G_2)\upharpoonright \mathtt{p} = \begin{cases} G_i\upharpoonright \mathtt{p} & \text{if } \mathtt{p} \in G_i \text{ and } \mathtt{p} \notin G_j,\ i \neq j \in \{1,2\} \\ \mathsf{end} & \text{if } \mathtt{p} \notin G_1 \text{ and } \mathtt{p} \notin G_2 \end{cases}$$

$$(\mathtt{p}_1 \to \mathtt{p}_2 : k\,\{l_j : G_j\}_{j\in J})\upharpoonright \mathtt{p} = \begin{cases} k \oplus \{l_j : (G_j\upharpoonright \mathtt{p})\}_{j\in J} & \text{if } \mathtt{p} = \mathtt{p}_1 \neq \mathtt{p}_2 \\ k\&\{l_j : (G_j\upharpoonright \mathtt{p})\}_{j\in J} & \text{if } \mathtt{p} = \mathtt{p}_2 \neq \mathtt{p}_1 \\ (\bigsqcup_{i\in I} G_i\upharpoonright \mathtt{p}) & \text{if } \mathtt{p} \neq \mathtt{p}_2 \wedge \mathtt{p} \neq \mathtt{p}_1 \\ & \text{and } \forall i, j \in I.G_i\upharpoonright \mathtt{p} \bowtie G_j\upharpoonright \mathtt{p} \end{cases}$$

PROJECTION (2)

Two-Buyers Protocol

$$\mathtt{B1} \to \mathtt{S} : s_1\langle \mathsf{string}\rangle.\quad \mathtt{S} \to \mathtt{B1} : b_1\langle \mathsf{int}\rangle.\quad \mathtt{S} \to \mathtt{B2} : b_2\langle \mathsf{int}\rangle.\quad \mathtt{B1} \to \mathtt{B2} : b_2'\langle \mathsf{int}\rangle.$$

$$\mathtt{B2} \to \mathtt{S} : s_2 \left\{ \begin{array}{l} \mathsf{ok} : \mathtt{B2} \to \mathtt{S} : s_2\,\langle \mathsf{string}\rangle.\mathtt{S} \to \mathtt{B2} : b_2\,\langle \mathsf{date}\rangle.\mathsf{end}, \\ \mathsf{quit} : \mathsf{end}\} \end{array} \right\}$$

For Buyer2:

$$b_2?\langle \mathsf{int}\rangle.\ b_2'?\langle \mathsf{int}\rangle.\ s_2 \oplus \{\mathsf{ok} : s_2!\langle \mathsf{string}\rangle.\ b_2?\langle \mathsf{date}\rangle.\ \mathsf{end},\ \mathsf{quit} : \mathsf{end}\}@\mathtt{B2}$$

MERGEABILITY&PROJECTION

Mergeability.

Smallest equiv. on local types closed under contexts and s.t.:

$$\frac{\forall i \in (I \cap J).\ \alpha_i \bowtie \beta_i \qquad \forall i \in I \setminus J.\ \forall j \in J \setminus I.\ l_i \neq l_j}{k \,\&\, \{l_i : \alpha_i\}_{i \in I} \bowtie k \,\&\, \{l_j : \beta_j\}_{j \in J}}$$

MERGEABILITY&PROJECTION

Mergeability.
Smallest equiv. on local types closed under contexts and s.t.:

$$\frac{\forall i \in (I \cap J).\ \alpha_i \bowtie \beta_i \qquad \forall i \in I \setminus J.\ \forall j \in J \setminus I.\ l_i \neq l_j}{k \,\&\, \{l_i : \alpha_i\}_{i \in I} \bowtie k \,\&\, \{l_j : \beta_j\}_{j \in J}}$$

Merge.
Partial (on mergeability) operator. It is the identity except from:

$$k \,\&\, \{l_i : \alpha_i\}_{i \in I} \sqcup k \,\&\, \{l_j : \beta_j\}_{j \in J} =$$

$$k \& (\{l_i : \alpha_i \sqcup \beta_i\}_{i \in I \cap J} \cup \{l_i : \alpha_i\}_{i \in I} \cup \{l_j : \beta_j\}_{j \in J})$$

MERGING EXAMPLE

$p_1 \to p_2 : k \{$
ok : $p_2 \to p_3 : k'$ {paymore : . . .},
quit : $p_2 \to p_3 : k'$ {refund : . . .}
$\}$

p_1 $k \oplus \{\text{ok}: \ldots, \text{quit}: \ldots,\}$

p_2 $k\&\{ \quad \text{ok}: k' \oplus \{\text{paymore}: \ldots\}, \quad \text{quit}: k' \oplus \{\text{reject}: \ldots\} \quad \}$

p_3 $k'\&\{ \text{paymore}: \ldots, \text{refund}: \ldots \}$

TYPING

Environments.

$$\Gamma ::= \emptyset \mid \Gamma, u : S \mid \Gamma, X : \Delta$$
$$\Delta ::= \emptyset \mid \Delta \cdot \tilde{s} : \{\alpha @ \mathrm{p}\}_{\mathrm{p} \in I}$$

Judgements.

$$\Gamma \vdash P \triangleright \Delta$$

Typing Rules: Init

$$\frac{\Gamma \vdash a : \langle G \rangle \quad \Gamma \vdash P \triangleright \Delta, \tilde{s} : (G \upharpoonright 1)@1 \quad |\tilde{s}| = \max(\mathsf{sid}(G))}{\Gamma \vdash \overline{a}_{[2..n]}(\tilde{s}).\, P \triangleright \Delta}$$

$$\frac{\Gamma \vdash a : \langle G \rangle \quad \Gamma \vdash P \triangleright \Delta, \tilde{s} : (G \upharpoonright \mathtt{p})@\mathtt{p} \quad |\tilde{s}| = \max(\mathsf{sid}(G))}{\Gamma \vdash a_{[\mathtt{p}]}(\tilde{s}).\, P \triangleright \Delta}$$

TYPING RULES: IN/OUT

$$\frac{\Gamma, x : \tilde{S} \vdash P \triangleright \Delta \cdot \tilde{s} : \alpha @ \mathtt{p}}{\Gamma \vdash s_k ?(\tilde{x}).\, P \triangleright \Delta \cdot \tilde{s} : k ?\langle \tilde{S} \rangle.\, \alpha @ \mathtt{p}}$$

$$\frac{\forall j.\, \Gamma \vdash e_j : S_j \qquad \Gamma \vdash P \triangleright \Delta \cdot \tilde{s} : \alpha @ \mathtt{p}}{\Gamma \vdash s_k !\langle \tilde{e} \rangle.\, P \triangleright \Delta \cdot \tilde{s} : k !\langle \tilde{S} \rangle.\, \alpha @ \mathtt{p}}$$

TYPING RULES: DELEGATION

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot \tilde{s} : \alpha @ \mathrm{p} \cdot \tilde{t} : \alpha' @ \mathrm{p}'}{\Gamma \vdash s_k\ ?((\tilde{t}))\mathbf{.}\ P \triangleright \Delta \cdot \tilde{s} : k?\langle \alpha' @ \mathrm{p}' \rangle\mathbf{.}\ \alpha @ \mathrm{p}}$$

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot \tilde{s} : \alpha @ \mathrm{p}}{\Gamma \vdash s_k !\langle\langle \tilde{t} \rangle\rangle\mathbf{.}\ P \triangleright \Delta \cdot \tilde{s} : k!\langle \alpha' @ \mathrm{p}' \rangle\mathbf{.}\ \alpha @ \mathrm{p} \cdot \tilde{t} : \alpha' @ \mathrm{p}'}$$

Typing rules: branching/selection

$$\frac{\Gamma \vdash P_j \triangleright \Delta \cdot \tilde{s} : \alpha_j @ \mathtt{p} \qquad \forall j \in J \qquad I \subseteq J}{\Gamma \vdash s_k \rhd \{l_j : P_j\}_{i \in J} \triangleright \Delta \cdot \tilde{s} : k \,\&\, \{l_i : \alpha_i\}_{i \in I} @ \mathtt{p}}$$

$$\frac{\Gamma \vdash P \triangleright \Delta \cdot \tilde{s} : \alpha_j @ \mathtt{p} \qquad j \in I}{\Gamma \vdash s_k \lhd l_j . P \triangleright \Delta \cdot \tilde{s} : k \oplus \{l_i : \alpha_i\}_{i \in I} @ \mathtt{p}}$$

RESULTS

Theorem (Subject Reduction).

If $\Gamma \vdash P \triangleright \emptyset$ and $P \longrightarrow^* Q$ then $\Gamma \vdash Q \triangleright \emptyset$

Theorem (Communication Safety).

MULTIPARTY: SUMMARY

- Multiparty Session Types
  - Global Types + Causal Analysis + Projection
  - Simple · Concise Syntax
  - Communication Error Freedom
  - Session Fidelity, Progress in a session

Related Work

Bonelli & Compagnoni TGC'07

Ongoing Work

Java with Session Types

W3C CDL (Choreography Description Language)

ISO UNIFI (UNIversal Financial Industry message scheme)

THE END

TERMINATION

- We may wish to terminate a try-catch block. This can be useful in many situation, e.g. when having the sequence operator ";":

$$\mathbf{try}\ \{\ \mathbf{0}\ \}\ \mathbf{catch}\ \{\ \kappa : Q\ \};\ R \longrightarrow R$$

TERMINATION

- We may wish to terminate a try-catch block. This can be useful in many situation, e.g. when having the sequence operator ";":

$$\mathbf{try}\ \{\ \mathbf{0}\ \}\ \mathbf{catch}\ \{\ \kappa : Q\ \};\ R \longrightarrow R$$

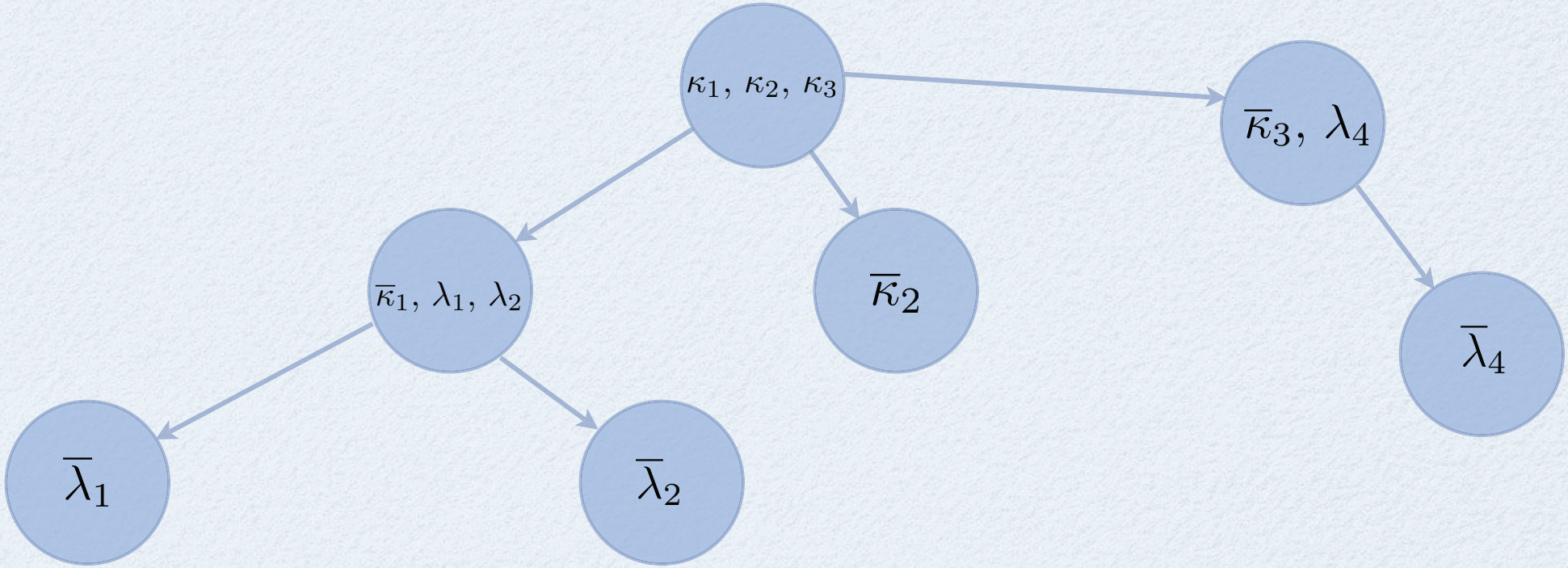
- But...there is a problem...

$$\mathbf{try}\ \{\ \mathbf{0}\ \}\ \mathbf{catch}\ \{\ \kappa : Q\ \} \mid \mathbf{try}\ \{\ \mathbf{throw}\ \}\ \mathbf{catch}\ \{\ \overline{\kappa} : Q'\ \}$$

- If the rhs disappears before the exception is thrown, how can we guarantee propagation?

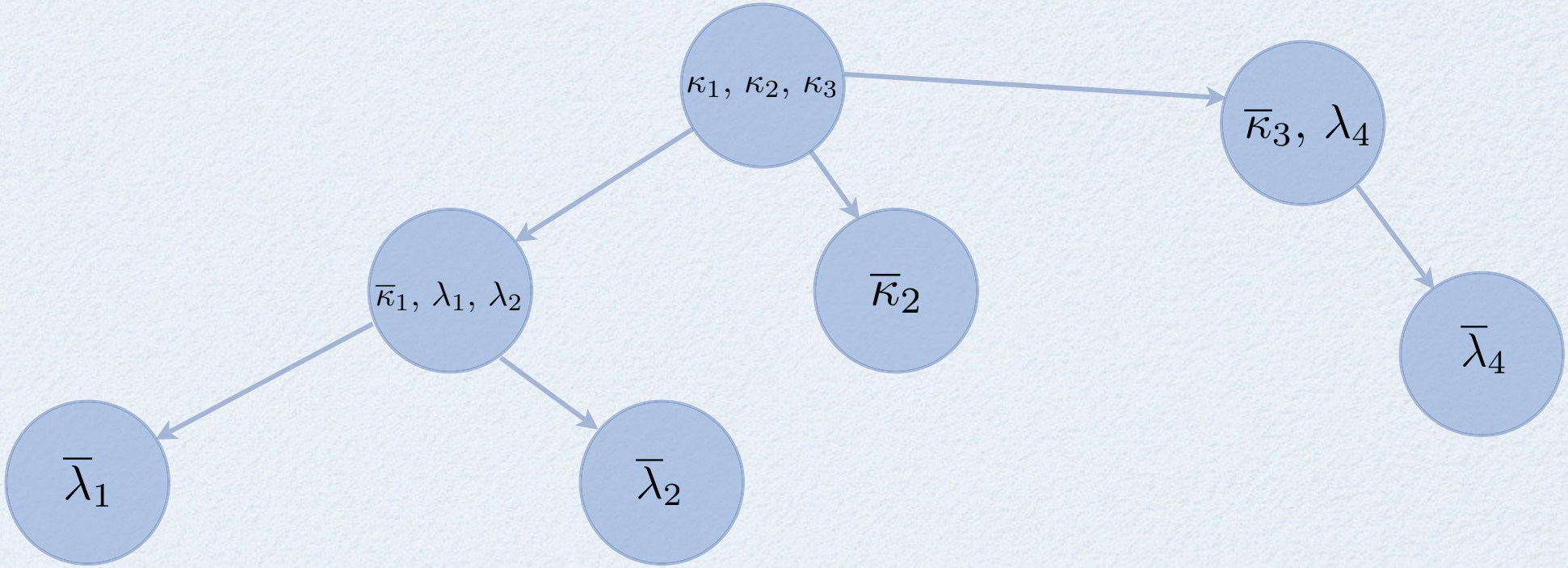
TERMINATION PROTOCOL

Consider the caller-callee relation on services (instances)



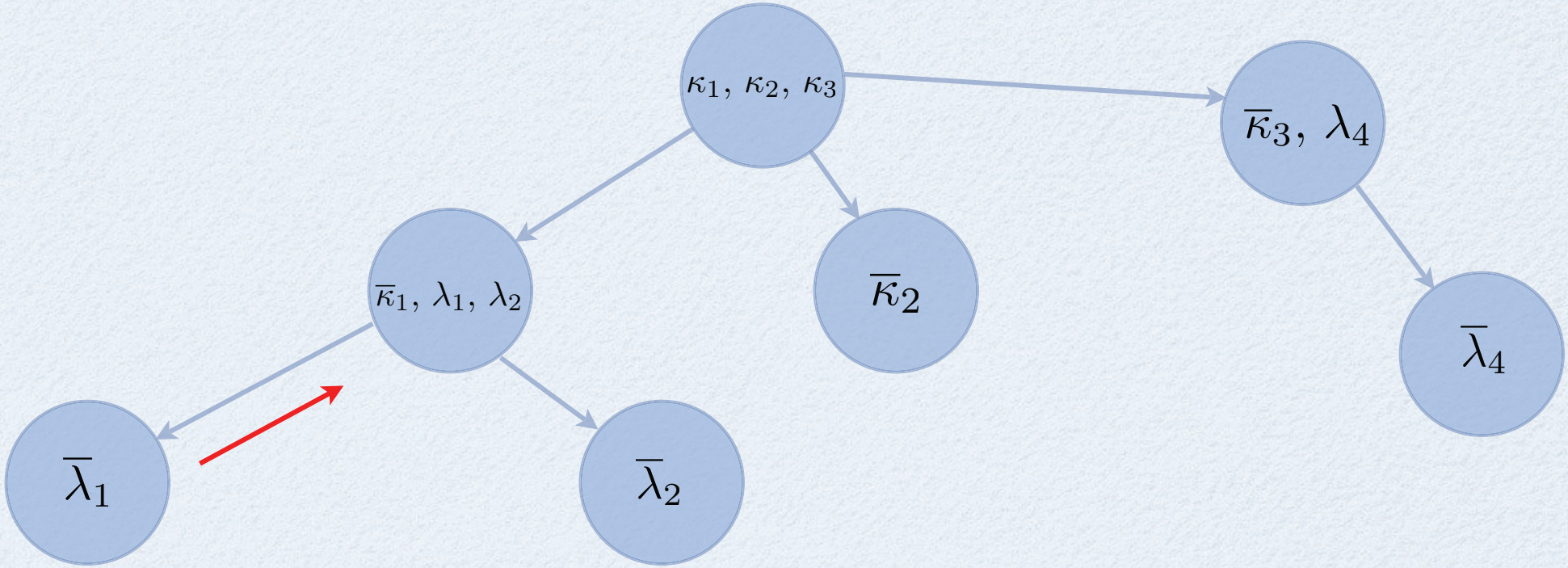
TERMINATION PROTOCOL

Consider the caller-callee relation on services (instances)



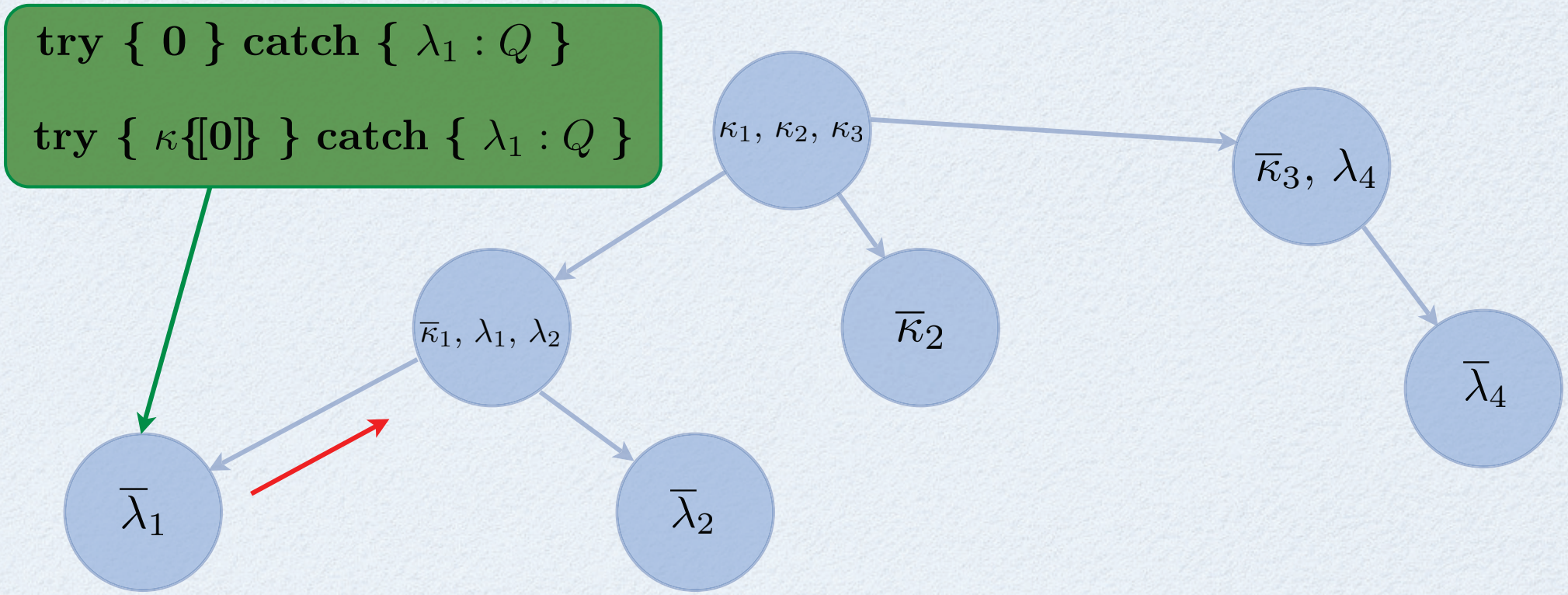
Theorem. The graph above is always a tree (for well-typed processes)

TERMINATION PROTOCOL



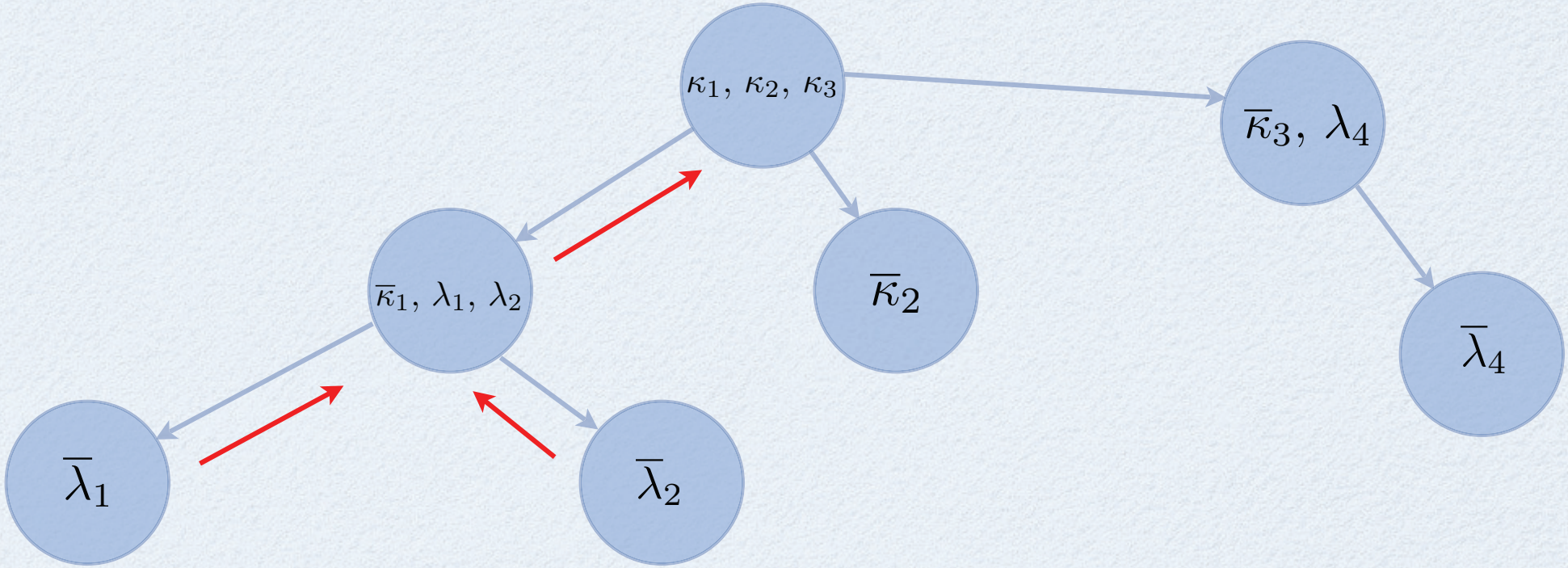
1) the leaves communicate their will to terminate to their father i.e. there are no active processes

TERMINATION PROTOCOL



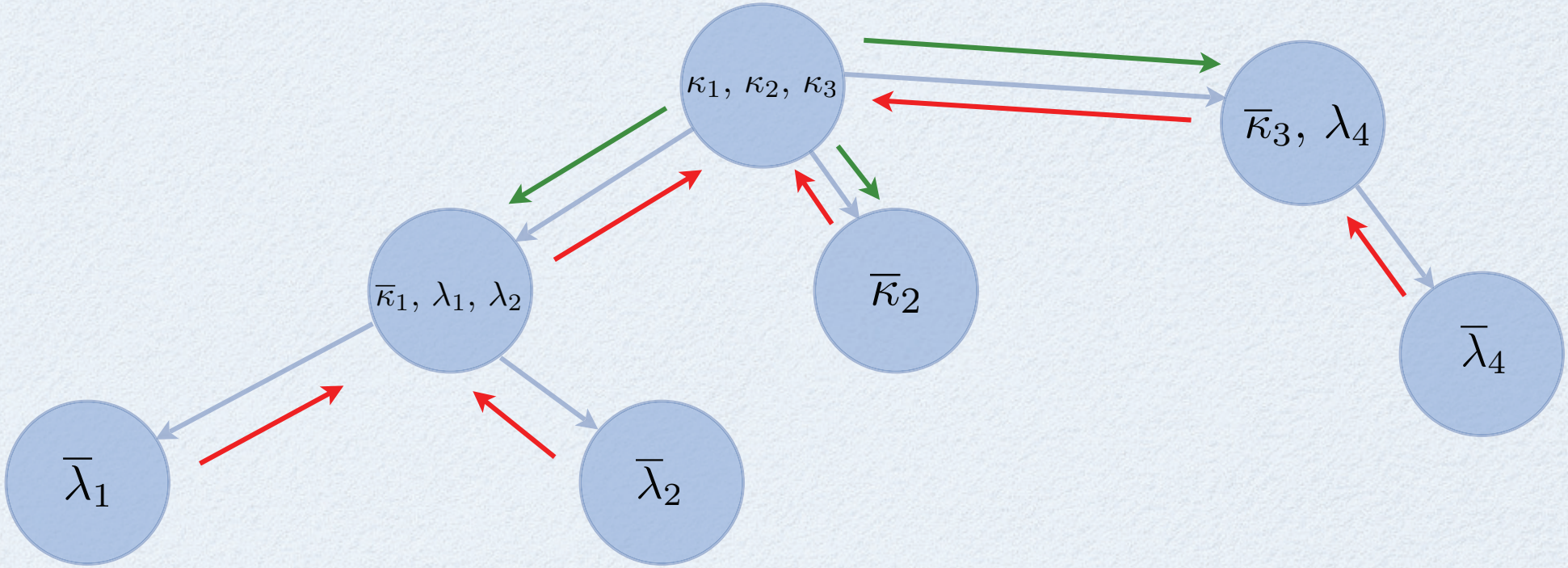
1) the leaves communicate their will to terminate to their father i.e. there are no active processes

TERMINATION PROTOCOL



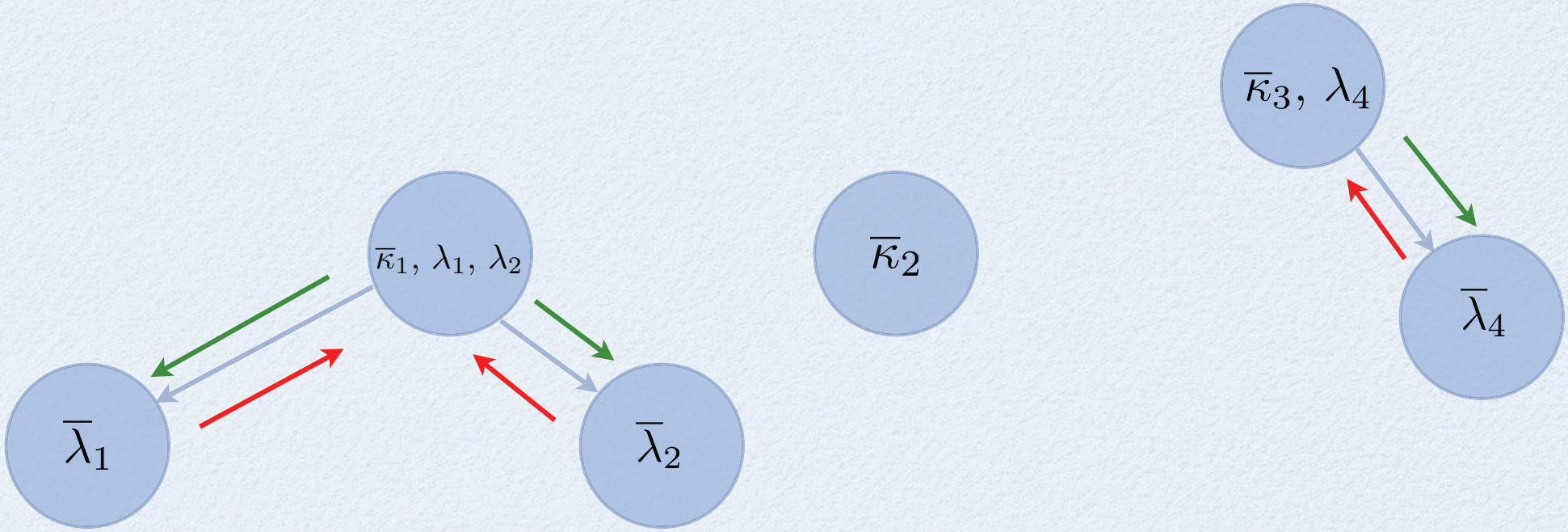
2) when a node receives a request from all children (and it is willing to terminate), it propagates upwards

TERMINATION PROTOCOL



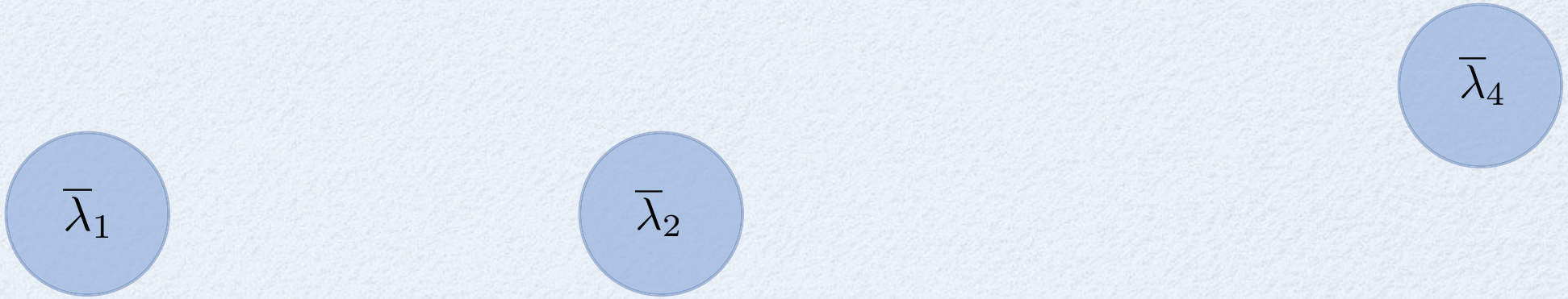
3) when the root collects a request from all its children then it terminates and sends ack to children

TERMINATION PROTOCOL



4) Each node sends acks and terminates.

TERMINATION PROTOCOL



4) Each node sends acks and terminates.

TERMINATION PROTOCOL

4) Each node sends acks and terminates.

Liveness

Theorem (Liveness).
If P is typable and $P \rightarrow^* Q$ then either Q is stable or $Q \rightarrow Q'$

*a process is stable whenever there are no pending outputs, inputs or try-catch blocks