Work Analysis with Resource-Aware Session Types

Ankush Das
Jan Hoffmann
Frank Pfenning

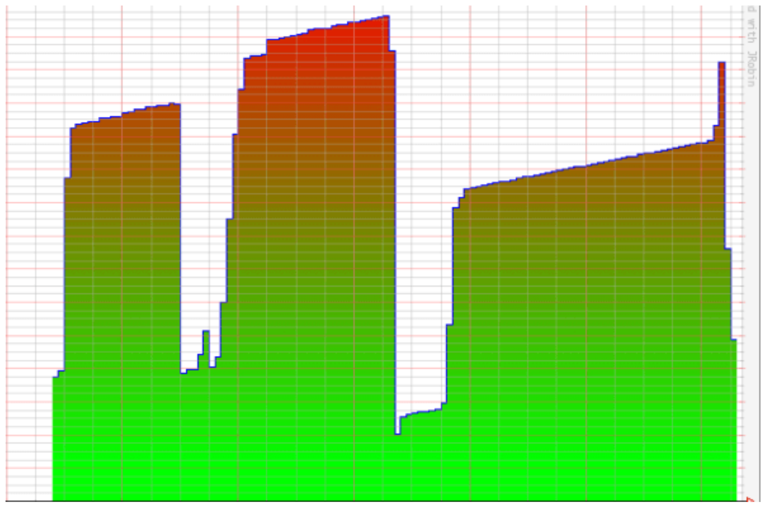
LICS, July 10, 2018

Carnegie Mellon University
Computer Science Department

Goal for this Talk

Resource Analysis
for Concurrent Programs

Execution Time



Memory Usage

Why Resource Analysis?

Why Resource Analysis?

Complexity of Parallel Algorithms

Çiçek et. al. (ESOP '15)

Why Resource Analysis?

Complexity of Parallel Algorithms

Çiçek et. al. (ESOP ’15)

Design of Optimal Scheduling Policies

Acar et. al. (JFP ’16)

Why Resource Analysis?

Complexity of Parallel Algorithms

Çiçek et. al. (ESOP ’15)

Design of Optimal Scheduling Policies

Acar et. al. (JFP ’16)

Prevention of Side-Channel Attacks

Ngo et. al. (S&P ’17)

Why Resource Analysis?

Complexity of Parallel Algorithms

Çiçek et. al. (ESOP ’15)

Design of Optimal Scheduling Policies

Acar et. al. (JFP ’16)

Prevention of Side-Channel Attacks

Ngo et. al. (S&P ’17)

Static and Dynamic Profiling

Haemmerlé et. al. (FLOPS ’17)

Measures of Execution Time

Measures of Execution Time

Work
Sequential Complexity

Execution time on one processor

Measures of Execution Time

Work
Sequential Complexity

Execution time
on one processor

Span
Parallel Complexity

Execution time on
arbitrarily many processors

Measures of Execution Time

Work
Sequential Complexity

Execution time
on one processor

Span
Parallel Complexity

Execution time on
arbitrarily many processors

Measures of Execution Time

Work
Sequential Complexity

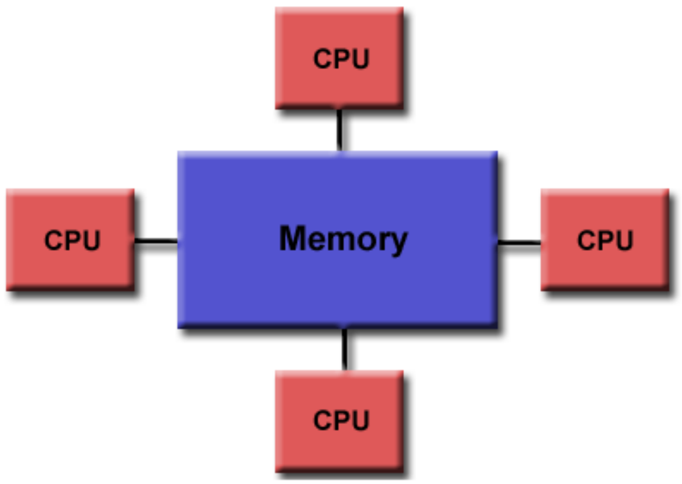
Execution time on one processor

Span
Parallel Complexity

Execution time on arbitrarily many processors

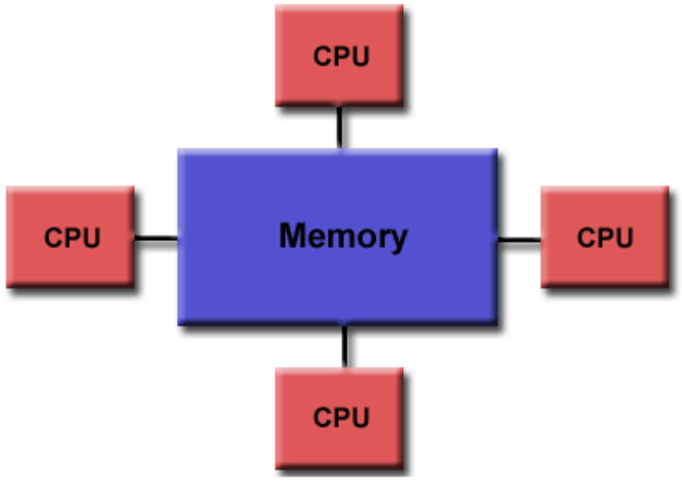
Concurrent Programs are hard to analyze!

Concurrent Programs are hard to analyze!



Shared Memory Read/Write Overhead

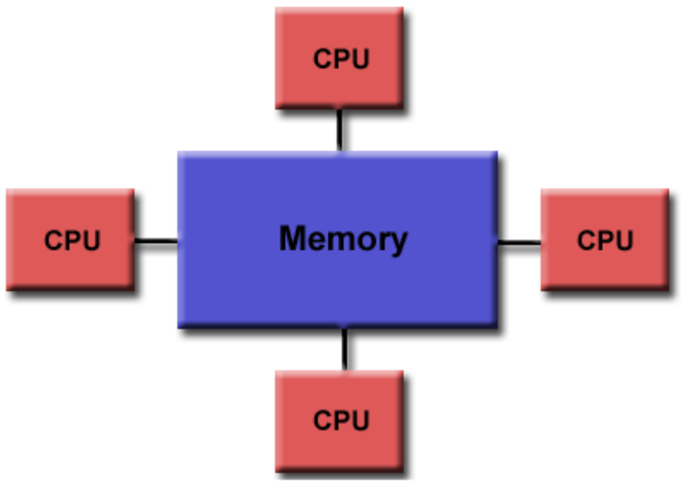
Concurrent Programs are hard to analyze!



Shared Memory Read/Write Overhead

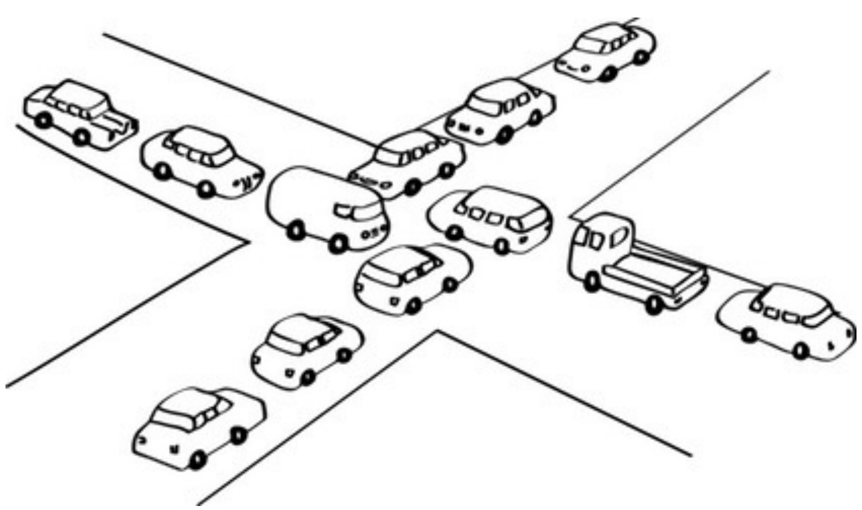
Communication Overhead

Concurrent Programs are hard to analyze!



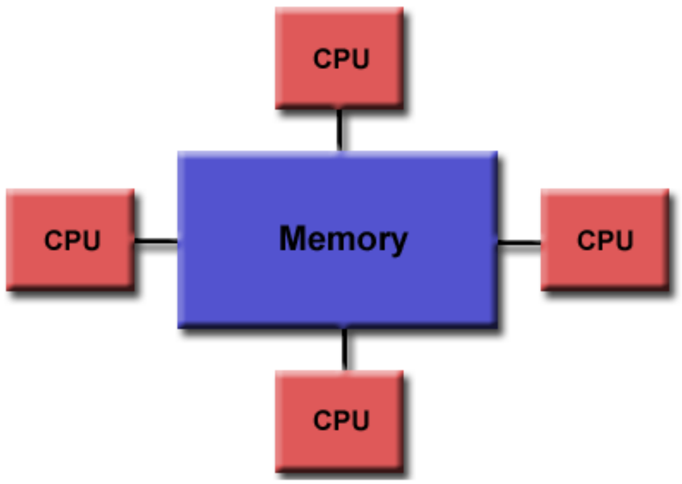
Shared Memory Read/Write Overhead

Communication Overhead



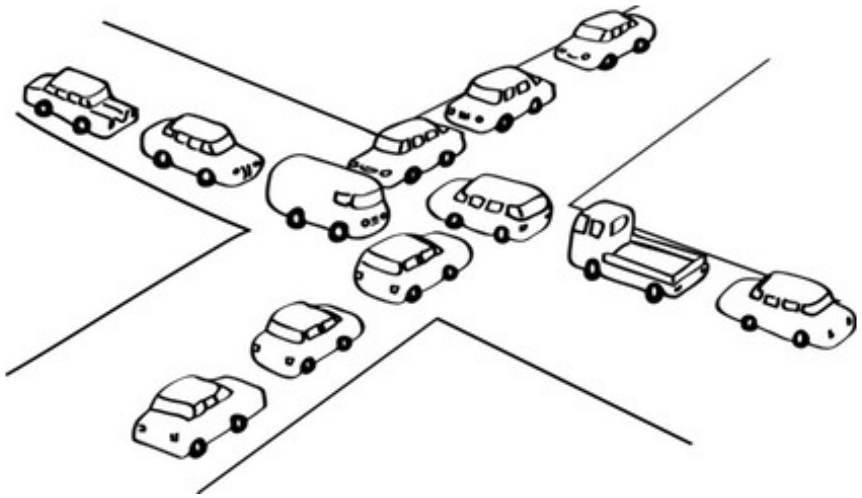
Deadlocks

Concurrent Programs are hard to analyze!



Shared Memory Read/Write Overhead

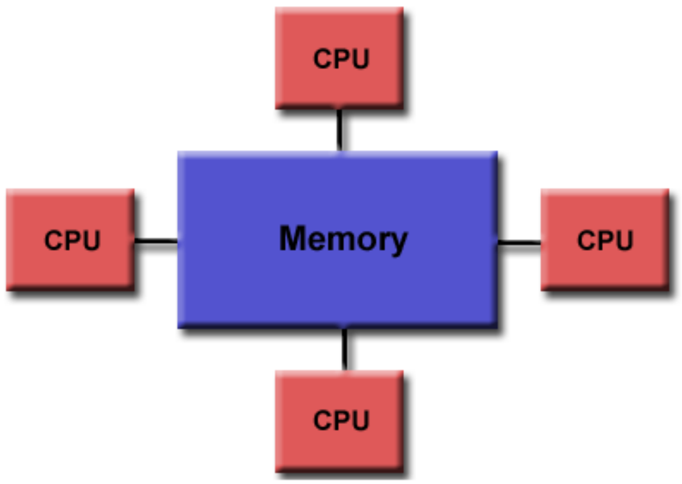
Communication Overhead



Deadlocks

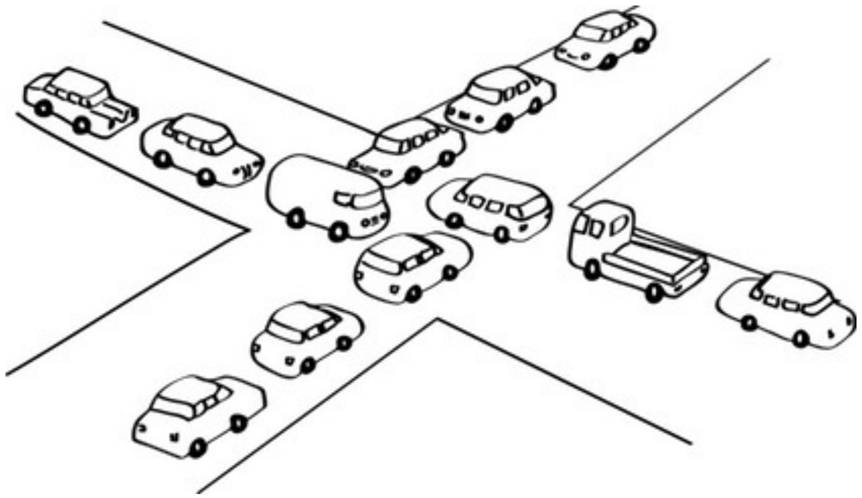
Non-Compositional

Session types can help :-)



Shared Memory
Read/Write Overhead

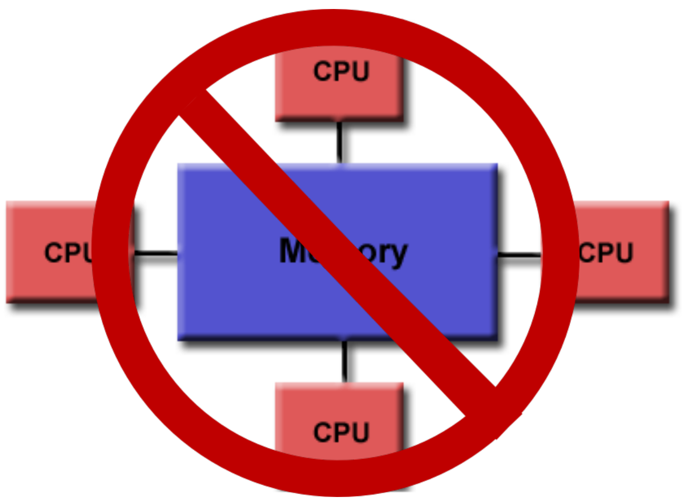
Communication Overhead



Deadlocks

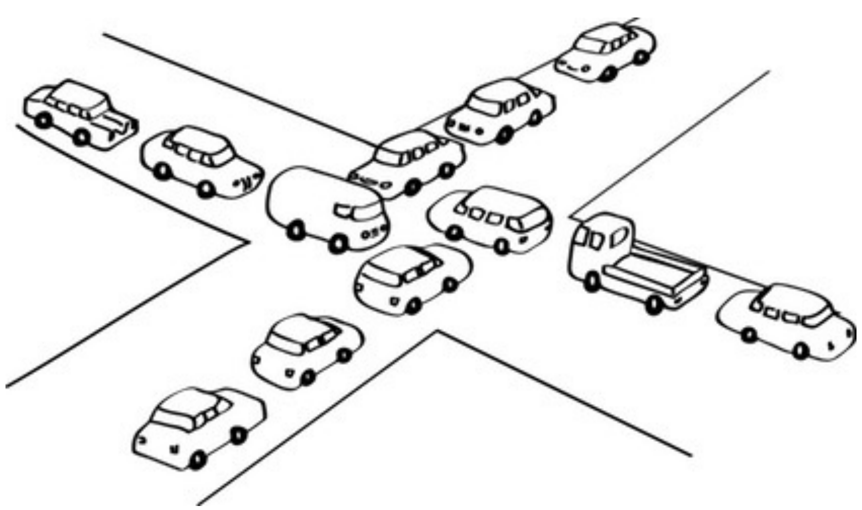
Non-Compositional

Session types can help :-)

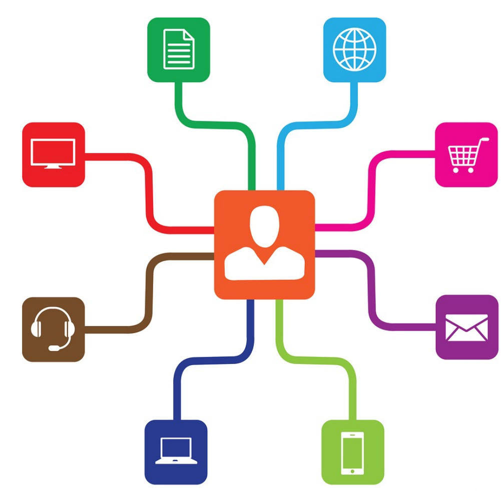


No Shared Memory

Communication Overhead

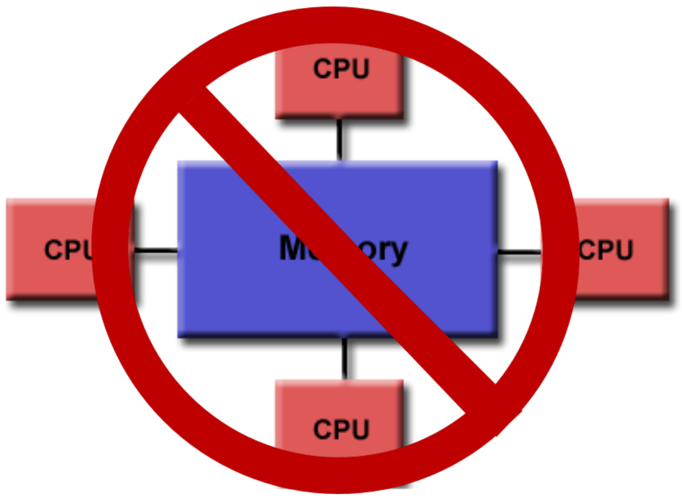


Deadlocks



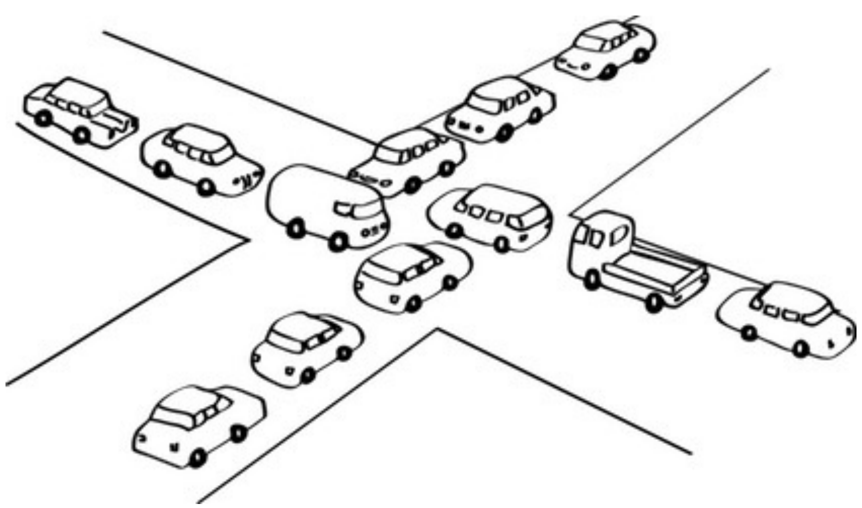
Non-Compositional

Session types can help :-)



No Shared Memory

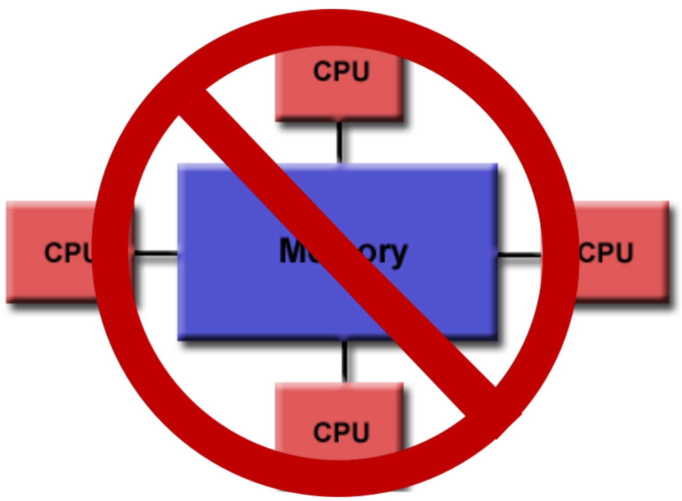
Types strictly enforce communication protocols



Deadlocks

Non-Compositional

Session types can help :-)



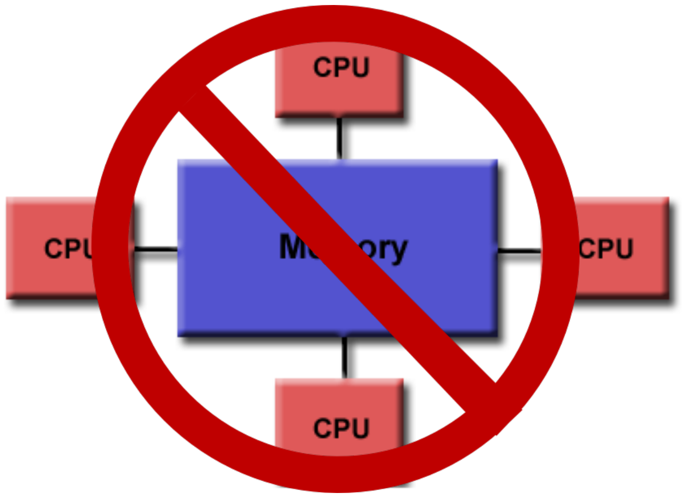
No Shared Memory

Types strictly enforce communication protocols

Deadlock Freedom

Non-Compositional

Session types can help :-)



No Shared Memory

Types strictly enforce communication protocols

Deadlock Freedom

Channels abstract over connected processes

What are Session Types?

- Implement message-passing concurrent programs
- Communication via bi-directional typed channels
- Curry-Howard isomorphism with intuitionistic linear logic

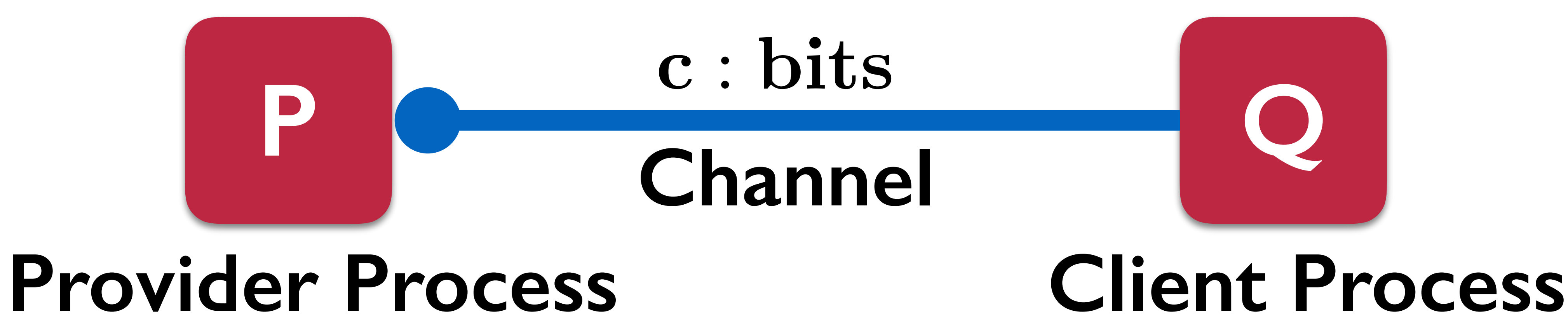
What are Session Types?

- Implement message-passing concurrent programs
- Communication via bi-directional typed channels
- Curry-Howard isomorphism with intuitionistic linear logic

What are Session Types?

- Implement message-passing concurrent programs
- Communication via bi-directional typed channels
- Curry-Howard isomorphism with intuitionistic linear logic

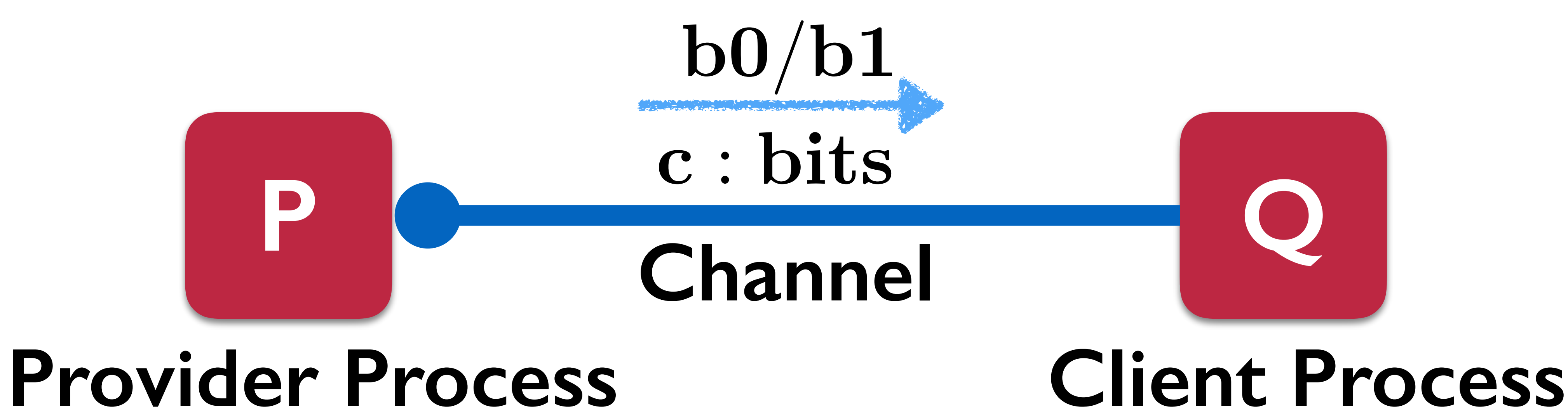
$$\mathrm{bits} = \oplus\{\mathrm{b0} : \mathrm{bits}, \mathrm{b1} : \mathrm{bits}\}$$



What are Session Types?

- Implement message-passing concurrent programs
- Communication via bi-directional typed channels
- Curry-Howard isomorphism with intuitionistic linear logic

$$\mathrm{bits} = \oplus\{\mathrm{b0} : \mathrm{bits}, \mathrm{b1} : \mathrm{bits}\}$$



Contributions

Contributions

Design a type system to analyze work of session-typed programs

Contributions

Design a type system to analyze work of session-typed programs

- based on amortized analysis
- can be parameterized to count different resources
- proved sound w.r.t. cost semantics
- conservative extension to standard session type system
- applied to all standard programs

Contributions

Design a type system to analyze work of session-typed programs

- based on amortized analysis
- can be parameterized to count different resources
- proved sound w.r.t. cost semantics
- conservative extension to standard session type system
- applied to all standard programs

messages exchanged

Contributions

Design a type system to analyze work of session-typed programs

- based on amortized analysis
- can be parameterized to count different resources
- proved sound w.r.t. cost semantics
- conservative extension to standard session type system
- applied to all standard programs

messages exchanged

processes spawned

Contributions

Design a type system to analyze work of session-typed programs

- based on amortized analysis
- can be parameterized to count different resources
- proved sound w.r.t. cost semantics
- conservative extension to standard session type system
- applied to all standard programs

messages exchanged

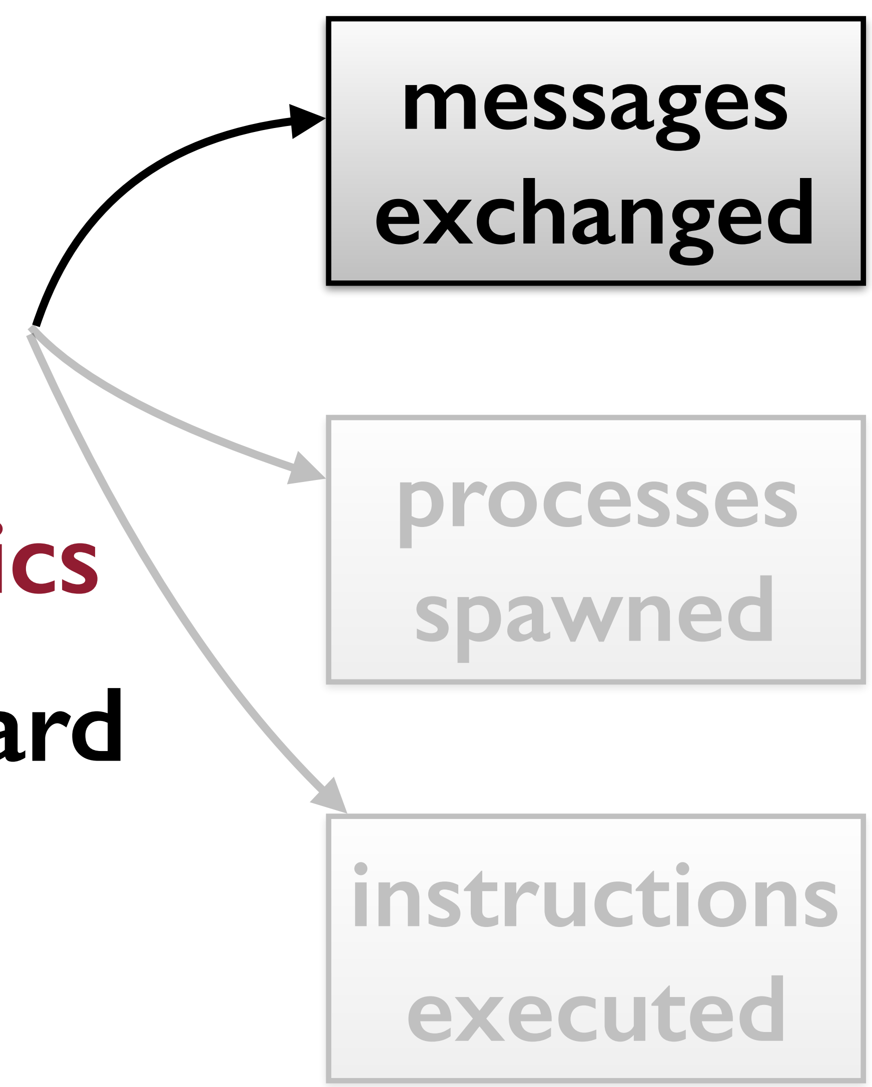
processes spawned

instructions executed

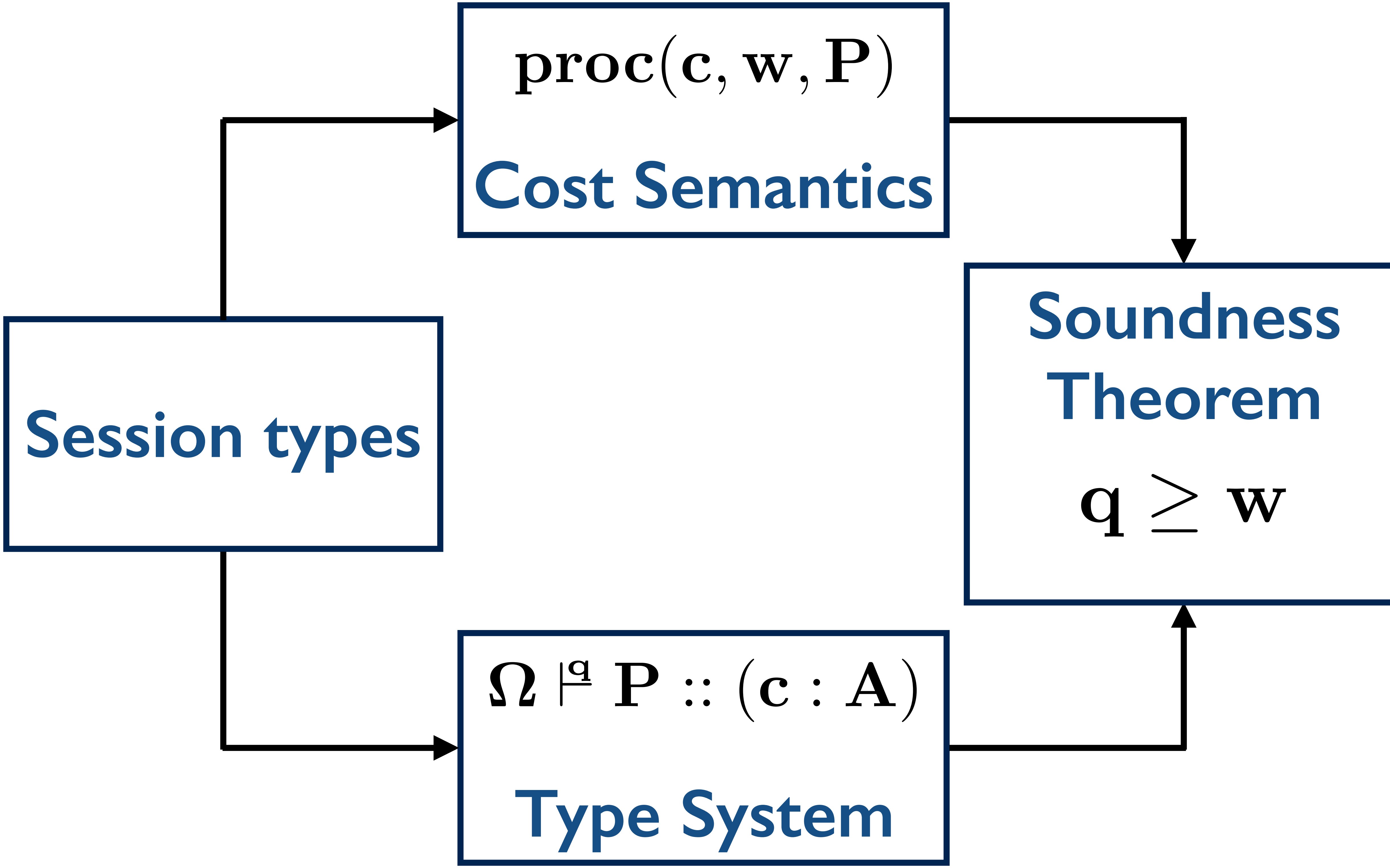
Contributions

Design a type system to analyze work of session-typed programs

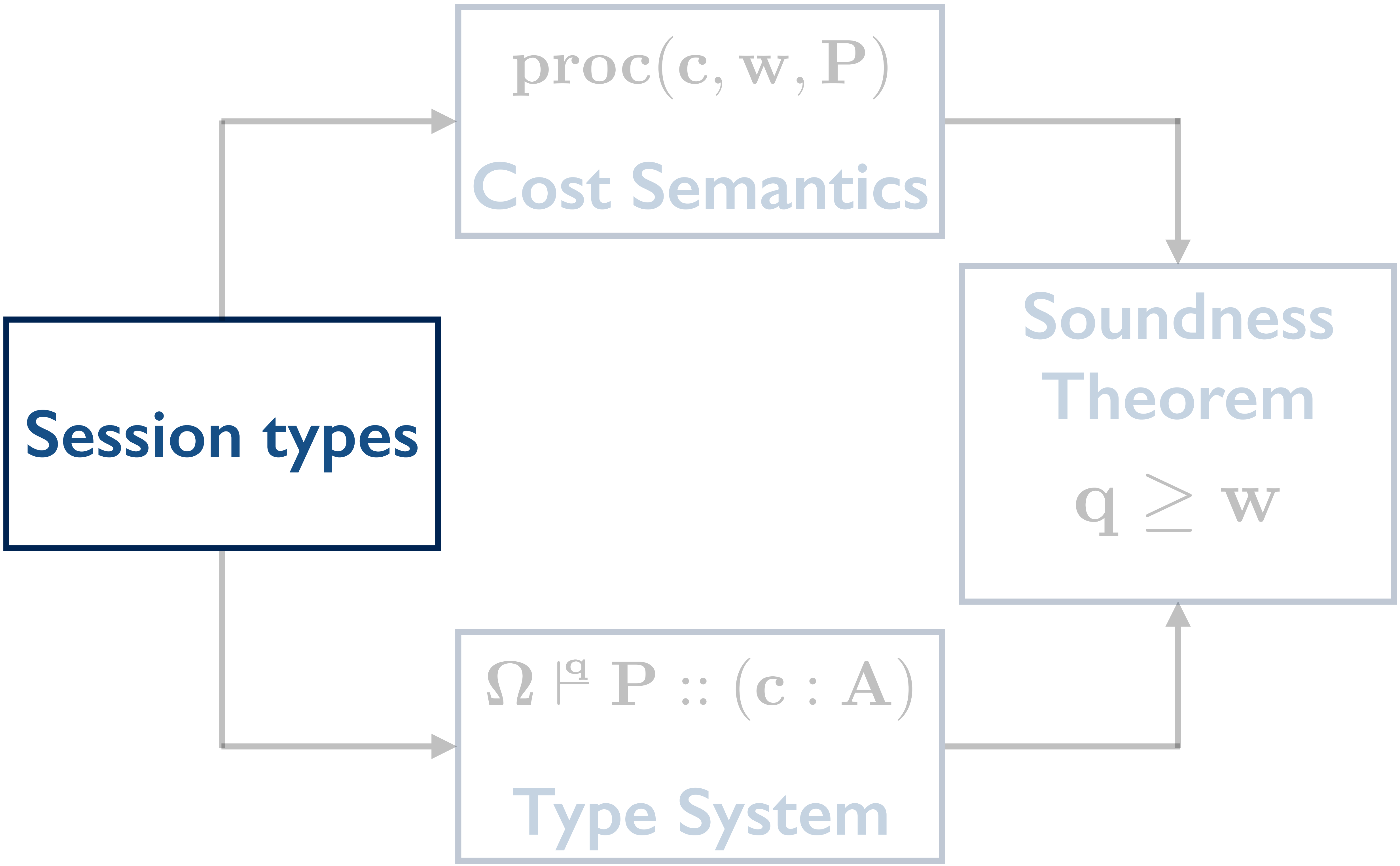
- based on amortized analysis
- can be parameterized to count different resources
- proved sound w.r.t. cost semantics
- conservative extension to standard session type system
- applied to all standard programs



Overview

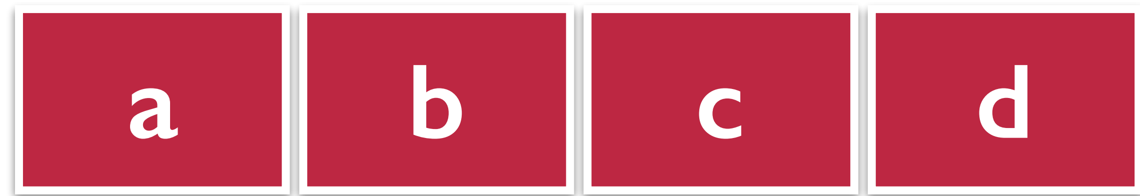

$\mathbf{proc(c, w, P)}$
Cost Semantics
Session types
Soundness Theorem
$\mathbf{q \geq w}$
$\mathbf{\Omega \vDash^{q} P :: (c : A)}$
Type System

Overview



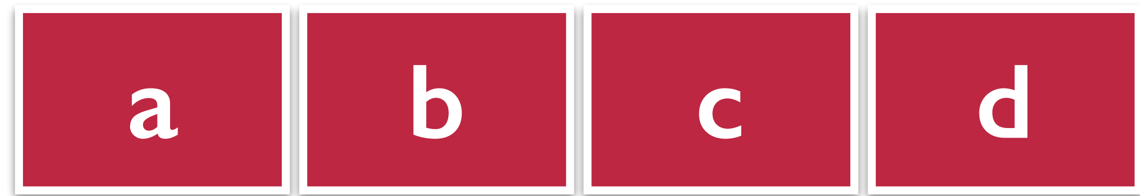
$\mathbf{proc}(c, w, P)$
Cost Semantics
Session types
Soundness Theorem
$q \geq w$
$\Omega \vDash^{q} P :: (c : A)$
Type System

Example: Queues



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

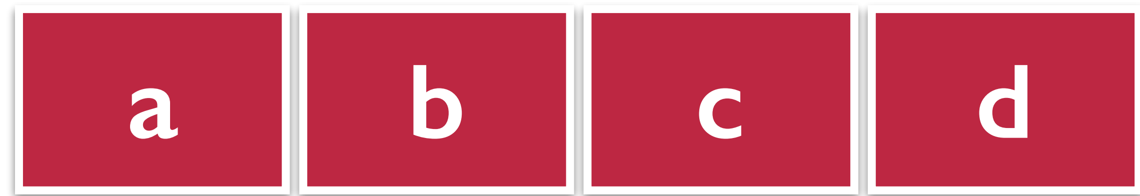
Example: Queues



offers choice of ins/del

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

Example: Queues

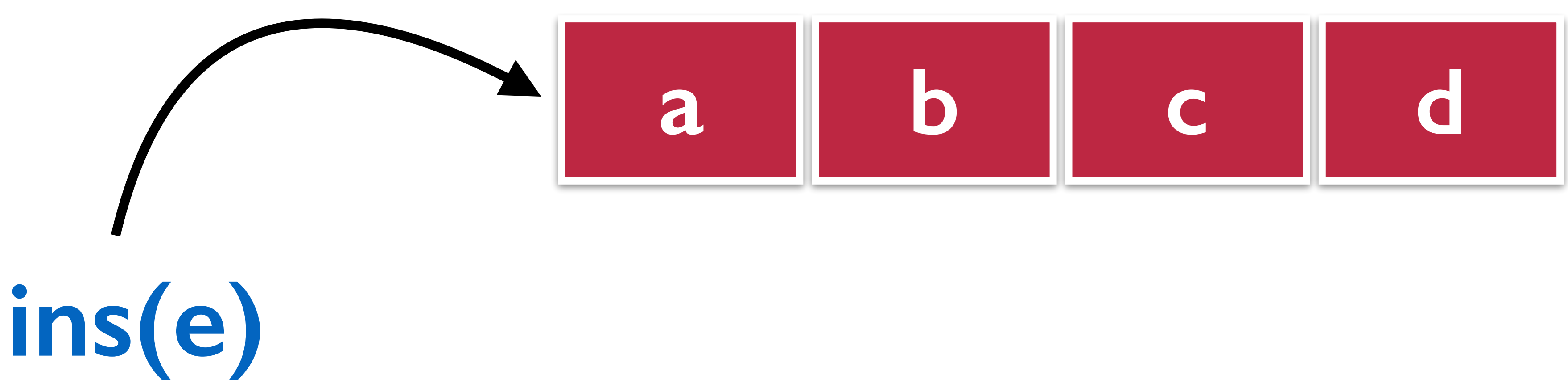


offers choice of ins/del

recv element of type A

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

Example: Queues

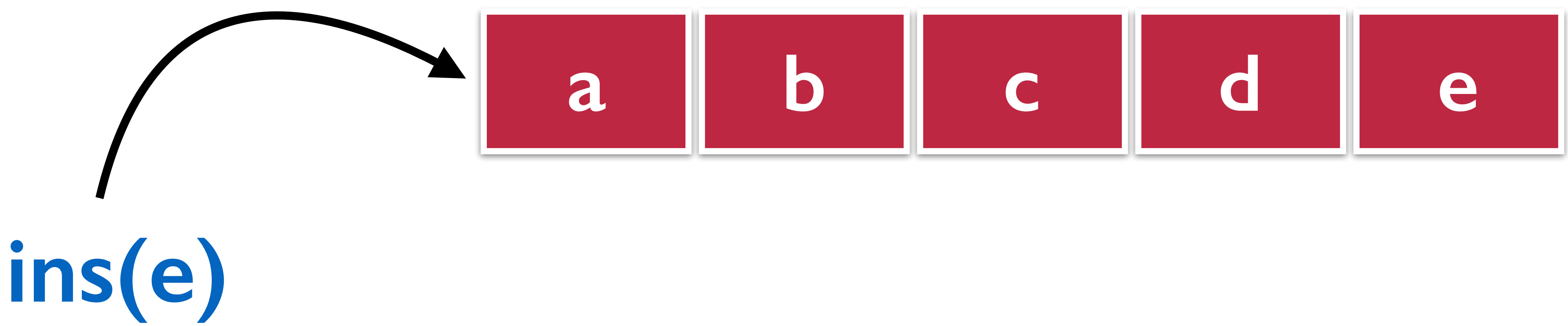


offers choice of ins/del

recv element of type A

$$\mathsf{queue}_{\mathbf{A}} = \&\{\mathsf{ins} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}},\ \mathsf{del} : \oplus\{\mathsf{none} : \mathbf{1},\ \mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}\}\}$$

Example: Queues

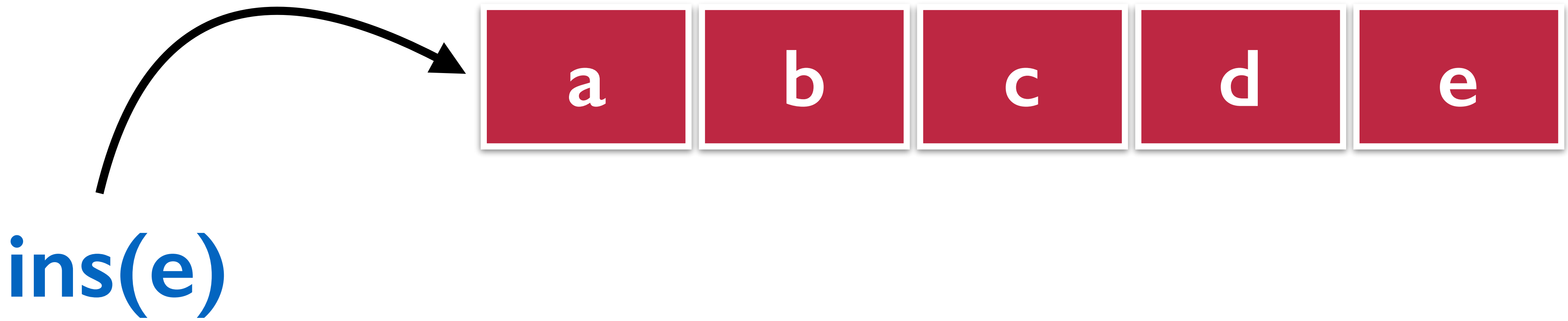


offers choice of ins/del

recv element of type A

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

Example: Queues



offers choice of ins/del

recv element of type A

behave as queue again

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

Example: Queues

offers choice of ins/del

$$\mathsf{queue}_{\mathbf{A}} = \&\{\mathsf{ins} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}},\ \mathsf{del} : \oplus\{\mathsf{none} : \mathbf{1},\ \mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}\}\}$$

Example: Queues

offers choice of ins/del

$$\mathsf{queue}_{\mathbf{A}} = \&\{\mathsf{ins} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}},\\ \mathsf{del} : \oplus\{\mathsf{none} : \mathbf{1},\\ \mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}\}\}$$

send none if queue is empty

Example: Queues

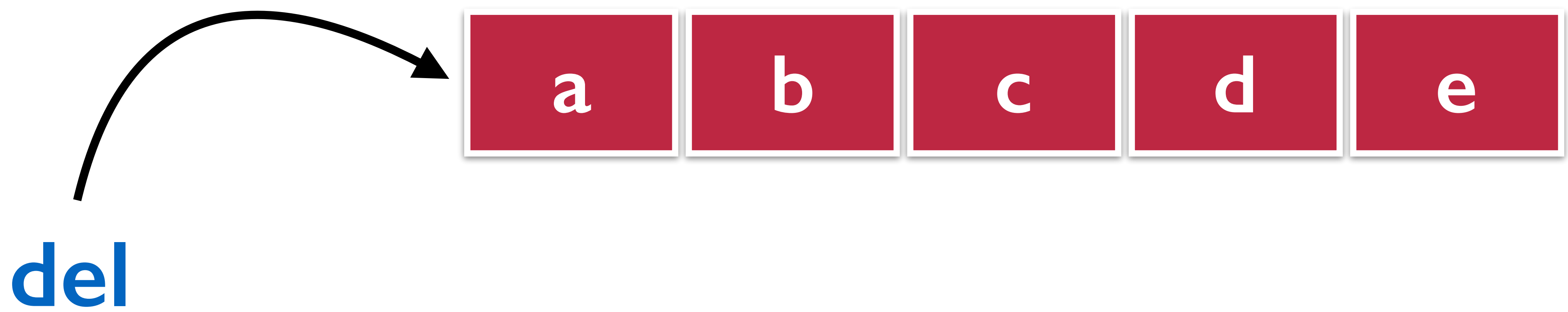
offers choice of ins/del

$$\mathsf{queue}_{\mathbf{A}} = \&\{\mathsf{ins} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}},\\ \mathsf{del} : \oplus\{\mathsf{none} : \mathbf{1},\\ \mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}\}\}$$

terminate

send none if queue is empty

Example: Queues



offers choice of ins/del

$$
\mathsf{queue}_{\mathbf{A}} = \&\{\mathsf{ins} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}},\ \mathsf{del} : \oplus\{\mathsf{none} : \mathbf{1},\ \mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}\}\}
$$

send some otherwise

Example: Queues

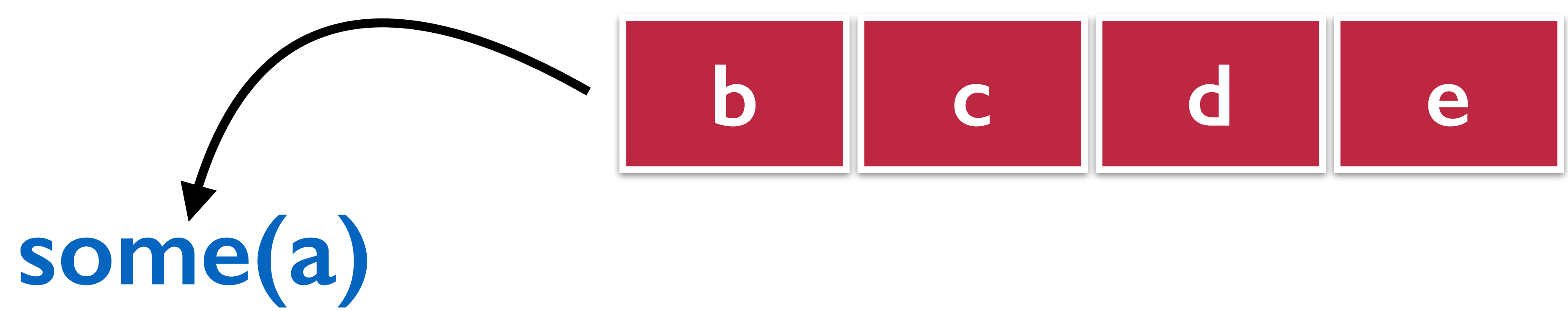
offers choice of ins/del

send element of type A

$$\mathrm{queue}_{\mathbf{A}} = \&\{\mathrm{ins} : \mathbf{A} \multimap \mathrm{queue}_{\mathbf{A}},\ \mathrm{del} : \oplus\{\mathrm{none} : \mathbf{1},\ \mathrm{some} : \mathbf{A} \otimes \mathrm{queue}_{\mathbf{A}}\}\}$$

send some otherwise

Example: Queues



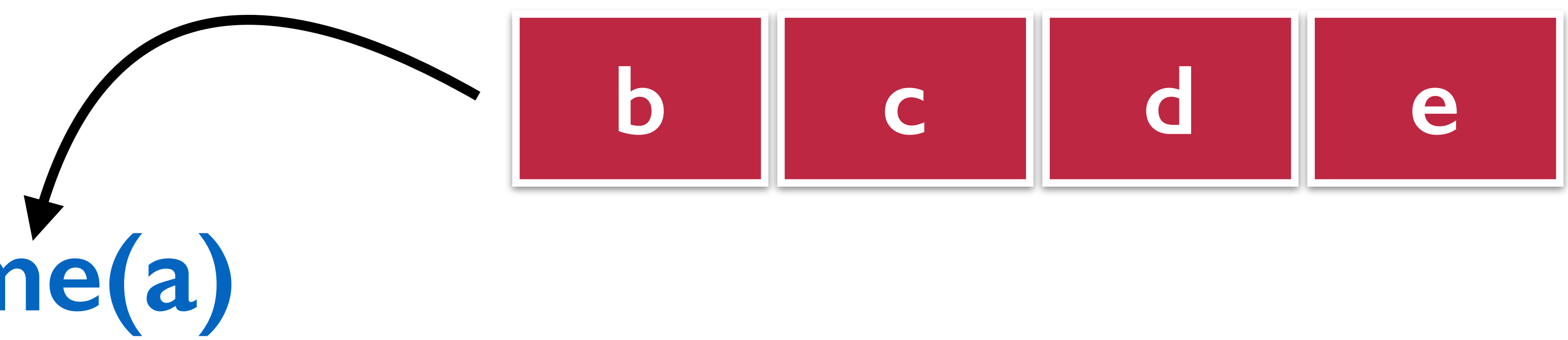
offers choice of ins/del

send element of type A

$$\mathsf{queue}_{\mathbf{A}} = \&\{\mathsf{ins} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}},\ \mathsf{del} : \oplus\{\mathsf{none} : \mathbf{1},\ \mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}\}\}$$

send some otherwise

Example: Queues



offers choice of ins/del

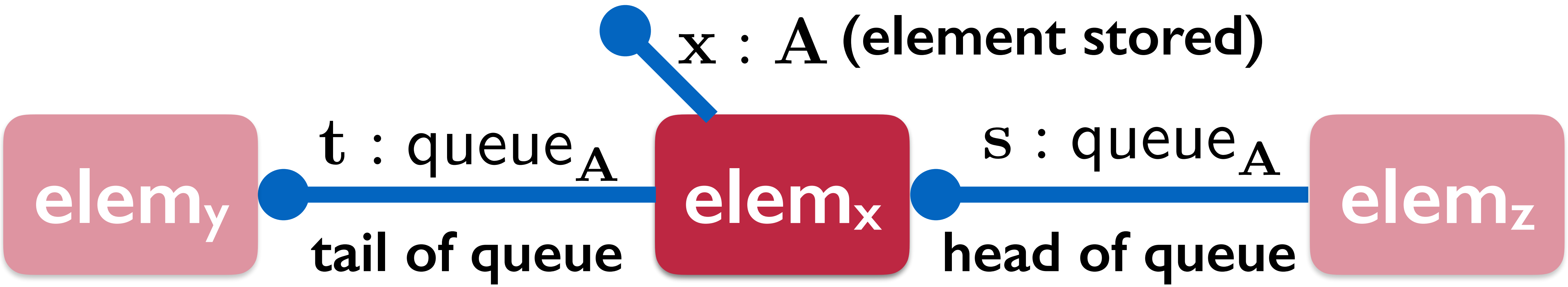
send element of type A

behave as queue again

$$\text{queue}_{\mathbf{A}} = \&\{\text{ins} : \mathbf{A} \multimap \text{queue}_{\mathbf{A}},\ \text{del} : \oplus\{\text{none} : \mathbf{1},\ \text{some} : \mathbf{A} \otimes \text{queue}_{\mathbf{A}}\}\}$$

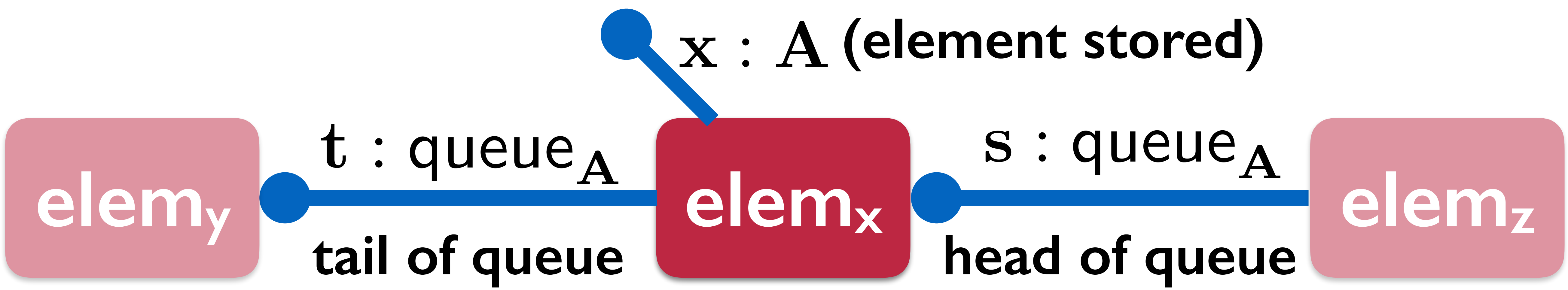
send some otherwise

Session-typed Program



$$
\begin{array}{l}
(x : A)\ (t : \mathsf{queue}_A) \vdash \mathit{elem} :: (s : \mathsf{queue}_A) \\
\quad s \leftarrow \mathit{elem} \leftarrow x\ t = \\
\qquad \mathsf{case}\ s\ (\mathsf{ins} \Rightarrow y \leftarrow \mathsf{recv}\ s\ ; \\
\qquad\qquad\qquad\quad t.\mathsf{ins}\ ; \\
\qquad\qquad\qquad\quad \mathsf{send}\ t\ y\ ; \\
\qquad\qquad\qquad\quad s \leftarrow \mathit{elem} \leftarrow x\ t \\
\qquad\qquad \mid \mathsf{del} \Rightarrow s.\mathsf{some}\ ; \\
\qquad\qquad\qquad\quad \mathsf{send}\ s\ x\ ; \\
\qquad\qquad\qquad\quad s \leftarrow t)
\end{array}
$$

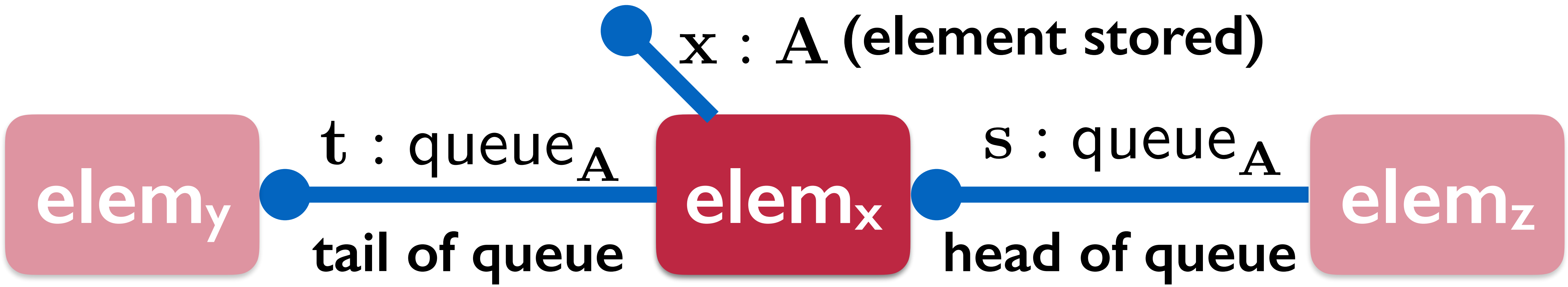
Session-typed Program



$$
\begin{array}{l}
(x : A)\ (t : \mathsf{queue}_A) \vdash \mathit{elem} :: (s : \mathsf{queue}_A) \\
\quad s \leftarrow \mathit{elem} \leftarrow x\ t = \\
\qquad \mathsf{case}\ s\ (\mathsf{ins} \Rightarrow y \leftarrow \mathsf{recv}\ s\ ; \\
\qquad\qquad\qquad\quad t.\mathsf{ins}\ ; \\
\qquad\qquad\qquad\quad \mathsf{send}\ t\ y\ ; \\
\qquad\qquad\qquad\quad s \leftarrow \mathit{elem} \leftarrow x\ t \\
\qquad\qquad\ \mid \mathsf{del} \Rightarrow s.\mathsf{some}\ ; \\
\qquad\qquad\qquad\quad \mathsf{send}\ s\ x\ ; \\
\qquad\qquad\qquad\quad s \leftarrow t)
\end{array}
$$

recv ‘ins’ and y

Session-typed Program

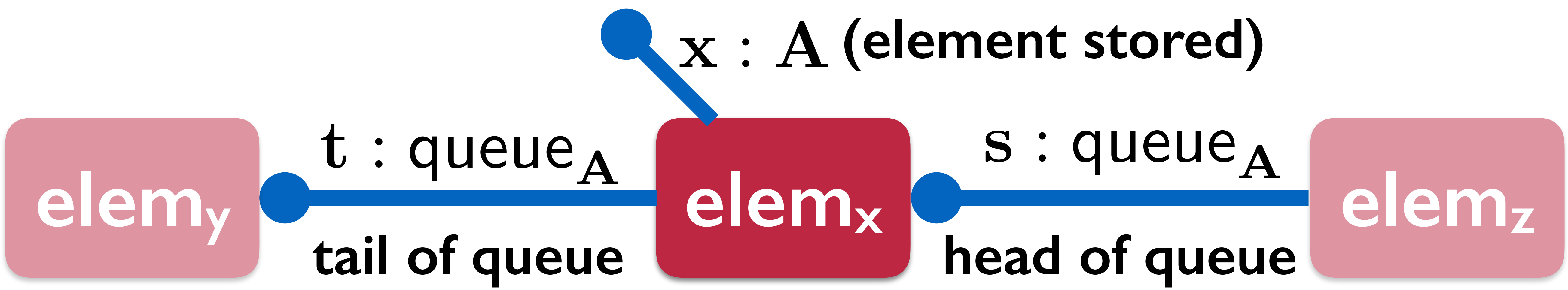


$$
\begin{array}{l}
(x : A)\ (t : \mathsf{queue}_A) \vdash \mathit{elem} :: (s : \mathsf{queue}_A) \\
\quad s \leftarrow \mathit{elem} \leftarrow x\ t = \\
\qquad \mathsf{case}\ s\ (\mathsf{ins} \Rightarrow y \leftarrow \mathsf{recv}\ s\ ; \\
\qquad\qquad\qquad\quad t.\mathsf{ins}\ ; \\
\qquad\qquad\qquad\quad \mathsf{send}\ t\ y\ ; \\
\qquad\qquad\qquad\quad s \leftarrow \mathit{elem} \leftarrow x\ t \\
\qquad\qquad \mid \mathsf{del} \Rightarrow s.\mathsf{some}\ ; \\
\qquad\qquad\qquad\quad \mathsf{send}\ s\ x\ ; \\
\qquad\qquad\qquad\quad s \leftarrow t)
\end{array}
$$

recv ‘ins’ and y

send ‘ins’ and y

Session-typed Program



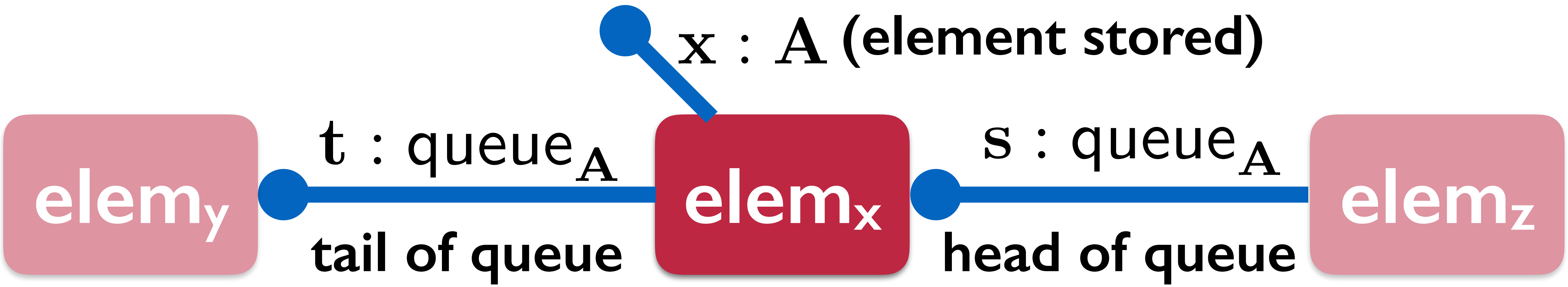
$$
\begin{array}{l}
(x : A)\ (t : \mathsf{queue}_A) \vdash \mathit{elem} :: (s : \mathsf{queue}_A) \\
\quad s \leftarrow \mathit{elem} \leftarrow x\ t = \\
\qquad\quad \mathsf{case}\ s\ (\mathsf{ins} \Rightarrow y \leftarrow \mathsf{recv}\ s\ ; \\
\qquad\qquad\qquad\qquad\quad t.\mathsf{ins}\ ; \\
\qquad\qquad\qquad\qquad\quad \mathsf{send}\ t\ y\ ; \\
\qquad\qquad\qquad\qquad\quad s \leftarrow \mathit{elem} \leftarrow x\ t \\
\qquad\qquad\qquad\ \mid \mathsf{del} \Rightarrow s.\mathsf{some}\ ; \\
\qquad\qquad\qquad\qquad\quad \mathsf{send}\ s\ x\ ; \\
\qquad\qquad\qquad\qquad\quad s \leftarrow t)
\end{array}
$$

recv 'ins' and y

send 'ins' and y

recurse

Session-typed Program



$$
\begin{array}{l}
(x : A)\ (t : \mathsf{queue}_A) \vdash \mathit{elem} :: (s : \mathsf{queue}_A) \\
\quad s \leftarrow \mathit{elem} \leftarrow x\ t = \\
\qquad \mathsf{case}\ s\ (\mathsf{ins} \Rightarrow y \leftarrow \mathsf{recv}\ s\ ; \\
\qquad\qquad\qquad\qquad t.\mathsf{ins}\ ; \\
\qquad\qquad\qquad\qquad \mathsf{send}\ t\ y\ ; \\
\qquad\qquad\qquad\qquad s \leftarrow \mathit{elem} \leftarrow x\ t \\
\qquad\qquad\quad \mid \mathsf{del} \Rightarrow s.\mathsf{some}\ ; \\
\qquad\qquad\qquad\qquad \mathsf{send}\ s\ x\ ; \\
\qquad\qquad\qquad\qquad s \leftarrow t)
\end{array}
$$

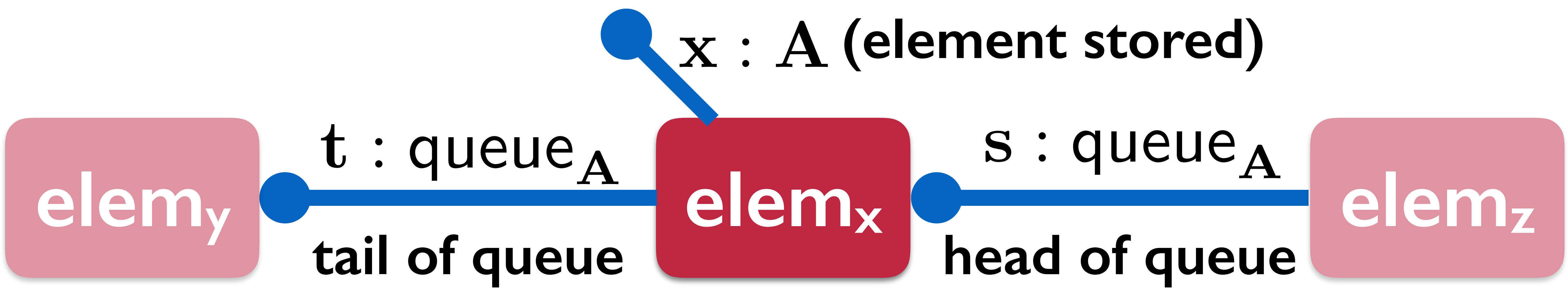
recv ‘ins’ and y

send ‘ins’ and y

recurse

send ‘some’, x

Session-typed Program



$$
\begin{array}{l}
(x : A)\ (t : \mathsf{queue}_A) \vdash elem :: (s : \mathsf{queue}_A) \\
\quad s \leftarrow elem \leftarrow x\ t = \\
\qquad \mathsf{case}\ s\ (\mathsf{ins} \Rightarrow y \leftarrow \mathsf{recv}\ s\ ; \\
\qquad\qquad\qquad\quad t.\mathsf{ins}\ ; \\
\qquad\qquad\qquad\quad \mathsf{send}\ t\ y\ ; \\
\qquad\qquad\qquad\quad s \leftarrow elem \leftarrow x\ t \\
\qquad\qquad\quad \mid \mathsf{del} \Rightarrow s.\mathsf{some}\ ; \\
\qquad\qquad\qquad\quad \mathsf{send}\ s\ x\ ; \\
\qquad\qquad\qquad\quad s \leftarrow t)
\end{array}
$$

recv ‘ins’ and y

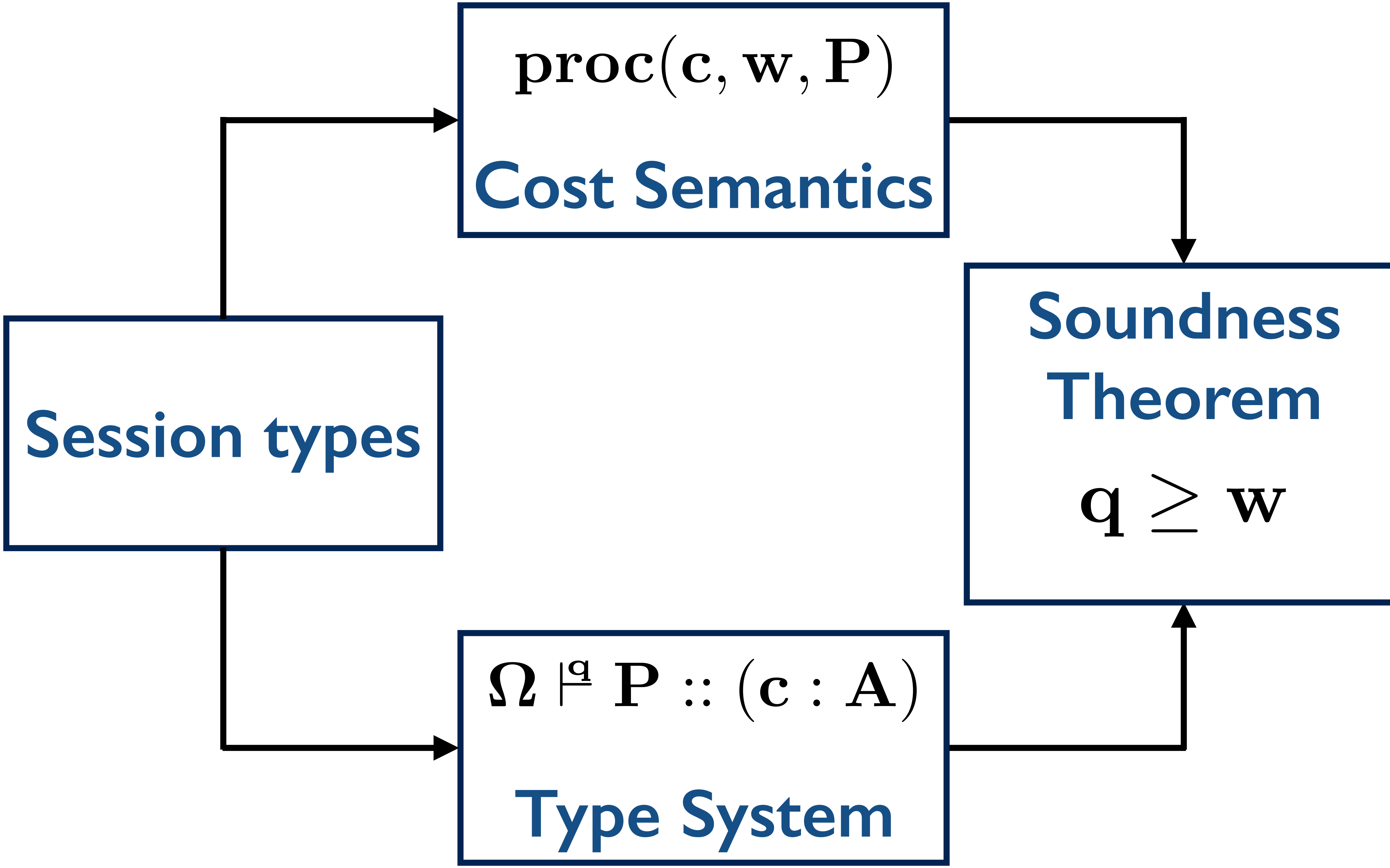
send ‘ins’ and y

recurse

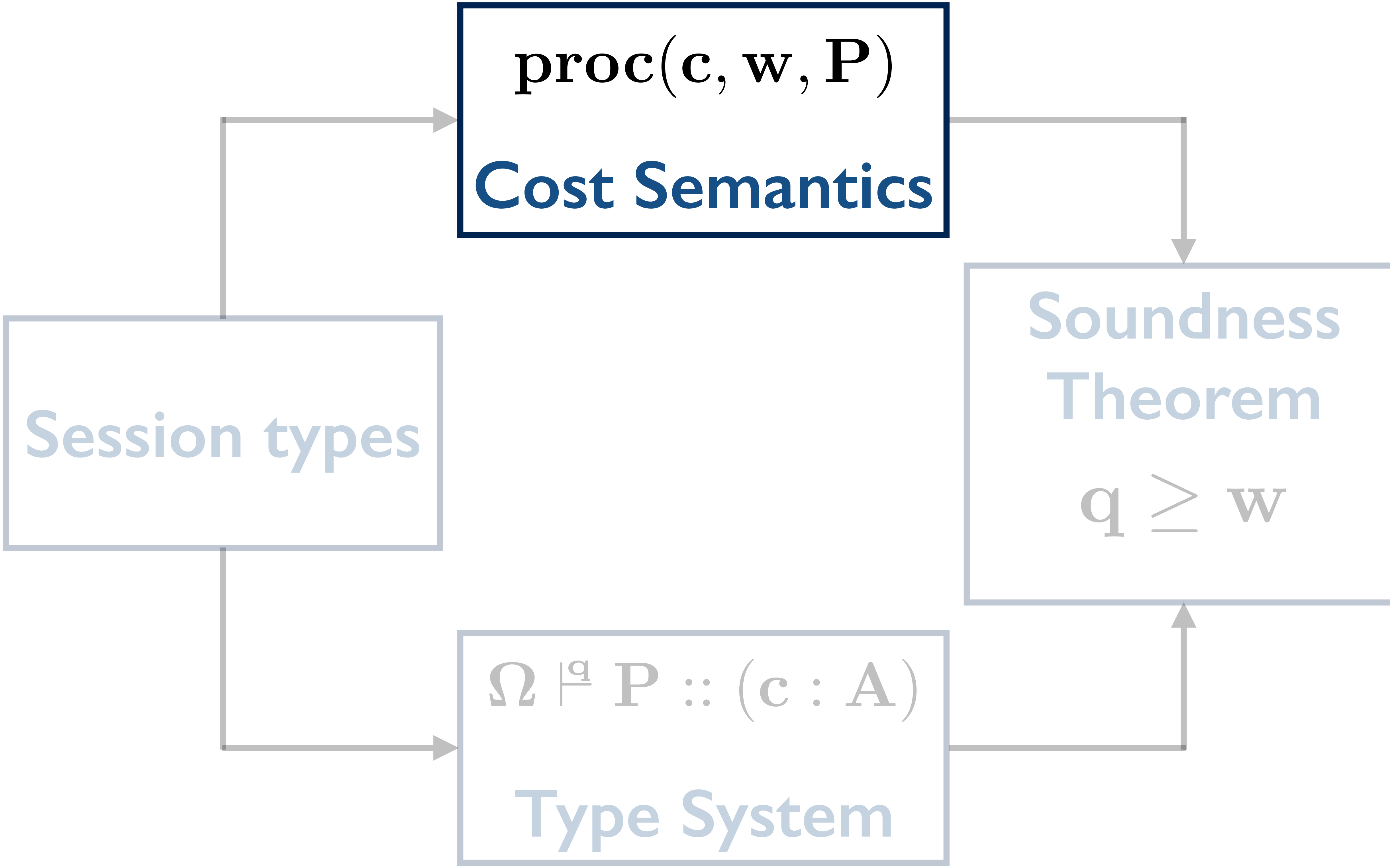
send ‘some’, x

terminate

Overview


$\mathbf{proc(c, w, P)}$
Cost Semantics
Session types
Soundness Theorem
$\mathbf{q \geq w}$
$\mathbf{\Omega \vdash^{q} P :: (c : A)}$
Type System

Overview


$\mathbf{proc}(\mathbf{c}, \mathbf{w}, \mathbf{P})$
Cost Semantics
Session types
Soundness Theorem
$\mathbf{q} \geq \mathbf{w}$
$\Omega \vDash^{q} \mathbf{P} :: (\mathbf{c} : \mathbf{A})$
Type System

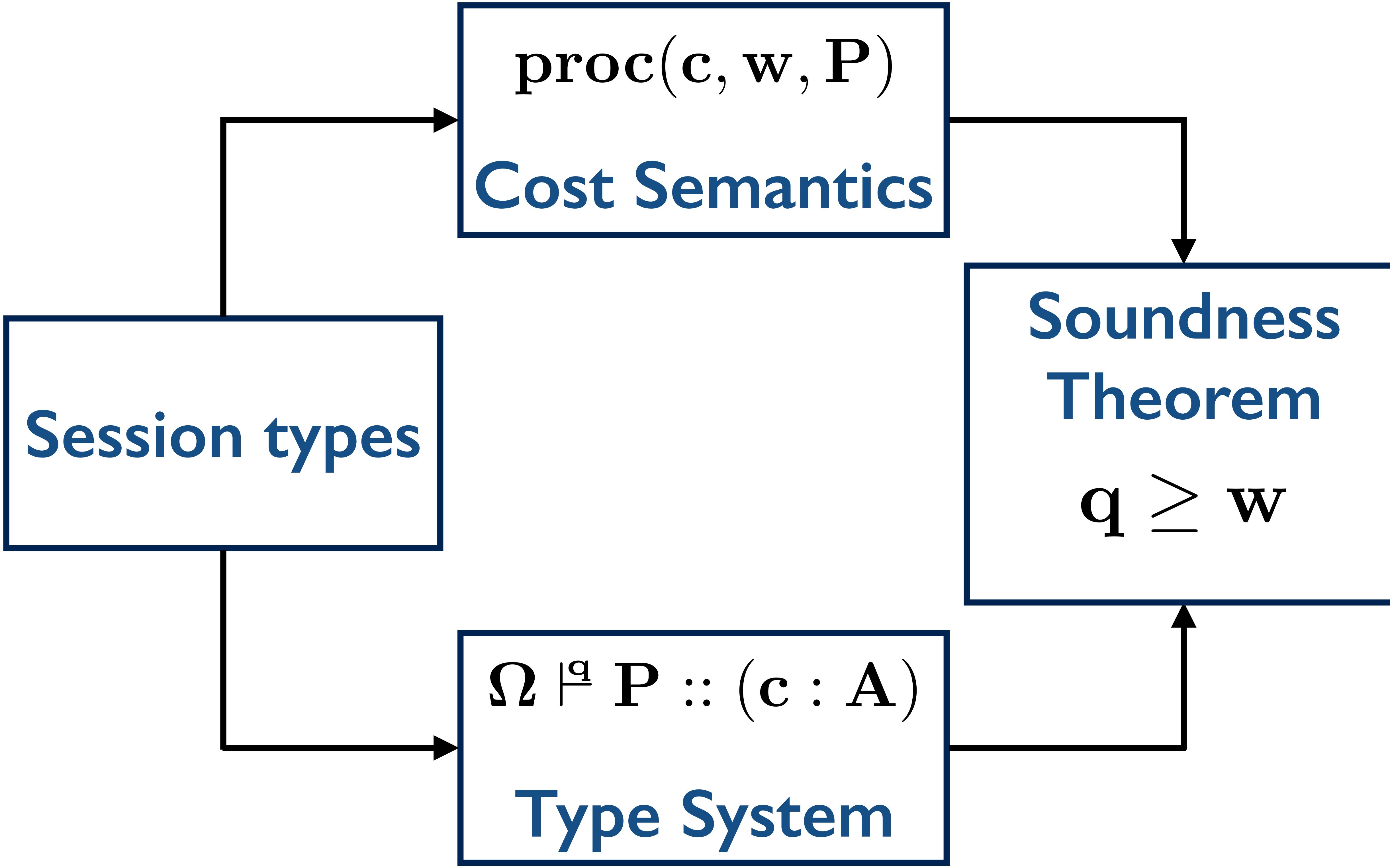
Cost Semantics

$$\mathbf{proc}(\mathbf{c}, \mathbf{w}, \mathbf{P})$$

Process P offering along channel c and has performed work w

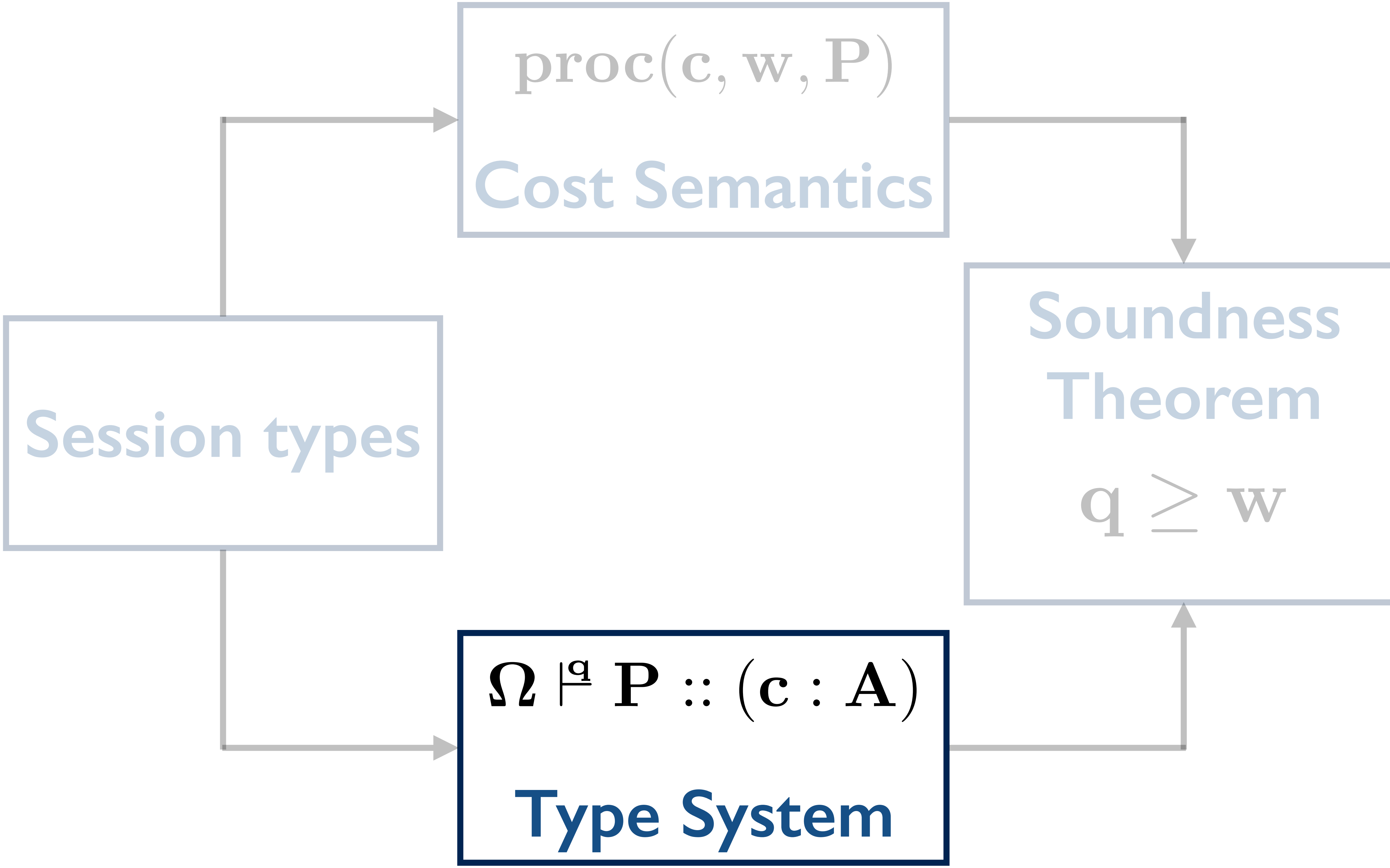
- Standard semantics extended with local work counters w for each process
- Total work of system is sum of local counters w
- w is incremented every time process P performs some 'work' (this talk: whenever message is sent)

Overview



$\mathbf{proc(c, w, P)}$
Cost Semantics
Session types
Soundness
Theorem
$\mathbf{q} \geq \mathbf{w}$
$\Omega \vDash^{q} \mathbf{P} :: (\mathbf{c} : \mathbf{A})$
Type System

Overview



$\mathrm{proc}(c, w, P)$
Cost Semantics
Session types
Soundness Theorem
$q \geq w$
$\Omega \vdash^{q} P :: (c : A)$
Type System

Type System

Based on Amortized Analysis!

- Store potential in each process
- Potential can be transferred via messages
- Potential is consumed to perform ‘work’

Type System

Based on Amortized Analysis!

- Store potential in each process
- Potential can be transferred via messages
- Potential is consumed to perform ‘work’

$$\Omega \vdash^{q} P :: (c : A)$$

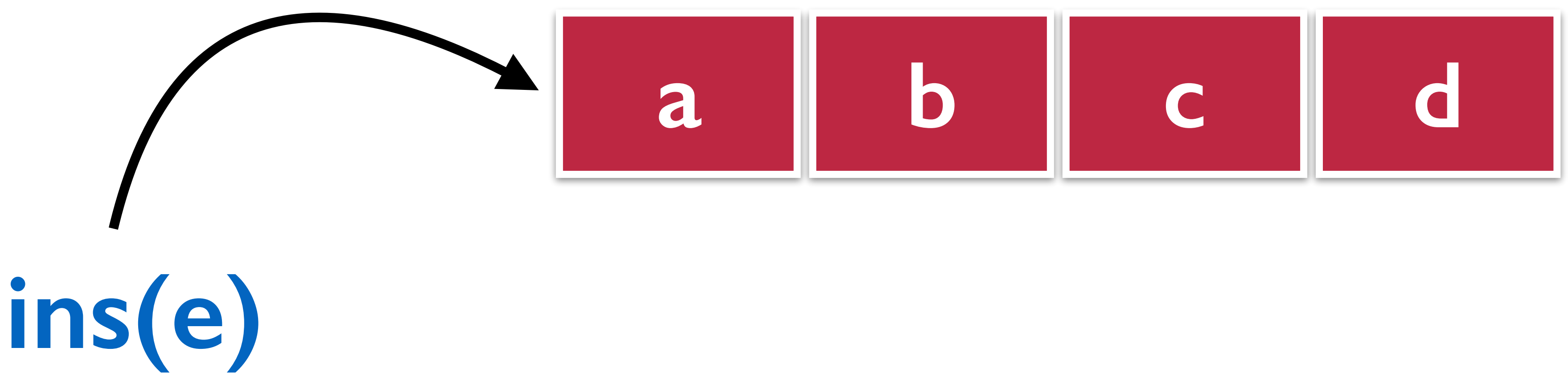
Process P offers along channel c,
acts as a client for channels in Ω
and storing potential q

Concurrent Queues

Count the messages!

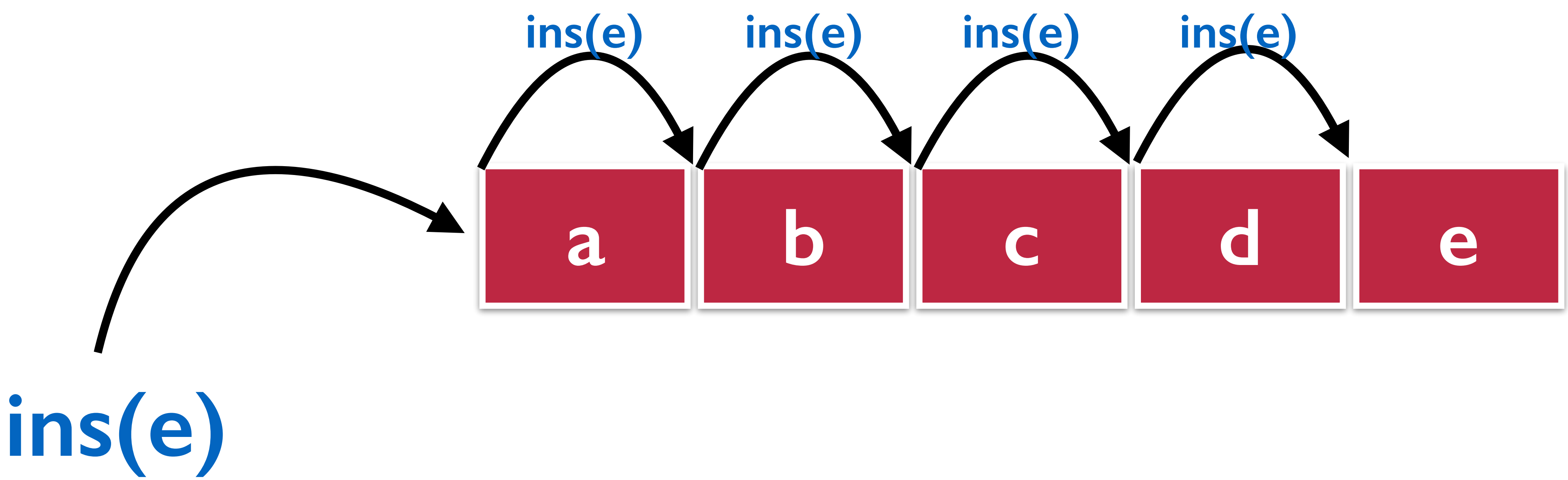
Concurrent Queues

Count the messages!



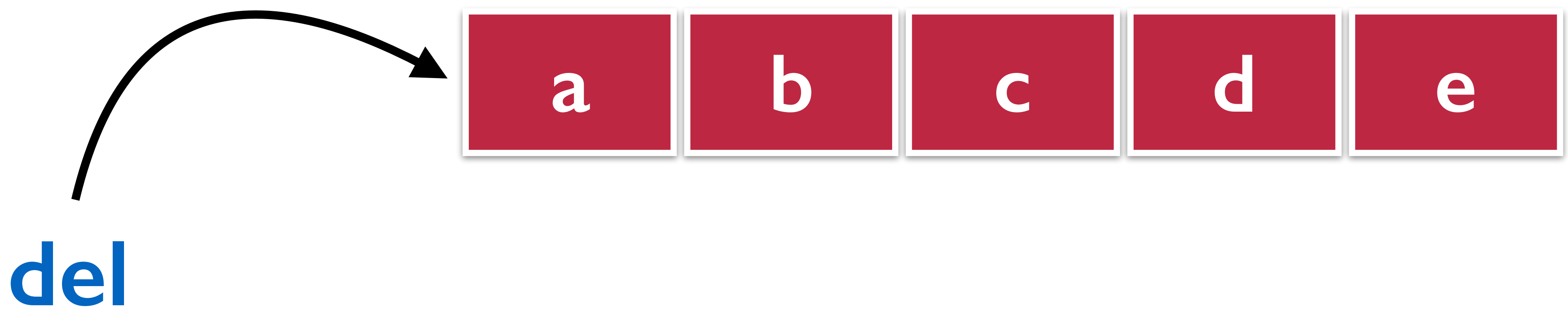
Concurrent Queues

Count the messages!



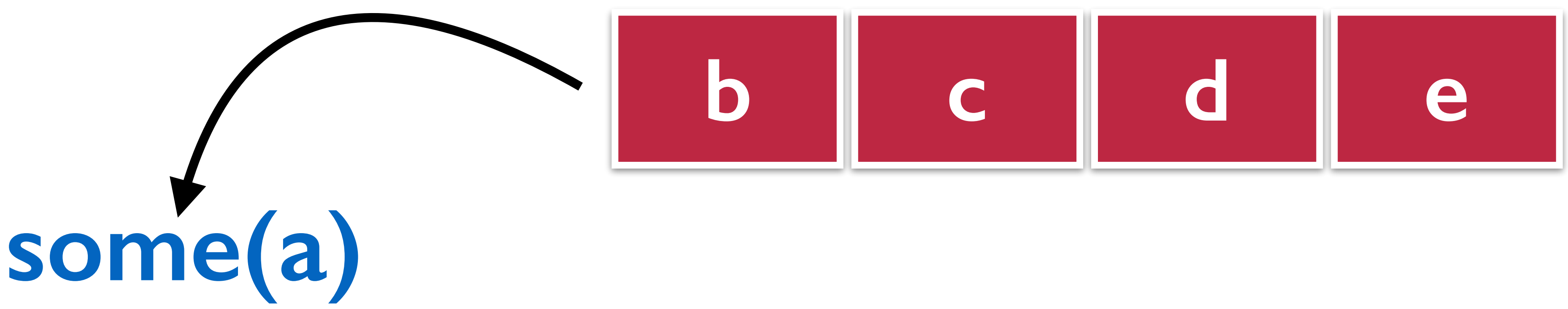
Concurrent Queues

Count the messages!



Concurrent Queues

Count the messages!



Concurrent Queues

Count the messages!

w_i = Work done to process insertion
= 2n (where n is the size of queue)
= ‘ins’ and ‘e’ travel to end of queue

w_d = Work done to process deletion
= 2 (sends back ‘some’ and ‘a’)

Type for Queues

$$\begin{aligned}
\mathsf{queue}_{\mathbf{A}}[\mathbf{n}] = \&\{&\mathsf{ins}^{\mathbf{2n}} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}}[\mathbf{n}+\mathbf{1}],\\
&\mathsf{del}^{\mathbf{2}} : \oplus\{\mathsf{none} : \mathbf{1},\\
&\qquad\qquad \mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}[\mathbf{n}-\mathbf{1}]\}\}
\end{aligned}$$

Type for Queues

$$\mathsf{queue}_{\mathbf{A}}[\mathbf{n}] = \&\{\mathsf{ins}^{\mathbf{2n}} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}}[\mathbf{n}+\mathbf{1}],$$
$$\mathsf{del}^{\mathbf{2}} : \oplus\{\mathsf{none} : \mathbf{1},$$
$$\mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}[\mathbf{n}-\mathbf{1}]\}\}$$

Resource-Aware Session Types

types augmented with potential information
sender pays potential with message
receiver gets potential with message

Type for Queues

$$\mathsf{queue}_{\mathbf{A}}[\mathbf{n}] = \&\{\mathsf{ins}^{\mathbf{2n}} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}}[\mathbf{n}+\mathbf{1}],$$
$$\mathsf{del}^{\mathbf{2}} : \oplus\{\mathsf{none} : \mathbf{1},$$
$$\mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}[\mathbf{n}-\mathbf{1}]\}\}$$

$$
\begin{array}{l}
(x : A)\ (t : \mathsf{queue}_A[n]) \vdash^{0} elem :: (s : \mathsf{queue}_A[n+1]) \\
\quad s \leftarrow elem \leftarrow x\ t = \\
\qquad \mathsf{case}\ s\ (\mathsf{ins} \Rightarrow y \leftarrow \mathsf{recv}\ s\ ; \\
\qquad\qquad\qquad\quad t.\mathsf{ins}\ ; \\
\qquad\qquad\qquad\quad \mathsf{send}\ t\ y\ ; \\
\qquad\qquad\qquad\quad s \leftarrow elem \leftarrow x\ t \\
\qquad\qquad \mid \mathsf{del} \Rightarrow s.\mathsf{some}\ ; \\
\qquad\qquad\qquad\quad \mathsf{send}\ s\ x\ ; \\
\qquad\qquad\qquad\quad s \leftarrow t)
\end{array}
$$

Type for Queues

$$\mathsf{queue}_{\mathbf{A}}[\mathbf{n}] = \&\{\mathsf{ins}^{\mathbf{2n}} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}}[\mathbf{n}+\mathbf{1}],$$
$$\mathsf{del}^{\mathbf{2}} : \oplus\{\mathsf{none} : \mathbf{1},$$
$$\mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}[\mathbf{n}-\mathbf{1}]\}\}$$

$$
\begin{array}{l}
(x : A)\ (t : \mathsf{queue}_A[n]) \vdash^{0} \mathit{elem} :: (s : \mathsf{queue}_A[n+1]) \\
\quad s \leftarrow \mathit{elem} \leftarrow x\ t = \\
\qquad \mathsf{case}\ s\ (\mathsf{ins} \Rightarrow y \leftarrow \mathsf{recv}\ s\ ; \\
\qquad\qquad\qquad t.\mathsf{ins}\ ; \\
\qquad\qquad\qquad \mathsf{send}\ t\ y\ ; \\
\qquad\qquad\qquad s \leftarrow \mathit{elem} \leftarrow x\ t \\
\qquad\qquad \mid \mathsf{del} \Rightarrow s.\mathsf{some}\ ; \\
\qquad\qquad\qquad \mathsf{send}\ s\ x\ ; \\
\qquad\qquad\qquad s \leftarrow t)
\end{array}
$$

recv 2(n+1) units

Type for Queues

$$\mathsf{queue}_{\mathbf{A}}[\mathbf{n}] = \&\{\mathsf{ins}^{\mathbf{2n}} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}}[\mathbf{n}+\mathbf{1}],$$
$$\mathsf{del}^{\mathbf{2}} : \oplus\{\mathsf{none} : \mathbf{1},$$
$$\mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}[\mathbf{n}-\mathbf{1}]\}\}$$

$$
\begin{array}{l}
(x : A)\ (t : \mathsf{queue}_A[n]) \vdash^{0} \mathit{elem} :: (s : \mathsf{queue}_A[n+1]) \\
\quad s \leftarrow \mathit{elem} \leftarrow x\ t = \\
\qquad \mathsf{case}\ s\ (\mathsf{ins} \Rightarrow y \leftarrow \mathsf{recv}\ s\ ; \\
\qquad\qquad\qquad\quad t.\mathsf{ins}\ ; \\
\qquad\qquad\qquad\quad \mathsf{send}\ t\ y\ ; \\
\qquad\qquad\qquad\quad s \leftarrow \mathit{elem} \leftarrow x\ t \\
\qquad\qquad\quad \mid \mathsf{del} \Rightarrow s.\mathsf{some}\ ; \\
\qquad\qquad\qquad\quad \mathsf{send}\ s\ x\ ; \\
\qquad\qquad\qquad\quad s \leftarrow t)
\end{array}
$$

recv 2(n+1) units

send 2n units

Type for Queues

$$\text{queue}_{\mathbf{A}}[\mathbf{n}] = \&\{\text{ins}^{\mathbf{2n}} : \mathbf{A} \multimap \text{queue}_{\mathbf{A}}[\mathbf{n}+\mathbf{1}],$$
$$\text{del}^{\mathbf{2}} : \oplus\{\text{none} : \mathbf{1},$$
$$\text{some} : \mathbf{A} \otimes \text{queue}_{\mathbf{A}}[\mathbf{n}-\mathbf{1}]\}\}$$

$$
\begin{array}{l}
(x : A)\ (t : \text{queue}_A[n]) \vdash^{0} \mathit{elem} :: (s : \text{queue}_A[n+1]) \\
\quad s \leftarrow \mathit{elem} \leftarrow x\ t = \\
\qquad \text{case } s\ (\text{ins} \Rightarrow y \leftarrow \text{recv } s\ ; \\
\qquad\qquad\qquad\quad t.\text{ins}\ ; \\
\qquad\qquad\qquad\quad \text{send } t\ y\ ; \\
\qquad\qquad\qquad\quad s \leftarrow \mathit{elem} \leftarrow x\ t \\
\qquad\qquad\quad \mid \text{del} \Rightarrow s.\text{some}\ ; \\
\qquad\qquad\qquad\quad \text{send } s\ x\ ; \\
\qquad\qquad\qquad\quad s \leftarrow t)
\end{array}
$$

recv 2(n+1) units

send 2n units

cost of 2

Type for Queues

$$
\begin{aligned}
\mathsf{queue}_{\mathbf{A}}[\mathbf{n}] = \&\{\mathsf{ins}^{\mathbf{2n}} &: \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}}[\mathbf{n}+\mathbf{1}], \\
\mathsf{del}^{\mathbf{2}} &: \oplus\{\mathsf{none} : \mathbf{1}, \\
&\qquad \mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}[\mathbf{n}-\mathbf{1}]\}\}
\end{aligned}
$$

$$
\begin{array}{l}
(x : A)\ (t : \mathsf{queue}_A[n]) \vdash^{0} \mathit{elem} :: (s : \mathsf{queue}_A[n+1]) \\
\quad s \leftarrow \mathit{elem} \leftarrow x\ t = \\
\qquad \mathsf{case}\ s\ (\mathsf{ins} \Rightarrow y \leftarrow \mathsf{recv}\ s\ ; \\
\qquad\qquad\qquad\qquad t.\mathsf{ins}\ ; \\
\qquad\qquad\qquad\qquad \mathsf{send}\ t\ y\ ; \\
\qquad\qquad\qquad\qquad s \leftarrow \mathit{elem} \leftarrow x\ t \\
\qquad\qquad\quad \mid \mathsf{del} \Rightarrow s.\mathsf{some}\ ; \\
\qquad\qquad\qquad\qquad \mathsf{send}\ s\ x\ ; \\
\qquad\qquad\qquad\qquad s \leftarrow t)
\end{array}
$$

recv 2(n+1) units

send 2n units

cost of 2

recv 2 units

Type for Queues

$$\mathsf{queue}_{\mathbf{A}}[\mathbf{n}] = \&\{\mathsf{ins}^{\mathbf{2n}} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}}[\mathbf{n}+\mathbf{1}],$$
$$\mathsf{del}^{\mathbf{2}} : \oplus\{\mathsf{none} : \mathbf{1},$$
$$\mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}[\mathbf{n}-\mathbf{1}]\}\}$$

$$
\begin{array}{l}
(x : A)\ (t : \mathsf{queue}_A[n]) \vdash^{0} \mathit{elem} :: (s : \mathsf{queue}_A[n+1]) \\
\quad s \leftarrow \mathit{elem} \leftarrow x\ t = \\
\qquad \mathsf{case}\ s\ (\mathsf{ins} \Rightarrow y \leftarrow \mathsf{recv}\ s\ ; \\
\qquad\qquad t.\mathsf{ins}\ ; \\
\qquad\qquad \mathsf{send}\ t\ y\ ; \\
\qquad\qquad s \leftarrow \mathit{elem} \leftarrow x\ t \\
\qquad\quad \mid \mathsf{del} \Rightarrow s.\mathsf{some}\ ; \\
\qquad\qquad \mathsf{send}\ s\ x\ ; \\
\qquad\qquad s \leftarrow t)
\end{array}
$$

recv 2(n+1) units

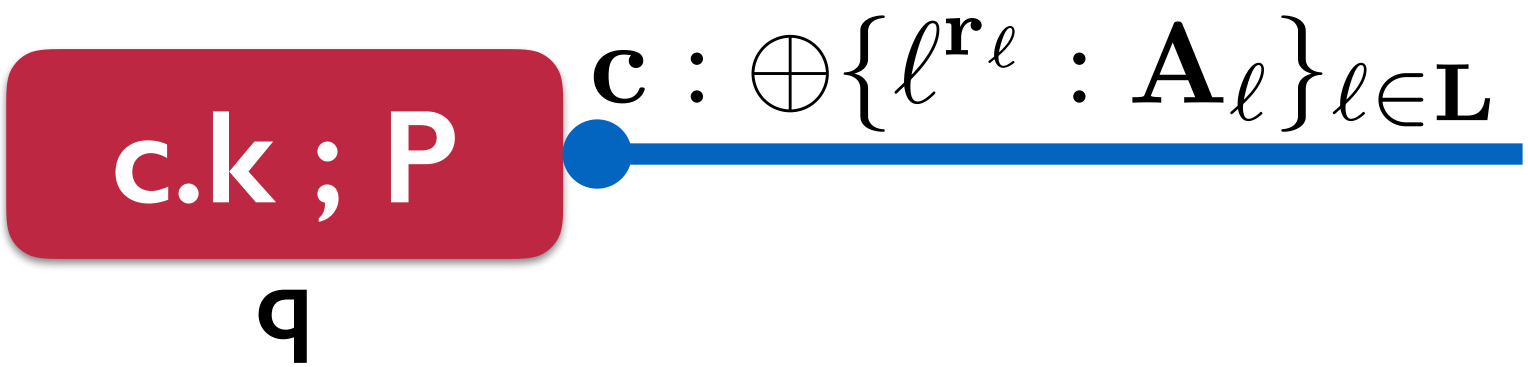
send 2n units

cost of 2

recv 2 units

cost of 2

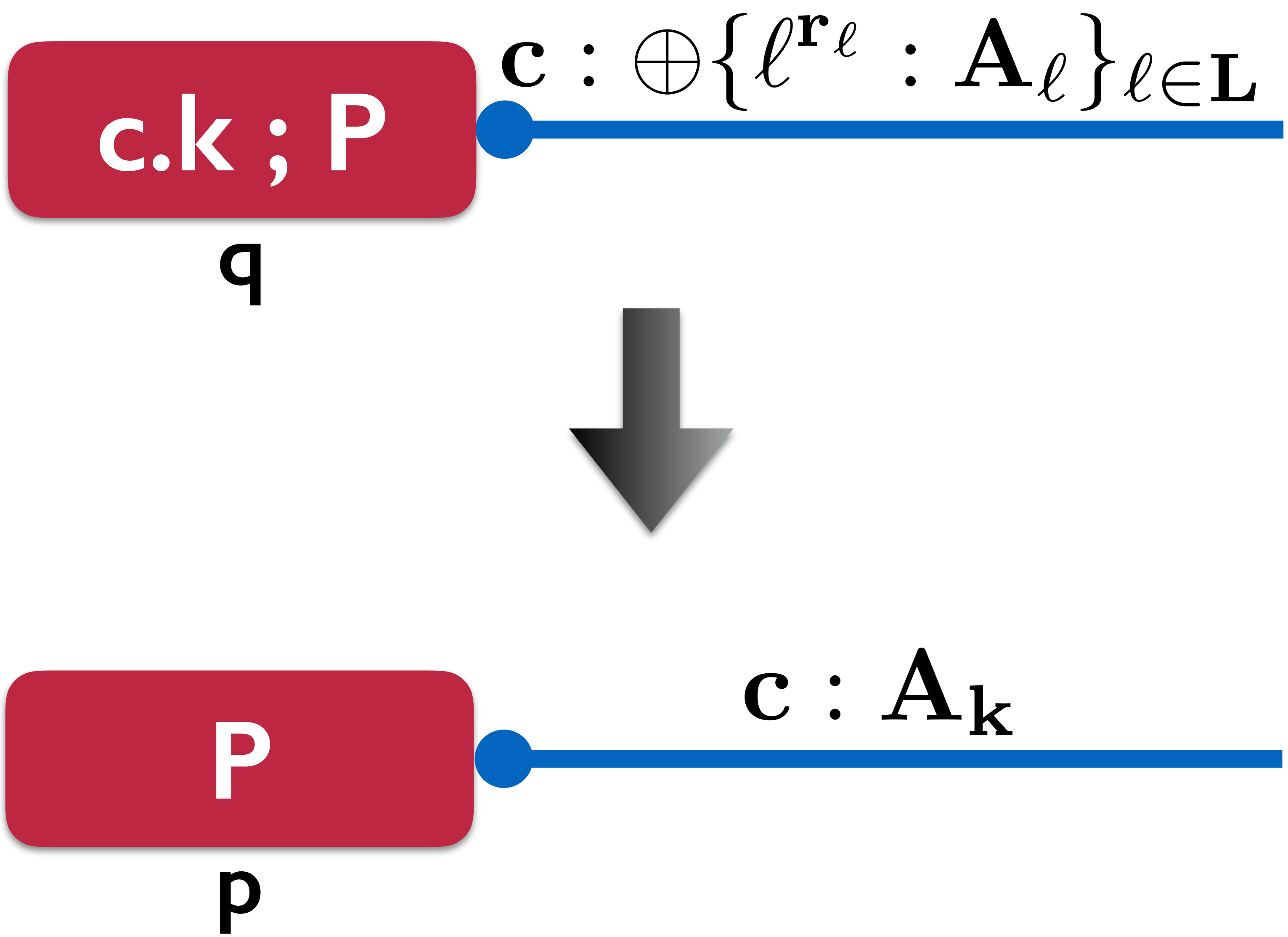
Rule: Sending a label



$\mathbf{\Omega} \models^{\mathbf{q}} \mathbf{P} :: (\mathbf{c} : \mathbf{A})$

Process P offers along channel c, acts as a client for channels in Ω and storing potential q

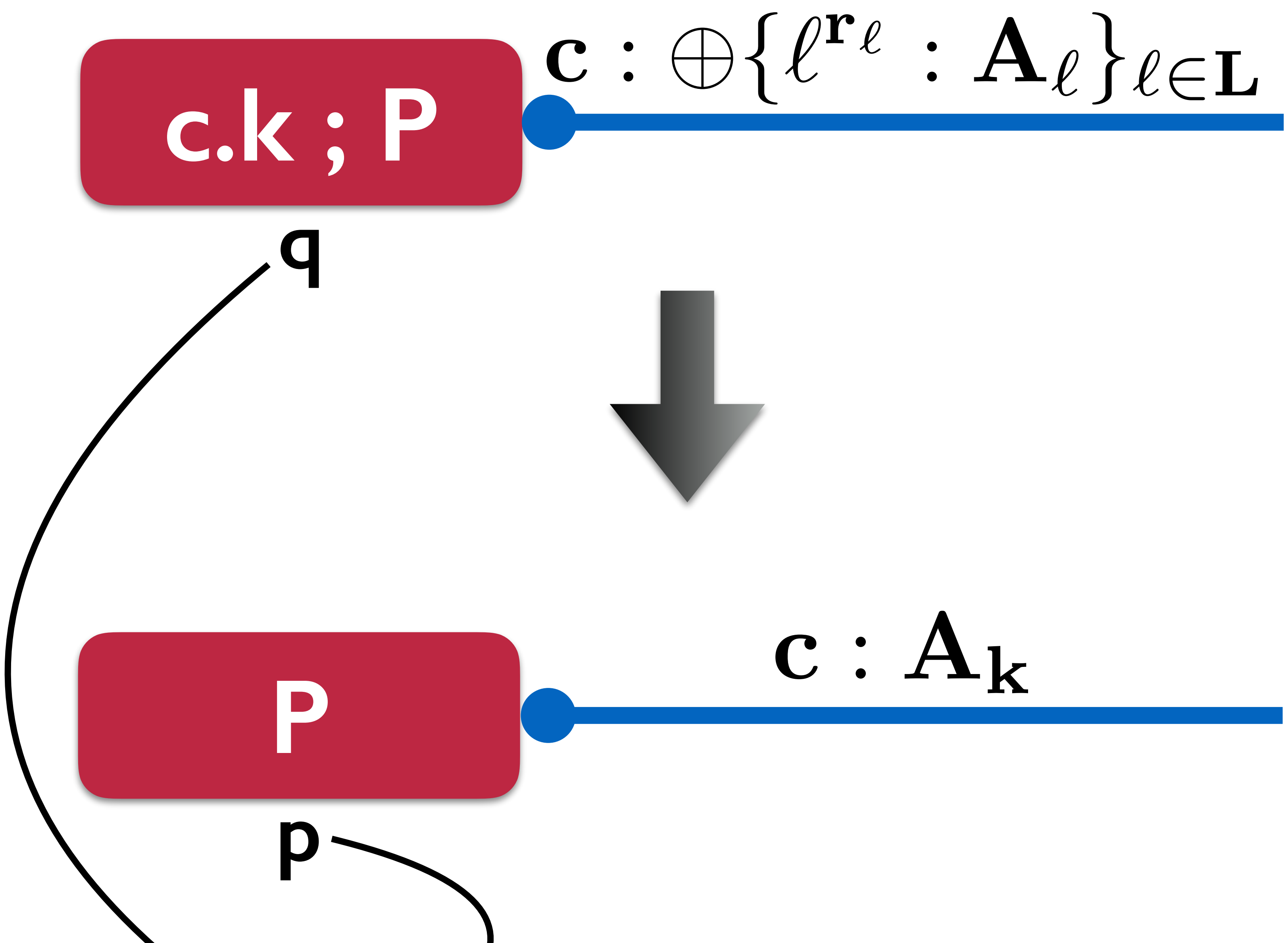
Rule: Sending a label



$\mathbf{\Omega} \vDash^{\mathbf{q}} \mathbf{P} :: (\mathbf{c} : \mathbf{A})$

Process P offers along channel c, acts as a client for channels in Ω and storing potential q

Rule: Sending a label



$\mathbf{\Omega} \overset{\mathbf{q}}{\vDash} \mathbf{P} :: (\mathbf{c} : \mathbf{A})$

Process P offers along channel c, acts as a client for channels in Ω and storing potential q

$$\frac{\mathbf{q} \geq \mathbf{p} + \mathbf{r_k} + 1 \qquad \mathbf{\Omega} \overset{\mathbf{p}}{\vDash} \mathbf{P} :: (\mathbf{c} : \mathbf{A_k})}{\mathbf{\Omega} \overset{\mathbf{q}}{\vDash} \mathbf{c.k} :: (\mathbf{c} : \oplus\{\ell^{\mathbf{r}_\ell} : \mathbf{A}_\ell\}_{\ell \in \mathbf{L}})}$$

Features of Type System

- **Flexible:** supports counting of different resources (e.g. messages exchanged, processes spawned, etc.) by being parametric in cost model
- **Compositional:** types describe individual processes, not just whole programs
- **Precise:** potential upper bounds work accurately
- **Conservative:** strict extension of type system
- **General:** works on most standard examples
- **Automatic:** future work

Examples!

Binary Counters

$$\mathsf{ctr}[\mathbf{n}] = \&\{\mathsf{inc}^{\mathbf{1}} : \mathsf{ctr}[\mathbf{n}+\mathbf{1}],$$
$$\mathsf{val}^{\mathbf{2}\lceil\mathbf{log}(\mathbf{n}+\mathbf{1})\rceil+\mathbf{2}} : \mathsf{bits}\}$$

- **Increment:**
  - **requires one unit of potential**
  - **uses amortized analysis!**
- **Value:**
  - **requires logarithmic potential**
  - **precise work bound**

Stacks vs Queues

$$\begin{aligned}\mathsf{stack}_{\mathbf{A}}[\mathbf{n}] = \&\{&\mathsf{ins}^{\mathbf{0}} : \mathbf{A} \multimap \mathsf{stack}_{\mathbf{A}}[\mathbf{n}+\mathbf{1}],\\ &\mathsf{del}^{\mathbf{2}} : \oplus\{\mathsf{none} : \mathbf{1},\\ &\qquad\qquad \mathsf{some} : \mathbf{A} \otimes \mathsf{stack}_{\mathbf{A}}[\mathbf{n}-\mathbf{1}]\}\}\end{aligned}$$

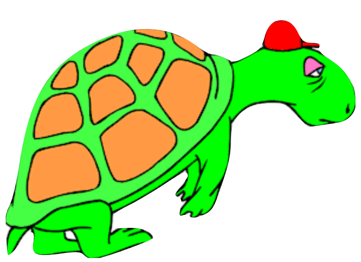
$$\begin{aligned}\mathsf{queue}_{\mathbf{A}}[\mathbf{n}] = \&\{&\mathsf{ins}^{\mathbf{2n}} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}}[\mathbf{n}+\mathbf{1}],\\ &\mathsf{del}^{\mathbf{2}} : \oplus\{\mathsf{none} : \mathbf{1},\\ &\qquad\qquad \mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}[\mathbf{n}-\mathbf{1}]\}\}\end{aligned}$$

Which one's more efficient?

Stacks vs Queues

$$\mathsf{stack}_{\mathbf{A}}[\mathbf{n}] = \&\{\mathsf{ins}^{\mathbf{0}} : \mathbf{A} \multimap \mathsf{stack}_{\mathbf{A}}[\mathbf{n}+\mathbf{1}],$$
$$\mathsf{del}^{\mathbf{2}} : \oplus\{\mathsf{none} : \mathbf{1},$$
$$\mathsf{some} : \mathbf{A} \otimes \mathsf{stack}_{\mathbf{A}}[\mathbf{n}-\mathbf{1}]\}\}$$

$$\mathsf{queue}_{\mathbf{A}}[\mathbf{n}] = \&\{\mathsf{ins}^{\mathbf{2n}} : \mathbf{A} \multimap \mathsf{queue}_{\mathbf{A}}[\mathbf{n}+\mathbf{1}],$$
$$\mathsf{del}^{\mathbf{2}} : \oplus\{\mathsf{none} : \mathbf{1},$$
$$\mathsf{some} : \mathbf{A} \otimes \mathsf{queue}_{\mathbf{A}}[\mathbf{n}-\mathbf{1}]\}\}$$



Which one's more efficient?

Contributions

Type System for Work Analysis

based on amortized analysis

types augmented with potential information

work is upper bounded by potential

Flexible, Compositional, Precise, Conservative, General, Automatic

Contributions

Type System for Work Analysis

based on amortized analysis

types augmented with potential information

work is upper bounded by potential

Flexible, Compositional, Precise, Conservative, General, Automatic

Soundness Theorem

Cost Semantics

Contributions

Type System for Work Analysis

based on amortized analysis

types augmented with potential information

work is upper bounded by potential

Flexible, Compositional, Precise, Conservative, General, Automatic

Soundness Theorem

Cost Semantics

Examples

stacks, queues, binary counters

efficiency comparison

list examples: append, map, fold