

Pretend Synchrony

Synchronous Verification of Asynchronous Distributed Programs

Pretend Synchrony

Synchronous Verification of Asynchronous Distributed Programs

How does it work?



Verifying Distributed Programs via Canonical Sequentialization

ALEXANDER BAKST, University of California, San Diego

KLAUS V. GLEISSENTHALL, University of California, San Diego

RAMI GÖKHAN KICI, University of California, San Diego

RANJIT JHALA, University of California, San Diego

We introduce canonical sequentialization, a new approach to verifying unbounded, asynchronous, message-passing programs at compile-time. Our approach builds upon the following observation: due the combinatorial explosion in complexity, programmers do not reason about their systems by case-splitting over all the possible execution orders. Instead, correct programs tend to be well-structured so that the programmer can reason about a small number of representative executions, which we call the program's *canonical sequentialization*. We have implemented our approach in a tool called BRISK that synthesizes canonical sequentializations for programs written in HASKELL, and evaluated it on a wide variety of distributed systems including benchmarks from the literature and implementations of MapReduce, two-phase commit, and a version of the Disco distributed file-system. BRISK verifies *unbounded* versions of the benchmarks in tens of *milliseconds*, yielding the first concurrency verification tool that is fast enough to be integrated into a design-implement-check cycle.



Pretend Synchrony*

Synchronous Verification of Asynchronous Distributed Programs

KLAUS V. GLEISSENTHALL, University of California, San Diego, USA

RAMI GÖKHAN KICI, University of California, San Diego, USA

ALEXANDER BAKST, University of California, San Diego, USA

DEIAN STEFAN, University of California, San Diego, USA

RANJIT JHALA, University of California, San Diego, USA

We present *pretend synchrony*, a new approach to verifying distributed systems, based on the observation that while distributed programs must execute asynchronously, we can often soundly treat them as if they were synchronous when verifying their correctness. To do so, we compute a *synchronization*, a semantically equivalent program where all sends, receives, and message buffers, have been replaced by simple assignments, yielding a program that can be verified using Floyd-Hoare style Verification Conditions and SMT. We implement our approach as a framework for writing verified distributed programs in Go and evaluate it with four challenging case studies— the classic two-phase commit, the Raft leader election protocol, single-decree Paxos protocol, and a Multi-Paxos based distributed key-value store. We find that pretend synchrony allows us to develop performant systems while making verification of functional correctness *simpler* by reducing manually specified invariants by a factor of 6, and *faster*, by reducing checking time by three orders of magnitude.

Rewrite the implementation to match programmer's mental model.

... and verify **that!**



Graydon Hoare
@graydon_pub



"Figure out how programmers are currently reasoning, formalize as language restricting validity to that model, profit!" is A+ research plan

4:03 AM · Sep 13, 2017 · TweetDeck

7 Retweets **1** Quote Tweet **33** Likes



Graydon Hoare

@graydon_pub



"Figure out how programmers are currently reasoning, formalize as language restricting validity to that model, profit!" is A+ research plan

4:03 AM · Sep 13, 2017 · TweetDeck

7 Retweets 1 Quote Tweet 33 Likes



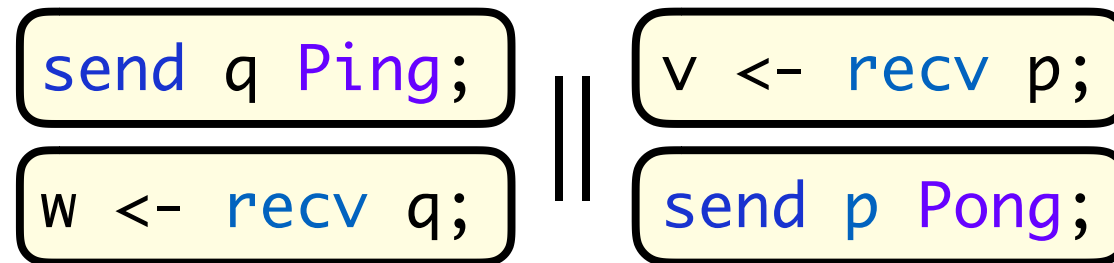
Graydon Hoare @graydon_pub · Sep 13, 2017



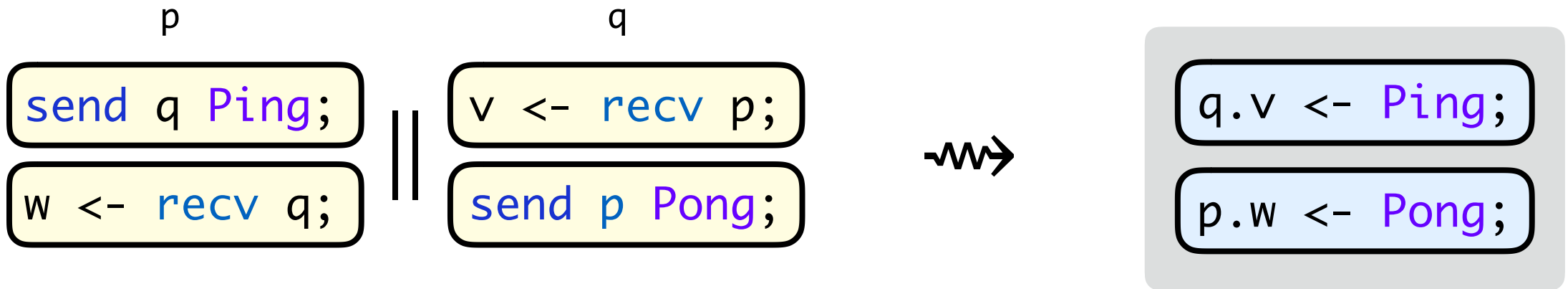
Yeah it's normal in many contexts, I'm just applauding a particularly pleasing example of it (that Canonical Sequentializatoon paper)



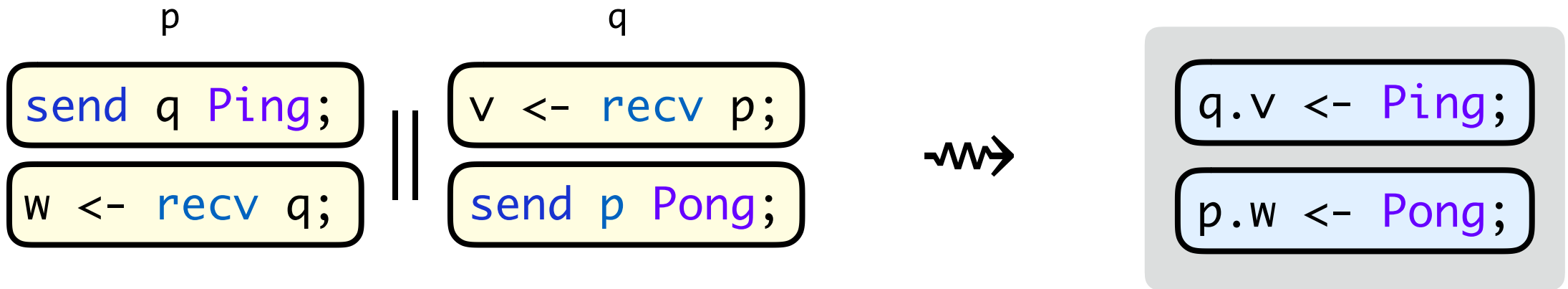
In many systems, *asynchrony* is **inconsequential** if we are concerned only with properties of halting states.



In many systems, *asynchrony* is **inconsequential** if we are concerned only with properties of halting states.



In many systems, *asynchrony* is **inconsequential** if we are concerned only with properties of halting states.



We can afford not to reason about *asynchrony* if we can guarantee that all schedules terminate in equivalent states.

When asynchrony is inconsequential

- processes block on receiving messages → execution is sequential
 - if every (logical) **send**
 - has exactly one matching (logical) **receive** (*reduction*)
 - one-to-one communication
 - **OR** has multiple matching **receives**, but they differ only in process IDs (*symmetry*)
 - one-to-many communication
- processes do not communicate → execution is parallel
(*almost symmetry* = processes that do not communicate + one-to-many sends)

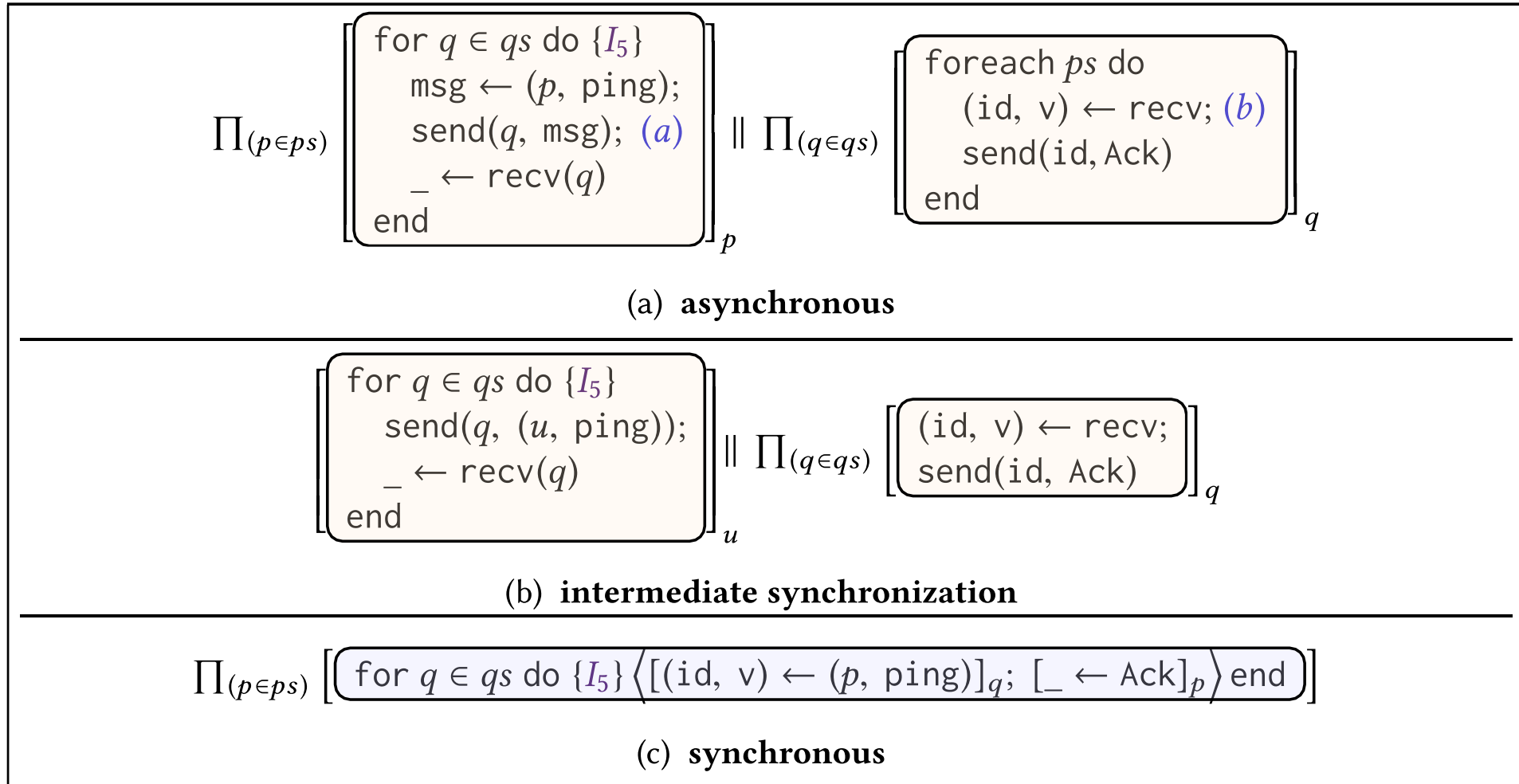


Fig. 6. Ex3 ping multi-cast.

When asynchrony is inconsequential

- processes block on receiving messages → execution is sequential
 - sequential sub-schedules = can be combined into one sequential schedule
- processes do not communicate → execution is parallel
 - non-interfering sub-schedules = every interleaving is equivalent

When asynchrony is inconsequential

- processes block on receiving messages → execution is sequential
→ sequential sub-schedules = can be combined into one sequential schedule
- processes do not communicate → execution is parallel
→ non-interfering sub-schedules = every interleaving is equivalent

synchronization

=

parallel composition of non-interfering sequential processes

(with no sends/receives; only variable reads/writes)

synchronization

=

parallel composition of non-interfering sequential processes

=

easy to automatically verify

“using Floyd-Hoare style Verification Conditions and SMT”

synchronization

=

parallel composition of non-interfering sequential processes

=

easy to automatically verify

“using Floyd-Hoare style Verification Conditions and SMT”

Programmer only needs to provide loop invariants.

parallel composition of non-interfering sequential processes

R-CONTEXT

$$\Gamma, \Delta, \Sigma, A \rightsquigarrow \Gamma', \Delta', \Sigma', A'$$

$$\Gamma, \Delta, \Sigma, A \circ B \rightsquigarrow \Gamma', \Delta', \Sigma', A' \circ B$$

program

parallel composition of non-interfering sequential processes

$\text{MSGs} \in \text{ID} \times \text{ID} \times \text{MsgType} \times \text{ROUND} \times \text{Exp} \rightarrow \text{INT}$

$\text{ASRT} ::= \emptyset \mid \text{ASRT} \cup \{\text{unfold}(p, x, ps)\}$

$\Gamma ::= (\text{MSGs}, \text{ASRT})$

R-CONTEXT

$$\Gamma, \Delta, \Sigma, A \rightsquigarrow \Gamma', \Delta', \Sigma', A'$$

$$\Gamma, \Delta, \Sigma, A \circ B \rightsquigarrow \Gamma', \Delta', \Sigma', A' \circ B$$

program

parallel composition of non-interfering sequential processes

$\text{MSGs} \in \text{ID} \times \text{ID} \times \text{MsgType} \times \text{ROUND} \times \text{Exp} \rightarrow \text{INT}$

$\text{ASRT} ::= \emptyset \mid \text{ASRT} \cup \{\text{unfold}(p, x, ps)\}$

$\Gamma ::= (\text{MSGs}, \text{ASRT})$

R-CONTEXT

$$\Gamma, \Delta, \Sigma, A \rightsquigarrow \Gamma', \Delta', \Sigma', A'$$

$$\Gamma, \Delta, \Sigma, A \circ B \rightsquigarrow \Gamma', \Delta', \Sigma', A' \circ B$$

program

parallel composition of non-interfering sequential processes

$\text{MSGS} \in \text{ID} \times \text{ID} \times \text{MsgType} \times \text{ROUND} \times \text{Exp} \rightarrow \text{INT}$

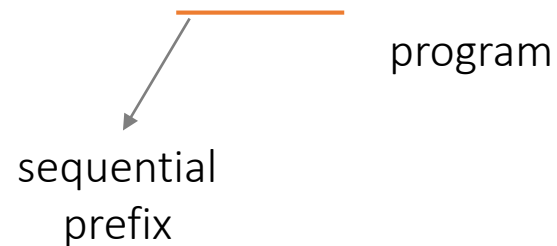
$\text{ASRT} ::= \emptyset \mid \text{ASRT} \cup \{\text{unfold}(p, x, ps)\}$

$\Gamma ::= (\text{MSGS}, \text{ASRT})$

R-CONTEXT

$$\Gamma, \Delta, \Sigma, A \rightsquigarrow \Gamma', \Delta', \Sigma', A'$$

$$\Gamma, \Delta, \Sigma, A \circ B \rightsquigarrow \Gamma', \Delta', \Sigma', A' \circ B$$

program

sequential
prefix

parallel composition of non-interfering sequential processes

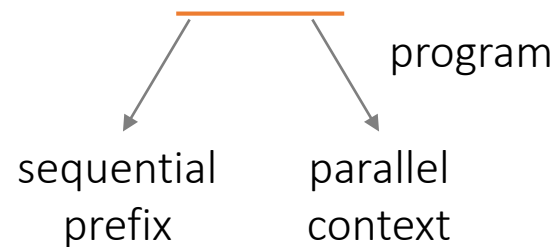
$\text{MSGS} \in \text{ID} \times \text{ID} \times \text{MsgType} \times \text{ROUND} \times \text{Exp} \rightarrow \text{INT}$

$\text{ASRT} ::= \emptyset \mid \text{ASRT} \cup \{\text{unfold}(p, x, ps)\}$

$\Gamma ::= (\text{MSGS}, \text{ASRT})$

R-CONTEXT

$$\Gamma, \Delta, \Sigma, A \rightsquigarrow \Gamma', \Delta', \Sigma', A'$$

$$\Gamma, \Delta, \Sigma, A \circ B \rightsquigarrow \Gamma', \Delta', \Sigma', A' \circ B$$


parallel composition of non-interfering sequential processes

$\text{MSGs} \in \text{ID} \times \text{ID} \times \text{MsgType} \times \text{ROUND} \times \text{Exp} \rightarrow \text{INT}$

$\text{ASRT} ::= \emptyset \mid \text{ASRT} \cup \{\text{unfold}(p, x, ps)\}$

$\Gamma ::= (\text{MSGs}, \text{ASRT})$

R-CONTEXT

$$\Gamma, \Delta, \Sigma, A \rightsquigarrow \Gamma', \Delta', \Sigma', A'$$

$$\Gamma, \Delta, \Sigma, A \circ B \rightsquigarrow \Gamma', \Delta', \Sigma', A' \circ B$$

program

parallel composition of non-interfering sequential processes

$\text{MSGs} \in \text{ID} \times \text{ID} \times \text{MsgType} \times \text{ROUND} \times \text{Exp} \rightarrow \text{INT}$

$\text{ASRT} ::= \emptyset \mid \text{ASRT} \cup \{\text{unfold}(p, x, ps)\}$

$\Gamma ::= (\text{MSGs}, \text{ASRT})$

R-CONTEXT

$$\Gamma, \Delta, \Sigma, A \rightsquigarrow \Gamma', \Delta', \Sigma', A'$$

$$\Gamma, \Delta, \Sigma, \underbrace{A \circ B}_{\text{program}} \rightsquigarrow \Gamma', \Delta', \Sigma', A' \circ B$$

parallel composition of non-interfering sequential processes

$\text{MSGs} \in \text{ID} \times \text{ID} \times \text{MsgType} \times \text{ROUND} \times \text{Exp} \rightarrow \text{INT}$

$\text{ASRT} ::= \emptyset \mid \text{ASRT} \cup \{\text{unfold}(p, x, ps)\}$

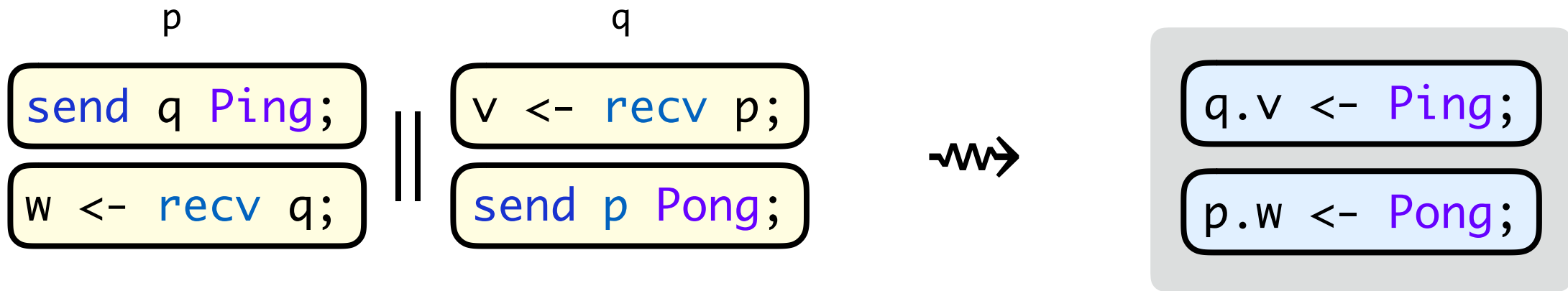
$\Gamma ::= (\text{MSGs}, \text{ASRT})$

R-CONTEXT

$$\Gamma, \Delta, \Sigma, A \rightsquigarrow \Gamma', \Delta', \Sigma', A'$$

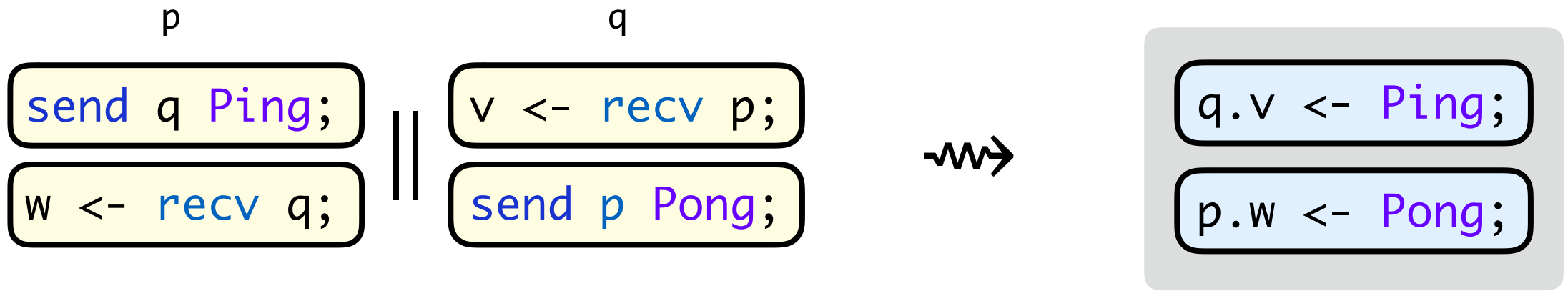
$$\Gamma, \Delta, \Sigma, A \circ B \rightsquigarrow \Gamma', \Delta', \Sigma', A' \circ B$$

program



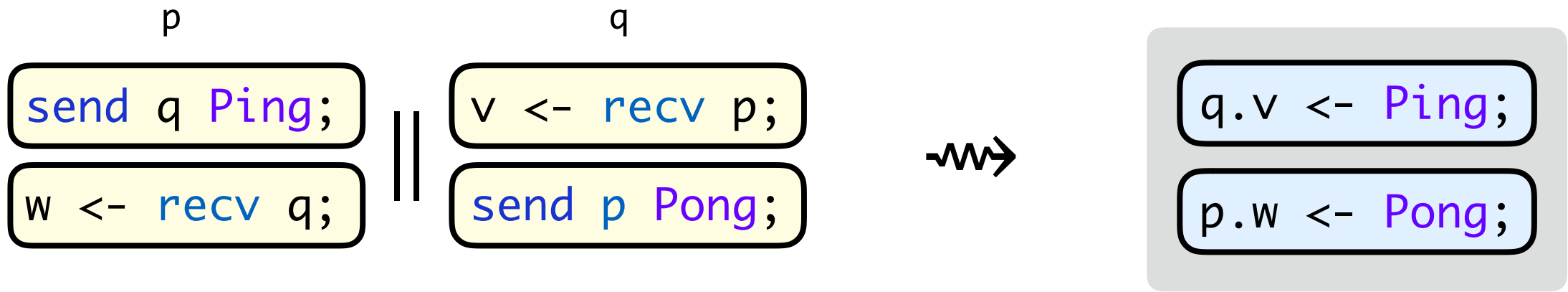
R-SEND

$$\frac{\Delta, \Sigma \models x = q \quad m \text{ fresh} \quad q \text{ is a PID} \quad \Gamma' = \Gamma \cup \{(p, q, t, r, m)\}}{\Gamma, \Delta, \Sigma, [\text{send}(x, n, t, r)]_p \rightsquigarrow \Gamma', (\Delta; [m \leftarrow n]_p), \Sigma, \text{skip}}$$



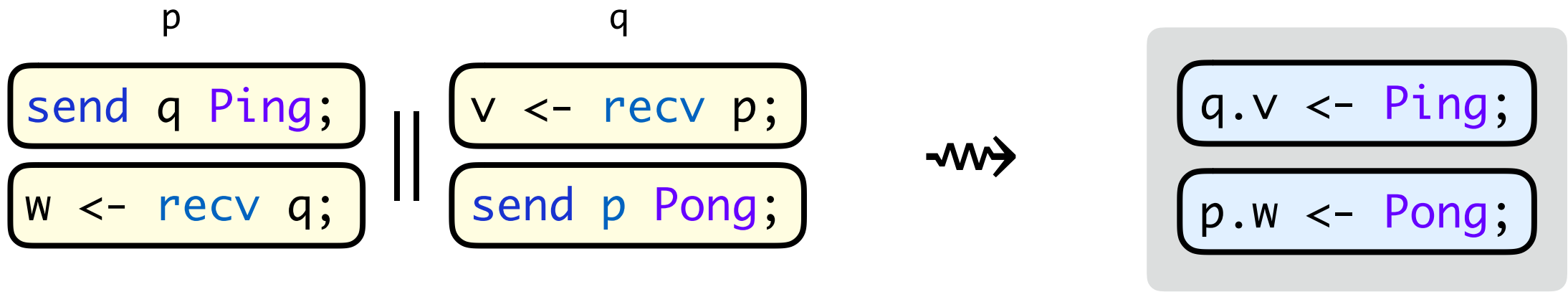
$$\begin{array}{c}
 \text{R-SEND} \\
 \frac{\Delta, \Sigma \models x = q \quad m \text{ fresh} \quad q \text{ is a PID} \quad \Gamma' = \Gamma \cup \{(p, q, t, r, m)\}}{\Gamma, \Delta, \Sigma, [\text{send}(x, n, t, r)]_p \rightsquigarrow \Gamma', (\Delta; [m \leftarrow n]_p), \Sigma, \text{skip}}
 \end{array}$$

\swarrow
 destination



R-SEND

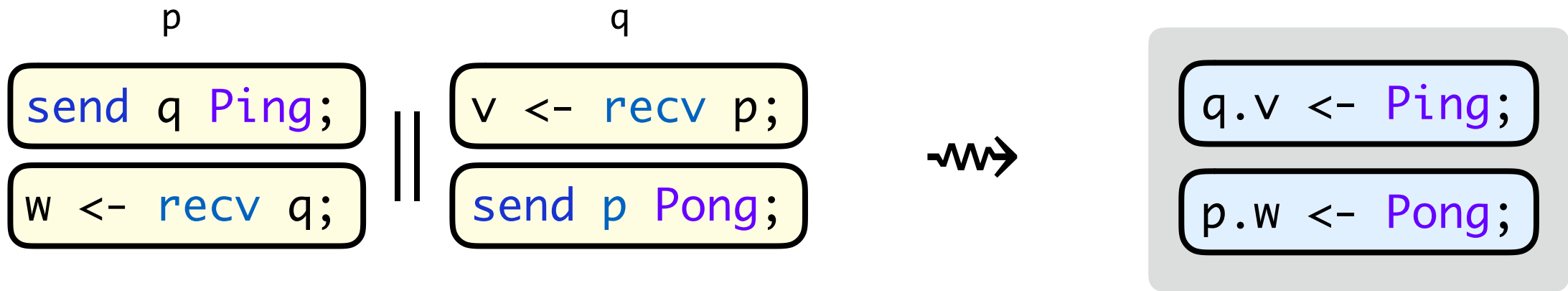
$$\frac{\begin{array}{l} \Delta, \Sigma \models x = q \\ m \text{ fresh} \end{array} \quad \begin{array}{l} q \text{ is a PID} \\ \Gamma' = \Gamma \cup \{(p, q, t, r, m)\} \end{array}}{\Gamma, \Delta, \Sigma, [\text{send}(x, n, t, r)]_p \rightsquigarrow \Gamma', (\Delta; [m \leftarrow n]_p), \Sigma, \text{skip}}$$



$$\begin{array}{c}
 \text{R-SEND} \\
 \frac{\Delta, \Sigma \models x = q \quad m \text{ fresh} \quad q \text{ is a PID} \quad \Gamma' = \Gamma \cup \{(p, q, t, r, m)\}}{\Gamma, \Delta, \Sigma, [\text{send}(x, n, t, r)]_p \rightsquigarrow \Gamma', (\Delta; [m \leftarrow n]_p), \Sigma, \text{skip}}
 \end{array}$$

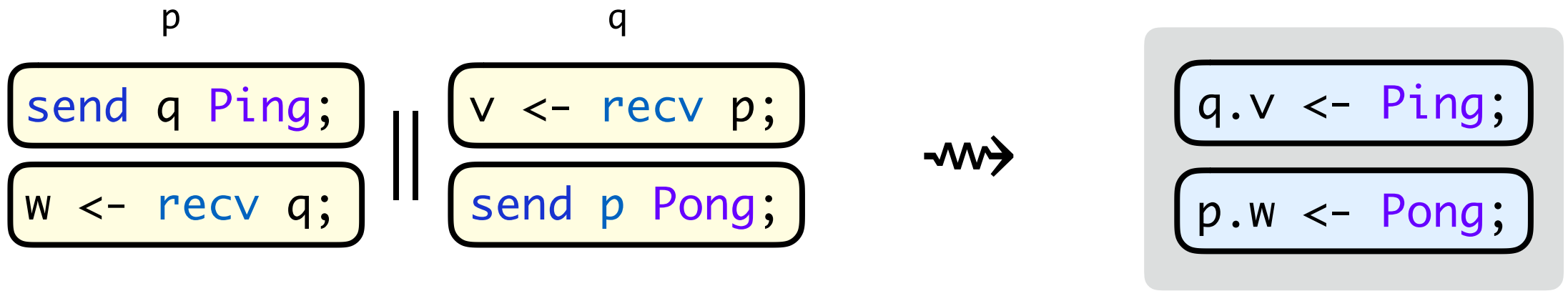
↓

message
value



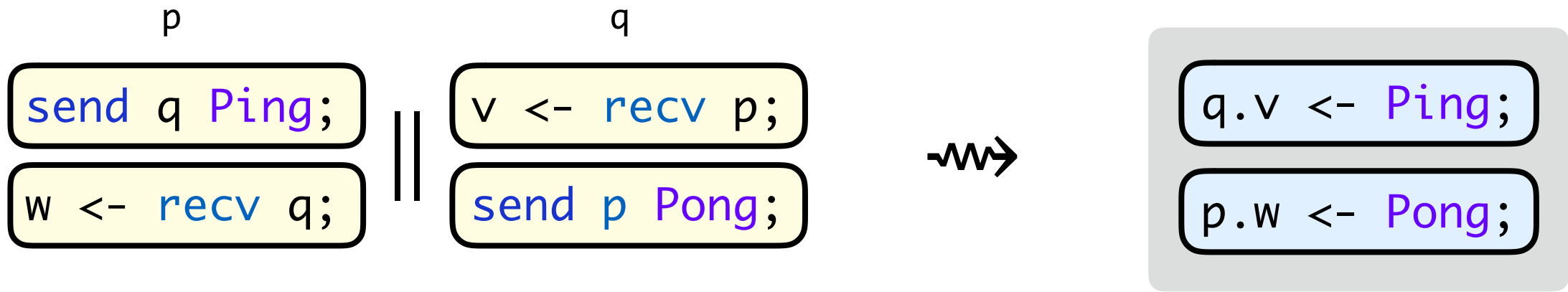
R-SEND

$$\frac{\Delta, \Sigma \models x = q \quad m \text{ fresh} \quad q \text{ is a PID} \quad \Gamma' = \Gamma \cup \{(p, q, t, r, m)\}}{\Gamma, \Delta, \Sigma, [\text{send}(x, n, t, r)]_p \rightsquigarrow \Gamma', (\Delta; [m \leftarrow n]_p), \Sigma, \text{skip}}$$



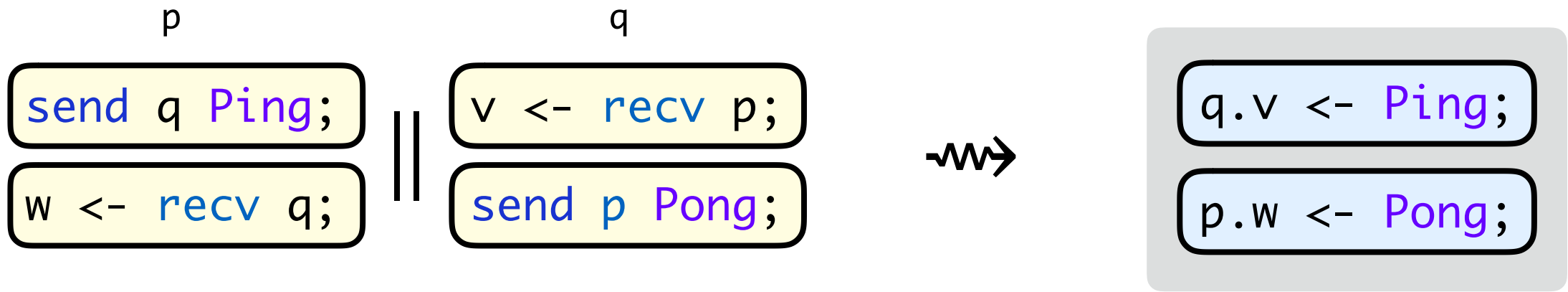
$$\begin{array}{c}
 \text{R-SEND} \\
 \frac{\Delta, \Sigma \models x = q \quad m \text{ fresh} \quad q \text{ is a PID} \quad \Gamma' = \Gamma \cup \{(p, q, t, r, m)\}}{\Gamma, \Delta, \Sigma, [\text{send}(x, n, t, r)]_p \rightsquigarrow \Gamma', (\Delta; [m \leftarrow n]_p), \Sigma, \text{skip}}
 \end{array}$$

message
 type



R-SEND

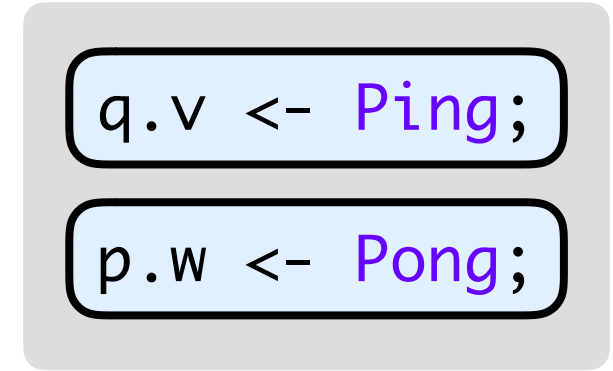
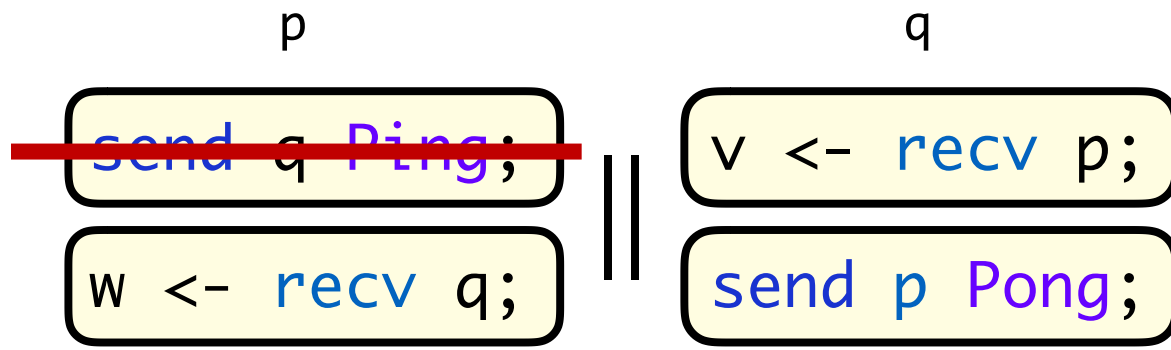
$$\frac{\Delta, \Sigma \models x = q \quad m \text{ fresh} \quad q \text{ is a PID} \quad \Gamma' = \Gamma \cup \{(p, q, t, r, m)\}}{\Gamma, \Delta, \Sigma, [\text{send}(x, n, t, r)]_p \rightsquigarrow \Gamma', (\Delta; [m \leftarrow n]_p), \Sigma, \text{skip}}$$



R-SEND

$$\frac{\begin{array}{l} \Delta, \Sigma \models x = q \\ m \text{ fresh} \end{array} \quad \begin{array}{l} q \text{ is a PID} \\ \Gamma' = \Gamma \cup \{(p, q, t, r, m)\} \end{array}}{\Gamma, \Delta, \Sigma, [\text{send}(x, n, t, r)]_p \rightsquigarrow \Gamma', (\Delta; [m \leftarrow n]_p), \Sigma, \text{skip}}$$

message is symbolic
(it contains variable name, not value)



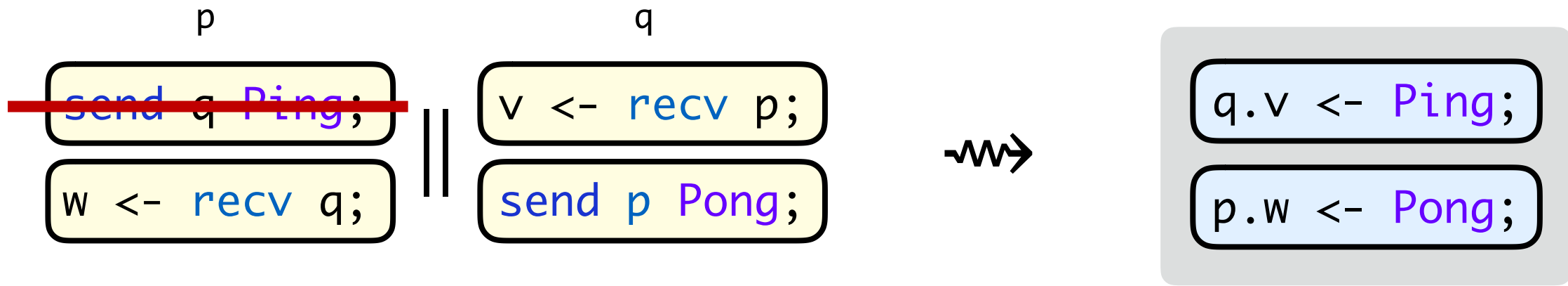
$\Gamma: (p, q, \text{Ping}, _, _m_0)$
 $[_m_0 \leftarrow \text{Ping}]_p$

R-SEND

$\Delta, \Sigma \models x = q$
 m fresh

q is a PID
 $\Gamma' = \Gamma \cup \{(p, q, t, r, m)\}$

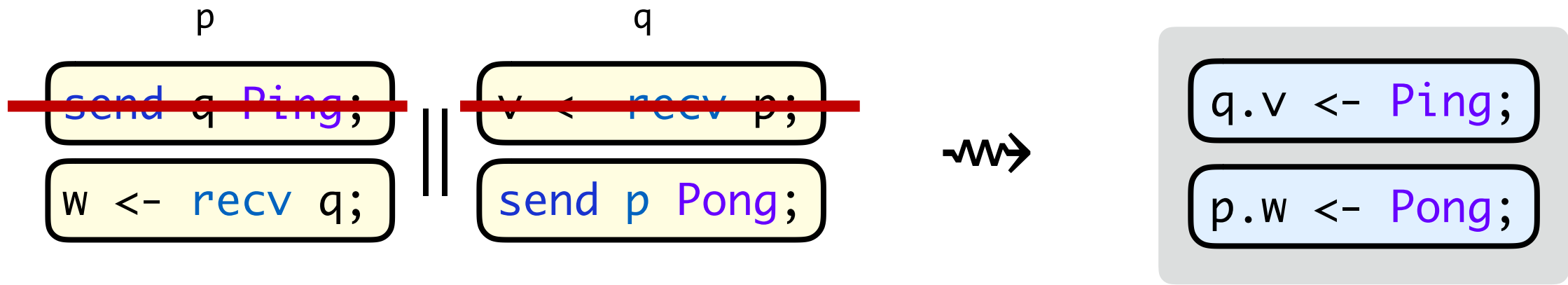
$\Gamma, \Delta, \Sigma, [\text{send}(x, n, t, r)]_p \rightsquigarrow \Gamma', (\Delta; [m \leftarrow n]_p), \Sigma, \text{skip}$



$\Gamma: (p, q, \text{Ping}, _, _m_0)$
 $[_m_0 \leftarrow \text{Ping}]_p$
 $[v \leftarrow p._m_0]_q$

R-RECV

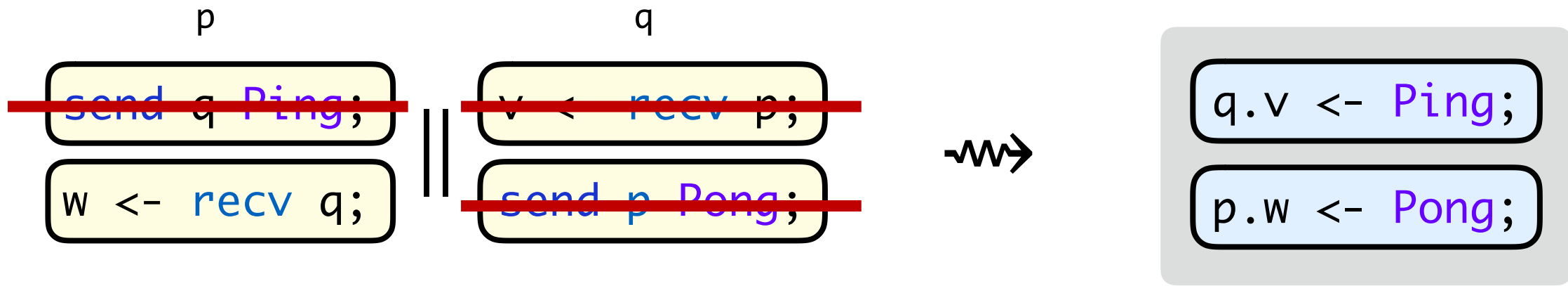
$$\frac{\Delta, \Sigma \models x = p \quad p \text{ is a PID} \quad (p, q, t, r, m) \in \Gamma \quad \Gamma' = \Gamma - \{(p, q, t, r, m)\}}{\Gamma, \Delta, \Sigma, [y \leftarrow \text{recv}(t, x, r)]_q \rightsquigarrow \Gamma', (\Delta; [y \leftarrow p.m]_q), \Sigma, \text{skip}}$$



$[_m_0 \leftarrow \text{Ping}]_p$
 $[v \leftarrow p._m0]_q$

R-RECV

$$\frac{\Delta, \Sigma \models x = p \quad p \text{ is a PID} \quad (p, q, t, r, m) \in \Gamma \quad \Gamma' = \Gamma - \{(p, q, t, r, m)\}}{\Gamma, \Delta, \Sigma, [y \leftarrow \text{recv}(t, x, r)]_q \rightsquigarrow \Gamma', (\Delta; [y \leftarrow p.m]_q), \Sigma, \text{skip}}$$



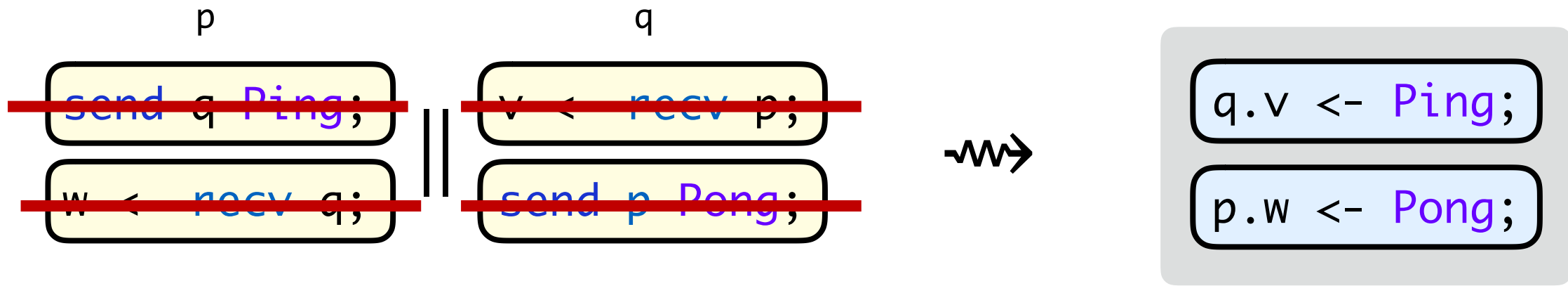
$\Gamma: (q, p, \text{Pong}, _, _m_1)$
 $[_m_0 \leftarrow \text{Ping}]_p$
 $[v \leftarrow p._m_0]_q$
 $[_m_1 \leftarrow \text{Pong}]_q$

R-SEND
 $\Delta, \Sigma \models x = q$ q is a PID
 m fresh $\Gamma' = \Gamma \cup \{(p, q, t, r, m)\}$

 $\Gamma, \Delta, \Sigma, [\text{send}(x, n, t, r)]_p \rightsquigarrow \Gamma', (\Delta; [m \leftarrow n]_p), \Sigma, \text{skip}$

R-RECV
 $\Delta, \Sigma \models x = p$ p is a PID
 $(p, q, t, r, m) \in \Gamma$ $\Gamma' = \Gamma - \{(p, q, t, r, m)\}$

 $\Gamma, \Delta, \Sigma, [y \leftarrow \text{recv}(t, x, r)]_q \rightsquigarrow \Gamma', (\Delta; [y \leftarrow p.m]_q), \Sigma, \text{skip}$



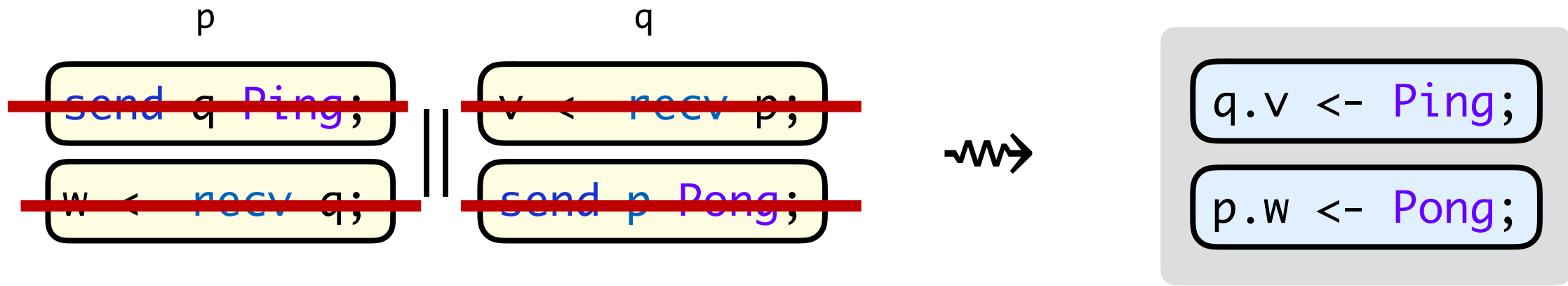
$[_m_0 \leftarrow \text{Ping}]_p$
 $[v \leftarrow p._m0]_q$
 $[_m_1 \leftarrow \text{Pong}]_q$
 $[w \leftarrow q._m1]_q$

R-SEND

$$\frac{\Delta, \Sigma \models x = q \quad m \text{ fresh} \quad \begin{array}{l} q \text{ is a PID} \\ \Gamma' = \Gamma \cup \{(p, q, t, r, m)\} \end{array}}{\Gamma, \Delta, \Sigma, [\text{send}(x, n, t, r)]_p \rightsquigarrow \Gamma', (\Delta; [m \leftarrow n]_p), \Sigma, \text{skip}}$$

R-RECV

$$\frac{\Delta, \Sigma \models x = p \quad \begin{array}{l} p \text{ is a PID} \\ (p, q, t, r, m) \in \Gamma \quad \Gamma' = \Gamma - \{(p, q, t, r, m)\} \end{array}}{\Gamma, \Delta, \Sigma, [y \leftarrow \text{recv}(t, x, r)]_q \rightsquigarrow \Gamma', (\Delta; [y \leftarrow p.m]_q), \Sigma, \text{skip}}$$



$[_m_0 \leftarrow \text{Ping}]_p$
 $[v \leftarrow p._m0]_q$
 $[_m_1 \leftarrow \text{Pong}]_q$
 $[w \leftarrow q._m1]_q$

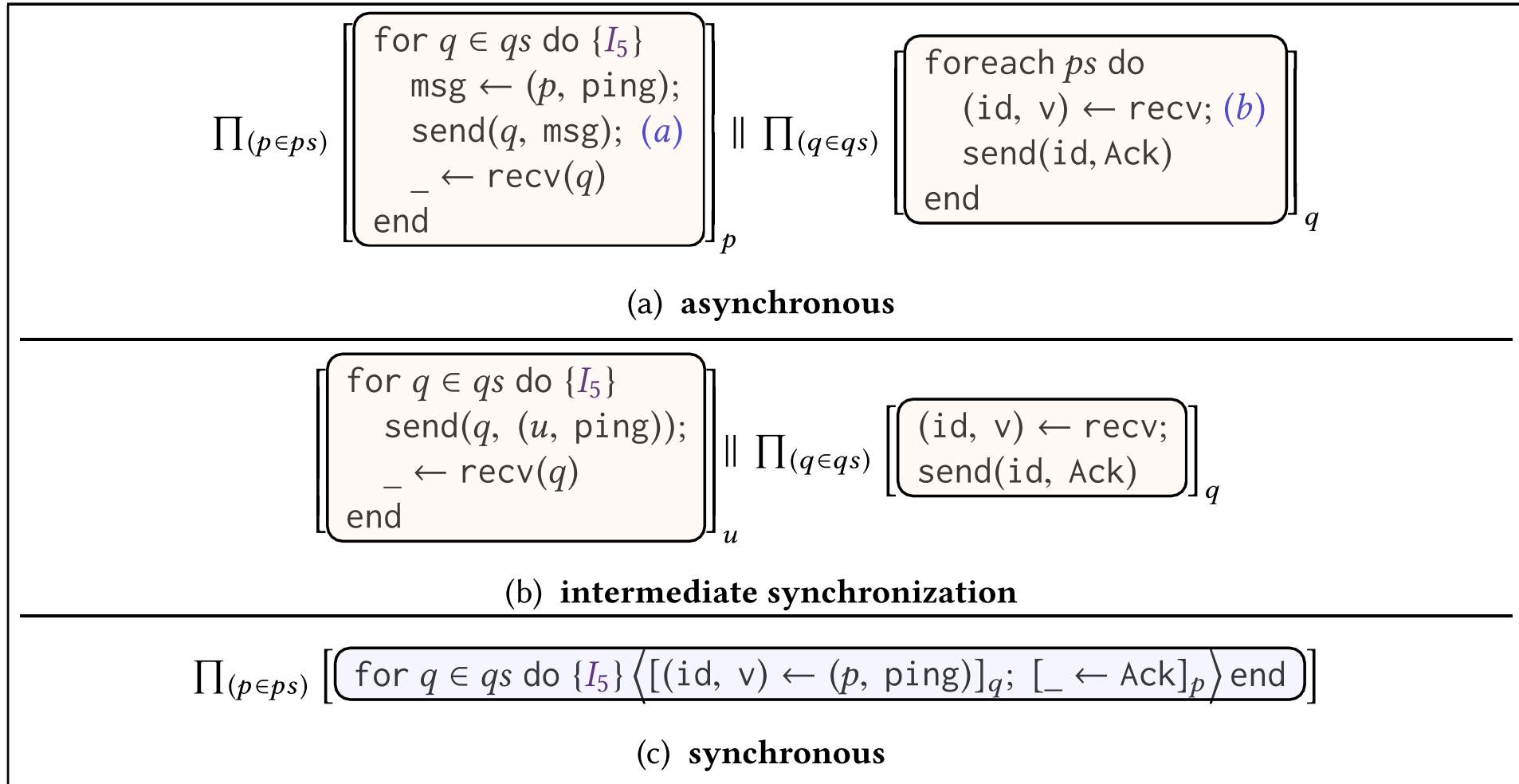
$q.v \leftarrow \text{Ping}$
 $p.w \leftarrow \text{Pong}$

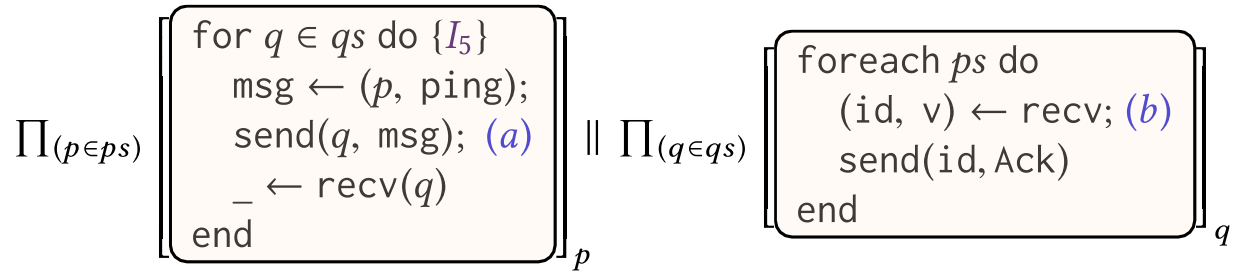
R-SEND

$$\frac{\Delta, \Sigma \models x = q \quad m \text{ fresh} \quad \begin{array}{l} q \text{ is a PID} \\ \Gamma' = \Gamma \cup \{(p, q, t, r, m)\} \end{array}}{\Gamma, \Delta, \Sigma, [\text{send}(x, n, t, r)]_p \rightsquigarrow \Gamma', (\Delta; [m \leftarrow n]_p), \Sigma, \text{skip}}$$

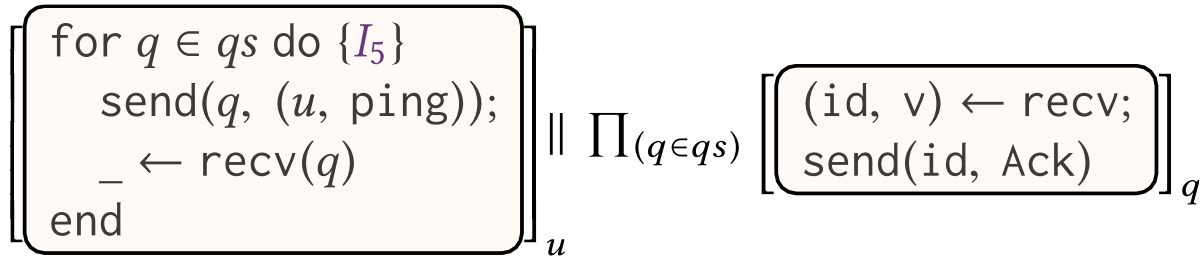
R-RECV

$$\frac{\Delta, \Sigma \models x = p \quad \begin{array}{l} p \text{ is a PID} \\ (p, q, t, r, m) \in \Gamma \quad \Gamma' = \Gamma - \{(p, q, t, r, m)\} \end{array}}{\Gamma, \Delta, \Sigma, [y \leftarrow \text{recv}(t, x, r)]_q \rightsquigarrow \Gamma', (\Delta; [y \leftarrow p.m]_q), \Sigma, \text{skip}}$$





(a) **asynchronous**

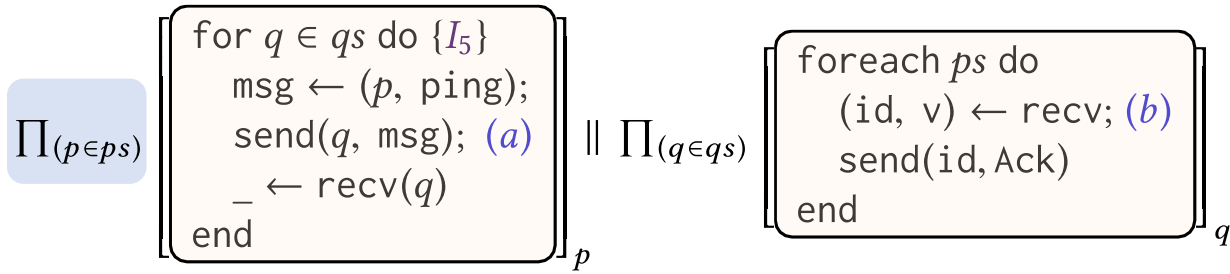


(b) **intermediate synchronization**

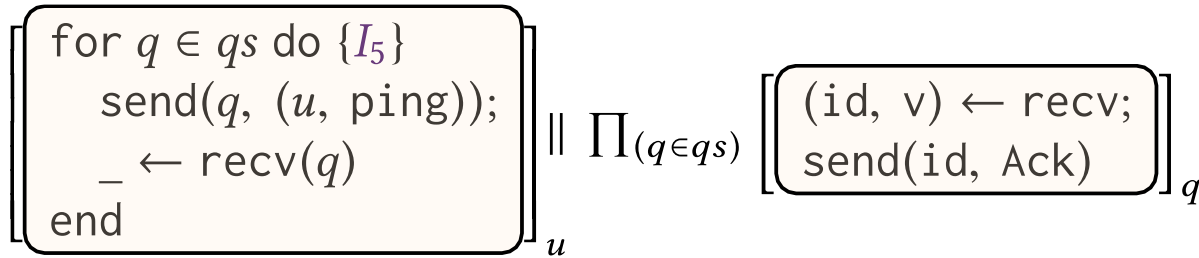
R-Focus

- (1) u fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, _, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Sigma' = (\Sigma \parallel \prod (p \in ps). \Delta^u[p/u])$
 - (4) $\Delta, \Sigma' \models I_C$ and $(\Delta_0; \Delta^u), \Sigma' \models I_C$
-
- $\Gamma_0, \Delta_0, \text{skip}, [A]_u \parallel \prod (q \in qs). [B]_q \rightsquigarrow \Gamma_0, (\Delta_0; \Delta^u), \text{skip}, \text{skip}$

$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A] \parallel \prod (q \in qs). [\text{foreach } ps \text{ do } B] \rightsquigarrow \Gamma, \Delta, \Sigma', \text{skip}$



(a) **asynchronous**

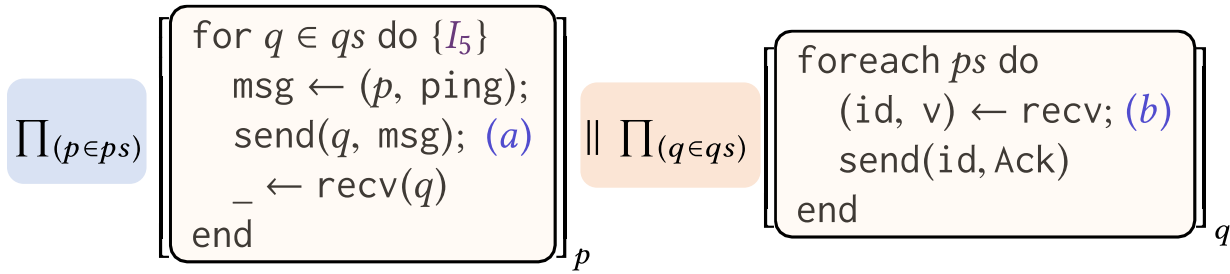


(b) **intermediate synchronization**

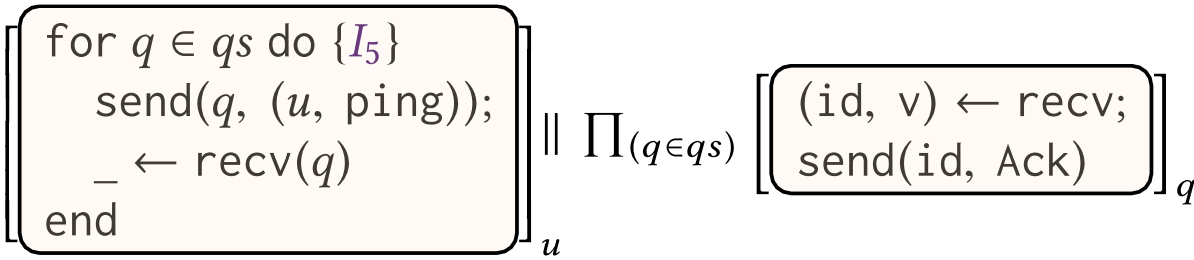
R-Focus

- (1) u fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, _, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Sigma' = (\Sigma \parallel \prod (p \in ps). \Delta^u[p/u])$
 - (4) $\Delta, \Sigma' \models I_C$ and $(\Delta_0; \Delta^u), \Sigma' \models I_C$
-
- $\Gamma_0, \Delta_0, \text{skip}, [A]_u \parallel \prod (q \in qs). [B]_q \rightsquigarrow \Gamma_0, (\Delta_0; \Delta^u), \text{skip}, \text{skip}$

$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A] \parallel \prod (q \in qs). [\text{foreach } ps \text{ do } B] \rightsquigarrow \Gamma, \Delta, \Sigma', \text{skip}$



(a) **asynchronous**

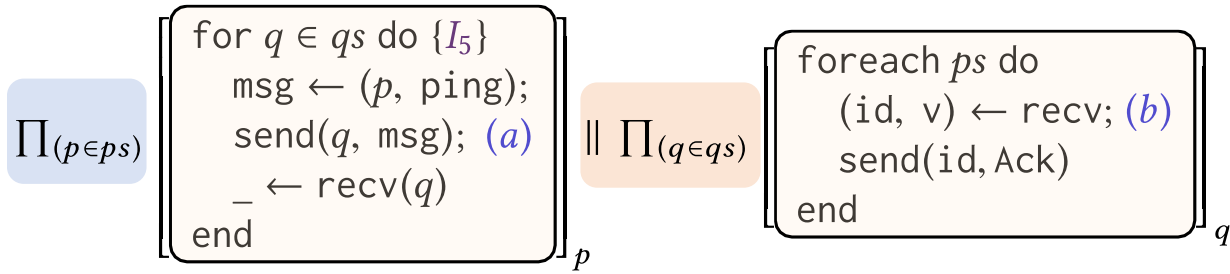


(b) **intermediate synchronization**

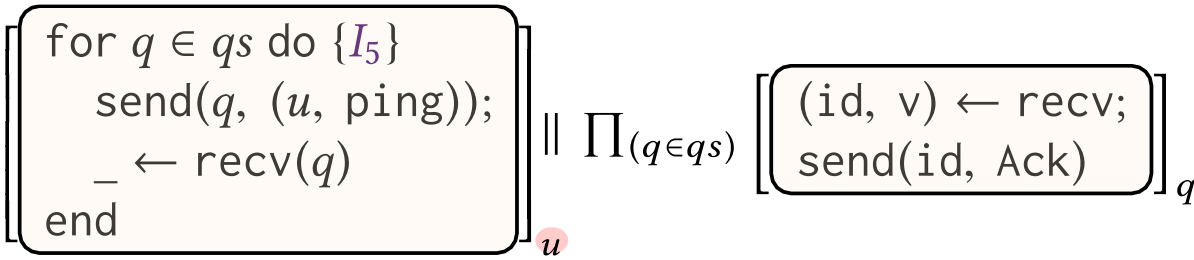
R-Focus

- (1) u fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, _, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Sigma' = (\Sigma \parallel \prod (p \in ps). \Delta^u[p/u])$
 - (4) $\Delta, \Sigma' \models I_C$ and $(\Delta_0; \Delta^u), \Sigma' \models I_C$
- $\Gamma_0, \Delta_0, \text{skip}, [A]_u \parallel \prod (q \in qs). [B]_q \rightsquigarrow \Gamma_0, (\Delta_0; \Delta^u), \text{skip}, \text{skip}$

$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A] \parallel \prod (q \in qs). [\text{foreach } ps \text{ do } B] \rightsquigarrow \Gamma, \Delta, \Sigma', \text{skip}$



(a) **asynchronous**

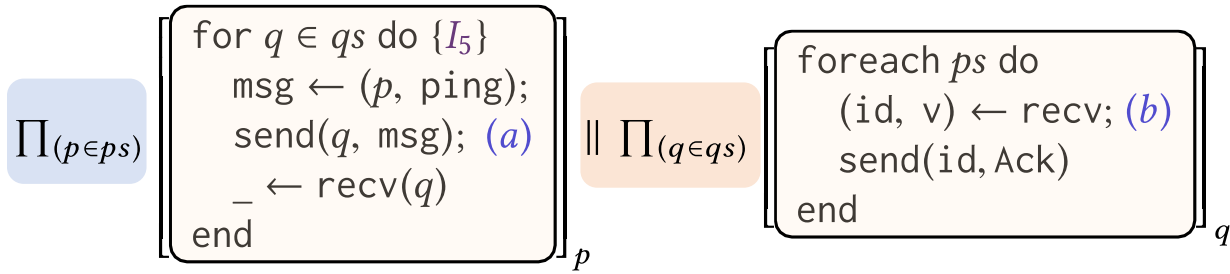


(b) **intermediate synchronization**

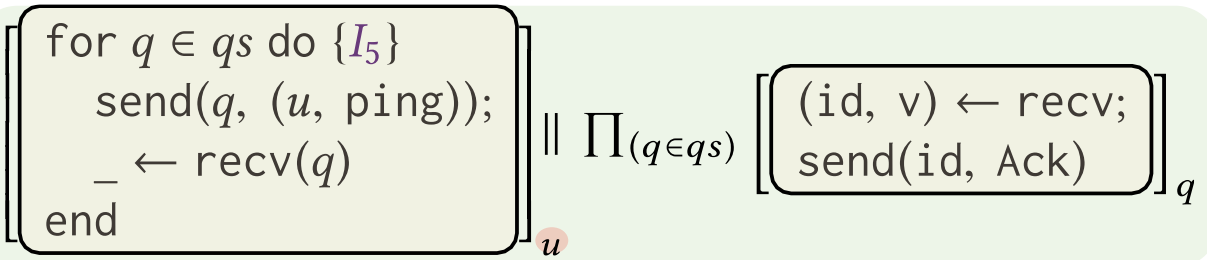
R-Focus

- (1) u fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, _, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Sigma' = (\Sigma \parallel \prod (p \in ps). \Delta^u[p/u])$
 - (4) $\Delta, \Sigma' \models I_C$ and $(\Delta_0; \Delta^u), \Sigma' \models I_C$
- $\Gamma_0, \Delta_0, \text{skip}, [A]_u \parallel \prod (q \in qs). [B]_q \rightsquigarrow \Gamma_0, (\Delta_0; \Delta^u), \text{skip}, \text{skip}$

$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A] \parallel \prod (q \in qs). [\text{foreach } ps \text{ do } B] \rightsquigarrow \Gamma, \Delta, \Sigma', \text{skip}$$



(a) asynchronous

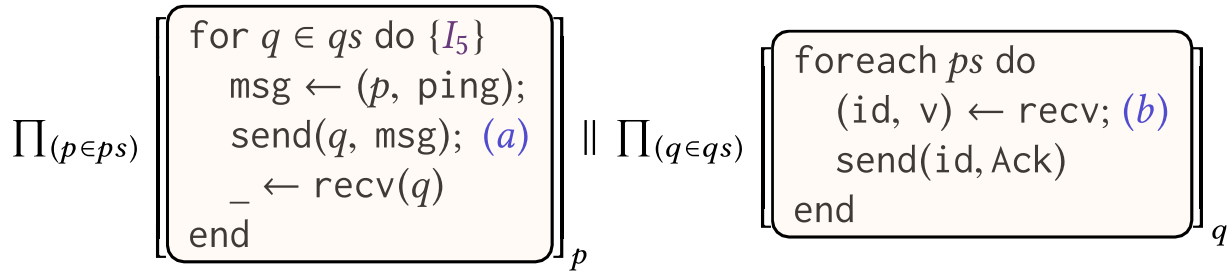


(b) intermediate synchronization

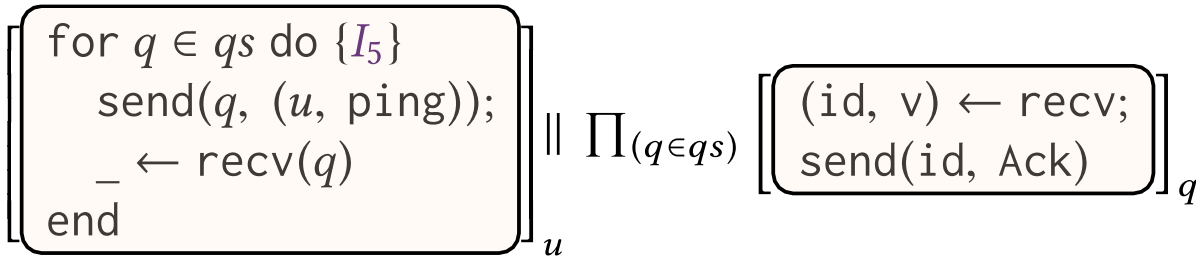
R-Focus

- (1) u fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, _, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Sigma' = (\Sigma \parallel \prod (p \in ps). \Delta^u[p/u])$
 - (4) $\Delta, \Sigma' \models I_C$ and $(\Delta_0; \Delta^u), \Sigma' \models I_C$
- $\Gamma_0, \Delta_0, \text{skip}. [A]_u \parallel \prod (q \in qs). [B]_q \rightsquigarrow \Gamma_0, (\Delta_0; \Delta^u), \text{skip}, \text{skip}$

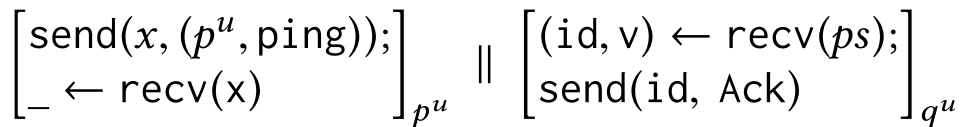
$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A] \parallel \prod (q \in qs). [\text{foreach } ps \text{ do } B] \rightsquigarrow \Gamma, \Delta, \Sigma', \text{skip}$$



(a) **asynchronous**



(b) **intermediate synchronization**



(c) **Obligation after applying R-Focus, R-Loop**

R-Focus

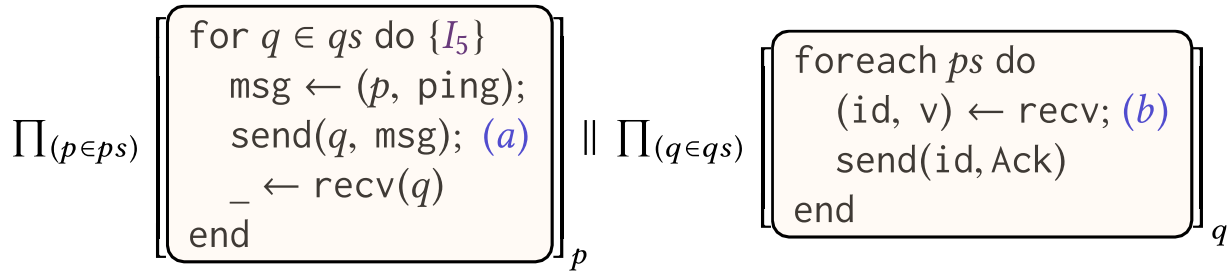
- (1) u fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, _, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Sigma' = (\Sigma \parallel \prod (p \in ps). \Delta^u[p/u])$
 - (4) $\Delta, \Sigma' \models I_C$ and $(\Delta_0; \Delta^u), \Sigma' \models I_C$
- $$\Gamma_0, \Delta_0, \text{skip}, [A]_u \parallel \prod (q \in qs). [B]_q \rightsquigarrow \Gamma_0, (\Delta_0; \Delta^u), \text{skip}, \text{skip}$$

$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A] \parallel \prod (q \in qs). [\text{foreach } ps \text{ do } B] \rightsquigarrow \Gamma, \Delta, \Sigma', \text{skip}$$

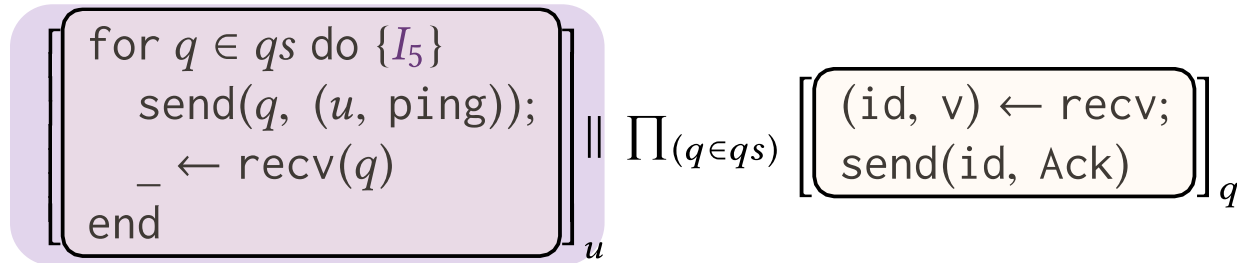
R-Loop

- (1) u, x fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, x, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Delta, \Sigma \models I_C$ and $(\Delta_0; \langle \Delta^u \rangle), \Sigma \models I_C$
- $$\Gamma_0, \Delta_0, \Sigma, [A]_u \parallel B[x/p] \rightsquigarrow \Gamma, (\Delta_0; \Delta^u), \Sigma, \text{skip}$$

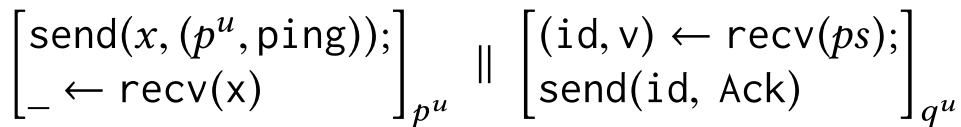
$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A]_p \parallel [\text{for } p \in ps \{I_S\} \text{ do } B \text{ end}]_q \rightsquigarrow \Gamma, [\text{for } p \in ps \text{ do } \langle I_S \triangleright \Delta^u[p/u] \rangle \text{ end}], \Sigma, \text{skip}$$



(a) **asynchronous**



(b) **intermediate synchronization**



(c) **Obligation after applying R-Focus, R-Loop**

R-Focus

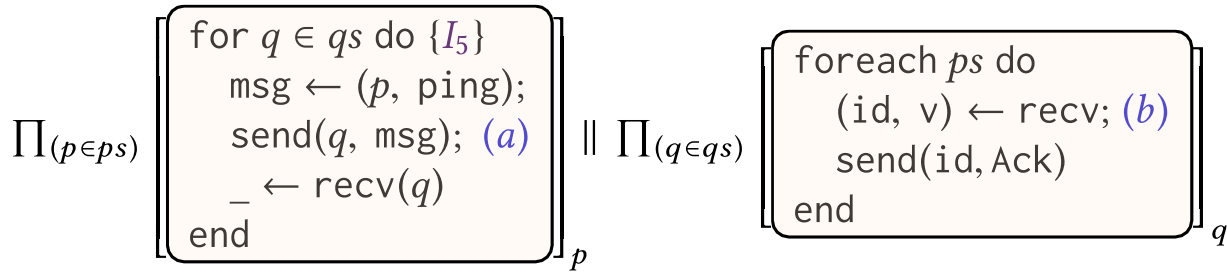
- (1) u fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, _, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Sigma' = (\Sigma \parallel \prod (p \in ps). \Delta^u[p/u])$
 - (4) $\Delta, \Sigma' \models I_C$ and $(\Delta_0; \Delta^u), \Sigma' \models I_C$
- $$\Gamma_0, \Delta_0, \text{skip}, [A]_u \parallel \prod (q \in qs). [B]_q \rightsquigarrow \Gamma_0, (\Delta_0; \Delta^u), \text{skip}, \text{skip}$$

$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A] \parallel \prod (q \in qs). [\text{foreach } ps \text{ do } B] \rightsquigarrow \Gamma, \Delta, \Sigma', \text{skip}$$

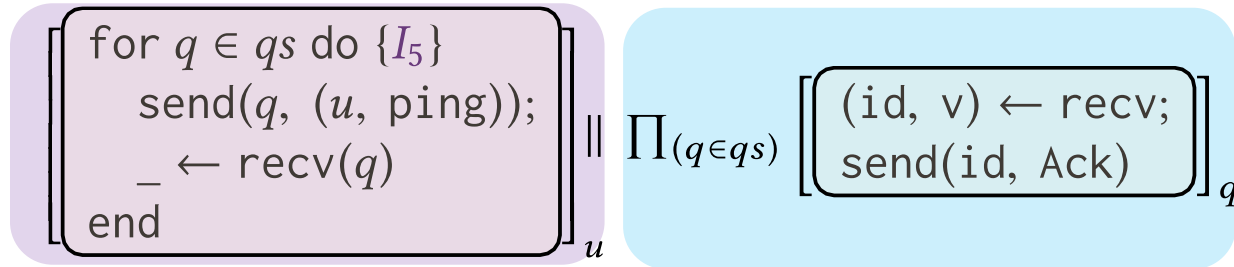
R-Loop

- (1) u, x fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, x, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Delta, \Sigma \models I_C$ and $(\Delta_0; \langle \Delta^u \rangle), \Sigma \models I_C$
- $$\Gamma_0, \Delta_0, \Sigma, [A]_u \parallel B[x/p] \rightsquigarrow \Gamma, (\Delta_0; \Delta^u), \Sigma, \text{skip}$$

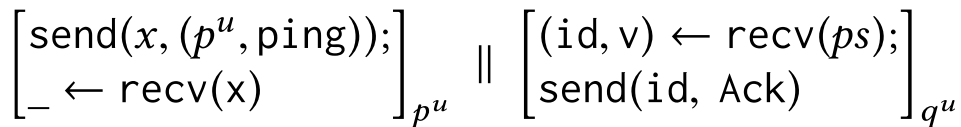
$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A]_p \parallel \left[\text{for } p \in ps \{I_S\} \text{ do } B \text{ end} \right]_q \rightsquigarrow \Gamma, \left[\text{for } p \in ps \text{ do } \langle I_S \triangleright \Delta^u[p/u] \rangle \text{ end} \right], \Sigma, \text{skip}$$



(a) **asynchronous**



(b) **intermediate synchronization**



(c) **Obligation after applying R-Focus, R-Loop**

R-Focus

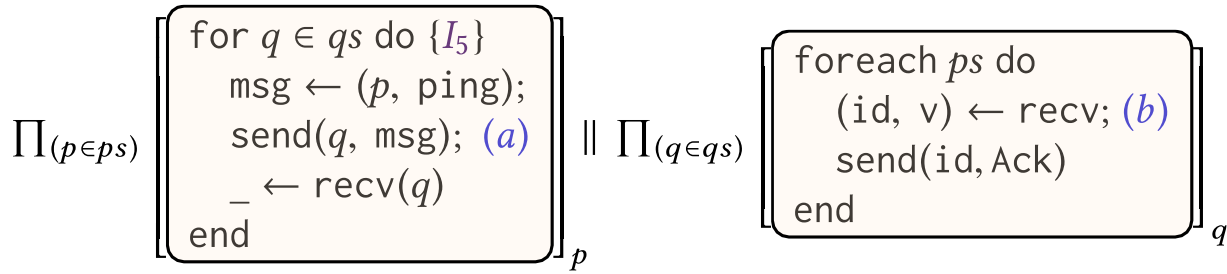
- (1) u fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, _, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Sigma' = (\Sigma \parallel \prod (p \in ps). \Delta^u[p/u])$
 - (4) $\Delta, \Sigma' \models I_C$ and $(\Delta_0; \Delta^u), \Sigma' \models I_C$
- $$\Gamma_0, \Delta_0, \text{skip}, [A]_u \parallel \prod (q \in qs). [B]_q \rightsquigarrow \Gamma_0, (\Delta_0; \Delta^u), \text{skip}, \text{skip}$$

$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A] \parallel \prod (q \in qs). [\text{foreach } ps \text{ do } B] \rightsquigarrow \Gamma, \Delta, \Sigma', \text{skip}$$

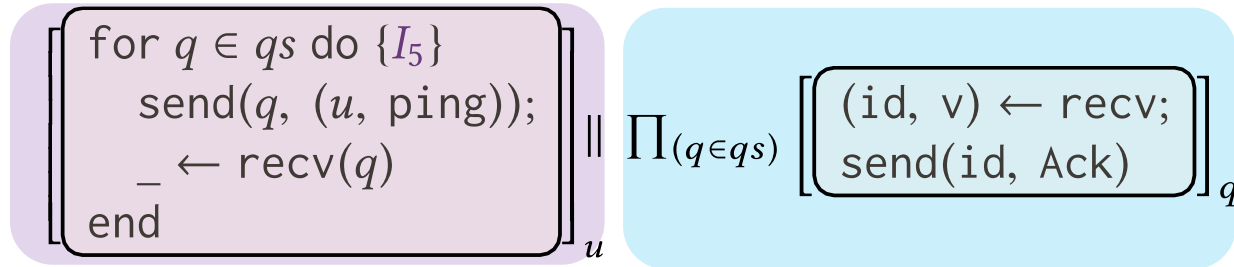
R-Loop

- (1) u, x fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, x, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Delta, \Sigma \models I_C$ and $(\Delta_0; \langle \Delta^u \rangle), \Sigma \models I_C$
- $$\Gamma_0, \Delta_0, \Sigma, [A]_u \parallel B[x/p] \rightsquigarrow \Gamma, (\Delta_0; \Delta^u), \Sigma, \text{skip}$$

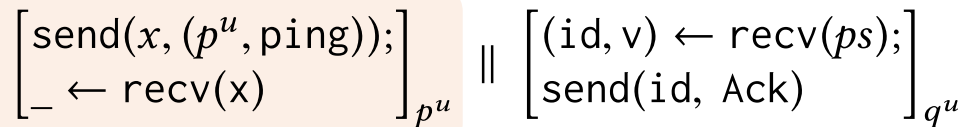
$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A]_p \parallel \left[\text{for } p \in ps \{I_S\} \text{ do } B \text{ end} \right]_q \rightsquigarrow \Gamma, \left[\text{for } p \in ps \text{ do } \langle I_S \triangleright \Delta^u[p/u] \rangle \text{ end} \right], \Sigma, \text{skip}$$



(a) **asynchronous**



(b) **intermediate synchronization**



(c) **Obligation after applying R-Focus, R-Loop**

R-Focus

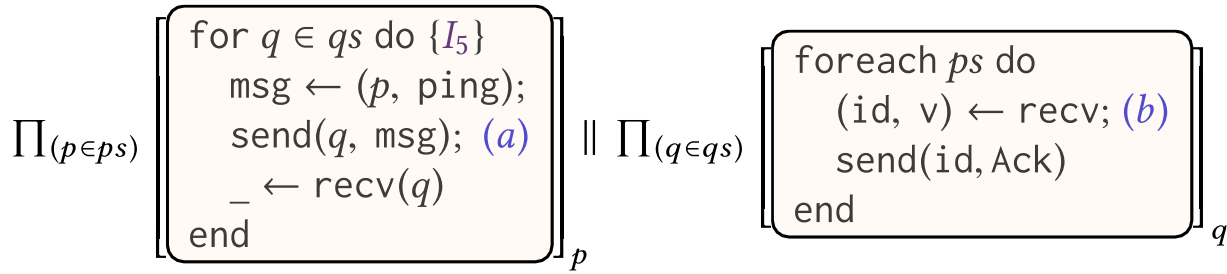
- (1) u fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, _, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Sigma' = (\Sigma \parallel \prod (p \in ps). \Delta^u[p/u])$
 - (4) $\Delta, \Sigma' \models I_C$ and $(\Delta_0; \Delta^u), \Sigma' \models I_C$
- $$\Gamma_0, \Delta_0, \text{skip}, [A]_u \parallel \prod (q \in qs). [B]_q \rightsquigarrow \Gamma_0, (\Delta_0; \Delta^u), \text{skip}, \text{skip}$$

$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A] \parallel \prod (q \in qs). [\text{foreach } ps \text{ do } B] \rightsquigarrow \Gamma, \Delta, \Sigma', \text{skip}$$

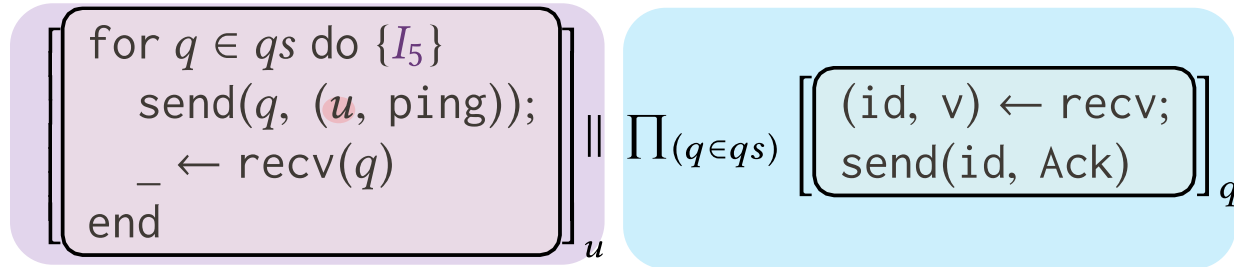
R-Loop

- (1) u, x fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, x, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Delta, \Sigma \models I_C$ and $(\Delta_0; \langle \Delta^u \rangle), \Sigma \models I_C$
- $$\Gamma_0, \Delta_0, \Sigma, [A]_u \parallel B[x/p] \rightsquigarrow \Gamma, (\Delta_0; \Delta^u), \Sigma, \text{skip}$$

$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A]_p \parallel \left[\text{for } p \in ps \{I_S\} \text{ do } B \text{ end} \right]_q \rightsquigarrow \Gamma, \left[\text{for } p \in ps \text{ do } \langle I_S \triangleright \Delta^u[p/u] \rangle \text{ end} \right], \Sigma, \text{skip}$$



(a) **asynchronous**



(b) **intermediate synchronization**



(c) **Obligation after applying R-Focus, R-Loop**

R-Focus

- (1) u fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, _, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Sigma' = (\Sigma \parallel \prod (p \in ps). \Delta^u[p/u])$
 - (4) $\Delta, \Sigma' \models I_C$ and $(\Delta_0; \Delta^u), \Sigma' \models I_C$
- $$\Gamma_0, \Delta_0, \text{skip}, [A]_u \parallel \prod (q \in qs). [B]_q \rightsquigarrow \Gamma_0, (\Delta_0; \Delta^u), \text{skip}, \text{skip}$$

$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A] \parallel \prod (q \in qs). [\text{foreach } ps \text{ do } B] \rightsquigarrow \Gamma, \Delta, \Sigma', \text{skip}$$

R-Loop

- (1) u, x fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, x, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Delta, \Sigma \models I_C$ and $(\Delta_0; \langle \Delta^u \rangle), \Sigma \models I_C$
- $$\Gamma_0, \Delta_0, \Sigma, [A]_u \parallel B[x/p] \rightsquigarrow \Gamma, (\Delta_0; \Delta^u), \Sigma, \text{skip}$$

$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A]_p \parallel [\text{for } p \in ps \{I_S\} \text{ do } B \text{ end}]_q \rightsquigarrow \Gamma, [\text{for } p \in ps \text{ do } \langle I_S \triangleright \Delta^u[p/u] \rangle \text{ end}], \Sigma, \text{skip}$$

R-Loop

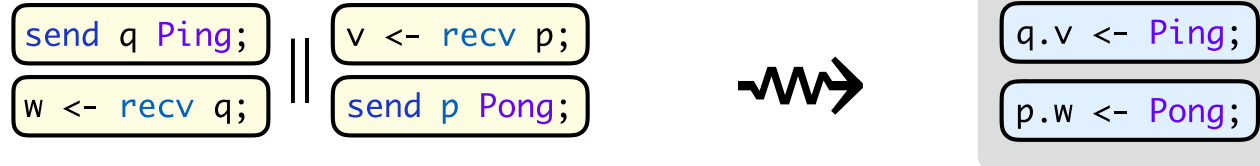
- (1) u, x fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, x, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Delta, \Sigma \models I_C$ and $(\Delta_0; \langle \Delta^u \rangle), \Sigma \models I_C$
- $$\Gamma_0, \Delta_0, \Sigma, [A]_u \parallel B[x/p] \rightsquigarrow \Gamma, (\Delta_0; \Delta^u), \Sigma, \text{skip}$$
-

$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A]_p \parallel \left[\text{for } p \in ps \{I_S\} \text{ do } B \text{ end} \right]_q \rightsquigarrow$$

$$\Gamma, \left[\text{for } p \in ps \text{ do } \langle I_S \triangleright \Delta^u[p/u] \rangle \text{ end} \right], \Sigma, \text{skip}$$

R-FOCUS

- (1) u fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, _, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Sigma' = (\Sigma \parallel \prod (p \in ps). \Delta^u[p/u])$
 - (4) $\Delta, \Sigma' \models I_C$ and $(\Delta_0; \Delta^u), \Sigma' \models I_C$
- $$\Gamma_0, \Delta_0, \text{skip}, [A]_u \parallel \prod (q \in qs). [B]_q \rightsquigarrow \Gamma_0, (\Delta_0; \Delta^u), \text{skip}, \text{skip}$$
-

$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A] \parallel \prod (q \in qs). \left[\text{foreach } ps \text{ do } B \right] \rightsquigarrow \Gamma, \Delta, \Sigma', \text{skip}$$


$$\prod (p \in ps) \left[\text{for } q \in qs \text{ do } \{I_S\} \left\langle [(\text{id}, v) \leftarrow (p, \text{ping})]_q; [_ \leftarrow \text{Ack}]_p \right\rangle \text{end} \right]$$

R-Loop

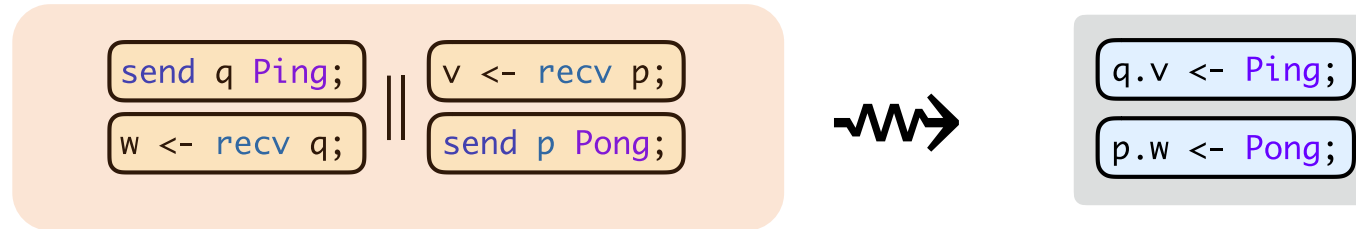
- (1) u, x fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, x, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Delta, \Sigma \models I_C$ and $(\Delta_0; \langle \Delta^u \rangle), \Sigma \models I_C$
- $$\Gamma_0, \Delta_0, \Sigma, [A]_u \parallel B[x/p] \rightsquigarrow \Gamma, (\Delta_0; \Delta^u), \Sigma, \text{skip}$$

$$\frac{\Gamma, \Delta, \Sigma, \prod (p \in ps). [A]_p \parallel [\text{for } p \in ps \{I_S\} \text{ do } B \text{ end}]_q \rightsquigarrow \Gamma, [\text{for } p \in ps \text{ do } \langle I_S \triangleright \Delta^u[p/u] \rangle \text{ end}], \Sigma, \text{skip}}$$

R-FOCUS

- (1) u fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, _, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Sigma' = (\Sigma \parallel \prod (p \in ps). \Delta^u[p/u])$
 - (4) $\Delta, \Sigma' \models I_C$ and $(\Delta_0; \Delta^u), \Sigma' \models I_C$
- $$\Gamma_0, \Delta_0, \text{skip}, [A]_u \parallel \prod (q \in qs). [B]_q \rightsquigarrow \Gamma_0, (\Delta_0; \Delta^u), \text{skip}, \text{skip}$$

$$\frac{\Gamma, \Delta, \Sigma, \prod (p \in ps). [A] \parallel \prod (q \in qs). [\text{foreach } ps \text{ do } B] \rightsquigarrow \Gamma, \Delta, \Sigma', \text{skip}}$$



$$\prod (p \in ps) \left[\text{for } q \in qs \text{ do } \{I_S\} \left\langle [(\text{id}, v) \leftarrow (p, \text{ping})]_q; [_ \leftarrow \text{Ack}]_p \right\rangle \text{end} \right]$$

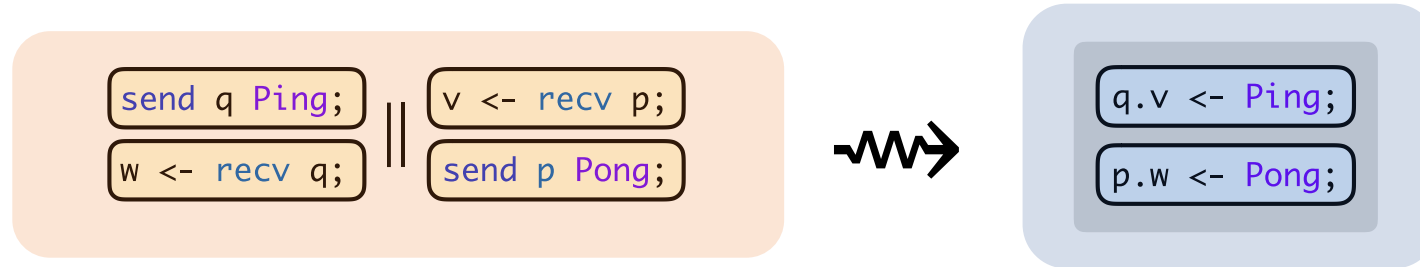
R-Loop

- (1) u, x fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, x, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Delta, \Sigma \models I_C$ and $(\Delta_0; \langle \Delta^u \rangle), \Sigma \models I_C$
- $$\Gamma_0, \Delta_0, \Sigma, [A]_u \parallel B[x/p] \rightsquigarrow \Gamma, (\Delta_0; \Delta^u), \Sigma, \text{skip}$$

$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A]_p \parallel [\text{for } p \in ps \{I_S\} \text{ do } B \text{ end}]_q \rightsquigarrow \Gamma, [\text{for } p \in ps \text{ do } \langle I_S \triangleright \Delta^u[p/u] \rangle \text{ end}], \Sigma, \text{skip}$$

R-FOCUS

- (1) u fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, _, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Sigma' = (\Sigma \parallel \prod (p \in ps). \Delta^u[p/u])$
 - (4) $\Delta, \Sigma' \models I_C$ and $(\Delta_0; \Delta^u), \Sigma' \models I_C$
- $$\Gamma_0, \Delta_0, \text{skip}, [A]_u \parallel \prod (q \in qs). [B]_q \rightsquigarrow \Gamma_0, (\Delta_0; \Delta^u), \text{skip}, \text{skip}$$

$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A] \parallel \prod (q \in qs). [\text{foreach } ps \text{ do } B] \rightsquigarrow \Gamma, \Delta, \Sigma', \text{skip}$$


$$\prod (p \in ps) \left[\text{for } q \in qs \text{ do } \{I_S\} \left\langle [(\text{id}, v) \leftarrow (p, \text{ping})]_q; [_ \leftarrow \text{Ack}]_p \right\rangle \text{end} \right]$$

R-Loop

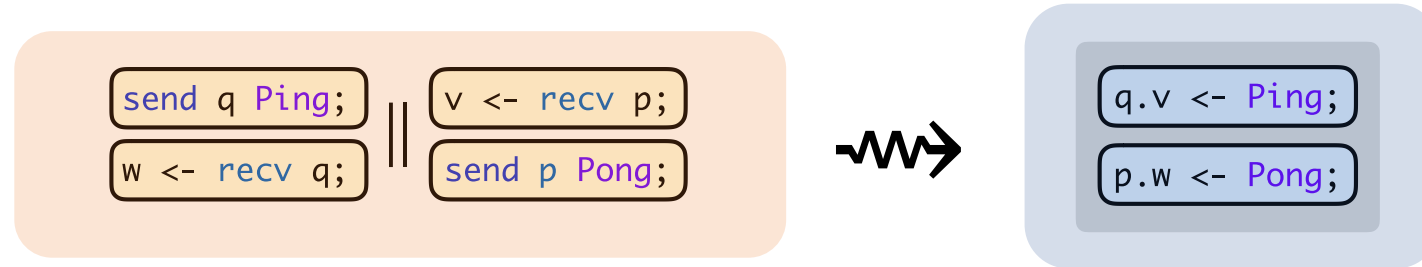
- (1) u, x fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, x, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Delta, \Sigma \models I_C$ and $(\Delta_0; \langle \Delta^u \rangle), \Sigma \models I_C$
- $$\Gamma_0, \Delta_0, \Sigma, [A]_u \parallel B[x/p] \rightsquigarrow \Gamma, (\Delta_0; \Delta^u), \Sigma, \text{skip}$$

$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A]_p \parallel [\text{for } p \in ps \{I_S\} \text{ do } B \text{ end}]_q \rightsquigarrow$$

$$\Gamma, [\text{for } p \in ps \text{ do } \langle I_S \triangleright \Delta^u[p/u] \rangle \text{ end}], \Sigma, \text{skip}$$

R-FOCUS

- (1) u fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, _, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Sigma' = (\Sigma \parallel \prod (p \in ps). \Delta^u[p/u])$
 - (4) $\Delta, \Sigma' \models I_C$ and $(\Delta_0; \Delta^u), \Sigma' \models I_C$
- $$\Gamma_0, \Delta_0, \text{skip}, [A]_u \parallel \prod (q \in qs). [B]_q \rightsquigarrow \Gamma_0, (\Delta_0; \Delta^u), \text{skip}, \text{skip}$$

$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A] \parallel \prod (q \in qs). [\text{foreach } ps \text{ do } B] \rightsquigarrow \Gamma, \Delta, \Sigma', \text{skip}$$


$$\prod (p \in ps) \left[\text{for } q \in qs \text{ do } \{I_S\} \left\langle [(\text{id}, v) \leftarrow (p, \text{ping})]_q; [_ \leftarrow \text{Ack}]_p \right\rangle \text{end} \right]$$

R-Loop

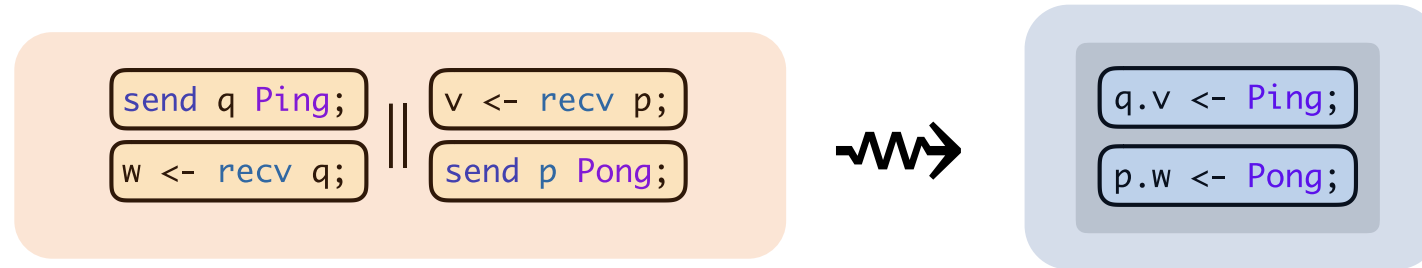
- (1) u, x fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, x, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Delta, \Sigma \models I_C$ and $(\Delta_0; \langle \Delta^u \rangle), \Sigma \models I_C$
- $$\Gamma_0, \Delta_0, \Sigma, [A]_u \parallel B[x/p] \rightsquigarrow \Gamma, (\Delta_0; \Delta^u), \Sigma, \text{skip}$$

$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A]_p \parallel [\text{for } p \in ps \{I_S\} \text{ do } B \text{ end}]_q \rightsquigarrow$$

$$\Gamma, [\text{for } p \in ps \text{ do } \langle I_S \triangleright \Delta^u[p/u] \rangle \text{ end}], \Sigma, \text{skip}$$

R-FOCUS

- (1) u fresh
 - (2) $\Gamma_0 \triangleq \Gamma \cup \{\text{unfold}(u, _, ps)\}$ and $\Delta_0 \triangleq \text{assume}(I_C)$
 - (3) $\Sigma' = (\Sigma \parallel \prod (p \in ps). \Delta^u[p/u])$
 - (4) $\Delta, \Sigma' \models I_C$ and $(\Delta_0; \Delta^u), \Sigma' \models I_C$
- $$\Gamma_0, \Delta_0, \text{skip}, [A]_u \parallel \prod (q \in qs). [B]_q \rightsquigarrow \Gamma_0, (\Delta_0; \Delta^u), \text{skip}, \text{skip}$$

$$\Gamma, \Delta, \Sigma, \prod (p \in ps). [A] \parallel \prod (q \in qs). [\text{foreach } ps \text{ do } B] \rightsquigarrow \Gamma, \Delta, \Sigma', \text{skip}$$


$$\prod (p \in ps) \left[\text{for } q \in qs \text{ do } \{I_S\} \left\langle [(\text{id}, v) \leftarrow (p, \text{ping})]_q; [_ \leftarrow \text{Ack}]_p \right\rangle \text{end} \right]$$

Name	#Inv Async	Time Async/ Dafny	#Inv Sync	Time Sync
2PC	30	12.8s	3	0.04s
Raft	50	301.6s	6	0.18s
Paxos	72	1141.3s	14	1.51s
Total	152	1455.8s	23	1.73s

Reduce Invariants by 6x

Reduce checking time by 3 orders of magnitude

Limitations

- Approach is not applicable to arbitrary protocols
 - if asynchrony is not *almost symmetric*, then it is not supported
- Very restrictive round non-interference condition
 - maybe can identify state-sharing patterns that produce decidable VCs?
- Only restrictive communication patterns are supported
 - effectively, if it does not look like a method call, it's not supported