

# DbC for multiparty distributed interactions: static & dynamic validation

Laura Bocchi, University of Leicester

Tzu-Chun Chen, University of London

Pierre-Malo Denielou, Imperial College London

Kohei Honda, University of London

Emilio Tuosto, University of Leicester

Nobuko Yoshida, Imperial College London

# BACKGROUND

- DbC: Assertions = **Types** + **Logical Formulae**

- Type signature

```
int foobar(int i)
```

- Assertion

```
int foobar(int i){  
    pre: {i>10}  
    post: {0< result < 1000}  
}
```

- Building systems on the basis of precise contracts
  - restrain **defensive programming**
  - provide **robustness**



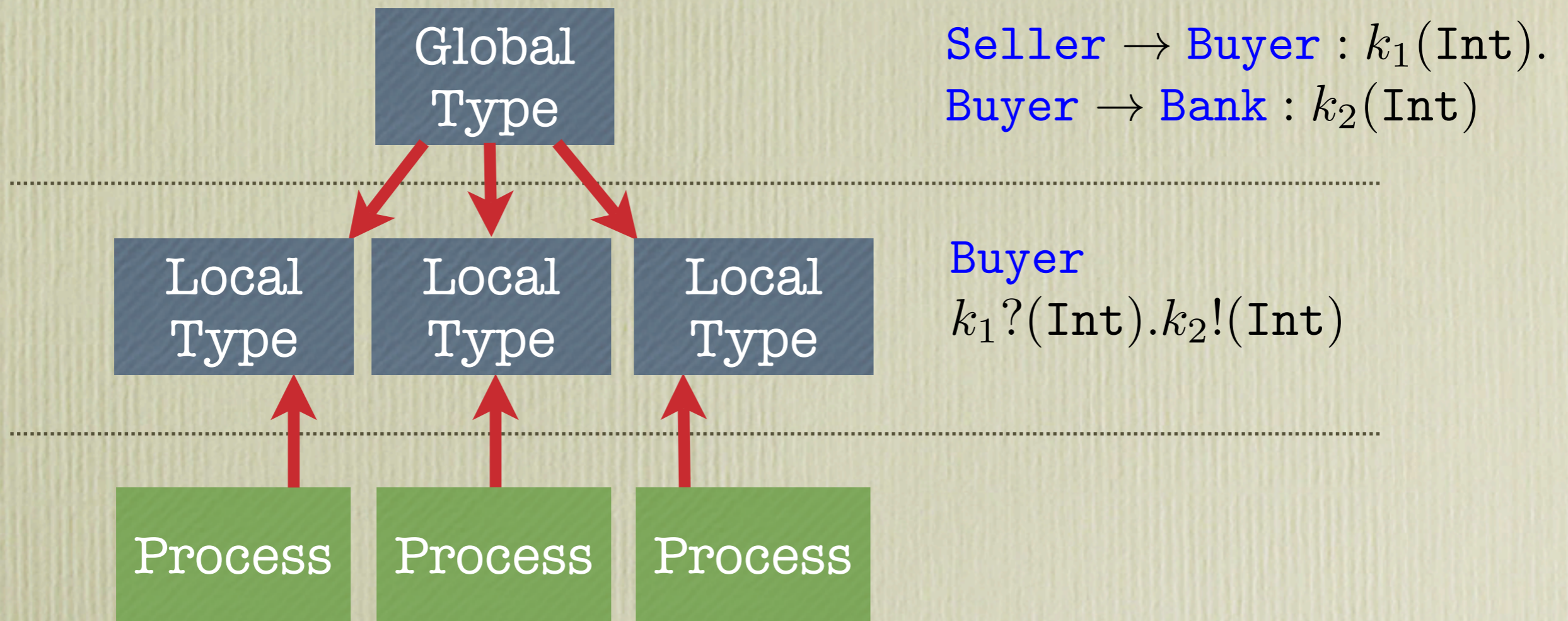
Bertrand Meyer  
**Applying “Design by Contract”**  
In Computer (IEEE), 25, 1992

# CHALLENGES

*Can we extend this framework to communications and concurrency?*

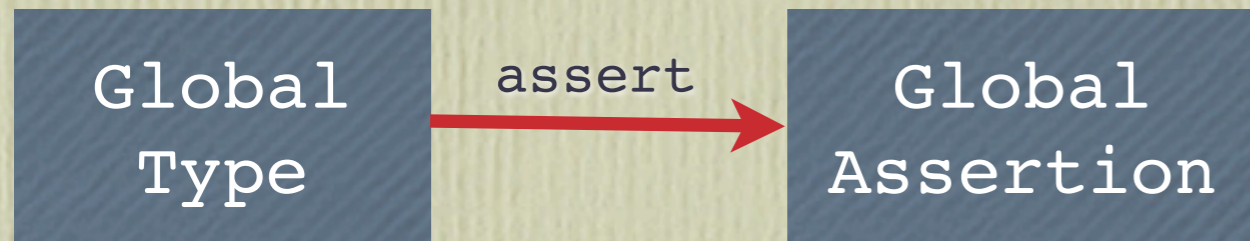
- **Distributed Setting** (asynchronous message passing)
- The **responsibilities** are **spread** among the participants
- Participants have **different views** of the contract, e.g., the condition of an interaction is
  - a post-condition for the sender
  - a pre-condition for the receiver
  - what about third parties?

# Multiparty Session Types



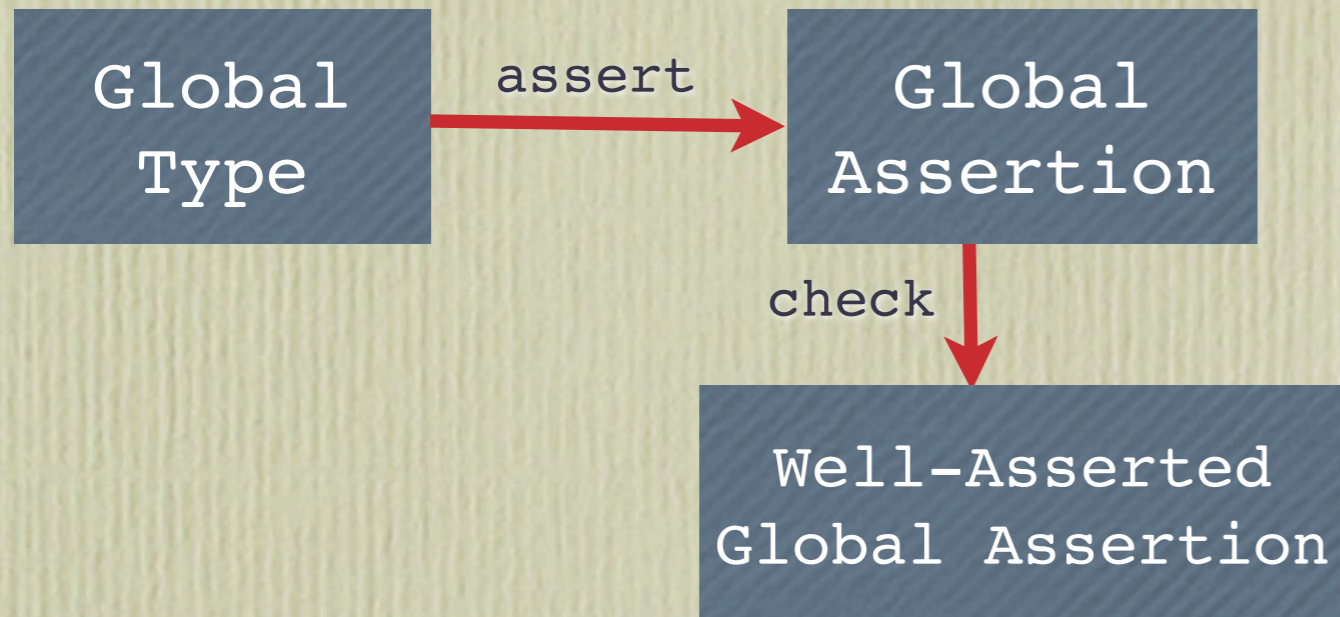
Kohei Honda, Nobuko Yoshida and Marco Carbone  
**Multiparty Asynchronous Session Types** (POPL 2008)

# OUTLINE



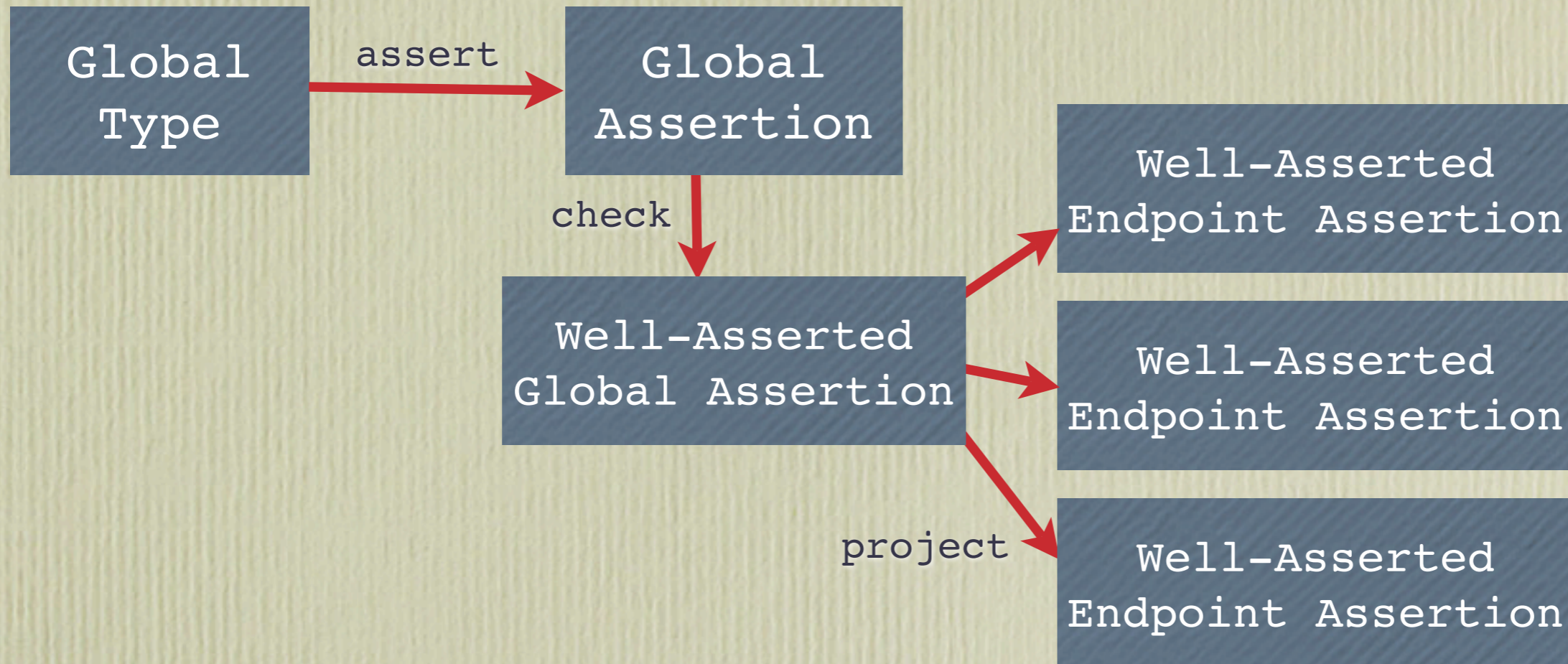
- A **global type** is used as a type signature describing the interactions of a multiparty **session**
- Each abstract action is **annotated** with a predicate

# OUTLINE



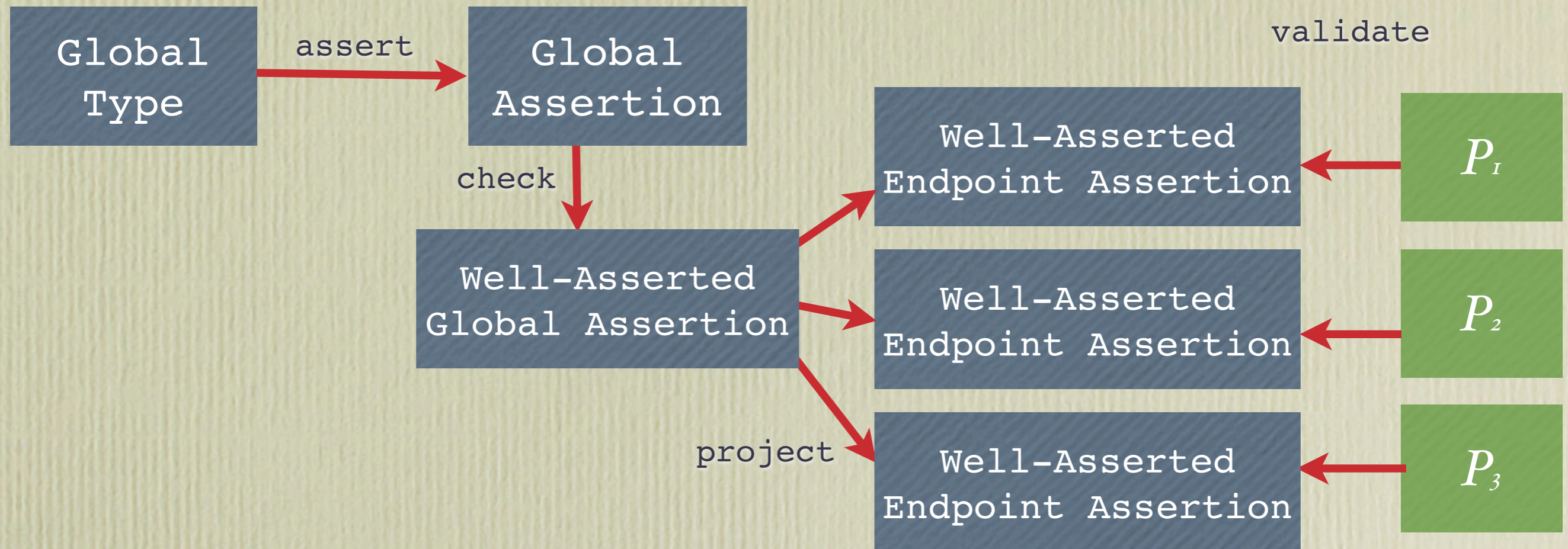
- **Consistency** of the global specification is checked

# OUTLINE



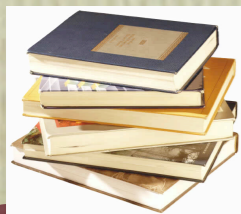
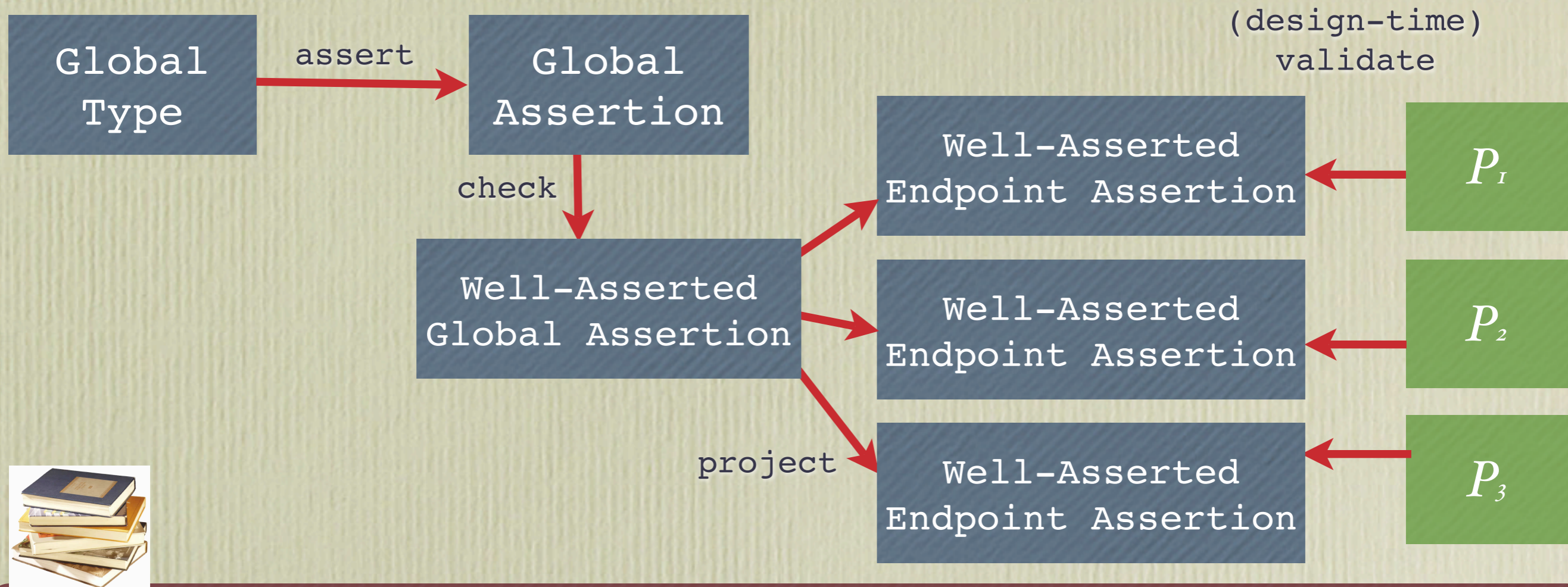
- The global assertion is **projected** onto each endpoint **preserving consistency**

# OUTLINE



- Each process is **validated** against its (one or more) local specification(s)

# STATIC VALIDATION

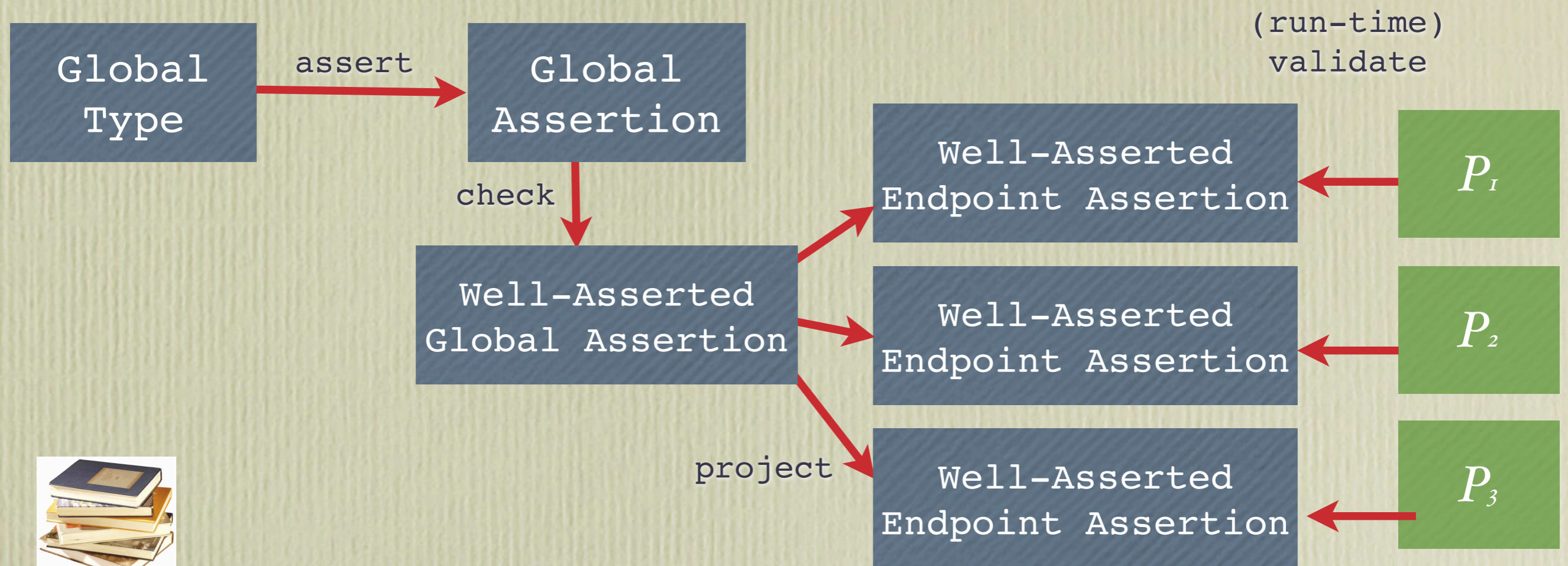


Laura Bocchi, Kohei Honda, Emilio Tuosto, and Nobuko Yoshida  
**A Theory of DbC for Multiparty Distributed Interactions** (CONCUR 2010)

- Key points: effective **well-assertedness**, **projection**, **validation**\*
- The proof system is **sound** and **relatively complete**

\*validation is effective up to the underlying logic

# RUNTIME VALIDATION



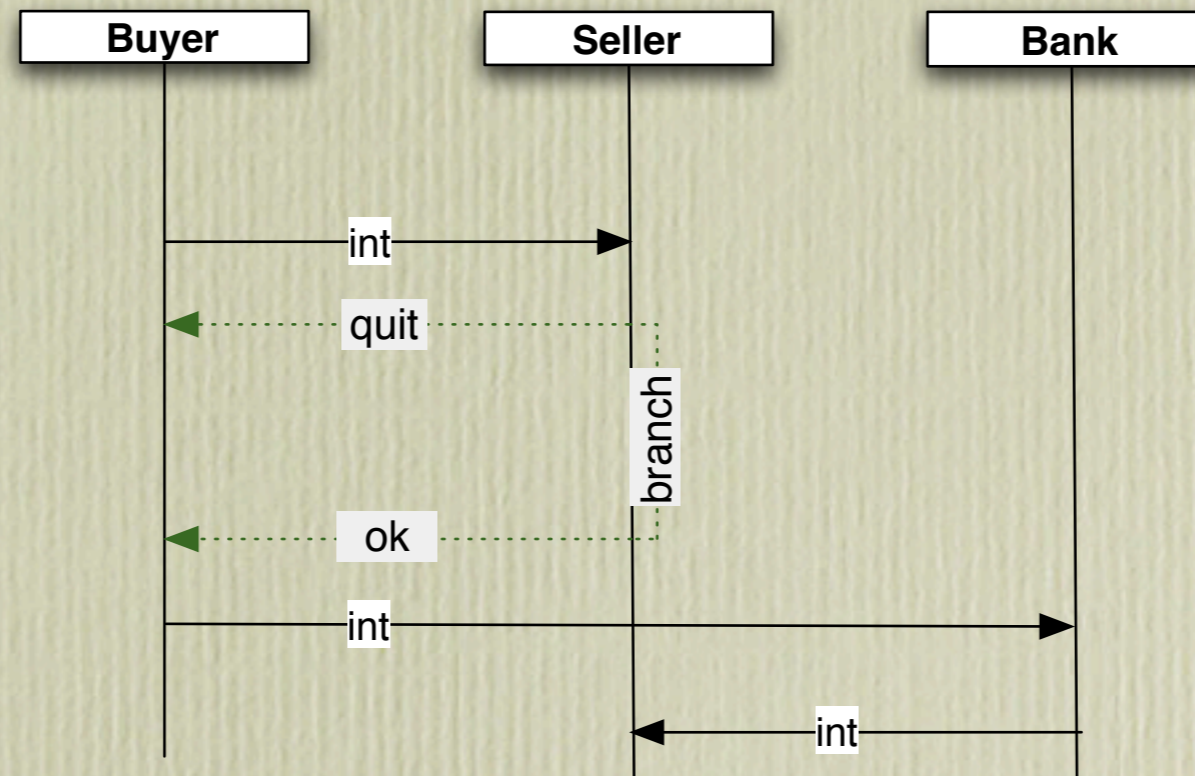
Tzu-Chun Chen, Laura Bocchi, Pierre-Malo Denielou, Kohei Honda, Nobuko Yoshida

**Distributed Monitoring for Multiparty Session Enforcement**

[http://www.eecs.qmul.ac.uk/~tcchen/monitoring\\_sessions.html](http://www.eecs.qmul.ac.uk/~tcchen/monitoring_sessions.html)

- From recent collaboration with **Ocean Observation Initiative (OOI)** on **large scale** distributed systems
- **Unsafe** endpoints in **multiple administrative domains**.
- Use previous theory to achieve **runtime enforcement**

# GLOBAL ASSERTIONS



Buyer  $\rightarrow$  Seller :  $k_1(\text{Int})$ .

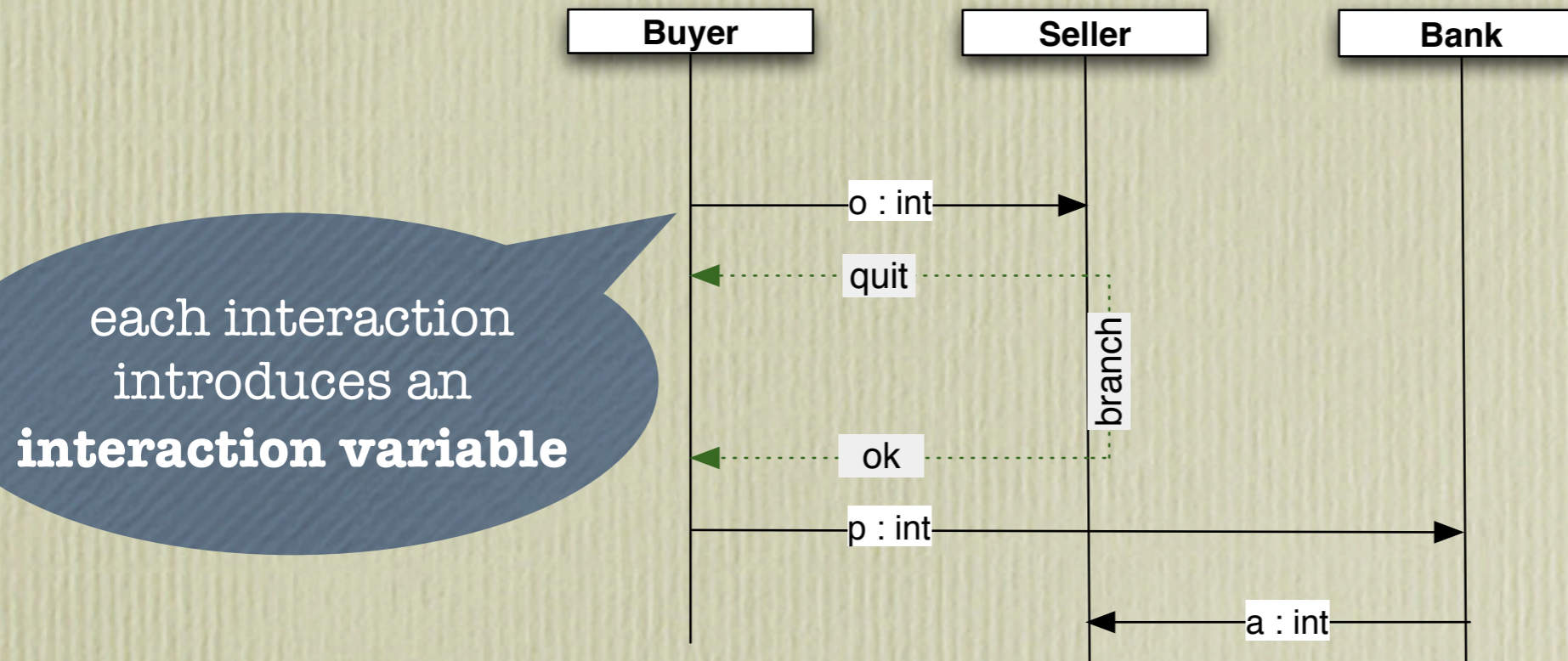
Seller  $\rightarrow$  Buyer :  $k_2\{\mathbf{quit} : \text{End},$

$\mathbf{ok} : \text{Buyer} \rightarrow \text{Bank} : k_3(\text{Int}).$

$\text{Bank} \rightarrow \text{Seller} : k_4(\text{Int})$

}

# GLOBAL ASSERTIONS



Buyer  $\rightarrow$  Seller :  $k_1(o : \text{Int})$ .

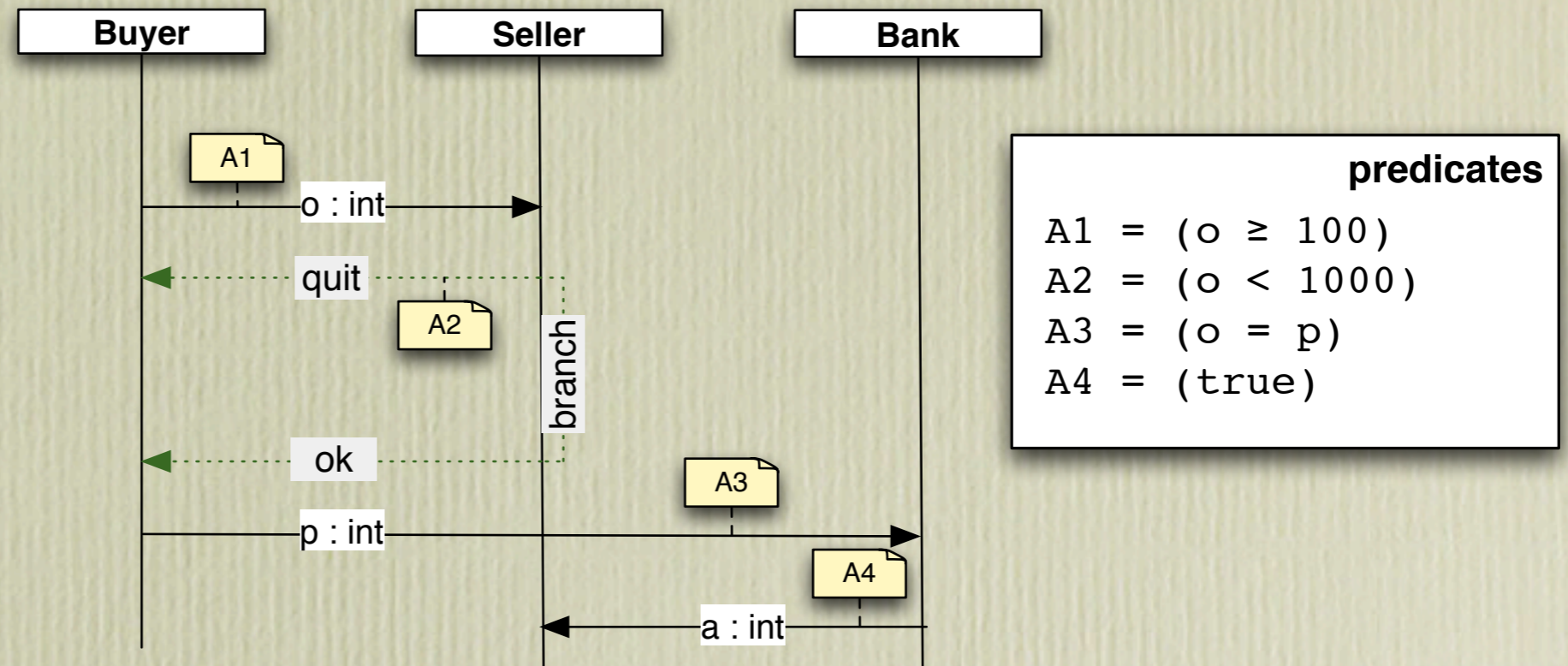
Seller  $\rightarrow$  Buyer :  $k_2\{\mathbf{quit} : \text{End},$

ok : Buyer  $\rightarrow$  Bank :  $k_3(p : \text{Int})$ .

Bank  $\rightarrow$  Seller :  $k_4(a : \text{Int})$

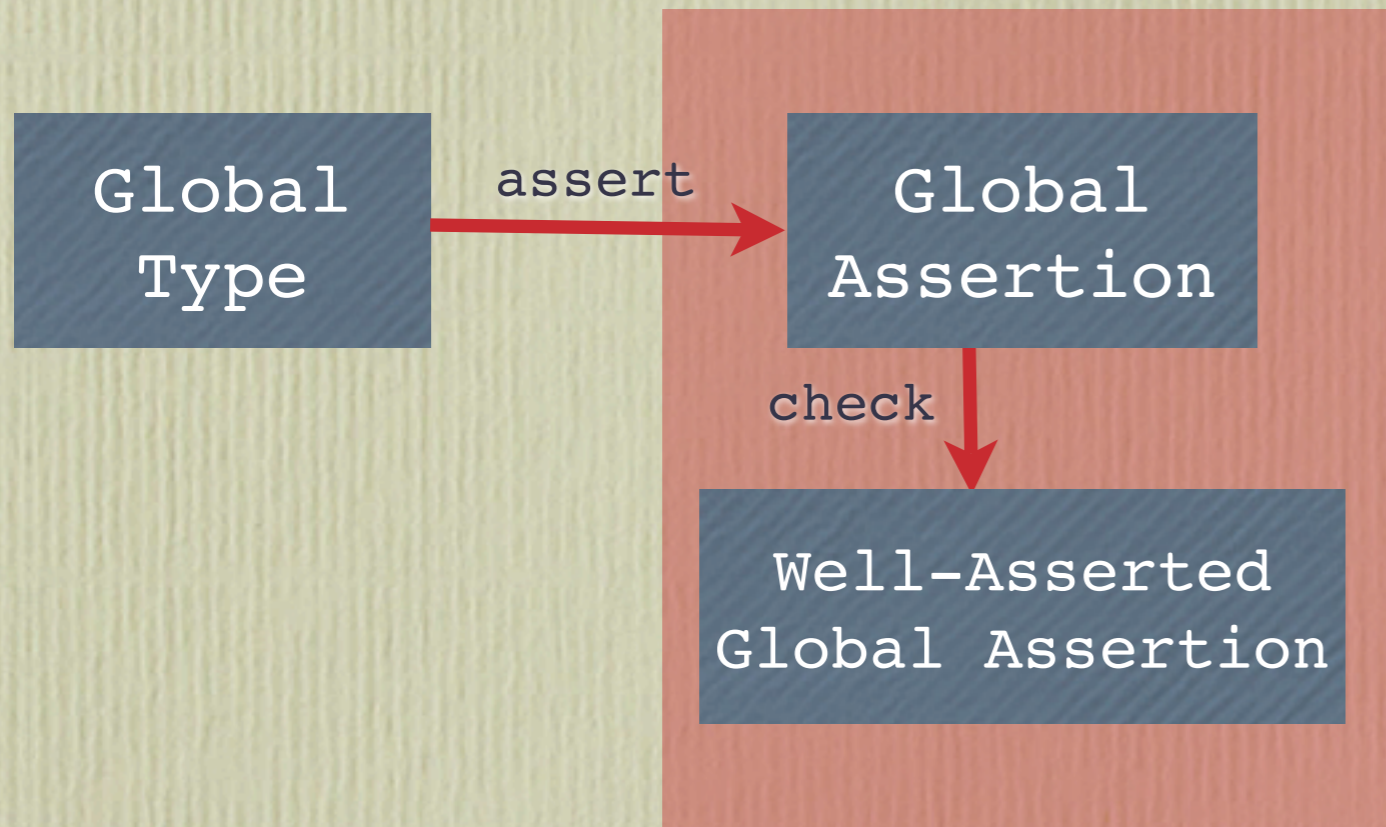
}

# GLOBAL ASSERTIONS



$\text{Buyer} \rightarrow \text{Seller} : k_1(o : \text{Int})\{A_1\}.$   
 $\text{Seller} \rightarrow \text{Buyer} : k_2\{\{A_2\} \text{ quit} : \text{End},$   
 $\quad \{true\} \text{ ok} : \text{Buyer} \rightarrow \text{Bank} : k_3(p : \text{Int})\{A_3\}.$   
 $\quad \text{Bank} \rightarrow \text{Seller} : k_4(a : \text{Int})\{A_4\}$   
 $\quad \}$

# CONSISTENCY CHECK



**When is a global assertion well designed?**

# HISTORY SENSITIVITY

“an interaction predicate can only contain those interaction variables that are known to its sender”

✗

Alice	→	Bob	:	( $u$ : Int)	{true}.	Alice	Bob	Carol
						$u$	$u$	
Bob	→	Carol	:	( $v$ : Int)	{true}.		$v$	$v$
Carol	→	Alice	:	( $z$ : Int)	{ $z > u$ }			

✓

Alice	→	Bob	:	( $u$ : Int)	{true}.
Bob	→	Carol	:	( $v$ : Int)	{ $v > u$ }.
Carol	→	Alice	:	( $z$ : Int)	{ $z > v$ }

Carol cannot  
guarantee  $z > u$  since she  
does not know  $u$

# TEMPORAL SATISFIABILITY

“a process can always find a valid forward path at each interaction point until it meets the end”

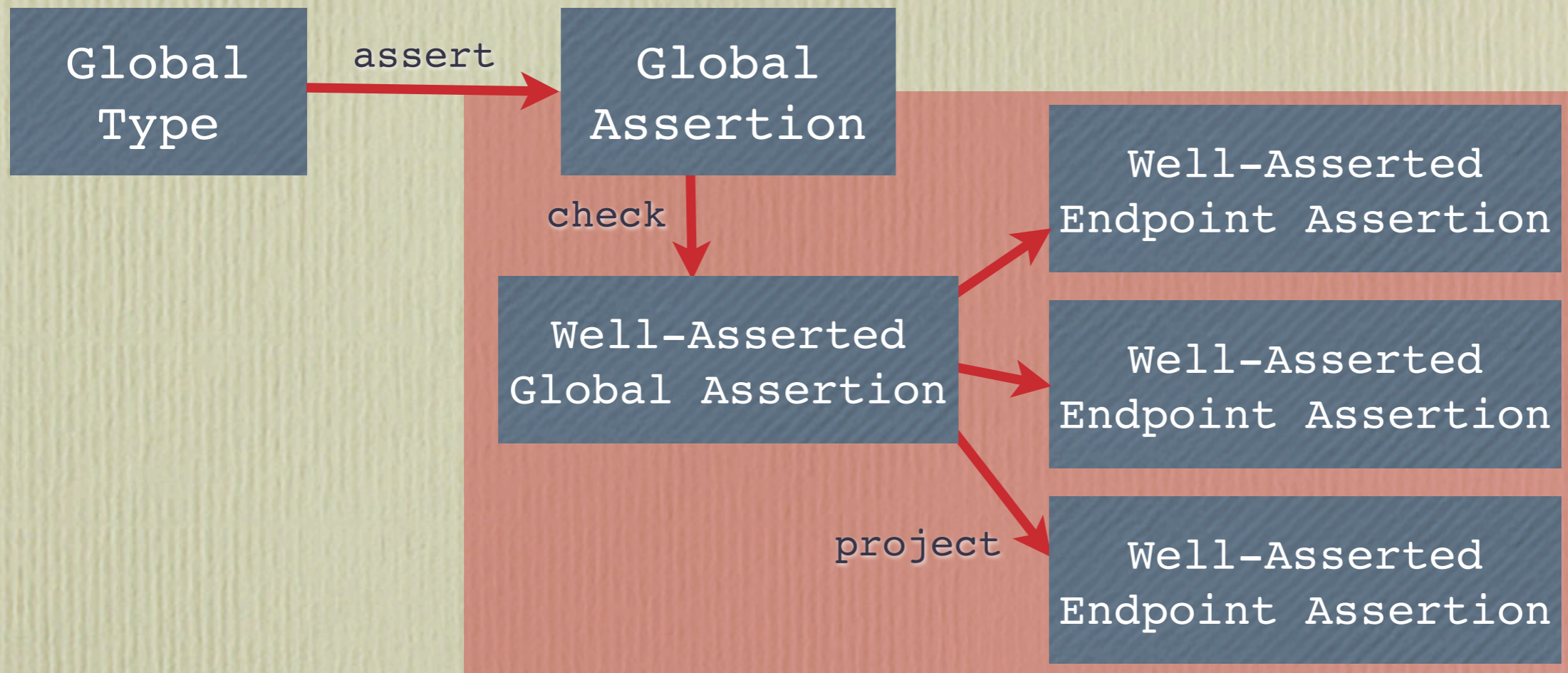
✗ Alice  $\rightarrow$  Bob :  $(v : \text{Int}) \{v > 10\}$ .  
Bob  $\rightarrow$  Alice :  $(z : \text{Int}) \{z < v \wedge z > 10\}$ .

✓ Alice  $\rightarrow$  Bob :  $(v : \text{Int}) \{v > 12\}$ .  
Bob  $\rightarrow$  Alice :  $(z : \text{Int}) \{z < v \wedge z > 10\}$ .

Had Alice  
chosen  $v=11$ ,  
Carol could not find  
a value for  $z$  s.t.  
 $z < 11$  and  $z > 10$

- **Well-assertedness** = History Sensitivity + Temporal Satisfiability
  - is **decidable** (as long as the logic is)
  - we provide **design-time checker**

# PROJECT



**How to project obligations and guarantees onto the endpoints?**

# ENDPOINT ASSERTIONS

## Global assertions

$p \rightarrow p' : k(\textcolor{red}{v} : \mathbb{S})\{\textcolor{red}{A}\}.\mathcal{G}$

$p \rightarrow p' : k\{\{\textcolor{red}{A}_i\}l_i : \mathcal{G}_i\}_{i \in I}$

$\mu \mathbf{t}\langle \textcolor{red}{e} \rangle (\textcolor{red}{v} : \mathbb{S})\{\textcolor{red}{A}\}.\mathcal{G}$

$\mathbf{t}\langle \textcolor{red}{e} \rangle$

$\mathcal{G}, \mathcal{G}'$

End

## Endpoint assertions

$k!(\textcolor{red}{v} : \mathbb{S})\{\textcolor{red}{A}\}; \mathcal{T}$

$k?(\textcolor{red}{v} : \mathbb{S})\{\textcolor{red}{A}\}; \mathcal{T}$

$k \oplus \{\{\textcolor{red}{A}_i\}l_i : \mathcal{T}_i\}_{i \in I}$

$k \& \{\{\textcolor{red}{A}_i\}l_i : \mathcal{T}_i\}_{i \in I}$

$\mu \mathbf{t}\langle \textcolor{red}{e} \rangle (\textcolor{red}{v} : \mathbb{S})\{\textcolor{red}{A}\}.\mathcal{T}$

$\mathbf{t}\langle \textcolor{red}{e} \rangle$

End

# PROJECTIONS

User  $\rightarrow$  Agent :  $k_1(c1 : \text{Command})\{c1 \neq \text{switch} - \text{off}\}$ .

Agent  $\rightarrow$  Instrument :  $k_2(c2 : \text{Command})\{c2 = c1\}$

- 
- A too naive projection on **Instrument**:

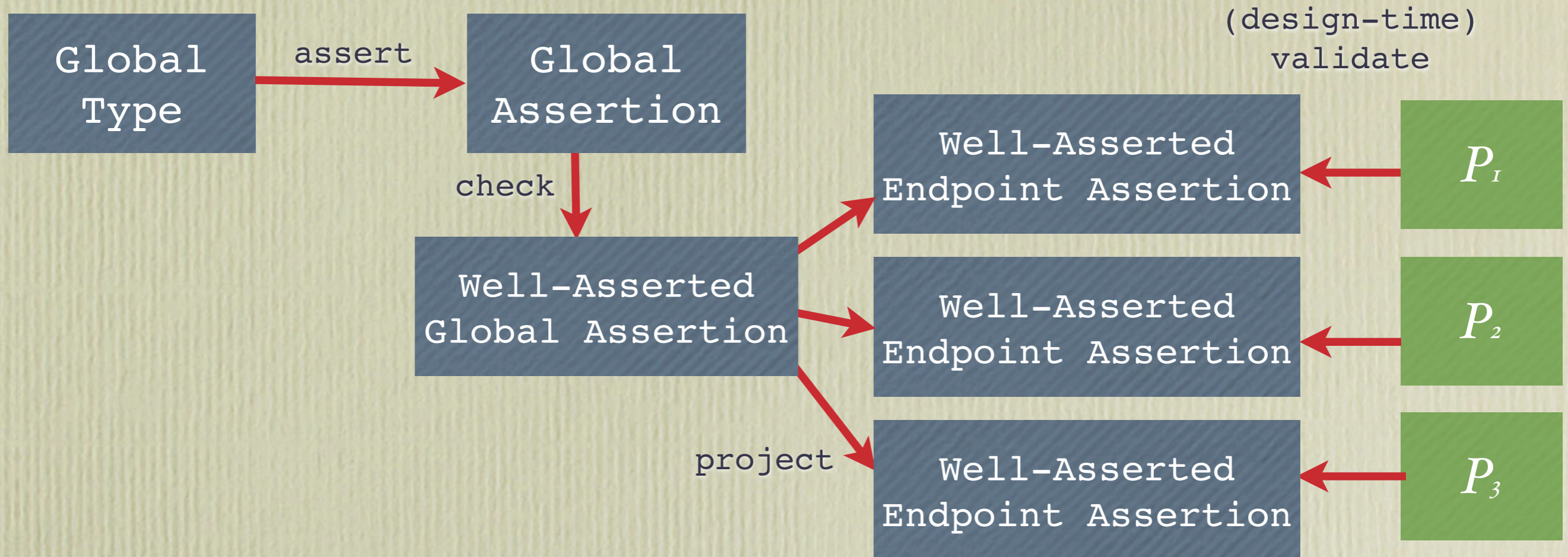
✗  $k_2?(c2 : \text{Command})\{c2 = c1\}$

---

✓  $k_2?(c2 : \text{Command})\{\exists c1.(c1 \neq \text{switch} - \text{off}) \wedge (c2 = c1)\}$

- We want to give **stronger preconditions** to prevent defensive programming
- We do not reveal the exact values exchanged between third parties

# STATIC VALIDATION



**How to ensure that a process satisfies a contract expressed as an assertion?**

# ASSERTED PROCESSES

## Programs

$$\begin{aligned}
 P ::= & \bar{a}[2..n](\tilde{s}).P \\
 & | a[p](\tilde{s}).P \\
 & | (\nu a)P \\
 & | s!\langle e \rangle(\nu)\{A\};P \\
 & | s?(\nu)\{A\};P \\
 & | s!\langle\langle \tilde{t} \rangle\rangle(\tilde{\nu})\{A\};P \\
 & | s?(\langle\langle \tilde{\nu} \rangle\rangle)\{A\};P
 \end{aligned}$$

request

accept

hide

send

receive

del-trw

del-cth

| if  $e$  then  $P$  else  $Q$

|  $s \triangleleft \{A\}l;P$

|  $s \triangleright \{\{A_i\}l_i : P_i\}_{i \in I}$

|  $P \mid Q$

|  $\mu X \langle e\tilde{t} \rangle (\nu\tilde{s}).P$

|  $X \langle e\tilde{s} \rangle$

|  $\mathbf{0}$

conditional

select

branch

parallel

rec def

rec call

idle

## Run-time processes

$$P_{rt} ::= P$$

|  $(\nu\tilde{s})P_{rt}$

|  $s:\tilde{h}$

| **errH**

| **errT**

**errH** notifies a violation in a send/select

**errT** notifies a violation in a receive/branch

Receive with no violation

$$s?(v)\{v \geq 10\};P \mid s:10 \cdot \tilde{h} \rightarrow P[10/v] \mid s:\tilde{h}$$

Receive with violation

$$s?(v)\{v \geq 10\};P \mid s:1 \cdot \tilde{h} \rightarrow \mathbf{errT} \mid s:\tilde{h}$$

# VALIDATION RULES

$\mathcal{C} ::= \text{true} \mid \mathcal{C} \wedge A$  (assertion environment)

$\Gamma ::= \emptyset \mid \Gamma, a:G \mid \Gamma, x:(\tilde{v}:\tilde{S}) L_1 @ p_1 \dots L_n @ p_n$  (typing environment)

$\Delta ::= \emptyset \mid \Delta, \tilde{s}:\tilde{T} @ p$  (assignment environment)

$\mathcal{C}; \Gamma \vdash P \triangleright \Delta$

$P$  is validated  
against  $\Delta$  and  $\Gamma$

$$\frac{\mathcal{C} \wedge A; \Gamma \vdash P \triangleright \Delta, s: T @ p}{\mathcal{C}; \Gamma \vdash s_k?(v)\{A\}; P \triangleright \Delta, \tilde{s}: k?(v: S)\{A\}; T @ p} \text{ [Rcv]}$$

$$\frac{\mathcal{C} \subset A[e/v] \quad \mathcal{C}; \Gamma \vdash P[e/v] \triangleright \Delta, \tilde{s}: T[e/v] @ p}{\mathcal{C}; \Gamma \vdash s_k!\langle e \rangle(v)\{A\}; P \triangleright \Delta, \tilde{s}: k!(v: S)\{A\}; T @ p} \text{ [Snd]}$$

# SOUNDNESS & COMPLETENESS

## Theorem (Soundness of Validation Rules)

*Let  $P$  be a closed program. Then  $\Gamma \vdash P \triangleright \Delta$  implies  $\Gamma \models P \triangleright \Delta$*

$P$  conditionally simulates  $\Delta$  and  $\Gamma$   
(the simulation only holds for valid inputs)

## Theorem (Completeness of Validation Rules)

*For each closed visible program  $P$ , if  $\Gamma \models P \triangleright \Delta$  then  $\Gamma \vdash P \triangleright \Delta$*

**Theorem (Error Freedom)** *Let  $P$  be a closed program.*

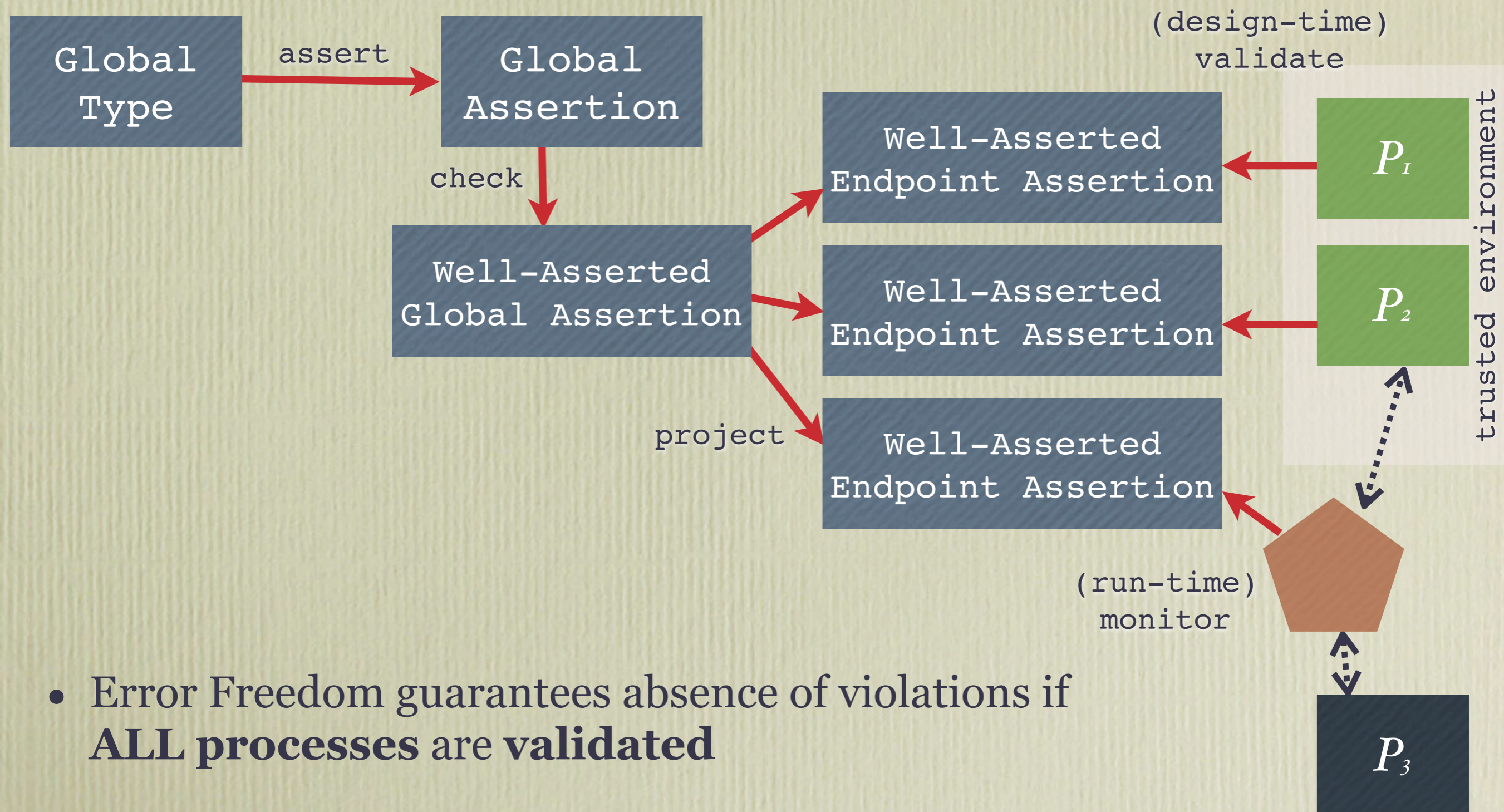
*Suppose*

1.  $\Gamma \vdash P \triangleright \Delta$ ,

2.  $P \xrightarrow{\ell_1 \dots \ell_n} P'$  such that  $\langle \Gamma, \Delta \rangle$  allows  $\ell_1 \dots \ell_n$ .

*Then  $P'$  contains neither  $errH$  nor  $errT$ .*

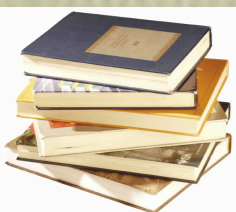
# RUNTIME VALIDATION



- Error Freedom guarantees absence of violations if **ALL processes are validated**
- What about systems with **unsafe** endpoints? **Monitoring!**

# OOI (Ocean Observation Initiative)

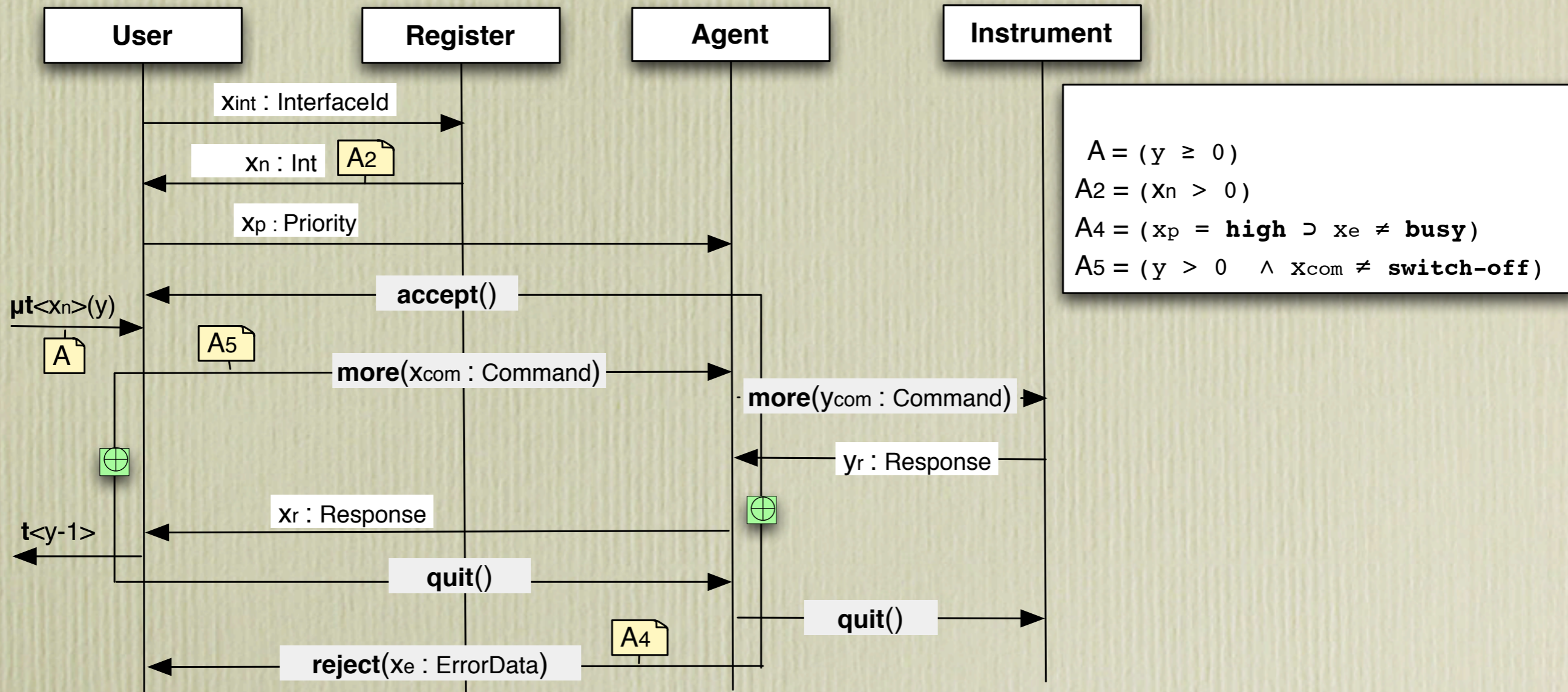
- Enabling environmental science observatories with persistent and interactive capabilities
- OOI cyberinfrastructure (OOI CI) based on **loosely coupled** distributed services and agents (e.g., seafloor instruments, on-shore research stations) communicating through a **common messaging infrastructure**.
- Systems are **large scale, distributed, multi-organizational**
- Applications built form application-level protocols
- Need for global safety ensurance by local validation with possibly unsafe endpoints



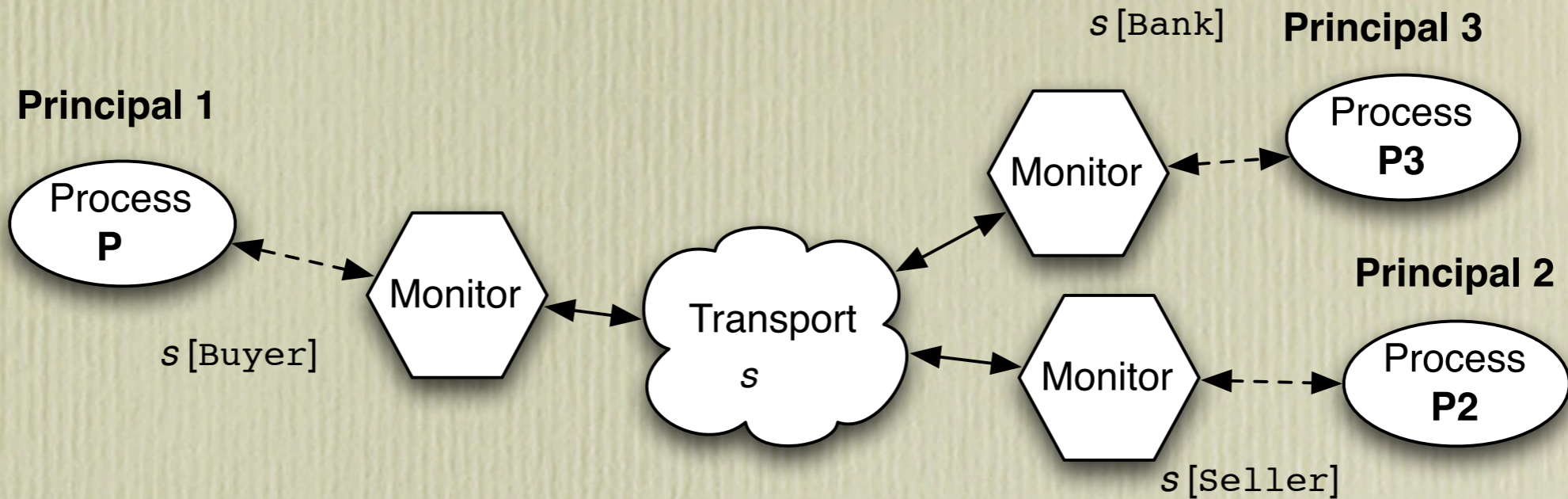
## OOI (Ocean Observatories Initiaitve)

<http://www.oceanleadership.org/programs-and-partnerships/ocean-observing/ooi/>

# INSTRUMENT COMMAND



# THE ARCHITECTURE



$$\mathcal{T} = \text{Buyer}!k(o : \text{Int})\{o \geq 100\}.\mathcal{T}'$$

Process  $P_i$

$$P = s_k!\langle 80 \rangle(o).P' \mid s[\text{Buyer}] : \emptyset$$

$$\downarrow \tau$$

$$P_1 = P' \mid s[\text{Buyer}] : \langle \text{Buyer}, \text{Seller}, \langle 80 \rangle \rangle$$

$$\downarrow s[\text{Buyer}, \text{Seller}]\langle 80 \rangle$$

$$P_2 = P'[80/o] \mid s[\text{Buyer}] : \emptyset$$

Monitor

$$\mathcal{M} = s[\text{Buyer}]^\bullet : \mathcal{T}$$

$$\downarrow \tau$$

$$\mathcal{M} = s[\text{Buyer}]^\bullet : \mathcal{T}$$

$$\downarrow s[\text{Buyer}, \text{Seller}]\langle 80 \rangle$$

$$\frac{\mathcal{M} \vdash v : S, A\{n/v\} \downarrow \text{true}, \mathcal{T} \leadsto \mathbf{p}_2!(v : S)\{A\}; \mathcal{T}'}{\mathcal{M}, s[\mathbf{p}_1]^\bullet : \mathcal{T} \xrightarrow{s[\mathbf{p}_1, \mathbf{p}_2]!(n)} \mathcal{M}, s[\mathbf{p}_1]^\bullet : \mathcal{T}'\{n/v\}}$$

# PROPERTIES

- **Local/global conformance:** a monitored process well- behaves and coherence is preserved in a network
- **Local/global transparency:** monitors do not alter well-behaved interactions
- **Session fidelity:** the interactions of a network are step-by-step conform to the corresponding global types

# PROPERTIES

**Theorem (Local Conformance)**  $\mathcal{M} \models \mathcal{M}[P]$  for all  $\mathcal{M}$  and  $P$

**Theorem (Global Conformance)**  $N \xrightarrow{\ell}_g N'$  with  $N$  coherent implies  $N'$  is coherent

**Theorem (Local Transparency)** If  $\mathcal{M} \models \mathcal{M}^\circ[P]$  then  $\mathcal{M} \models \mathcal{M}^\circ[P] \sim \mathcal{M}[P]$

**Theorem (Global Transparency)** Suppose  $N$  is coherent and locally conformant. Then  $N \sim \text{erase}(N)$

**Theorem (Session Fidelity)** If  $\mathcal{E} \vdash N$  and  $N \xrightarrow{\ell}_g N'$  then  $\mathcal{E} \xrightarrow{\ell}_g \mathcal{E}'$  such that  $\mathcal{E}' \vdash N'$

# CONCLUSIONS

- We enabled DbC for distributed interactions through the elaboration of MPSTs with logic formulae

## *Design time*

- Local validation of global safety
- Sound+relatively complete validation system
- Effectiveness

## *Runtime*

- Local enforcement of global safety with unsafe endpoints
- Prototype: framework for interoperable processes (Scala, Java, OCaml)
- Efficiency

# RELATED WORK

## HML

- M. Berger, K. Honda, and N. Yoshida. *Completeness and logical full abstraction for modal logics for the typed pi-calculus*. ICALP 2008
- M. Dam. *Proof systems for pi-calculus logics*. In *Logic for Concurrency and Synchronisation, Trends in Logic*, 2003

## Contracts

- L. Acciai and M. Borale. *A type system for client progress in a service-oriented calculus*. In *Concurrency, Graphs and Models*, 2008
- M. Bravetti and G. Zavattaro. *A foundational theory of contracts for multi-party service composition*. *Fundamenta Informaticae*, XX:1–28, 2008
- L. Caires and H. T. Vieira. *Conversation types*. ESOP 2009
- G. Castagna and L. Padovani. *Contracts for mobile processes*. CONCUR 2009
- K. Honda, N. Yoshida, and M. Carbone. *Multiparty asynchronous session types*. POPL 2008.

# RELATED WORK

## Assertions for functional programming

- S. Peyton Jones et al. *Composing contracts: an adventure in financial engineering*. ICFP 2000.
- D. Xu and S. Peyton Jones. *Static contract checking for Haskell*. POPL 2009

## DBC

- P. Nienaltowski, B. Meyer, and J. S. Ostroff. *Contracts for concurrency*. Form. Asp. Comput., 21(4):305–318, 2009.

## Corresponding assertions, refinement/dependent types

- E. Bonelli, A. Compagnoni, and E. Gunter. *Correspondence assertions for process synchronization in concurrent communications*. JFC, 15(2):219–247, 2005
- K. Bhargavan, C. Fournet, and A. D. Gordon. *Modular verification of security protocol code by typing*. POPL 2010.
- T. Freeman and F. Pfenning. *Refinement types for ML*. SIGPLAN Not., 26(6):268–277, 1991.
- H. Xi and F. Pfenning. *Dependent types in practical programming*. POPL 1999.