A theory of DbC for multiparty distributed interactions

Laura Bocchi
University of Leicester

Kohei Honda
Queen Mary, University of London

Emilio Tuosto
University of Leicester

Nobuko Yoshida
Imperial College London

Aims & Objectives

- Support description/engineering of multiparty protocols
  - format of messages
  - discipline interactions in conversations
  - values carried in messages
- Devise a theoretical framework
  - analyze protocols
  - specify obligations/guarantees of participants

Reading list

- Assertion methods

C. A. R. Hoare
An axiomatic basis of computer programming
In CACM, 12, 1969.

- Design by Contract (DbC)

B. Meyer
Applying "Design by Contract"
In Computer (IEEE), 25, 1992.

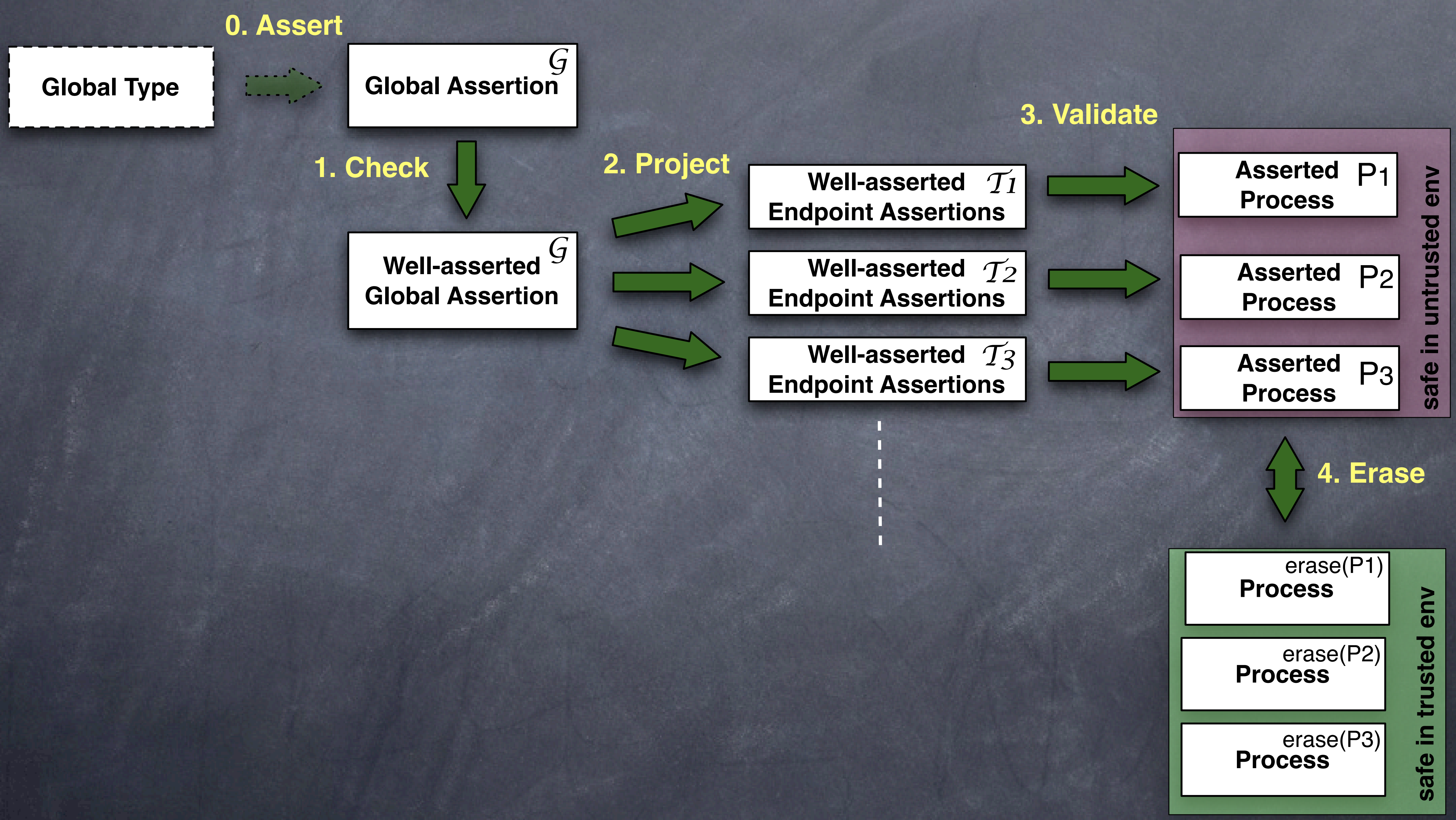
- Multiparty Asynchronous Session Types

K. Honda, N. Yoshida and M. Carbone
Multyparty Asynchronous Session Types
In POPL 2008.

- Global assertions

L. Bocchi, K. Honda, E. Tuosto, and N. Yoshida
A theory of DbC for multiparty distributed
http://www.cs.le.ac.uk/people/lb148/assertedtypes.html

Outline

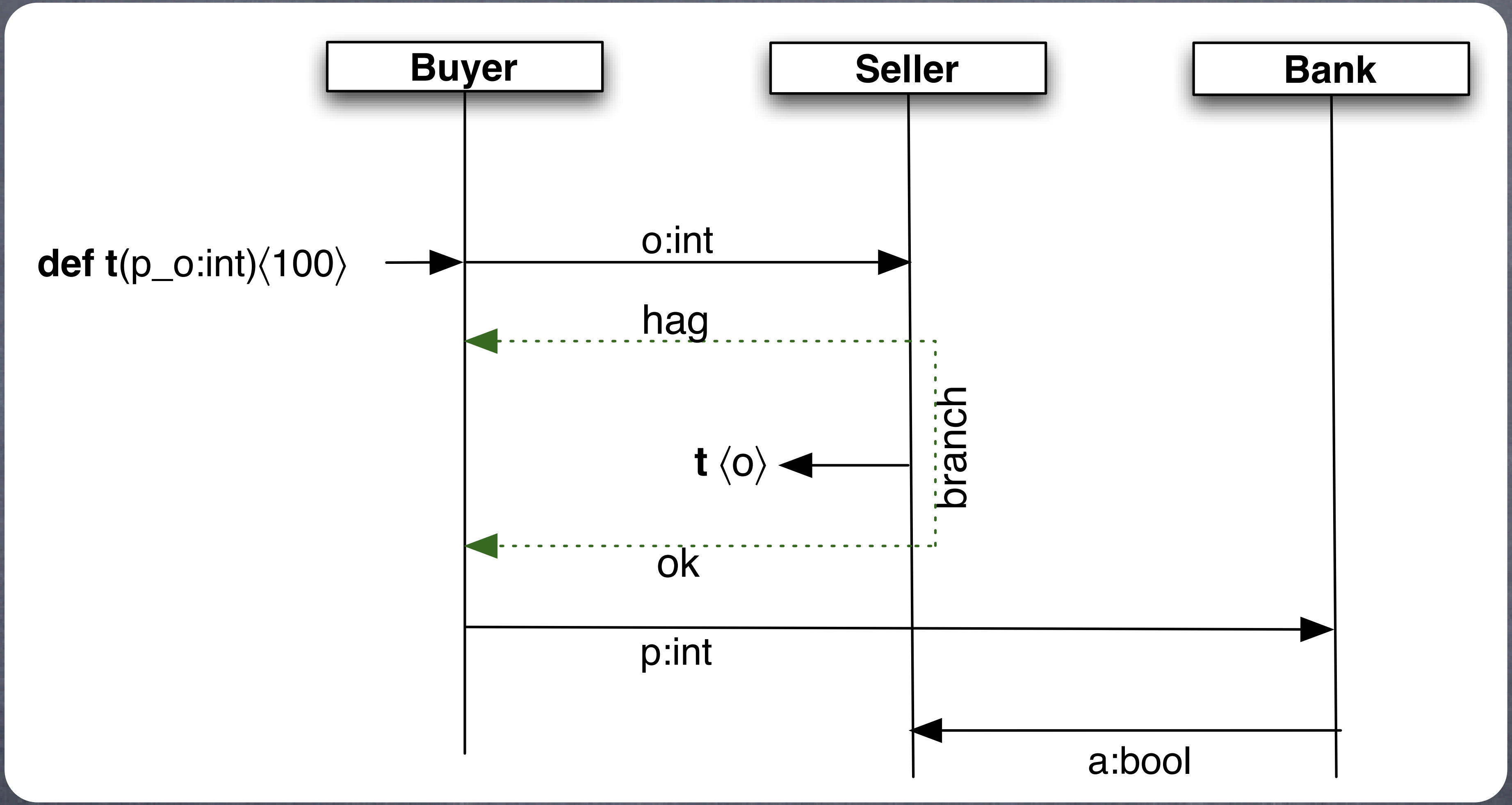


Global Type
0. Assert
Global Assertion $\mathcal{G}$
1. Check
Well-asserted Global Assertion $\mathcal{G}$
2. Project
Well-asserted Endpoint Assertions $\mathcal{T}1$
Well-asserted Endpoint Assertions $\mathcal{T}2$
Well-asserted Endpoint Assertions $\mathcal{T}3$
3. Validate
Asserted Process P1
Asserted Process P2
Asserted Process P3
safe in untrusted env
4. Erase
erase(P1) Process
erase(P2) Process
erase(P3) Process
safe in trusted env

Design by Contract

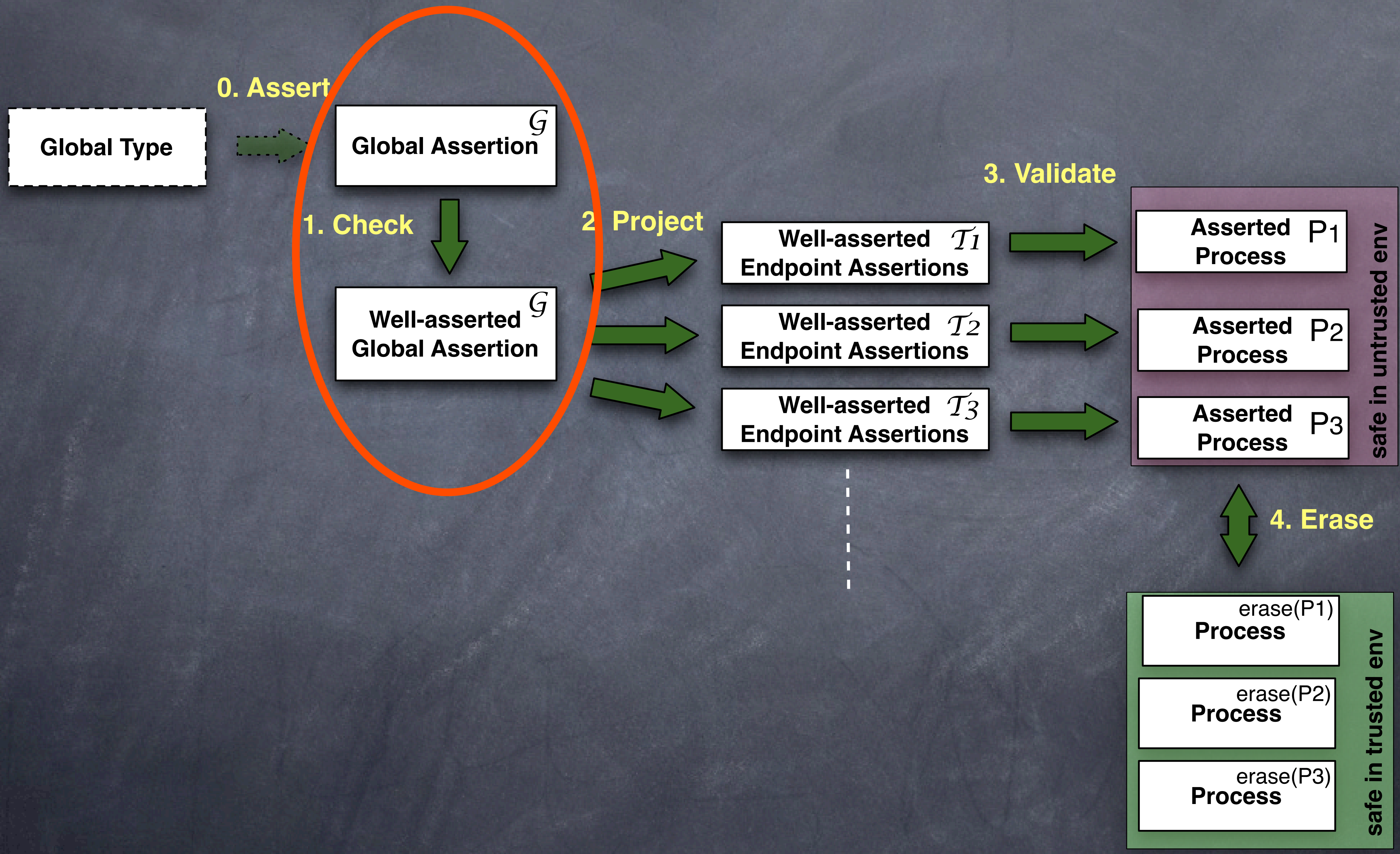
- Type signatures to constraint computation
  - the method m of an object of class C should be invoked with a string and an integer; m will return (if ever) a string
- DbC = Types + Assertions
  - if m is invoked with a string representing a date $2007 \leq d \leq 2008$ and an integer $n \leq 1000$ then it will (if ever) return the date n days after d
- In a distributed setting each party has
  - guarantees (e.g., on the content of the received messages)
  - obligations (e.g., on the content of the sent messages)

A simple global type



μ t(p_o : int)⟨100⟩.
Buyer → Seller : ChSeller (o : int).
Seller → Buyer : ChBuyer {
ok: Buyer → Bank : ChBank (p : int). Bank → Seller : ChSeller(a : bool),
hag: t⟨o⟩
}

Outline



0. Assert
Global Type
Global Assertion $\mathcal{G}$
1. Check
Well-asserted Global Assertion $\mathcal{G}$
2. Project
Well-asserted Endpoint Assertions $\mathcal{T}1$
Well-asserted Endpoint Assertions $\mathcal{T}2$
Well-asserted Endpoint Assertions $\mathcal{T}3$
3. Validate
Asserted Process P1
Asserted Process P2
Asserted Process P3
safe in untrusted env
4. Erase
erase(P1) Process
erase(P2) Process
erase(P3) Process
safe in trusted env

Syntax for Global Assertions

$$
\begin{array}{lll}
\mathcal{G} & ::= & \mathtt{p} \to \mathtt{p}' : k\ (\tilde{v} : \tilde{S})\{A\}.\mathcal{G} \\
& \mid & \mathtt{p} \to \mathtt{p}' : k\ \{\{A_j\}l_j : \mathcal{G}_j\}_{j \in J} \\
& \mid & \mu\ \mathbf{t}(\ldots v_i : S_i\ @\ \mathtt{L_i} \ldots)\langle \tilde{e} \rangle \{A\}.\mathcal{G} \\
& \mid & \mathbf{t}\langle \tilde{e} \rangle \\
& \mid & \mathcal{G}, \mathcal{G}' \\
& \mid & \mathsf{end}
\end{array}
$$

$$
\begin{array}{lll}
S & ::= & \mathsf{bool} \mid \mathsf{int} \mid \ldots \mid \mathcal{G} \\
\mathtt{L} & ::= & \{\mathtt{p}, \mathtt{p}'\}
\end{array}
$$

What is A?

Logical Language

- The investigation of the most suitable logic is left as a future work
  - The logical language is likely to be application dependent
- good candidates are first-order decidable logics (e.g. Presburger arithmetic)

$$A \quad ::= \quad e_1 = e_2 \ | \ e_1 > e_2 \ | \ \phi(e_1, \ldots, e_n) \quad | \quad A_1 \wedge A_2 \ | \ \neg A \ | \ \exists v(A)$$

A global assertion

G_{hag} = μ t(o : int@{Buyer, Seller})⟨10⟩{o≥10}.

Buyer → Seller: ChSeller (p : int) {p≥10}.

Seller → Buyer: ChBuyer{

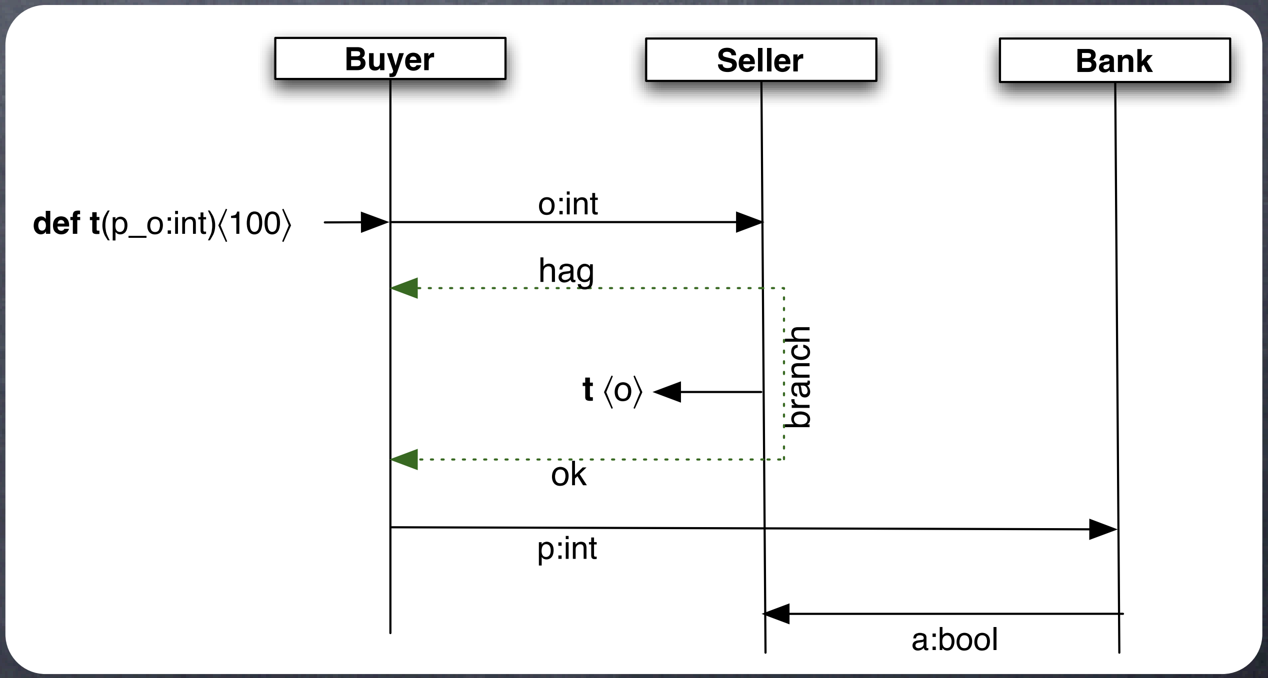
{true} ok:

Buyer → Bank: ChBank (c : int) {c=p}.

Bank → Seller: ChSeller (a : bool) {true},

{p>o} hag: t⟨p⟩

}



Correctness of Global Assertions

- Global assertions must respect 3 principles
  - History sensitivity
  - Locality
  - Temporal satisfiability

History sensitivity principle

An interaction predicate can only constraint variables known by the sender

~~Alice → Bob : ChB (u:int) {true}.~~
~~Bob → Carol : ChC (v:int) {true}.~~
~~Carol → Alice : ChA (z:int) {z<u}~~

Carol cannot guarantee z<u as she doesn't know u

Alice → Bob : ChB (u:int) {true}.
Bob → Carol : ChC (v:int) {v<u}.
Carol → Alice : ChA (z:int) {z<v}

z indirectly depends on u...

...but Carol can choose the right value since she knows v and the predicates ensure that the dependencies are respected

Locality principle

Predicates can only constraint variables which they introduce

~~Alice → Bob : ChB (u:int) {u>0}.~~
~~Bob → Carol : ChC (v:int) {true}.~~
~~Carol → Alice : ChA (z:int) {z≥v ∧ v>1}~~

Carol strengthens the constraints on v withough being entitled

Alice → Bob : ChB (u:int) {u>0}.
Bob → Carol : ChC (v:int) {true}.
Carol → Alice : ChA (z:int) {z≥v ∧ z>1}

Temporal-satisfiability principle

- For each value satisfying a predicate A
  - there is always a branch enabled
  - for each subsequent predicate A′, it is always possible to find values that satisfy A′

Alice → Bob : ChB (u:int) {u<10}.
Bob → Alice : ChA (v:int) {v<u ∧ v>6}

had Alice sent 6 or 7, Bob couldn't meet his obbligation!

Being true to our principles

- HSP can be statically checked

$$\frac{\Gamma, \tilde{v} : \tilde{S} \,@\, \{\mathtt{p}, \mathtt{p}'\} \vdash \mathcal{G} \;\; \forall u \in var(A) \setminus \tilde{v}, \Gamma \vdash u \,@\, \mathtt{p}}{\Gamma \vdash \mathtt{p} \to \mathtt{p}' \colon k\; (\tilde{v} : \tilde{S})\{A\}.\mathcal{G}} \qquad \frac{\forall j \in J,\; \Gamma \vdash \mathcal{G}_j \;\; \forall u \in \bigcup_{j \in J} var(A_j) \Gamma \vdash u \,@\, \mathtt{p}}{\Gamma \vdash \mathtt{p} \to \mathtt{p}' \colon k\; \{\{A_j\} l_j : \mathcal{G}_j\}_{j \in J}}$$

$$\frac{\Gamma \vdash \mathcal{G} \quad \Gamma \vdash \mathcal{G}'}{\Gamma \vdash \mathcal{G}, \mathcal{G}'} \qquad \frac{}{\Gamma \vdash \mathsf{end}} \qquad \frac{\Gamma \vdash e_1 : S_1 \,@\, \mathtt{L_1} \;\; \cdots \;\; \Gamma \vdash e_n : S_n \,@\, \mathtt{L_n}}{\Gamma, \mathbf{t} : S_1 \,@\, \mathtt{L_1} \ldots S_n \,@\, \mathtt{L_n} \vdash \mathbf{t}\langle \tilde{e} \rangle}$$

$$\frac{\Gamma' = \Gamma, \mathbf{t} : S_1 \,@\, \mathtt{L_1} \ldots S_n \,@\, \mathtt{L_n} \quad \Gamma' \vdash \mathcal{G} \quad dom(\Gamma') \supseteq var(A) \quad \forall i. \Gamma \vdash v_i : S_i \,@\, \mathtt{L_i}, e_i : S_i \,@\, \mathtt{L_i}}{\Gamma \vdash \mu \mathbf{t} \langle \tilde{e} \rangle (v_1 : S_1 \,@\, \mathtt{L_1}, \ldots, v_n : S_n \,@\, \mathtt{L_n})\{A\}.\mathcal{G}}$$

Being true to our principles

- TSP implies LP, and we give a checker GSat(G,A)

1. $\mathcal{G} = \mathtt{p}_1 \to \mathtt{p}_2 \colon k(\tilde{v} : \tilde{S})\{A'\}.\mathcal{G}' \begin{cases} \text{if } A \supset \exists \tilde{v}(A') \text{ then } GSat(\mathcal{G}, A) = GSat(\mathcal{G}', A \wedge A') \\ \text{otherwise } GSat(\mathcal{G}, A) = \mathsf{false} \end{cases}$

1. $\mathcal{G} = \mathtt{p}_1 \to \mathtt{p}_2 \colon k\,(\tilde{v} : \tilde{S})\{A'\}.\mathcal{G}'$

$$\begin{cases} \text{if } A \supset \exists \tilde{v}(A') \text{ then } GSat(\mathcal{G}, A) = GSat(\mathcal{G}', A \wedge A') \\ \text{otherwise } GSat(\mathcal{G}, A) = \mathsf{false} \end{cases}$$

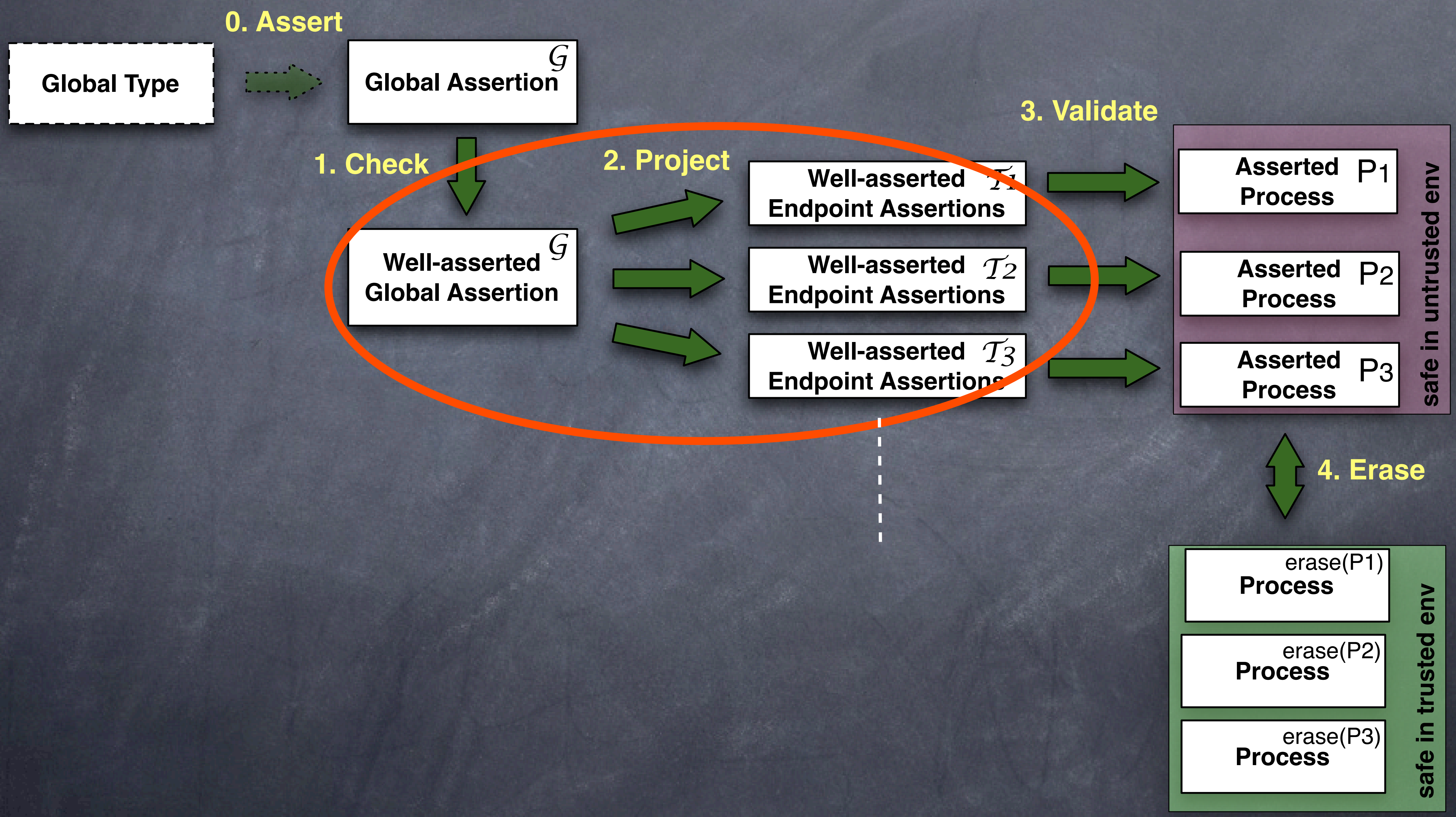
2. ...

6. $\mathcal{G} = \mathsf{end}$ then $GSat(\mathcal{G}, A) = \mathsf{true}$

Well-assertedness

- A global assertion G is well-asserted when
  - G is history-sensitive and
  - GSat(G,true) = true

Outline



0. Assert
Global Type
Global Assertion
1. Check
Well-asserted Global Assertion
2. Project
Well-asserted Endpoint Assertions
Well-asserted Endpoint Assertions
Well-asserted Endpoint Assertions
3. Validate
Asserted Process P1
Asserted Process P2
Asserted Process P3
safe in untrusted env
4. Erase
erase(P1) Process
erase(P2) Process
erase(P3) Process
safe in trusted env

End-point assertions

$$
\begin{array}{lcl}
\mathcal{T} & ::= & k!(\tilde{v} : \tilde{S})\{A\};\mathcal{T} \\
 & \mid & k?(\tilde{v} : \tilde{S})\{A\};\mathcal{T} \\
 & \mid & k \oplus \{\{A_j\}l_i : \mathcal{T}_i\}_{i\in I} \\
 & \mid & k\&\{\{A_i\}l_i : \mathcal{T}_i\}_{i\in I} \\
 & \mid & \mu\mathbf{t}\langle\tilde{e}\rangle(\tilde{v} : \tilde{S})\{A\}.\mathcal{T} \\
 & \mid & \mathbf{t}\langle\tilde{e}\rangle \\
 & \mid & \mathsf{end}
\end{array}
$$

Endpoint assertions specify the behaviour of processes involved in a session.

Projections & “third parties”

Seller → Buyer : ChBuyer(p : int) {p > 10}.

Buyer → Bank : ChBank (c : int) {c ≥ p}

A too naive projection wrt Bank would give

ChBank?(c:Int) {c ≥ p}

which is meaningless because Bank ignores the value of p so it cannot check if Buyer meets its obbligation.

Projecting global assertions

$$Proj(\mathtt{p}_1 \to \mathtt{p}_2 \colon k\,(\tilde{v} : \tilde{S})\{A\}.\mathcal{G}', A_{Proj}, \mathtt{p}) = \begin{cases} k!(\tilde{v} : \tilde{S})\{A\}.\mathcal{G}_{Proj} & \text{if } \mathtt{p} = \mathtt{p}_1 \\ k?(\tilde{v} : \tilde{S})\{\exists V_{ext}(A \wedge A_{Proj})\}.\mathcal{G}_{Proj} & \text{if } \mathtt{p} = \mathtt{p}_2 \\ \mathcal{G}_{Proj} & \text{otw} \end{cases}$$

$$\mathcal{G}_{Proj} = Proj(\mathcal{G}', A \wedge A_{Proj}, \mathtt{p}) \text{ and } V_{ext} = var(A_{Proj}) \setminus \mathcal{I}(\mathcal{G}) \restriction \mathtt{p}$$

Seller → Buyer : ChBuyer(p : int) {p > 10}.

Buyer → Bank : ChBank (c : int) {c ≥ p}

projected wrt **Bank**

ChBank?(c:Int) {∃p. p≥10 ∧ c ≥ p}

Projection in action

T_{hag} = μ t(o : int)⟨10⟩ {o≥10}.

ChSeller?(p : int) {p≥10 ∧ o≥10};

ChBuyer⊕{

{true} ok: ChSeller?(a : bool) {∃c. p≥10 ∧ o≥10 ∧ c=p}

{p>o} hag: t⟨o⟩,

}

G_{hag} = μ t(o : int@{Buyer, Seller})⟨10⟩{o≥10}.

Buyer → Seller: ChSeller (p : int) {p≥10}.

Seller → Buyer: ChBuyer{

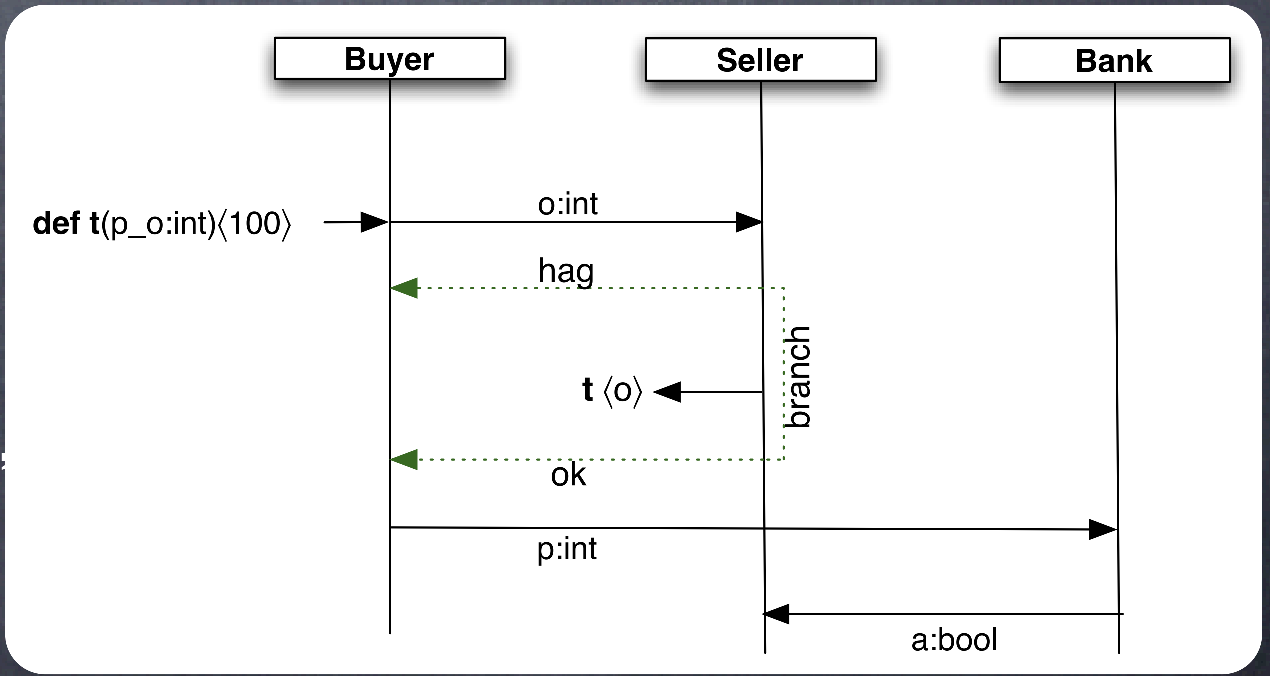
{true} ok:

Buyer → Bank: ChBank (c : int) {c=p}.

Bank → Seller: ChSeller (a : bool) {true},

{p>o} hag: t⟨p⟩

}



Well-assertedness for endpoint assertions

- Similarly to global assertions, we define LSat(T,A) to check if T is satisfiable under assertion A
  - Notice that for endpoint assertions only TSP is important as
    - TSP implies LP
    - HSP is vacuously guaranteed
- Proj(G,true,p) preserves well-assertedness

Asserted processes

$P ::= \overline{a}[2..n]\,(\tilde{s}).P$	request		$\mid \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q$	conditional
$\mid a[p]\,(\tilde{s}).P$	accept			error
$\mid s!\langle\tilde{e}\rangle(\tilde{v})\{A\}; P$	send		$\mid P \mid Q$	parallel
$\mid s?(\tilde{v})\{A\}; P$	reception		$\mid \mathbf{0}$	idle
$\mid s \lhd \{A\}l; P$	select		$\mid (\nu a)P$	hiding
$\mid s \rhd \{\{A_i\}l_i : P_i\}_{i\in I}$	branch		$\mid \mathsf{def}\ D\ \mathsf{in}\ P \mid X\langle\tilde{e}\tilde{s}\rangle$	rec def/call
$D ::= \{\langle X_i(\tilde{v}_i\tilde{s}_i) = P_i\rangle\}_{i\in I}$	rec dec		$n ::= a \mid \mathsf{true} \mid \mathsf{false}$	values
$e ::= n \mid e \wedge e' \mid \neg e\ \ldots$	expressions			

Semantics

$$\overline{a}[2..\mathtt{n}]\,(\tilde{s}).P \mid a[2]\,(\tilde{s}).P_2 \mid \ldots \mid a[\mathtt{n}]\,(\tilde{s}).P_n \to (\nu\tilde{s})(P_1 \mid P_2 \mid \ldots \mid P_n \mid s_1 : \emptyset \mid \ldots \mid s_n : \emptyset)$$

$$s!\langle\tilde{e}\rangle(\tilde{v})\{A\}; P \mid s : \tilde{h} \to P[\tilde{\mathtt{n}}/\tilde{v}] \mid s_k : \tilde{h} \cdot \tilde{\mathtt{n}} \quad (\tilde{e} \downarrow \tilde{\mathtt{n}} \wedge A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true})$$

$$s?(\tilde{v})\{A\}; P \mid s : \tilde{\mathtt{n}} \cdot \tilde{h} \to P[\tilde{\mathtt{n}}/\tilde{v}] \mid s : \tilde{h} \quad (A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{true})$$

$$s \rhd \{\{A_i\}l_i : P_i\}_{i \in I} \mid s : l_j \cdot \tilde{h} \to P_j \mid s : \tilde{h} \quad (j \in I \text{ and } A_j \downarrow \mathsf{true})$$

$$s \lhd \{A\}l; P \mid s : \tilde{h} \to P \mid s : \tilde{h} \cdot l \quad (A \downarrow \mathsf{true})$$

$$\mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \to P \quad (e \downarrow \mathsf{true}) \qquad \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ Q \to Q \quad (e \downarrow \mathsf{false})$$

$$\mathsf{def}\ D\ \mathsf{in}\ C[X\langle\tilde{e}\tilde{s}\rangle] \to \mathsf{def}\ D\ \mathsf{in}\ Q (\text{where } \langle X\langle\tilde{v}\tilde{s}\rangle = P\rangle \in D \text{ and } C[P[\tilde{e}/\tilde{v}]] \to Q)$$

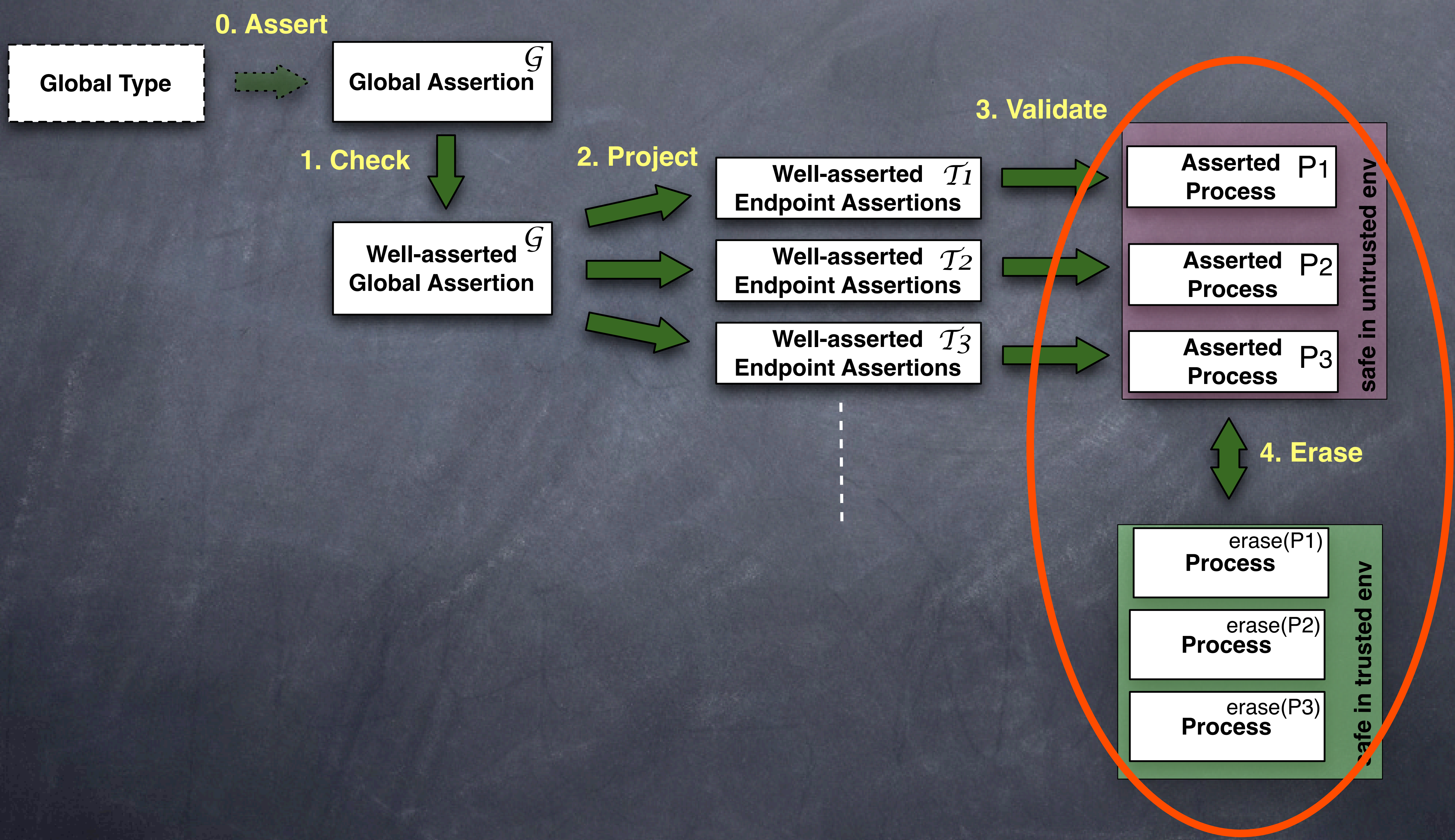
$$s!\langle\tilde{\mathtt{n}}\rangle(\tilde{v})\{A\}; P \to \mathsf{errH} \quad (A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{false})$$

$$s?(\tilde{v})\{A\}; P \mid s : \tilde{\mathtt{n}} \cdot \tilde{h} \to \mathsf{errT} \mid s : \tilde{h} \quad (A[\tilde{\mathtt{n}}/\tilde{v}] \downarrow \mathsf{false})$$

$$s \rhd \{\{A_i\}l_i : P_i\}_{i \in I} \mid s : l_j \cdot \tilde{h} \to \mathsf{errT} \mid s : \tilde{h} \quad (j \in I \text{ and } A_j \downarrow \mathsf{false})$$

$$s \lhd \{A\}l; P \to \mathsf{errH} \quad (A \downarrow \mathsf{false})$$

Outline



0. Assert
Global Type
Global Assertion $\mathcal{G}$
1. Check
Well-asserted Global Assertion $\mathcal{G}$
2. Project
Well-asserted Endpoint Assertions $\mathcal{T}1$
Well-asserted Endpoint Assertions $\mathcal{T}2$
Well-asserted Endpoint Assertions $\mathcal{T}3$
3. Validate
Asserted Process P1
Asserted Process P2
Asserted Process P3
safe in untrusted env
4. Erase
erase(P1) Process
erase(P2) Process
erase(P3) Process
safe in trusted env

Validating asserted processes

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

under the assertion environment C and the sorting Γ, P is validated w.r.t. the assertion assignment Δ

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

$$\frac{\mathcal{C} \wedge A_i, \Gamma \vdash P_i \triangleright \Delta, \tilde{s} : \mathcal{T}_i \,@\, \mathtt{p} \quad \forall i \in I}{\mathcal{C};\Gamma \vdash \ s_k \rhd \{\{A_i\}l_i : P_i\}_{i\in I} \triangleright \Delta, \tilde{s} : k\&\{\{A_i\}l_i : \mathcal{T}_i\}_{i\in I} \,@\, \mathtt{p}}$$

Main Results

- Validated processes can be “simulated” by their end-point assertions
- End-point assertions do not yield errors (by construction)
- A validated process never reaches errors
- Validation is decidable

Main results 2

- In a trusted environment, all assertions of validate processes can be turned into true
- A monitor can be automatically deduced from validated processes (it has to check sent/selection messages)
- In an untrusted environment, the monitor may guard processes and help in debugging

Future work

- Study properties of suitable logics for global assertions
  - tractability/decidability
  - complexity
- Play with implementations
- Apply this context to financial protocols

Thank you…