

Trusting Outsourced Components in Flight-Critical Systems

Temesghen Kahsai

NASA Ames / CMU



Carnegie
Mellon
University

Joint work with ...



Dr. Kasper Luckow
(*CMU*)



Dr. Arie Gurfinkel
(*SEI / CMU*)



Prof. Cesare Tinelli
(*The University of Iowa*)



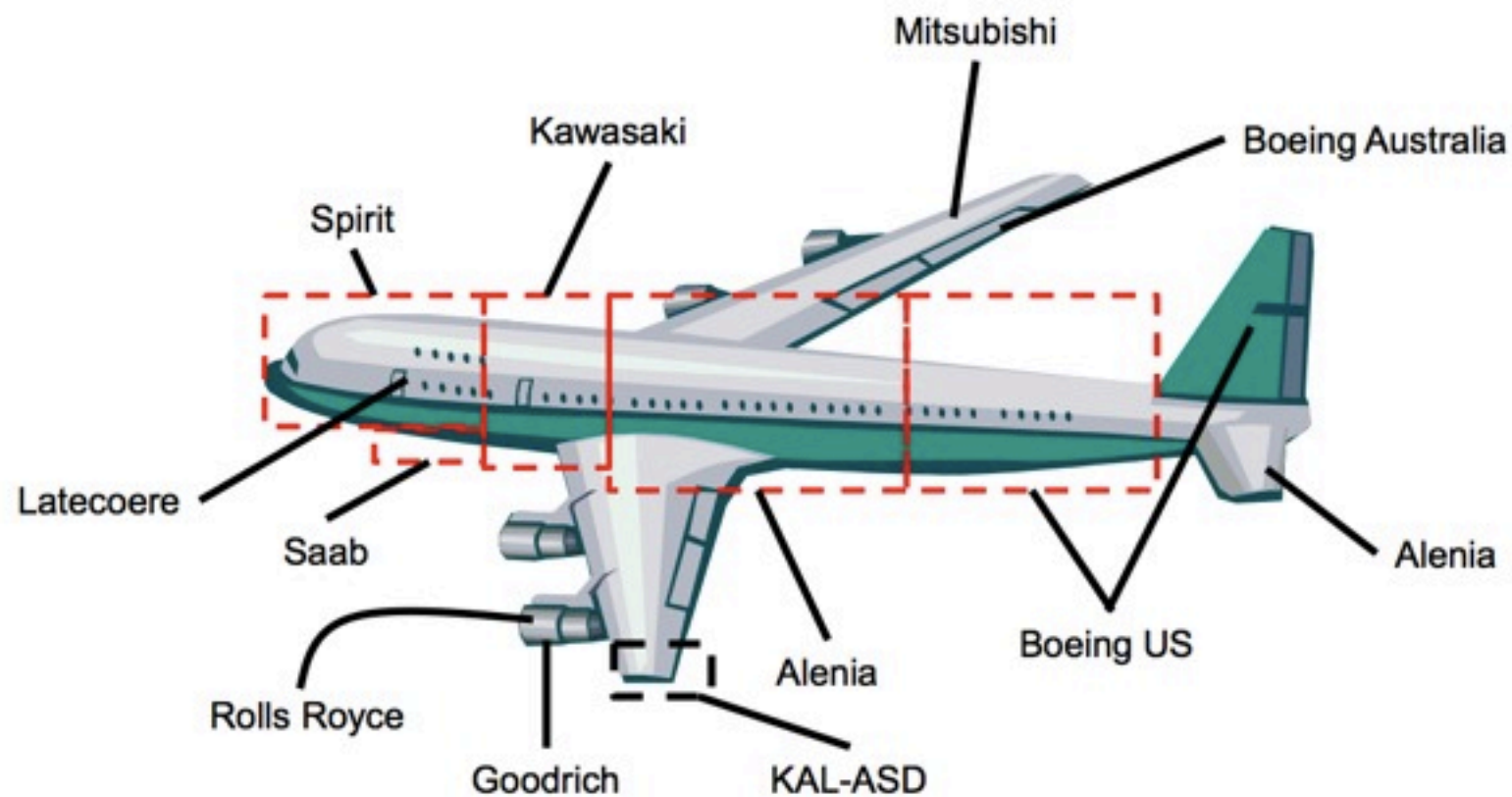
Dr. Adrien Champion
(*The University of Iowa*)

Outsourcing in the aerospace industry

Outsourcing in the aerospace industry

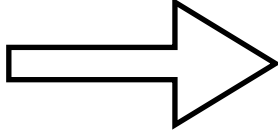


The Boeing 787 Dreamliner's flight critical, embedded software is built on the WRS ARINC 653 system and is assembled from software components by multiple subcontractors

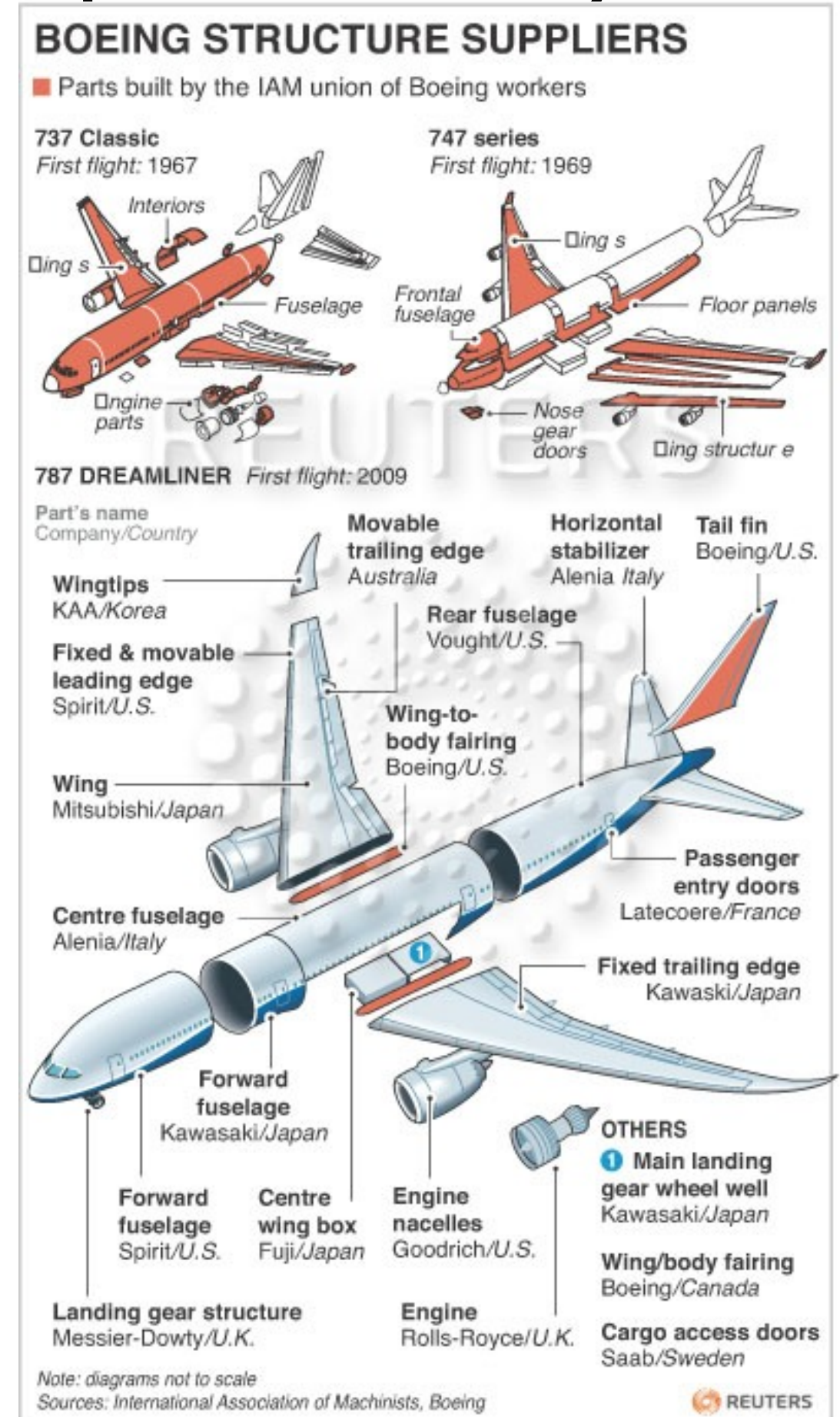


Source: Boeing / Reuters

Outsourcing in the aerospace industry

Boeing 737 and 747 = 35-50% 

Boeing 787 = 70% 



Outsourcing in the aerospace industry



- The delivery date was pushed back 4 times and was late more than 4 years
- The aft fuselage consisted of 6,000 components, and many of those components failed to conform to Boeing's specified tolerances, resulting in **significant cost** and **schedule delays**
- The first Dreamliner to arrive at the company's assembly place was missing tens of thousands of parts

Outsourcing in the aerospace industry



Outsourcing in the aerospace industry



- January 2013: 50 Dreamliner was grounded due to issues with the lithium-ion batteries.
- On balance with just under 60 aircraft in service, the 787 has had 6 reported mechanical incidents in 2013.
- All the individual parts worked in **isolation**. But, together, **under certain circumstances**, the parts failed.

Outsourcing in the aerospace industry



*“While we can’t completely eliminate failures, the answer lies in system engineering. This involves a process of careful design and architecture ... as well as a **staged integration of the entire system, and extensive qualification, verification and validation testing.**” Prof. S. Eppinger (MIT)*

* <http://executive.mit.edu/blog/will-risk-result-in-reward-for-boeings-dreamliner>

This talk ...



... outsourcing in flight critical software

... virtual integration of outsourced components

Assurance of Flight Critical Systems (FCS)

Aim:

- Develop multidisciplinary V&V tools and techniques that advance **safety assurance and certification**
- Flight-critical systems: any systems that **directly controls** the safe conduct of an aircraft's flight, i.e. **air** and **ground** systems

Technical Challenges:

1. Argument-based safety assurance
2. Integrated distributed systems
3. Authority and Autonomy
4. Software intensive systems
5. Assessment environments

Assurance of Flight Critical Systems (FCS)

Aim:

- Develop multidisciplinary V&V tools and techniques that advance **safety assurance and certification**
- Flight-critical systems: any systems that **directly controls** the safe conduct of an aircraft's flight, i.e. **air** and **ground** systems

Technical Challenges:

1. **Argument-based safety assurance**
2. Integrated distributed systems
3. Authority and Autonomy
4. **Software intensive systems**
5. Assessment environments

Assurance of Flight Critical Systems (FCS)

Topic: “Support for verification of black-box FCS”

Assurance of Flight Critical Systems (FCS)

Topic: “Support for verification of black-box FCS”

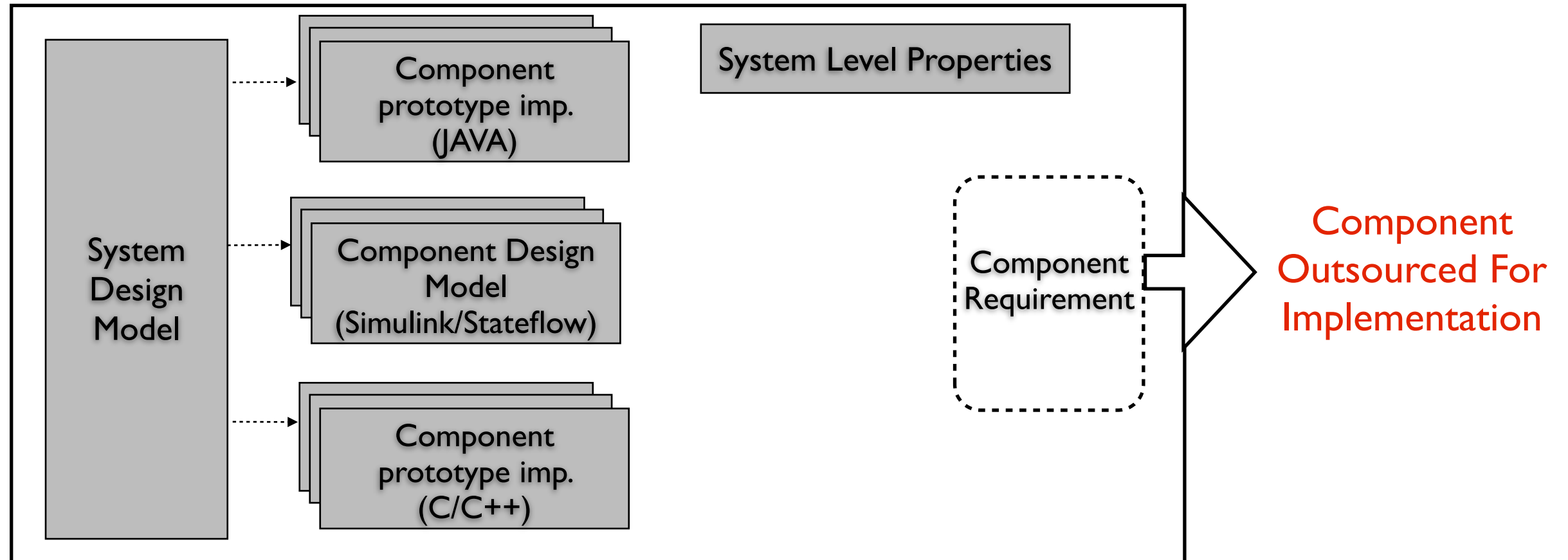
Context:

- More and more **design** and **implementation** of FCS is contracted out to external companies
- Example: FAA contracts out the implementation of most of the **air traffic systems**
- Integration of FCS from **Commercial Off-The-Shelf (COTS)** components
- Current technique is based on **black-box testing**
- Many of those systems have been first prototyped in-house
- Example: Many FAA systems has been prototyped by **MIT Lincoln Lab**, **NASA** etc. (e.g. TCAS, ACAS-X, TSAFE, etc.)

Assurance of Flight Critical Systems (FCS)

Topic: “Support for verification of black-box FCS”

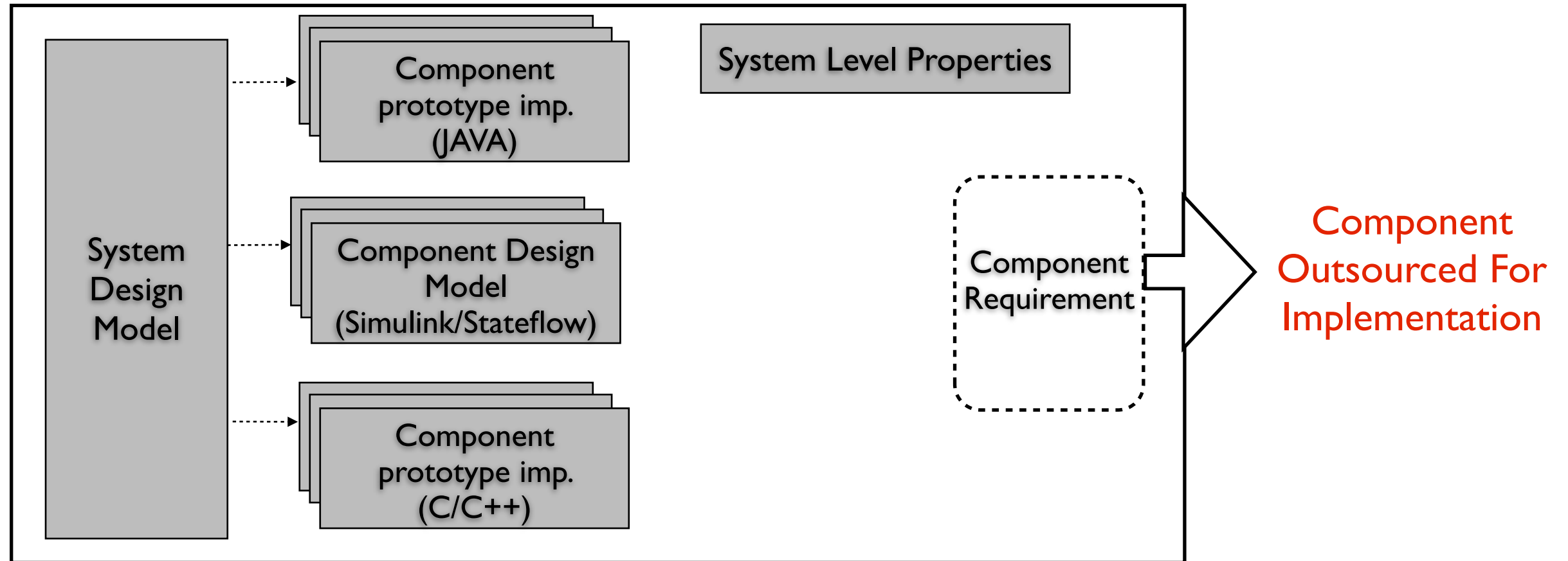
In house prototyping



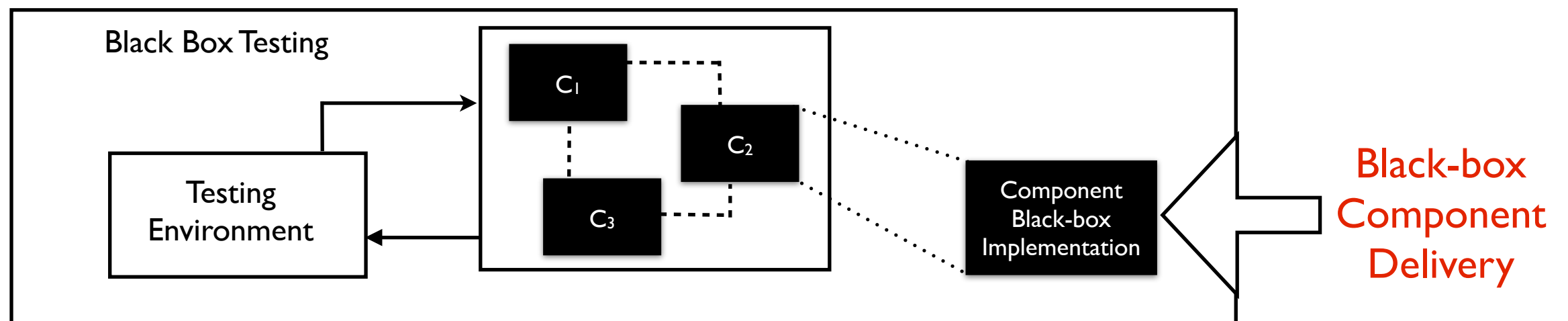
Assurance of Flight Critical Systems (FCS)

Topic: “Support for verification of black-box FCS”

In house prototyping



In house assembling



Contract-based Compositional verification for outsourced FCS (CoCo)



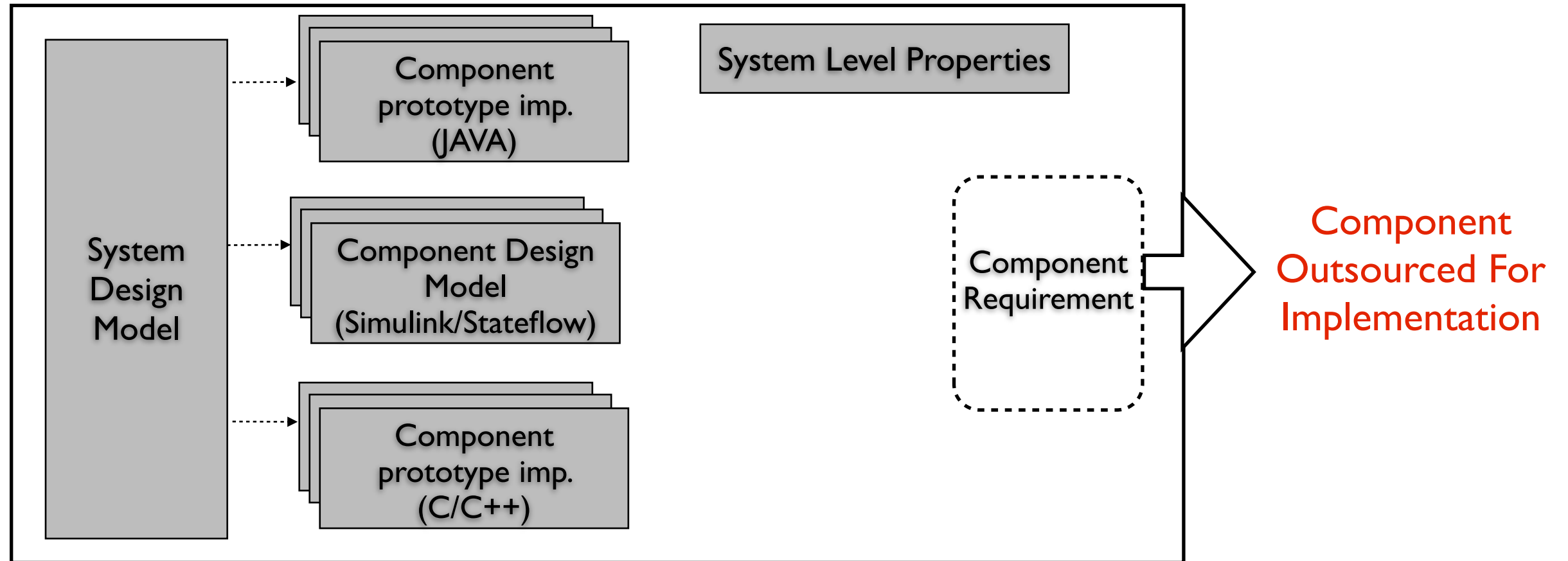
Outline

- Two stage solution for virtual integration
- 1st stage: contract generation
- 2nd stage: contract compliance
- Flight critical system case studies

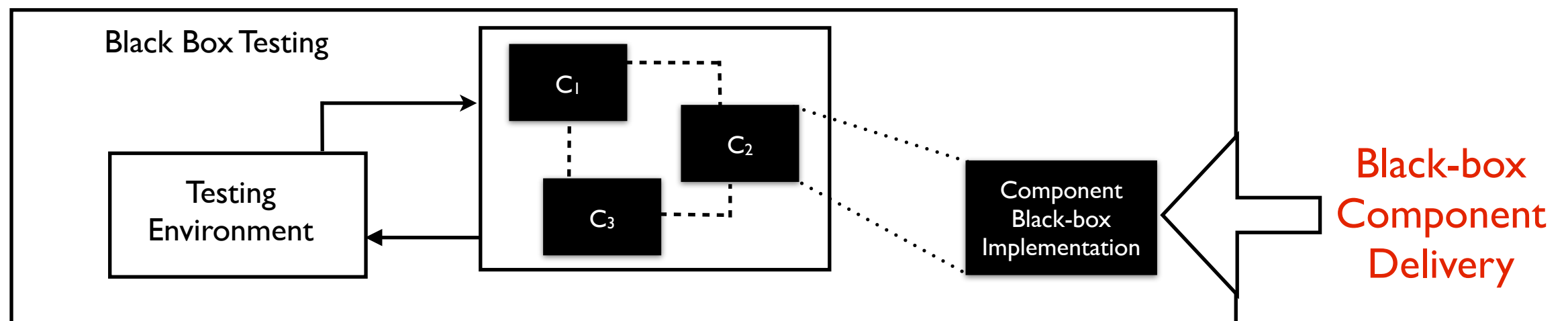
Two Stage solution for virtual integration

Two Stage solution for virtual integration

In house prototyping

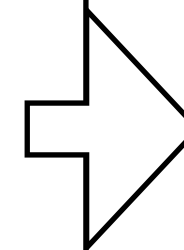
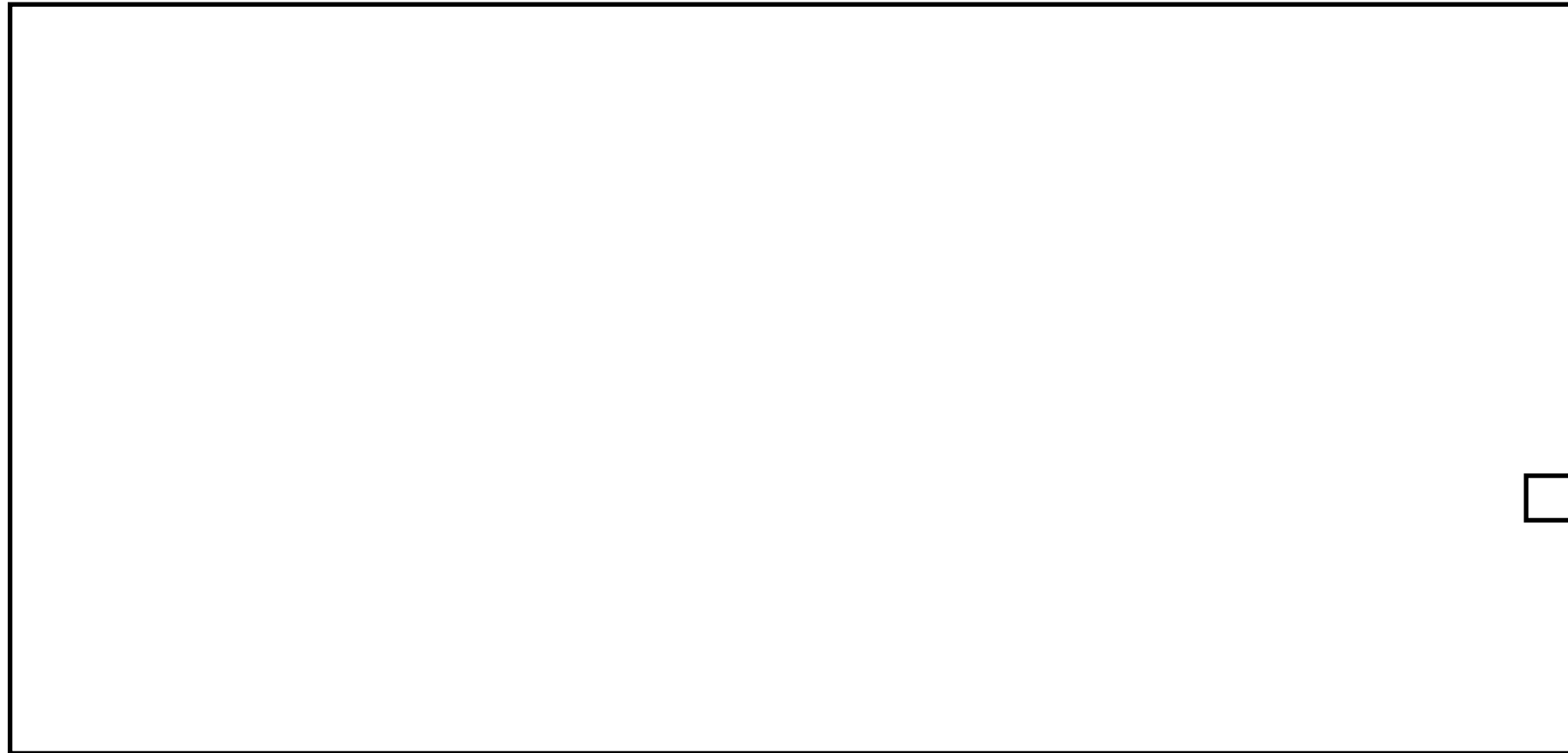


In house assembling



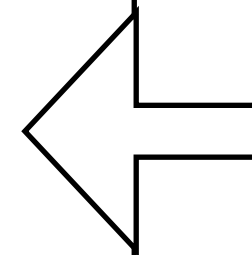
Two Stage solution for virtual integration

Pre-Delivery Stage



Component
Outsourced For
Implementation

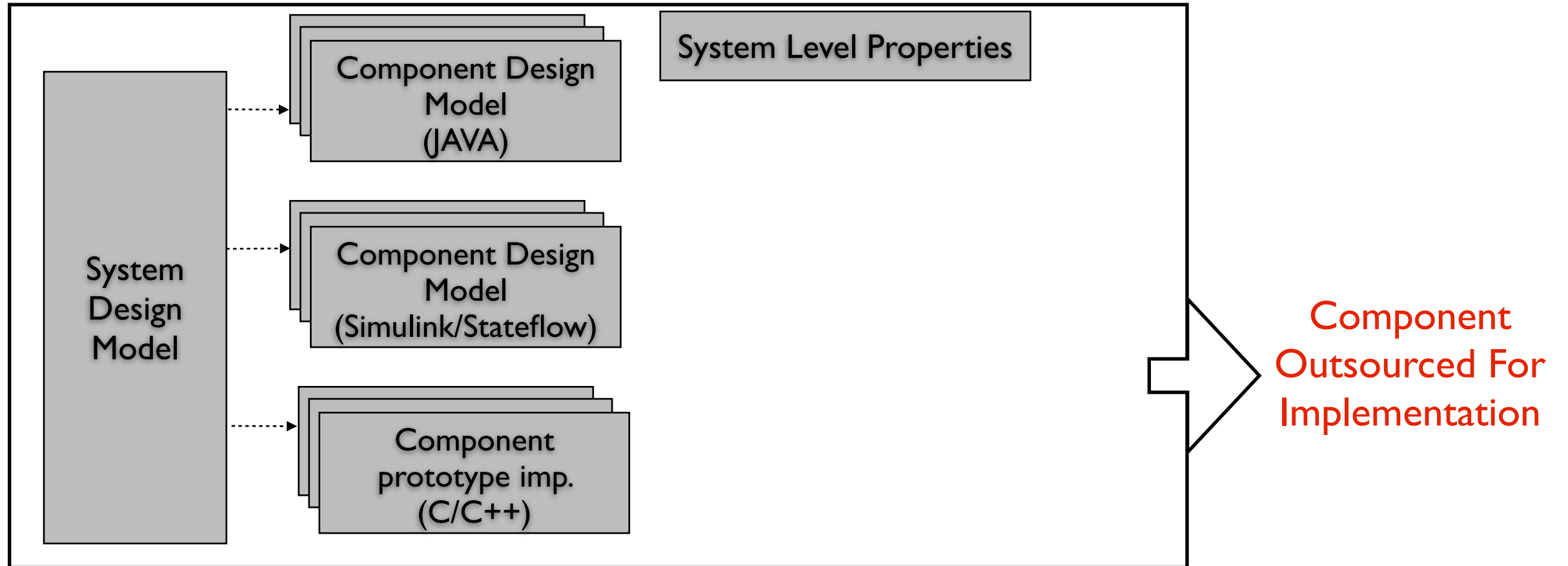
Post-Delivery stage



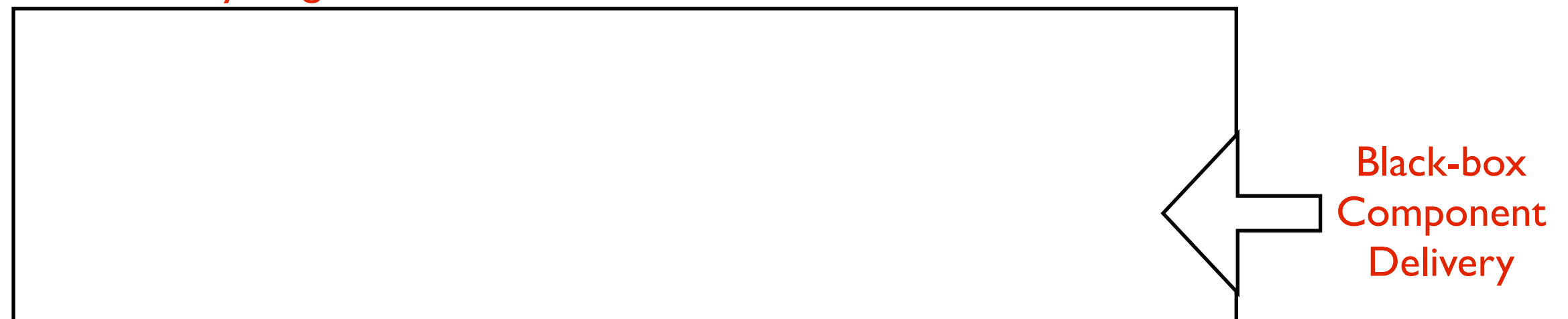
Black-box
Component
Delivery

Two Stage solution for virtual integration

Pre-Delivery Stage

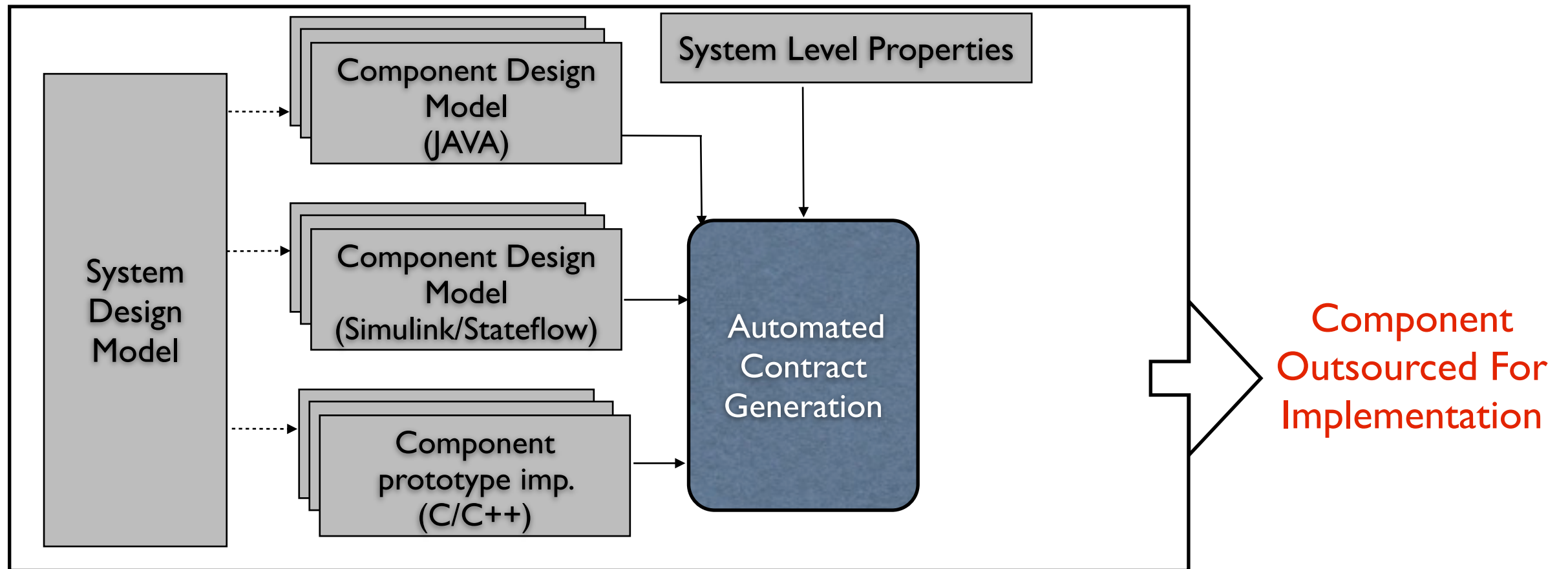


Post-Delivery stage

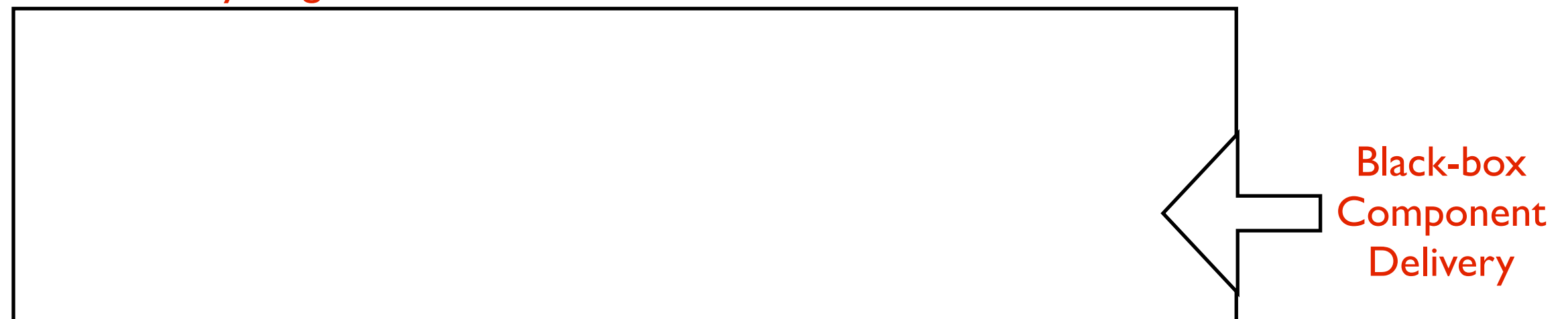


Two Stage solution for virtual integration

Pre-Delivery Stage

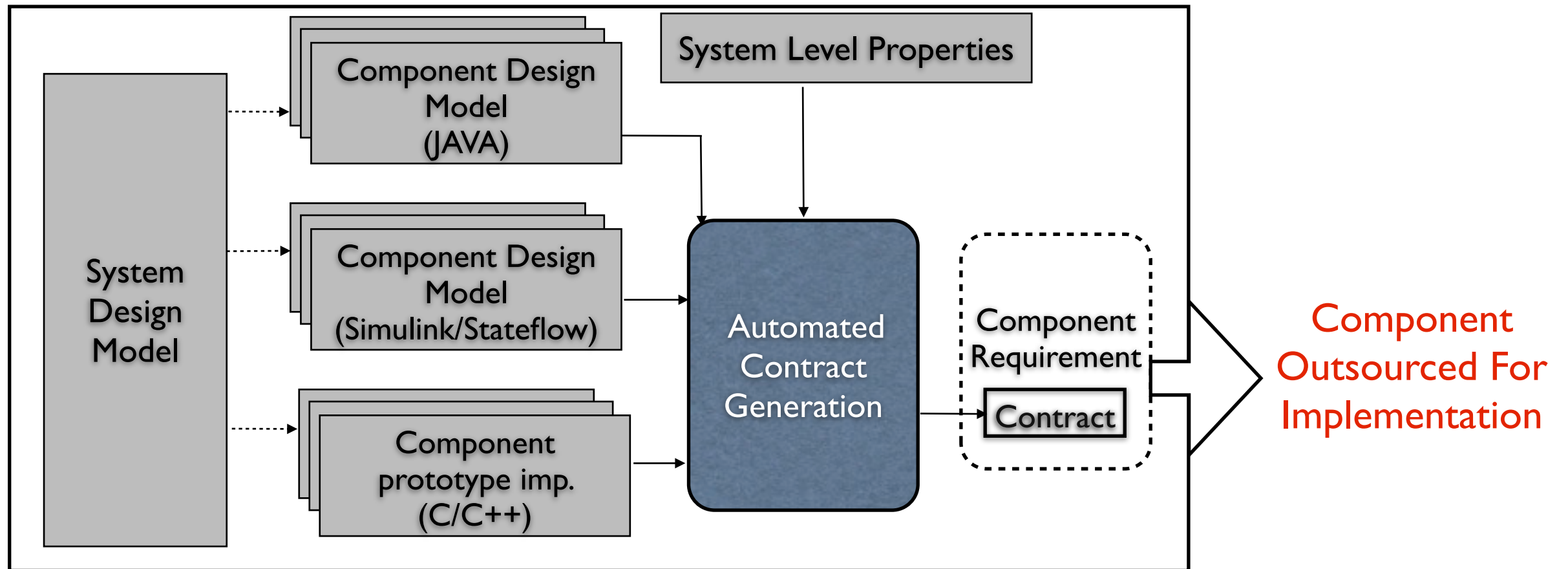


Post-Delivery stage

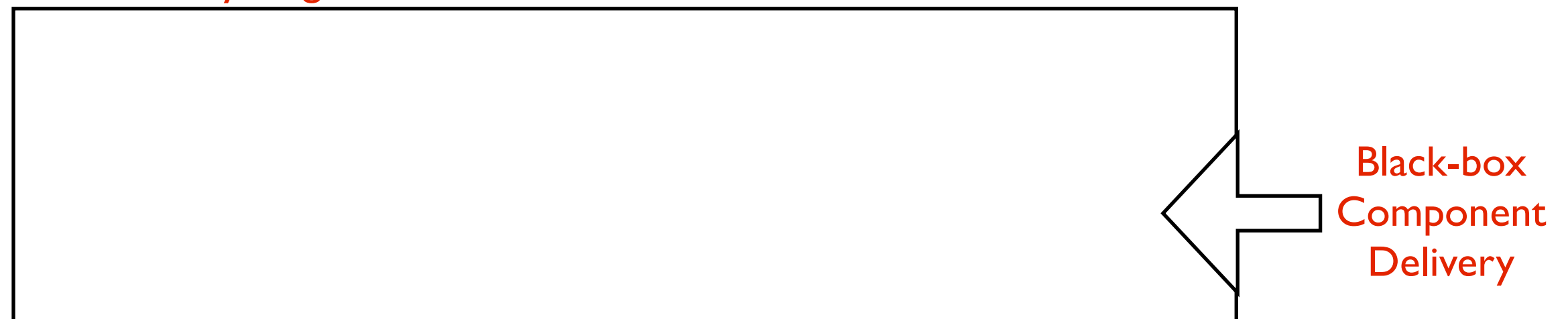


Two Stage solution for virtual integration

Pre-Delivery Stage

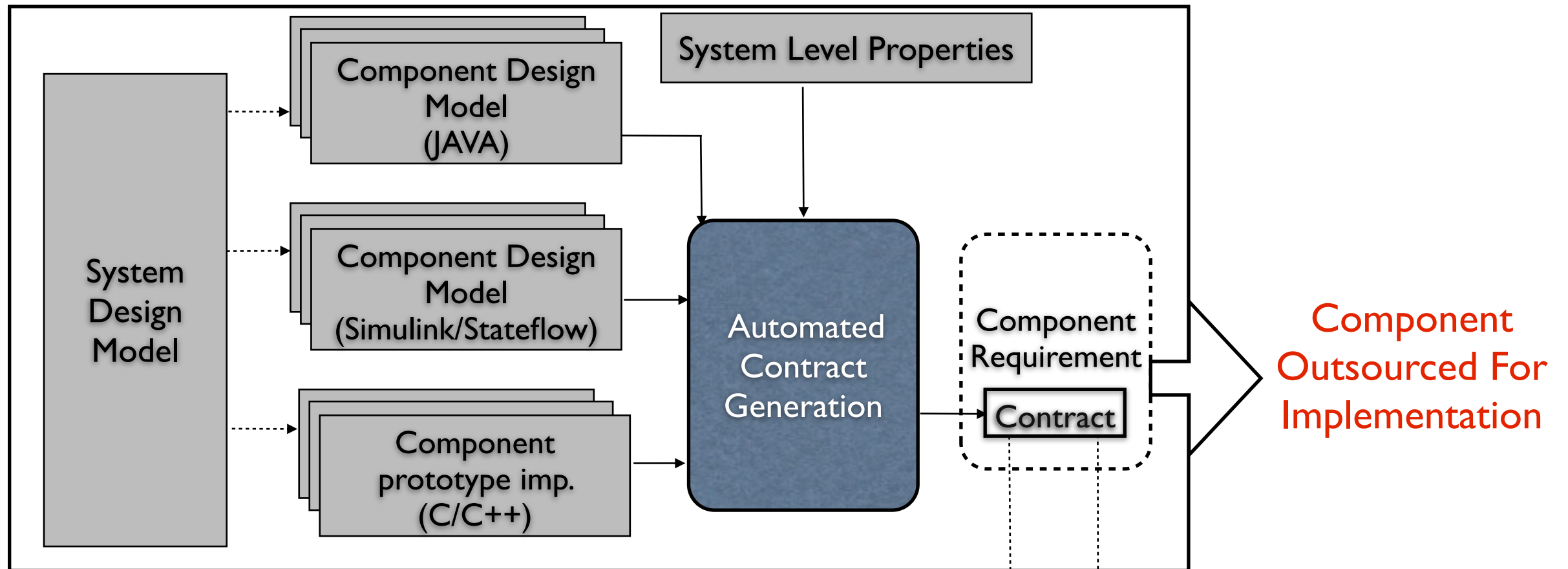


Post-Delivery stage

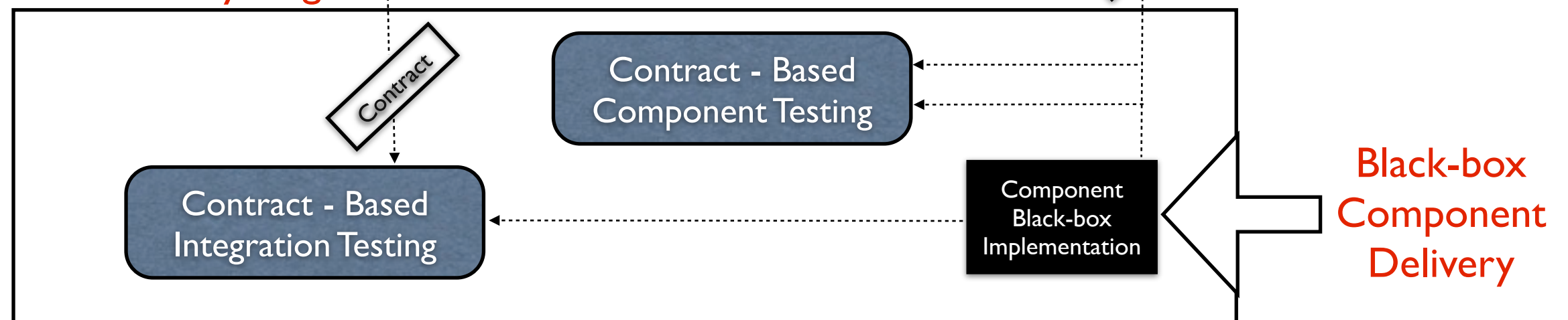


Two Stage solution for virtual integration

Pre-Delivery Stage



Post-Delivery stage



Pre-delivery verification stage

Pre-delivery Verification Stage

How to generate **formal contracts** from models and prototypical code?

1. Define a notion of a **component contract**
 - system property based
 - allows obtain a **higher degree of assurance**
2. Design a **uniform intermediate modeling formalism**
 - to facilitate the integration of different techniques
 - to target heterogeneous in-house system prototypes
3. Develop **(semi)-automated techniques** to generate contract from models and prototypical code

Notion of a Formal Contract

- Contracts as a method to organize and integrate component-based systems
- Specify precisely the information necessary to reason about a component interactions
- Contracts specify I/O behavior of a component:
 - Define the **component guarantees** provided that its environment obey certain **assumptions**.

Notion of a Formal Contract

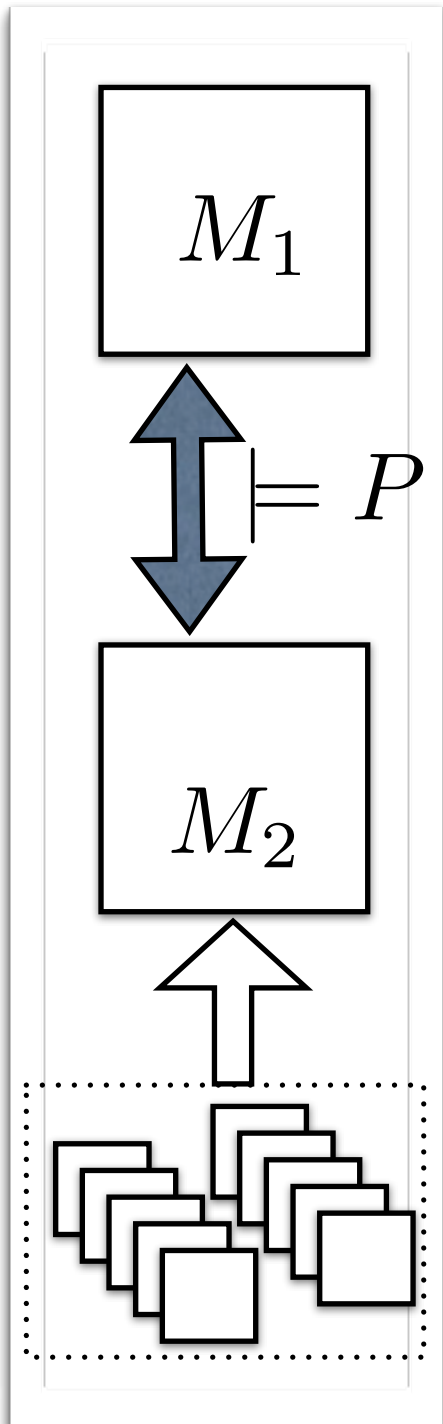
- Contracts as a method to organize and integrate component-based systems
- Specify precisely the information necessary to reason about a component interactions
- Contracts specify I/O behavior of a component:
 - Define the **component guarantees** provided that its environment obey certain **assumptions**.

Different notions of formal contract, e.g.:

- **Othello**: Trace-based contract framework [Tonetta et. al.]
- **AGREE**: Contract language for AADL [Cofer et. al.]
- ACSL, JML, SPARK, etc : Contract in Programming Languages.

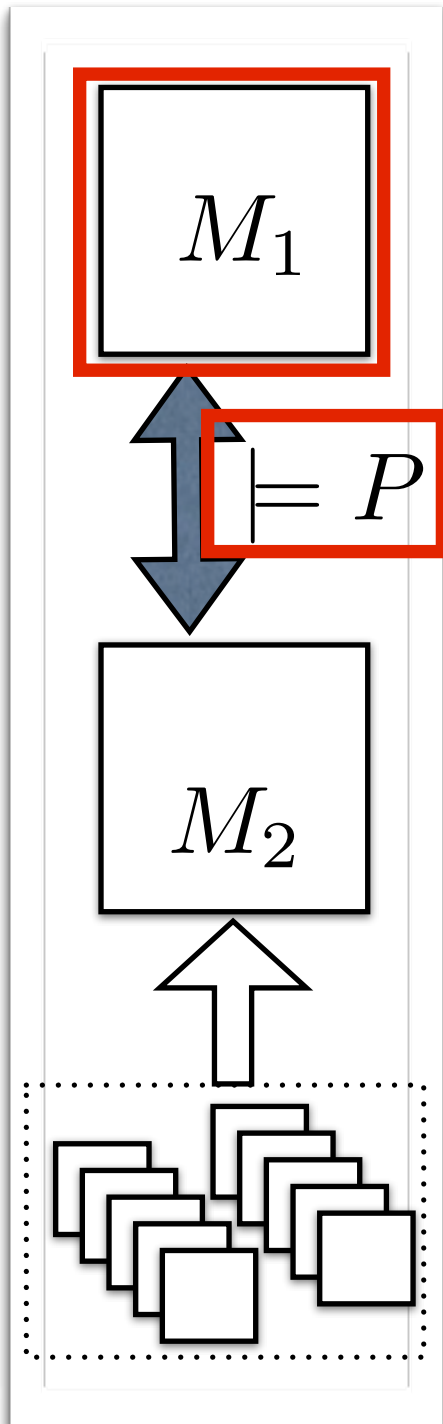
Compositional Verification

- Check P on entire system: **too complicated (e.g. many states)**
- Use system's natural decomposition into components to break-up the verification task
- Check components in isolation: $M_1 \models P?$
- ... typically a component is designed to satisfy its requirements in specific contexts



Compositional Verification

- Check P on entire system: **too complicated (e.g. many states)**
- Use system's natural decomposition into components to break-up the verification task
- Check components in isolation: $M_1 \models P$?
- ... typically a component is designed to satisfy its requirements in specific contexts

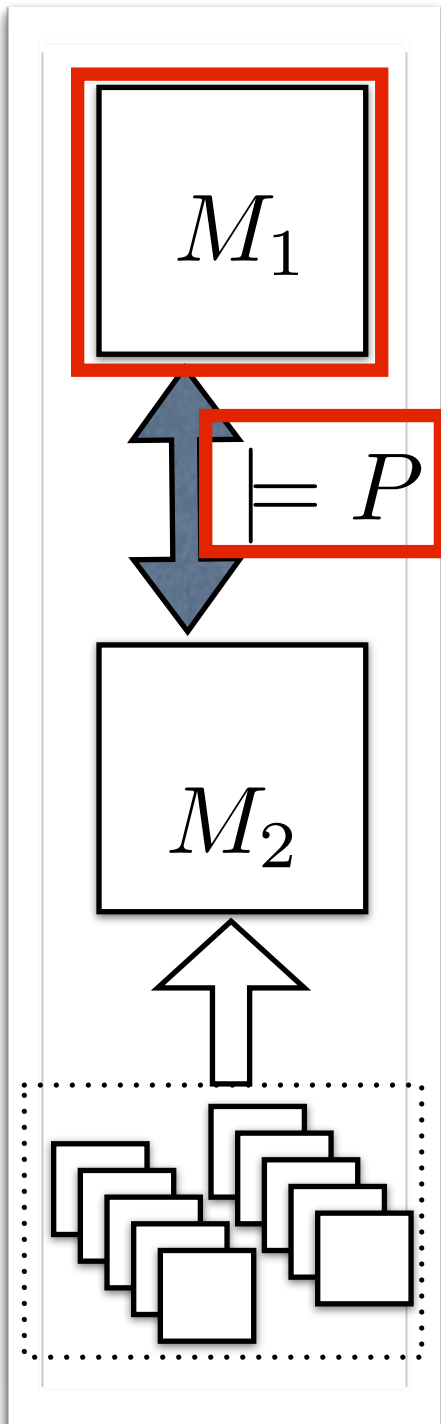


Compositional Verification

- Check P on entire system: **too complicated** (e.g. many states)
- Use system's natural decomposition into components to break-up the verification task
- Check components in isolation: $M_1 \models P$?
- ... typically a component is designed to satisfy its requirements in specific contexts

- **Assume-Guarantee** reasoning

- *Misra & Chandy 81, Jones 83, Pnueli 84, Pasareanu 01*

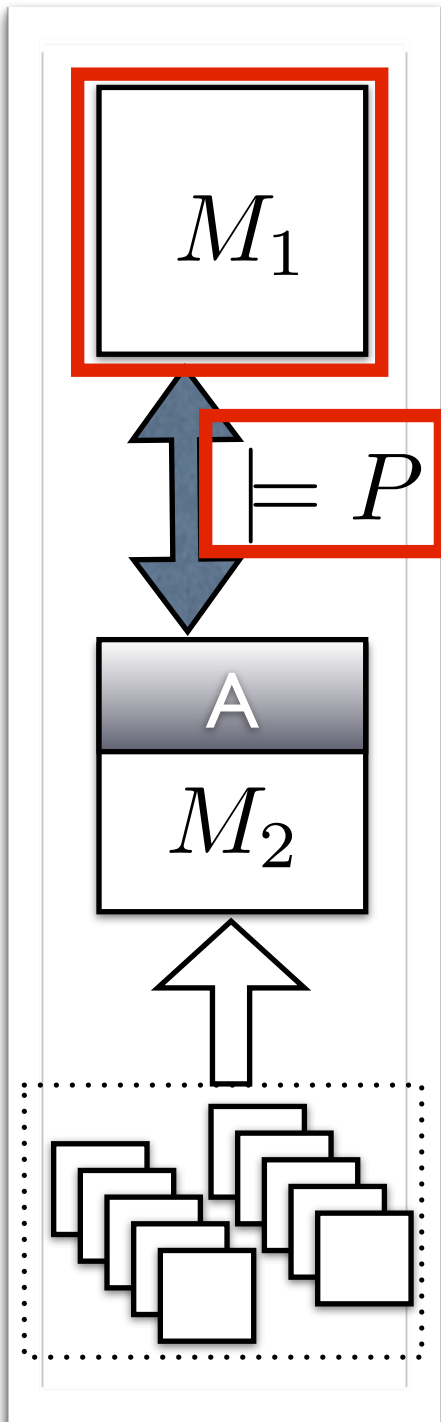


Compositional Verification

- Check P on entire system: **too complicated** (e.g. many states)
- Use system's natural decomposition into components to break-up the verification task
- Check components in isolation: $M_1 \models P$?
- ... typically a component is designed to satisfy its requirements in specific contexts

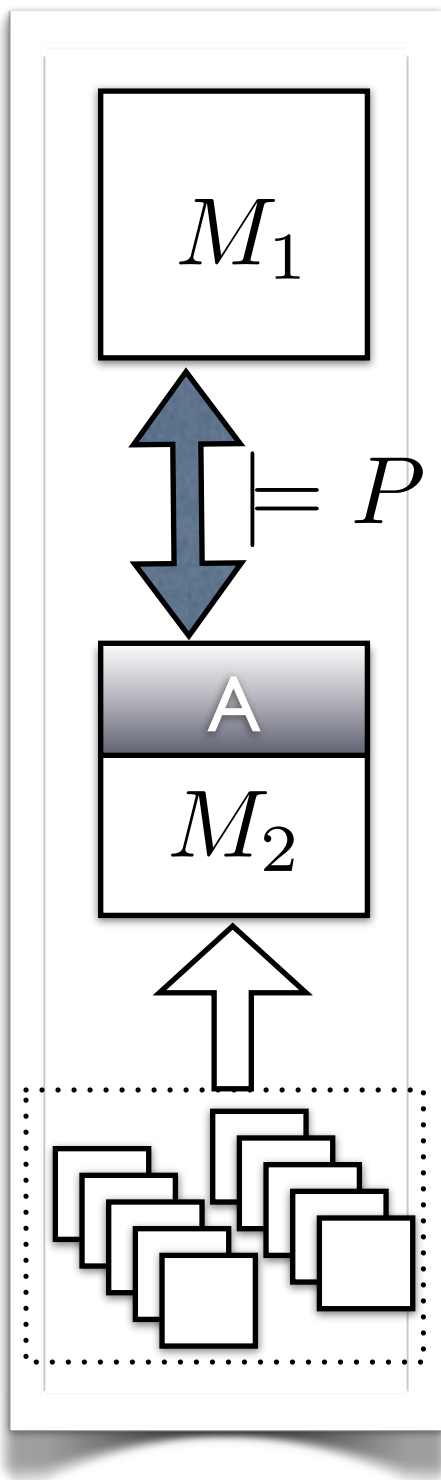
- **Assume-Guarantee** reasoning
- *Misra & Chandy 81, Jones 83, Pnueli 84, Pasareanu 01*

- introduces **assumption** A representing M_1 's context



Compositional Verification

$\langle A \rangle M \langle P \rangle$ is true if whenever M is part of a system that satisfies A , then the system must also guarantee P



Simplest assume-guarantee rule (Asym)

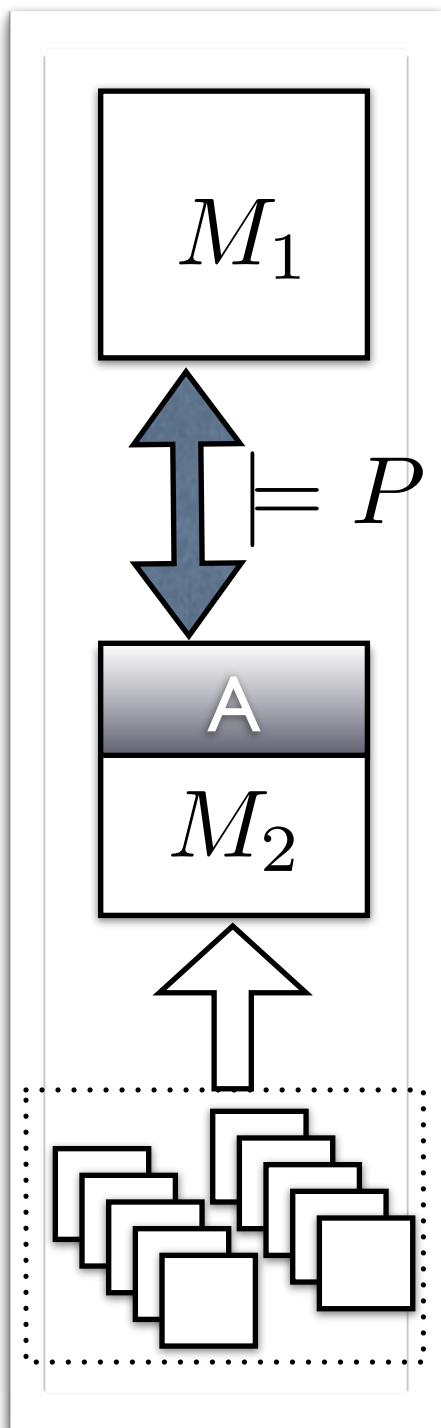
$$1. \langle A \rangle M_1 \langle P \rangle$$

$$2. \langle true \rangle M_2 \langle A \rangle$$

$$\frac{}{\langle true \rangle M_1 \parallel M_2 \langle P \rangle}$$

Compositional Verification

$\langle A \rangle M \langle P \rangle$ is true if whenever M is part of a system that satisfies A , then the system must also guarantee P



Simplest assume-guarantee rule (Asym)

$$\frac{\begin{array}{l} 1. \langle A \rangle M_1 \langle P \rangle \\ 2. \langle true \rangle M_2 \langle A \rangle \end{array}}{\langle true \rangle M_1 \parallel M_2 \langle P \rangle}$$

- * Cobleigh et. al “*Learning assumption for compositional verification*”. TACAS’01
- * Emmi et. al “*Assume Guarantee Verification for Interface Automata*”. FM’08
- * Giannakopoulou et. al “*Symbolic Learning of component interfaces*”. SAS’12
- * Howar et. al “*Hybrid learning: interface generation through static, dynamic, and symbolic analysis*” ISSTA’13.

Compositional Verification

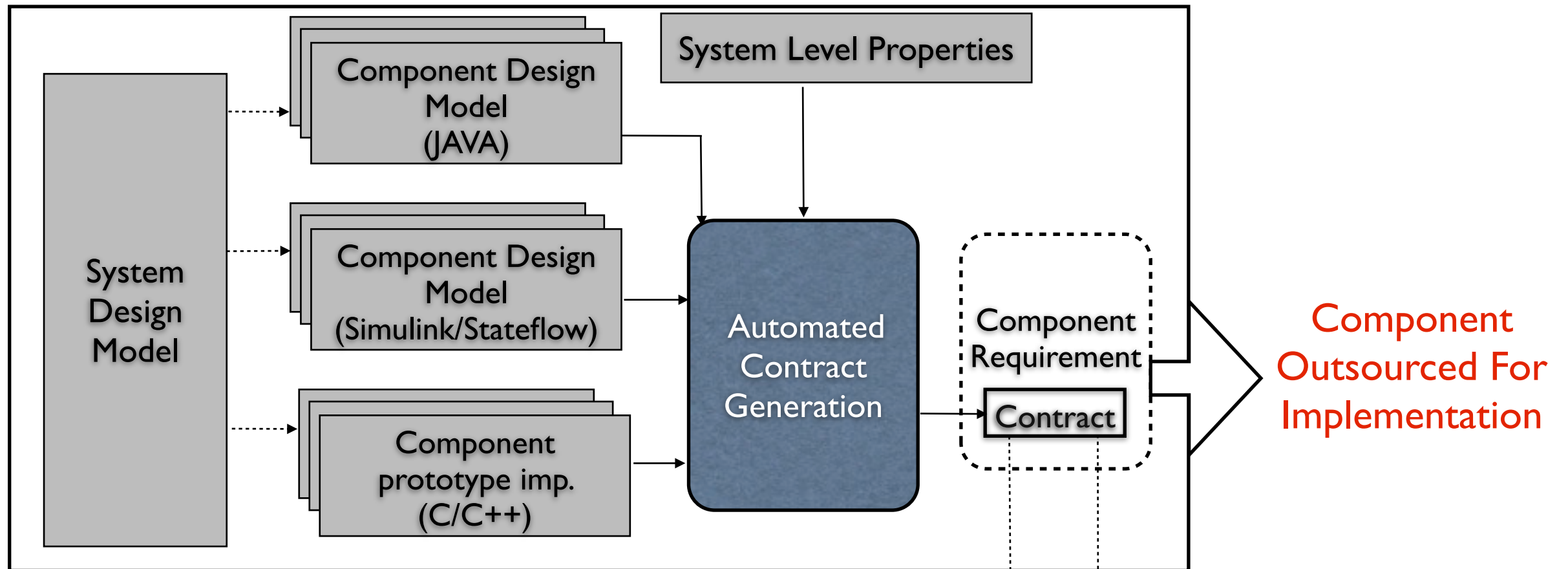
Example of assumptions (*)

- no file “close” before “open”
- access to shared variable “X” must be protected by lock “L”
- *(rover executive)* whenever thread “T” reads variable “V”, no other thread can read “V” before thread “T” clears it first
- *(spacecraft flight phases)* a docking maneuver can only be invoked if the launch abort system has previously been jettisoned from the spacecraft

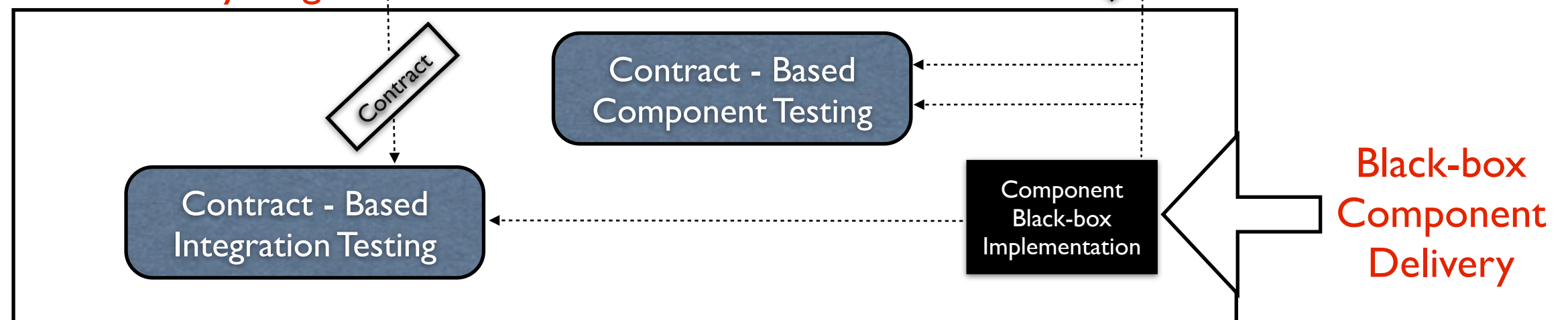
(*) C. Pasareanu slides on compositional verification from SSFT 2012

Two Stage solution for virtual integration

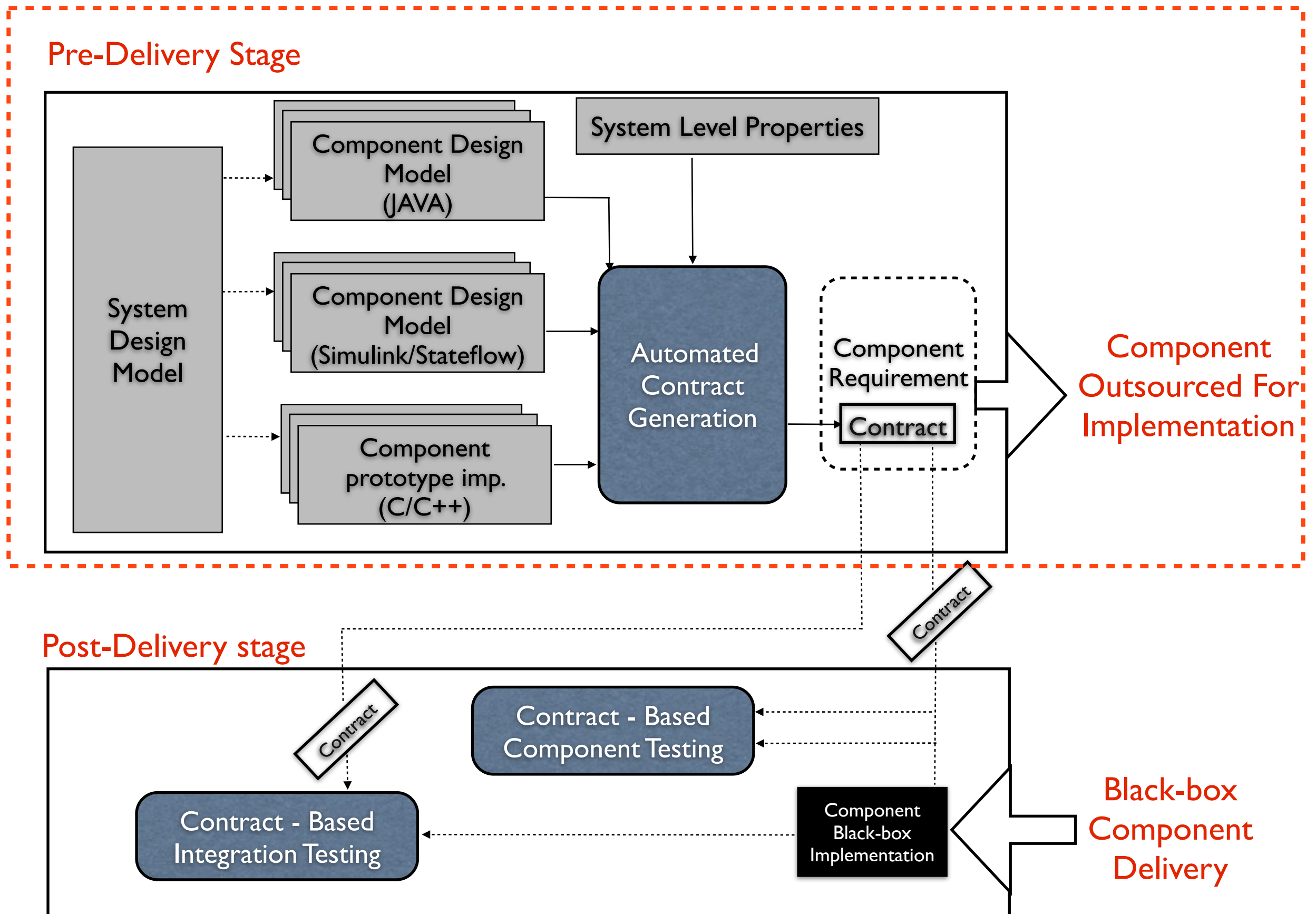
Pre-Delivery Stage



Post-Delivery stage

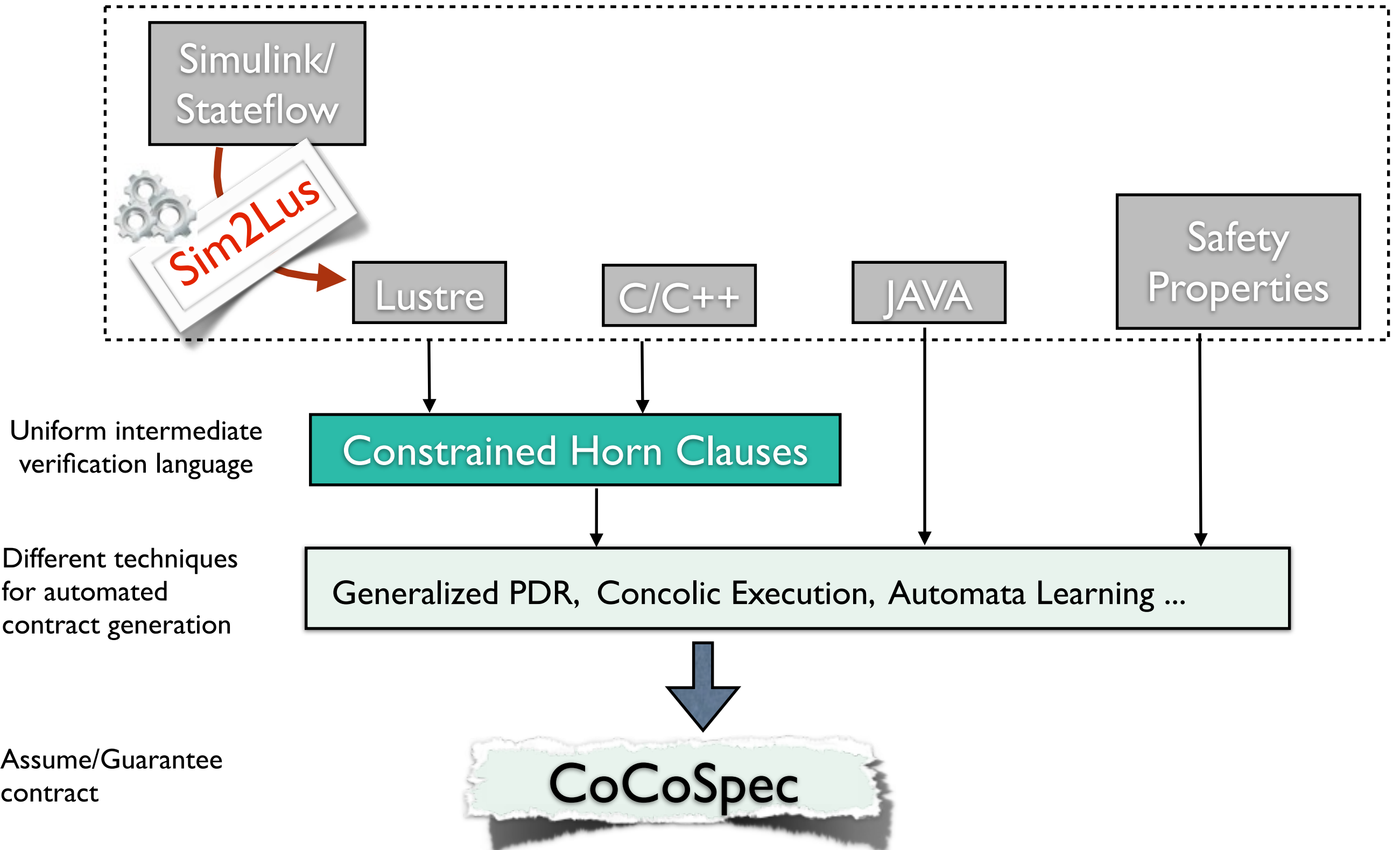


Two Stage solution for virtual integration



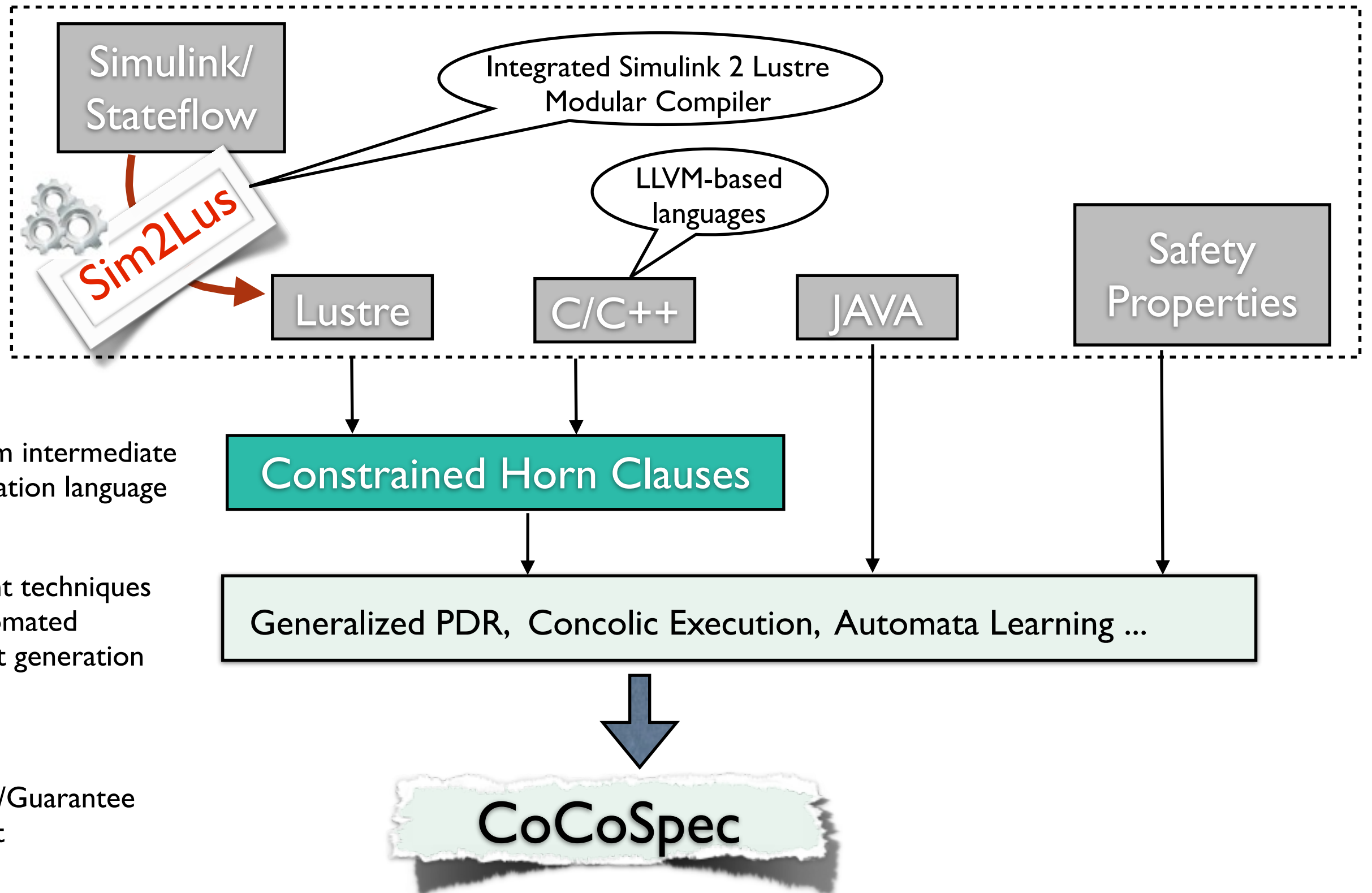
Pre-delivery Verification Stage

Our current approach:



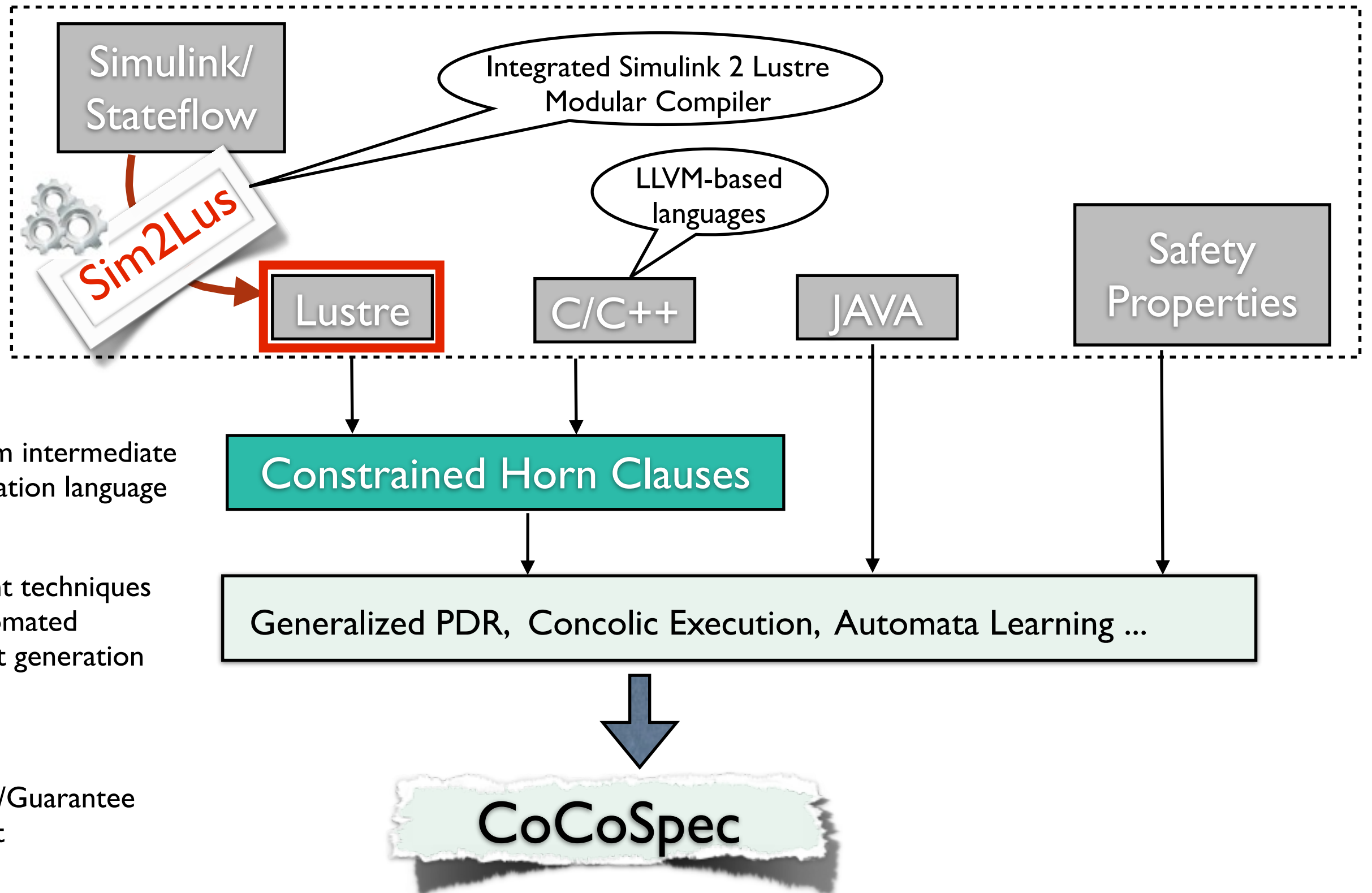
Pre-delivery Verification Stage

Our current approach:



Pre-delivery Verification Stage

Our current approach:



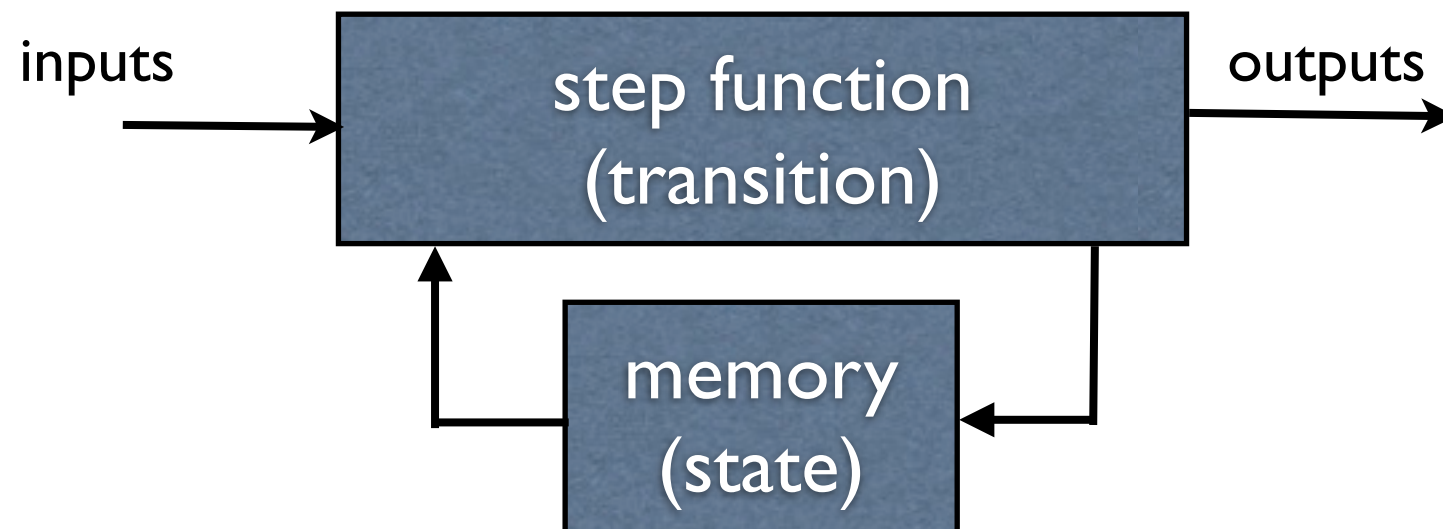
Lustre

- **Declarative** and **deterministic** specification language
- Lustre programs = systems of **equational constraints** between input and output **streams**

```
node therm_control (actual: real; up, dn: bool )  
    returns (heat, cool : bool)  
var desired, margin : real;  
let  
    margin = 1.5;  
    desired = 21.0 → if dn then (pre desired) - 1.0  
                    else if up then (pre desired) + 1.0  
                    else (pre desired);  
    cool = (actual - desired) > margin;  
    heat = (actual - desired) < -margin;  
tel
```

Lustre

- A Lustre program models an **I/O automaton**



Implementing a Lustre program

- Read inputs
- Compute next state and outputs
- Write outputs
- Update state

Repeat at every trigger
(external event)

Lustre

A Lustre program is a collection of nodes: $L = [N_0, N_1, \dots, N_m]$

Lustre

A Lustre program is a collection of nodes: $L = [N_0, N_1, \dots, N_m]$

$$N_i = (\mathcal{I}_i, \mathcal{O}_i, \mathcal{L}_i, \text{Init}_i, \text{Trans}_i)$$

- $\mathcal{I}_i, \mathcal{O}_i, \mathcal{L}_i$: set of input/output/local vars
- $\text{Init}_i, \text{Trans}_i$: set of formulas for the initial states and transition relation

Lustre

A Lustre program is a collection of nodes: $L = [N_0, N_1, \dots, N_m]$

$$N_i = (\mathcal{I}_i, \mathcal{O}_i, \mathcal{L}_i, Init_i, Trans_i)$$

- $\mathcal{I}_i, \mathcal{O}_i, \mathcal{L}_i$: set of input/output/local vars
- $Init_i, Trans_i$: set of formulas for the initial states and transition relation

$$\bigwedge_{i \in \mathbb{N}} v_i = \rho(s_i)$$

- $v_i \in \mathcal{O}_i \cup \mathcal{L}_i$ and $Vars(s_i) \subseteq \mathcal{I}_i \cup \mathcal{O}_i \cup \mathcal{L}_i$
- s_i arbitrary Lustre expression including node calls $N_j(u_1, \dots, u_n)$
- ρ function maps expression to expression

$$a \rightarrow b \text{ is projected as } \begin{cases} a \text{ in } Init_i \\ b \text{ in } Trans_i \end{cases}$$

Lustre

A Lustre program is a collection of nodes: $L = [N_0, N_1, \dots, N_m]$

$$N_i = (\mathcal{I}_i, \mathcal{O}_i, \mathcal{L}_i, \text{Init}_i, \text{Trans}_i)$$

- $\mathcal{I}_i, \mathcal{O}_i, \mathcal{L}_i$: set of input/output/local vars
- $\text{Init}_i, \text{Trans}_i$: set of formulas for the initial states and transition relation

$$\bigwedge_{i \in \mathbb{N}} v_i = \rho(s_i)$$

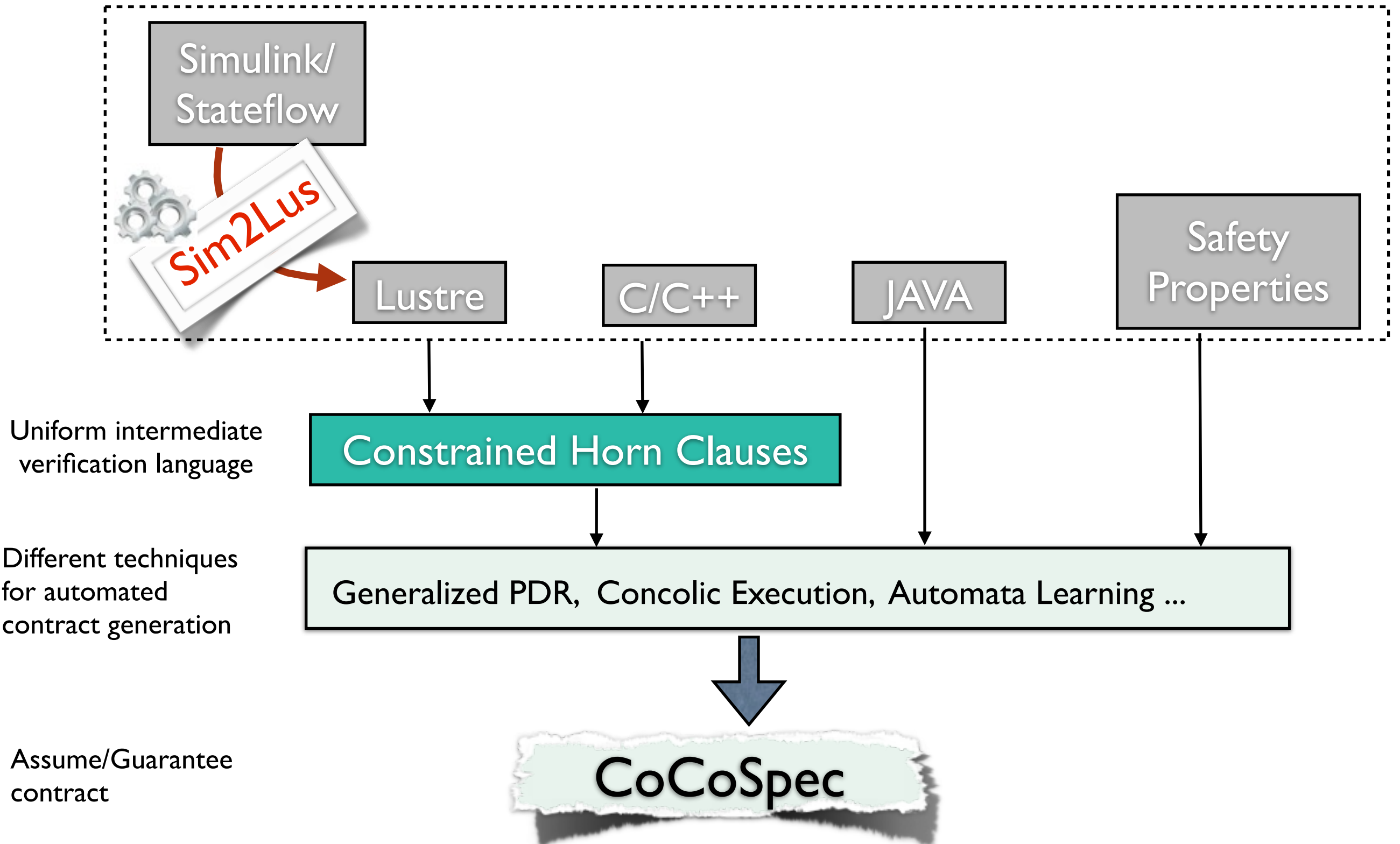
- $v_i \in \mathcal{O}_i \cup \mathcal{L}_i$ and $\text{Vars}(s_i) \subseteq \mathcal{I}_i \cup \mathcal{O}_i \cup \mathcal{L}_i$
- s_i arbitrary Lustre expression including node calls $N_j(u_1, \dots, u_n)$
- ρ function maps expression to expression

$$a \rightarrow b \text{ is projected as } \begin{cases} a \text{ in } \text{Init}_i \\ b \text{ in } \text{Trans}_i \end{cases}$$

- A safety property P is any Lustre expression over the main node N_0

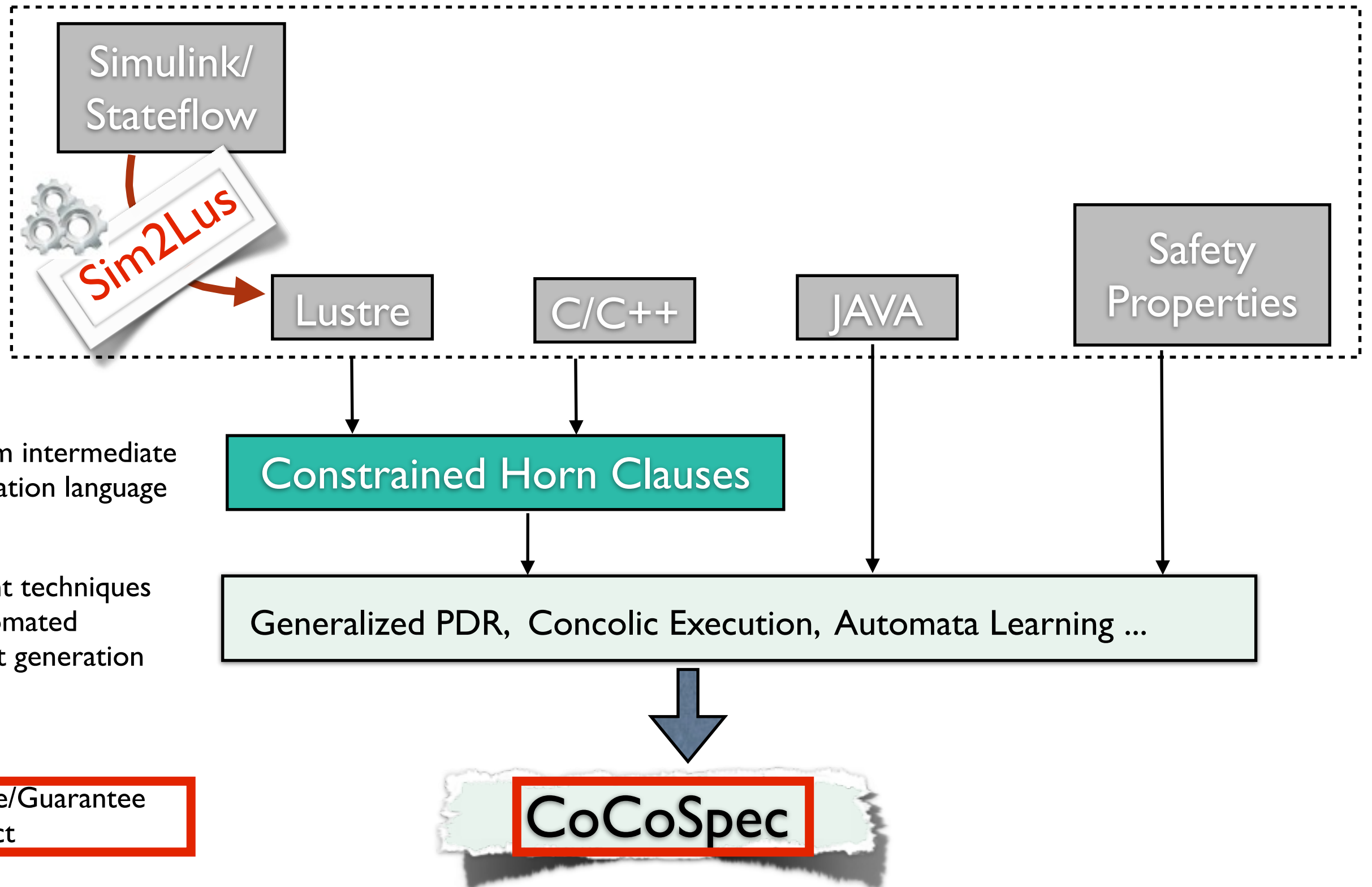
Pre-delivery Verification Stage

Our current approach:



Pre-delivery Verification Stage

Our current approach:



Assume-Guarantee contracts

consist of

- an **assumption** \mathcal{A} : how the component must be used
- a **guarantee** \mathcal{G} : how the component must behave, assuming \mathcal{A}

Assume-Guarantee contracts

consist of

- an **assumption** \mathcal{A} : how the component must be used
- a **guarantee** \mathcal{G} : how the component must behave, assuming \mathcal{A}

If $\langle \mathcal{A}, \mathcal{G} \rangle$ is a **contract** for component C , then if \mathcal{A} is always true so is \mathcal{G} :

$$(\Box \mathcal{A}) \Rightarrow (\Box \mathcal{G}) \quad \text{holds for } C$$

In practice, usually weakened to

$$(\text{hist } \mathcal{A}) \Rightarrow \mathcal{G} \quad \text{is an invariant of } C$$

Assume-Guarantee contracts

consist of

- an **assumption** \mathcal{A} : how the component must be used
- a **guarantee** \mathcal{G} : how the component must behave, assuming \mathcal{A}

If $\langle \mathcal{A}, \mathcal{G} \rangle$ is a **contract** for component C , then if \mathcal{A} is always true so is \mathcal{G} :

$$(\Box \mathcal{A}) \Rightarrow (\Box \mathcal{G}) \quad \text{holds for } C$$

In practice, usually weakened to

$$(\text{hist } \mathcal{A}) \Rightarrow \mathcal{G} \quad \text{is an invariant of } C$$

If component C' uses C , then \mathcal{A}_{cs} (\mathcal{A} at call site) must always be true:

$$\mathcal{A}_{cs} \quad \text{is an invariant of } C'$$

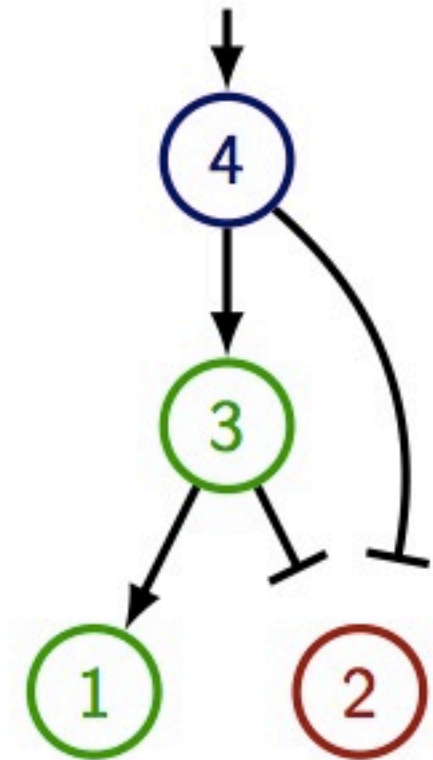
Assume-Guarantee contracts

Improves scalability of the verification of hierarchical systems by abstracting components by their contract.

The analysis is bottom-up:

- *leaves* are analyzed as usual, which can **succeed** or **fail**.
- for *nodes*, we first abstract the subcomponents, which can **succeed**, or **fail**.

In case of **failure** we can restart the analysis after (soundly) refining the abstraction, possibly several times.



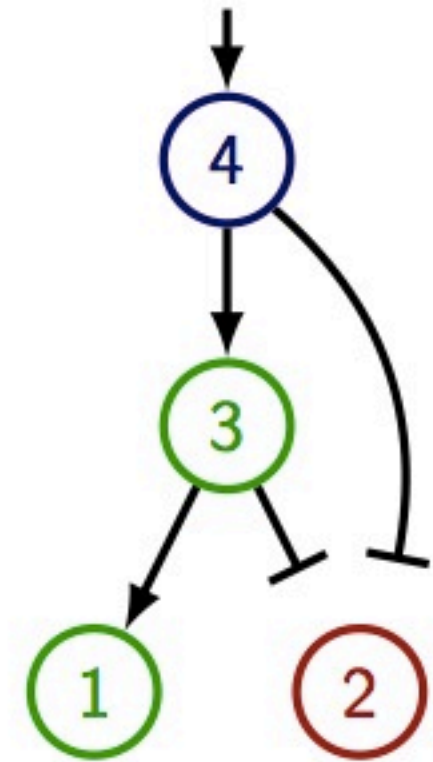
Assume-Guarantee contracts

Improves scalability of the verification of hierarchical systems by abstracting components by their contract.

The analysis is bottom-up:

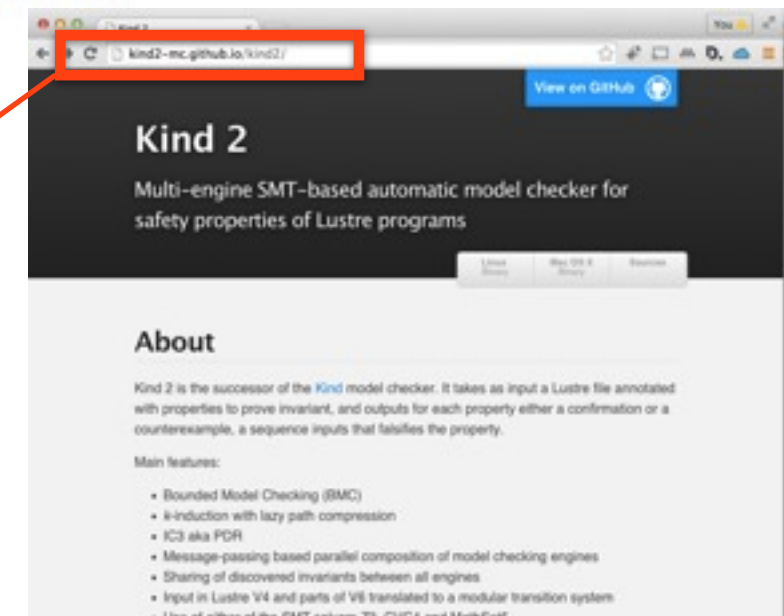
- *leaves* are analyzed as usual, which can **succeed** or **fail**.
- for *nodes*, we first abstract the subcomponents, which can **succeed**, or **fail**.

In case of **failure** we can restart the analysis after (soundly) refining the abstraction, possibly several times.



Implemented in Kind2: a multi engine model checker for Lustre programs

<http://kind2-mc.github.io/kind2/>



CoCoSpec

- An Assume/Guarantee-based Contract Language on top of Lustre

A **CocoSpec** contract is a pair $\langle \mathcal{A}, \mathcal{G} \rangle$

Assumption — how the component must be used:

$$\mathcal{A} \equiv \bigvee (require_i)$$

Guarantee — how the component behaves:

$$\mathcal{G} \equiv \bigwedge (require_i \Rightarrow ensure_i)$$

CoCoSpec

- An Assume/Guarantee-based Contract Language

```
node component(n1, n2:int; chaos:bool)
    returns (out: bool; corrupted, warning:bool) ;
--!contract : contr      ;

let
    -- Implementation.
tel

contract contr(n1, n2:int; chaos:bool)
    returns (out: bool; corrupted, warning:bool) ;

let
    require (-7 <= n1) and (7 <= n1); -- n1 legal input
    require (-11 <= n2) and (11 <= n2); -- n2 legal input
    ensure (-42 <= out) and (42 <= out); -- out is bounded
tel
```


CoCoSpec

- An Assume/Guarantee-based Contract Language

```
node component(n1, n2:int; chaos:bool)
    returns (out: bool; corrupted, warning:bool) ;
--!contract : contr          ;

let
    -- Implementation.
tel

contract contr(n1, n2:int; chaos:bool)
    returns (out: bool; corrupted, warning:bool) ;

let
    require (-7 <= n1) and (7 <= n1); -- n1 legal input
    require (-11 <= n2) and (11 <= n2); -- n2 legal input
    ensure (-42 <= out) and (42 <= out); -- out is bounded
tel
```

CoCoSpec

- An Assume/Guarantee-based Contract Language

```
node component(n1, n2:int; chaos:bool)
    returns (out: bool; corrupted, warning:bool) ;
--!contract : contr          ;

let
    -- Implementation.
tel
```

```
contract contr(n1, n2:int; chaos:bool)
    returns (out: bool; corrupted, warning:bool) ;

let
    require (-7 <= n1) and (7 <= n1); -- n1 legal input
    require (-11 <= n2) and (11 <= n2); -- n2 legal input
    ensure (-42 <= out) and (42 <= out); -- out is bounded
tel
```

CoCoSpec

- An Assume/Guarantee-based Contract Language

```
node component(on, off: bool) returns (active: bool) ;
--!contract : nop ;
--!contract : inhibited ;
let
  -- Implementation.
tel


contract inhibited(on, off: bool) returns (active: bool) ;
var
  act_raise: bool ; last_act_raise: int ;
let
  active_raise = false -> active and not pre active ;
  last_act_raise = 0 -> if pre active_raise then 1
                        else 1 + pre last_act_raise ;
  require last_act_raise <= n ;
  ensure active ;
tel
```


CoCoSpec

- An Assume/Guarantee-based Contract Language

```
node component(on, off: bool) returns (active: bool) ;
--!contract : nop ;
--!contract : inhibited ;
let
  -- Implementation.
tel

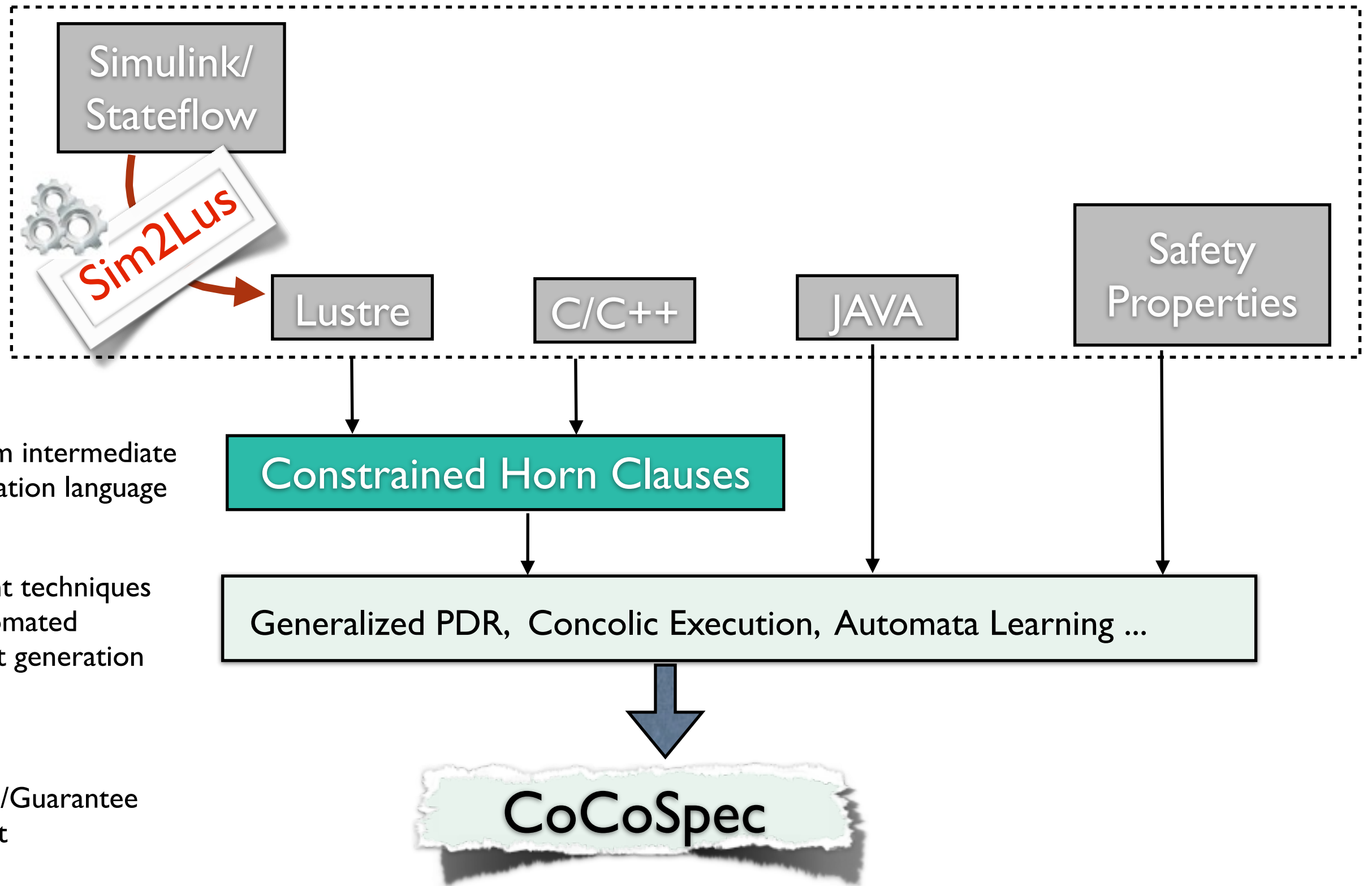
contract inhibited(on, off: bool) returns (active: bool) ;
var
  act_raise: bool ; last_act_raise: int ;
let
  active_raise = false -> active and not pre active ;
  last_act_raise = 0 -> if pre active_raise then 1
                        else 1 + pre last_act_raise ;
  require last_act_raise <= n ;
  ensure active ;
tel
```



ghost variable

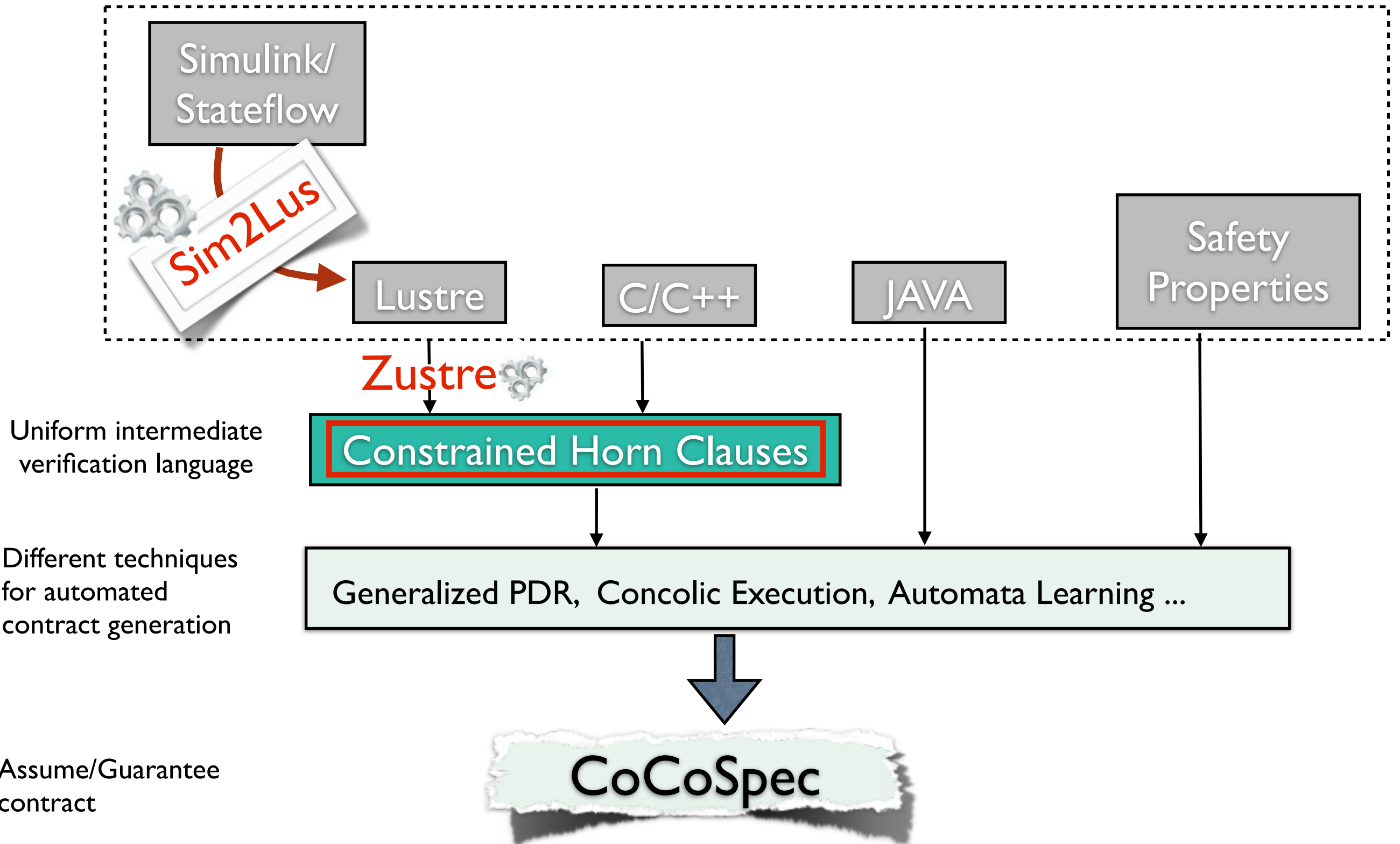
Pre-delivery Verification Stage

Our current approach:



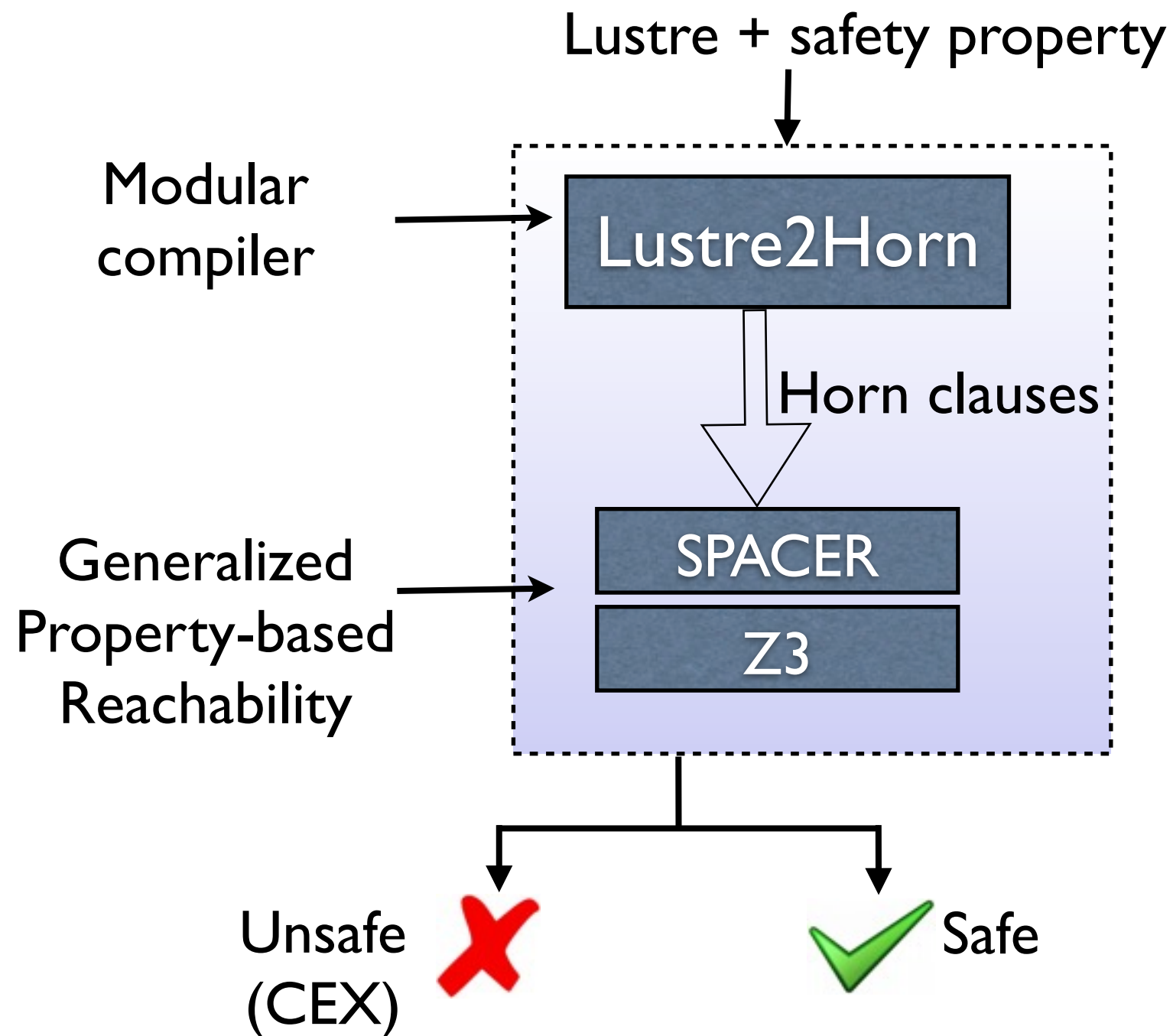
Pre-delivery Verification Stage

Our current approach:



Zustre

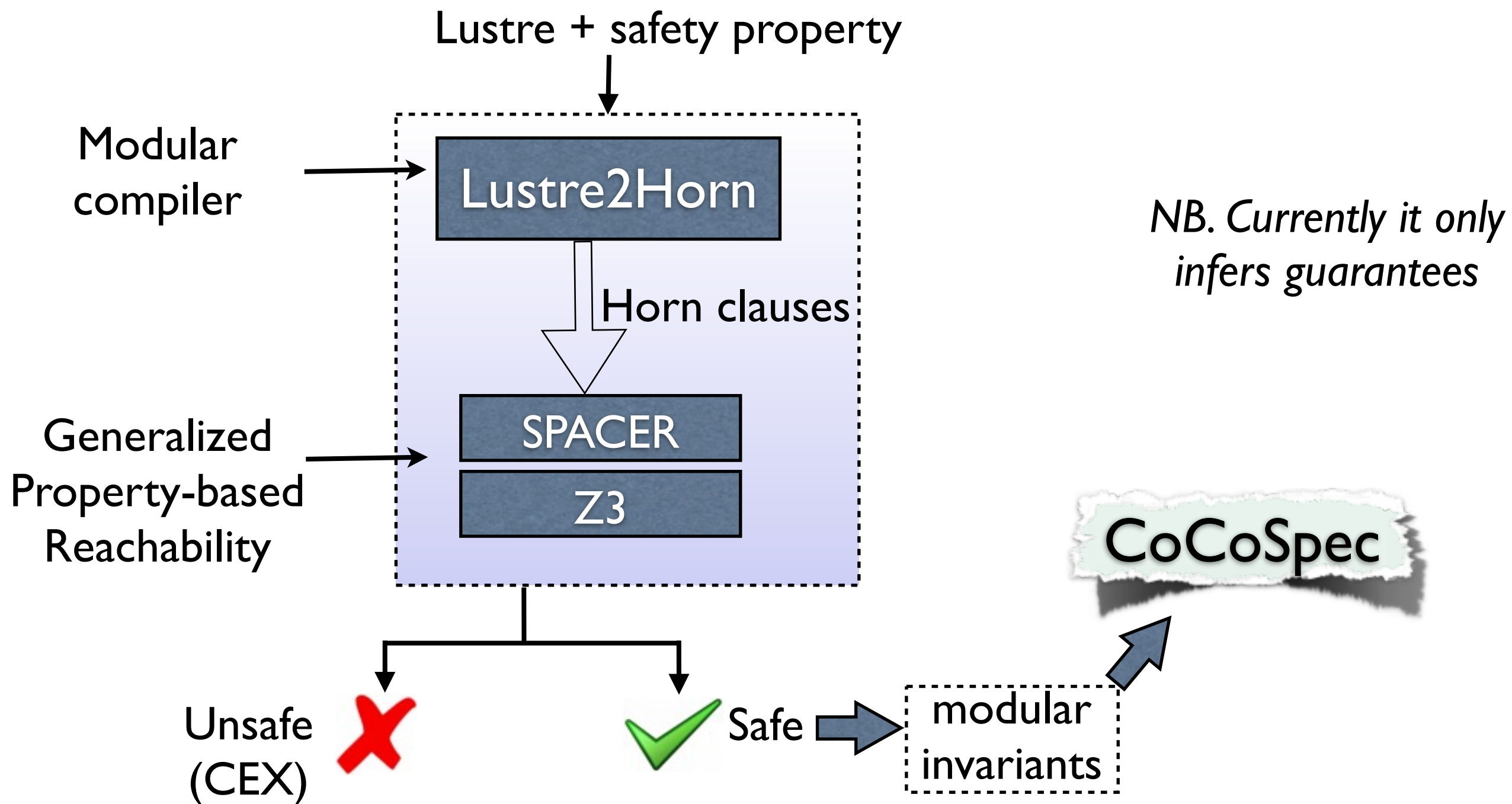
A verification engine and CoCoSpec generator for Lustre program



T. Kahsai, PL. Garoche, A. Gurfinkel: "Synthesizing modular invariants for synchronous code". In HCVS 2014.

Zustre

A verification engine and CoCoSpec generator for Lustre program



T. Kahsai, PL. Garoche, A. Gurfinkel: "Synthesizing modular invariants for synchronous code". In HCVS 2014.

Constrained Horn Clause

Constrained Horn Clause

- A fragment of First Order Logic.
- A uniform way to represent transition systems for verification.

\mathcal{F} : set of function symbols

\mathcal{P} : set of predicate symbols

\mathcal{V} : set of variables

Constrained Horn Clause (CHC) is a formula:

$$\forall \mathcal{V} \cdot (\phi \wedge p_1[X_1] \wedge \cdots \wedge p_n[X_n] \rightarrow h[X]), \text{ for } n \geq 0$$

ϕ : constraint over $\mathcal{F} \cup \mathcal{V}$ with respect to some background theory
e.g. arithmetic, arrays, SMT

$X_i, X \subseteq \mathcal{V}$: (possibly empty) vectors of variables

p_1, \dots, p_n, h : n -ary predicates

$p_i[X_i]$: application $p(t_1, \dots, t_n)$ of an n -ary predicate symbol

Example 1

```
node therm_control (actual: real; up, dn: bool )  
    returns (heat, cool : bool)  
var desired, margin : real;  
let  
    margin = 1.5;  
    desired = 21.0 → if dn then (pre desired) - 1.0  
                    else if up then (pre desired) + 1.0  
                    else (pre desired);  
    cool = (actual - desired) > margin;  
    heat = (actual - desired) < -margin;  
tel
```

Example 1

```
node therm_control (actual: real; up, dn: bool )
  returns (heat, cool : bool)
  var desired, margin : real;
  let
    margin = 1.5;
    desired = 21.0 → if dn then (pre desired) - 1.0
                    else if up then (pre desired) + 1.0
                    else (pre desired);
    cool = (actual - desired) > margin;
    heat = (actual - desired) < -margin;
  tel
```

$[\textit{margin} = 1.5$
 $\wedge \textit{desired} = 21.0$
 $\wedge \textit{cool} = \textit{actual} - \textit{desired} > \textit{margin}$
 $\wedge \textit{heat} = \dots] \Rightarrow TC_{init}(\textit{actual}, \textit{up}, \textit{dn}, \textit{heat}, \textit{cool}, \textit{desired})$

Initial states

$[\textit{margin} = 1.5$
 $\wedge \textit{desired}' = \textit{ite}(\textit{dn} (\textit{desired} - 1.0) (\textit{ite}\dots))$
 $\wedge \textit{cool} = \textit{actual} - \textit{desired}' > \textit{margin}$
 $\wedge \textit{heat} = \dots] \Rightarrow TC_{trans}(\textit{actual}, \textit{up}, \textit{dn}, \textit{heat}, \textit{cool}, \textit{desired}, \textit{desired}')$

Transition relation

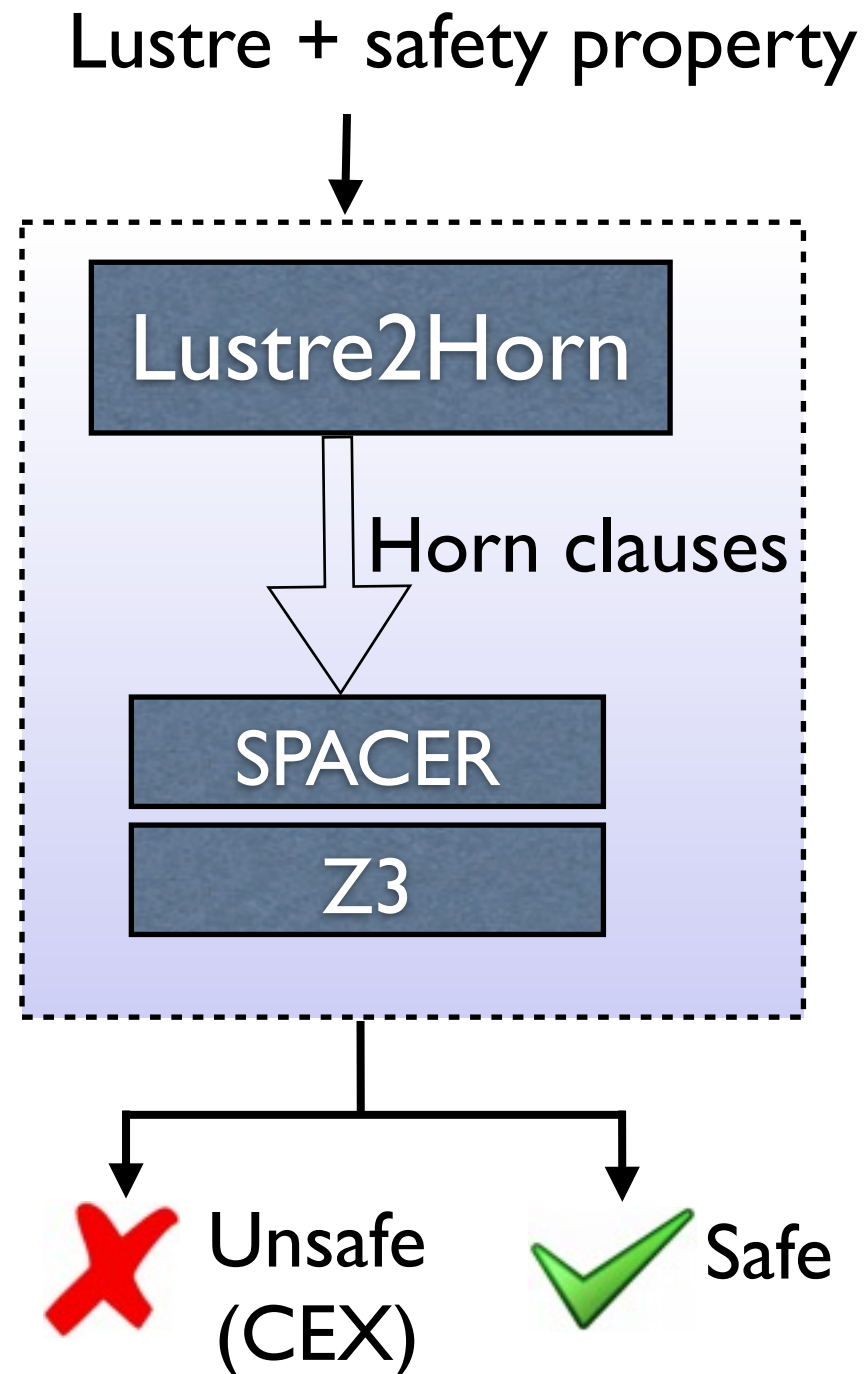
$TC_{init}(\textit{actual}, \textit{up}, \textit{dn}, \textit{heat}, \textit{cool}, \textit{desired}) \Rightarrow \textit{Loop}(\textit{actual}, \textit{up}, \textit{dn}, \textit{heat}, \textit{cool}, \textit{desired})$

$\textit{Loop}(\textit{actual}', \textit{up}', \textit{dn}', \textit{heat}', \textit{cool}', \textit{desired})$
 $\wedge TC_{trans}(\textit{actual}, \textit{up}, \textit{dn}, \textit{heat}, \textit{cool}, \textit{desired}, \textit{desired}')$
 $\Rightarrow \textit{Loop}(\textit{actual}, \textit{up}, \textit{dn}, \textit{heat}, \textit{cool}, \textit{desired}')$

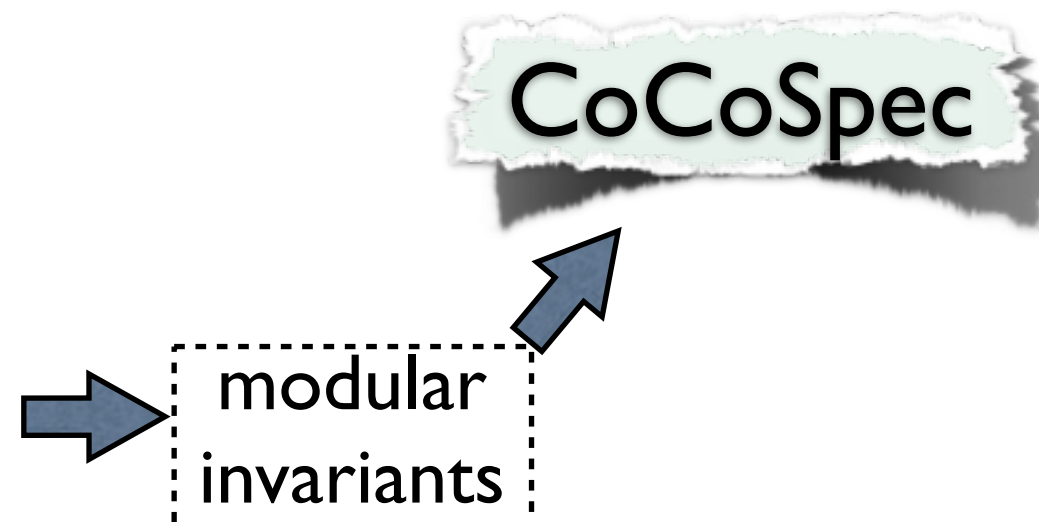
Loop

Zustre

A verification engine and CoCoSpec generator for Lustre program



- $N(\mathcal{I}, \mathcal{O}, \mathcal{S}, \mathcal{S}') = \bigwedge \varphi$: an invariant for a node N
- **Modular invariants** : invariants for each node



From Horn Clauses to CoCoSpec

```
node Sofar( X : bool ) returns ( Y : bool );
let
  Y = (true -> pre Y) and X;
tel

-- assignment other

node Store( Delta : int ) returns ( Total : int );
var Prev : int;
let
  Prev = 0 -> pre Total;
  Total = if Delta < 0 and Prev > 0 then Prev+Delta
         else if Delta > 0 and Prev < 10 then Prev+Delta
         else Prev;
tel

node top( Delta : int ) returns ( OK : bool );
var Total : int;
  S: bool;
  -- Delta_const : int;
let
  -- Delta_const = Delta -> pre Delta_const;
  Total = Store( Delta );

  S = Sofar( -1 <= Delta and Delta <= 1 );
  OK = S => 0 <= Total and Total <= 20;
  --!PROPERTY : OK=true;
  --!MAIN:true;
tel
```

From Horn Clauses to CoCoSpec

```
node Sofar( X : bool ) returns ( Y : bool );
let
  Y = (true -> pre Y) and X;
tel

-- assignment other

node Store( Delta : int ) returns ( Total : int );
var Prev : int;
let
  Prev = 0 -> pre Total;
  Total = if Delta < 0 and Prev > 0 then Prev+Delta
          else if Delta > 0 and Prev < 10 then Prev+Delta
          else Prev;
tel

node top( Delta : int ) returns ( OK : bool );
var Total : int;
  S: bool;
  -- Delta_const : int;
let
  -- Delta_const = Delta -> pre Delta_const;
  Total = Store( Delta );
  S = Sofar( -1 <= Delta and Delta <= 1 );
  OK = S => 0 <= Total and Total <= 20;
  --!PROPERTY : OK=true;
  --!MAIN:true;
tel
```



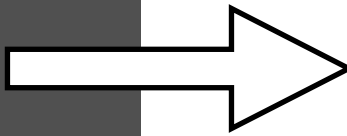
From Horn Clauses to CoCoSpec

```
node Sofar( X : bool ) returns ( Y : bool );
let
  Y = (true -> pre Y) and X;
tel

-- assignment other

node Store( Delta : int ) returns ( Total : int );
var Prev : int;
let
  Prev = 0 -> pre Total;
  Total = if Delta < 0 and Prev > 0 then Prev+Delta
         else if Delta > 0 and Prev < 10 then Prev+Delta
         else Prev;
tel

node top( Delta : int ) returns ( OK : bool );
var Total : int;
  S: bool;
  -- Delta_const : int;
let
  -- Delta_const = Delta -> pre Delta_const;
  Total = Store( Delta );
  S = Sofar( -1 <= Delta and Delta <= 1 );
  OK = S => 0 <= Total and Total <= 20;
  --!PROPERTY : OK=true;
  --!MAIN:true;
tel
```



```
And(Not(Not(Sofar.__Sofar_2_x) == Sofar.X),
     Not(Not(Sofar.Y) == Sofar.X))
And(Not(Sofar.Y == Or(Not(Sofar.__Sofar_2_c), Not(Sofar.X))),
     Not(Not(Sofar.__Sofar_2_x) == Sofar.Y))
```



From Horn Clauses to CoCoSpec

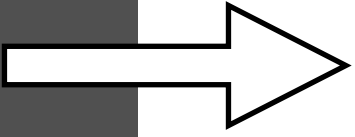
```
node Sofar( X : bool ) returns ( Y : bool );
let
  Y = (true -> pre Y) and X;
tel

-- assignment other

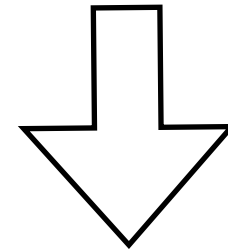
node Store( Delta : int ) returns ( Total : int );
var Prev : int;
let
  Prev = 0 -> pre Total;
  Total = if Delta < 0 and Prev > 0 then Prev+Delta
          else if Delta > 0 and Prev < 10 then Prev+Delta
          else Prev;
tel

node top( Delta : int ) returns ( OK : bool );
var Total : int;
  S: bool;
  -- Delta_const : int;
let
  -- Delta_const = Delta -> pre Delta_const;
  Total = Store( Delta );

S = Sofar( -1 <= Delta and Delta <= 1 );
OK = S => 0 <= Total and Total <= 20;
--!PROPERTY : OK=true;
--!MAIN:true;
tel
```



```
And(Not(Not(Sofar.__Sofar_2_x) == Sofar.X),
     Not(Not(Sofar.Y) == Sofar.X))
And(Not(Sofar.Y == Or(Not(Sofar.__Sofar_2_c), Not(Sofar.X))),
     Not(Not(Sofar.__Sofar_2_x) == Sofar.Y))
```

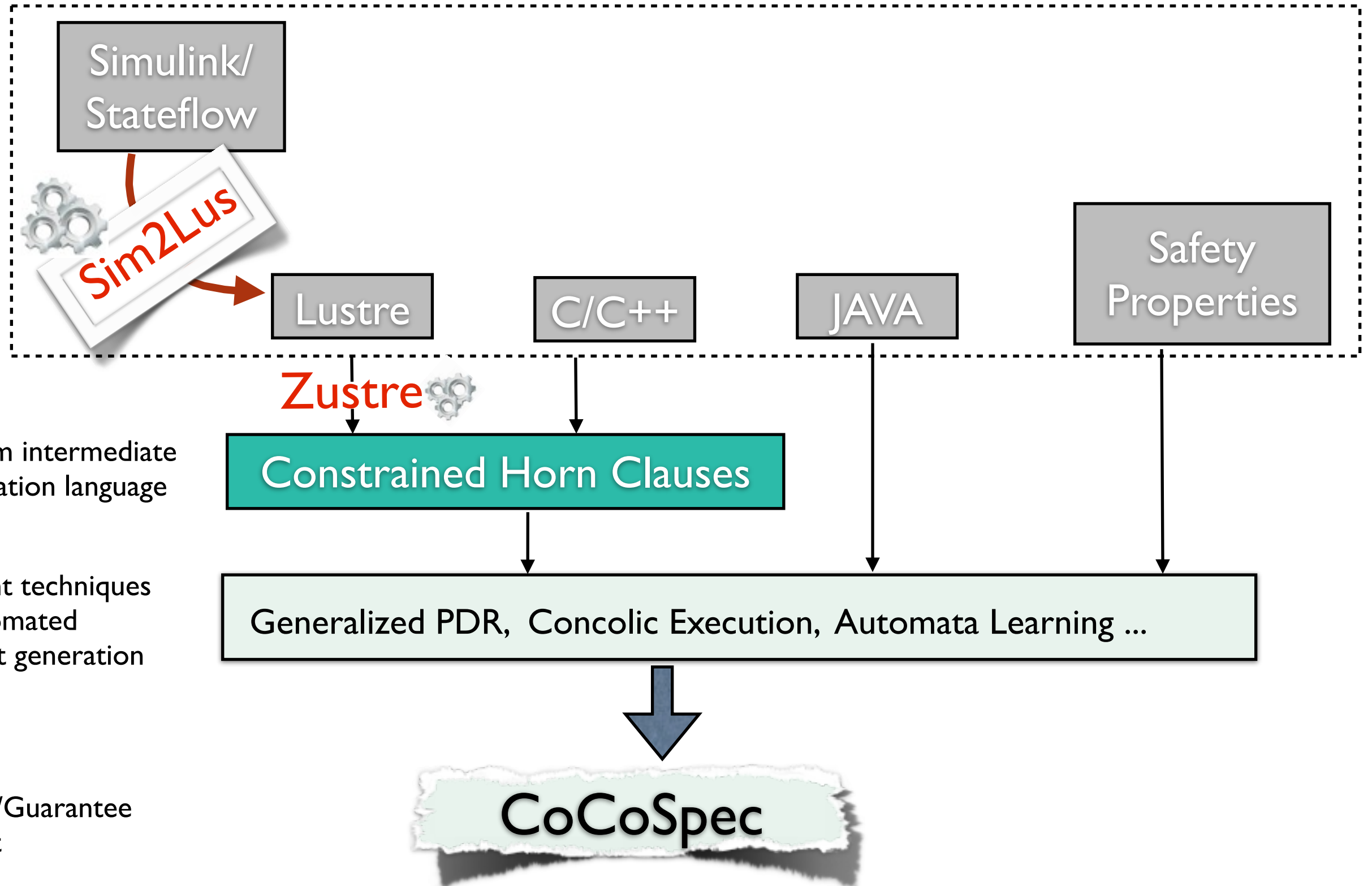


```
contract Sofar (X:bool) returns (Y:bool);
let
  ensure (Y = X) -> (Y = (pre(Y) and X));
tel
```

NB. We use Z3 tactics to simplify (manipulate) formulas

Pre-delivery Verification Stage

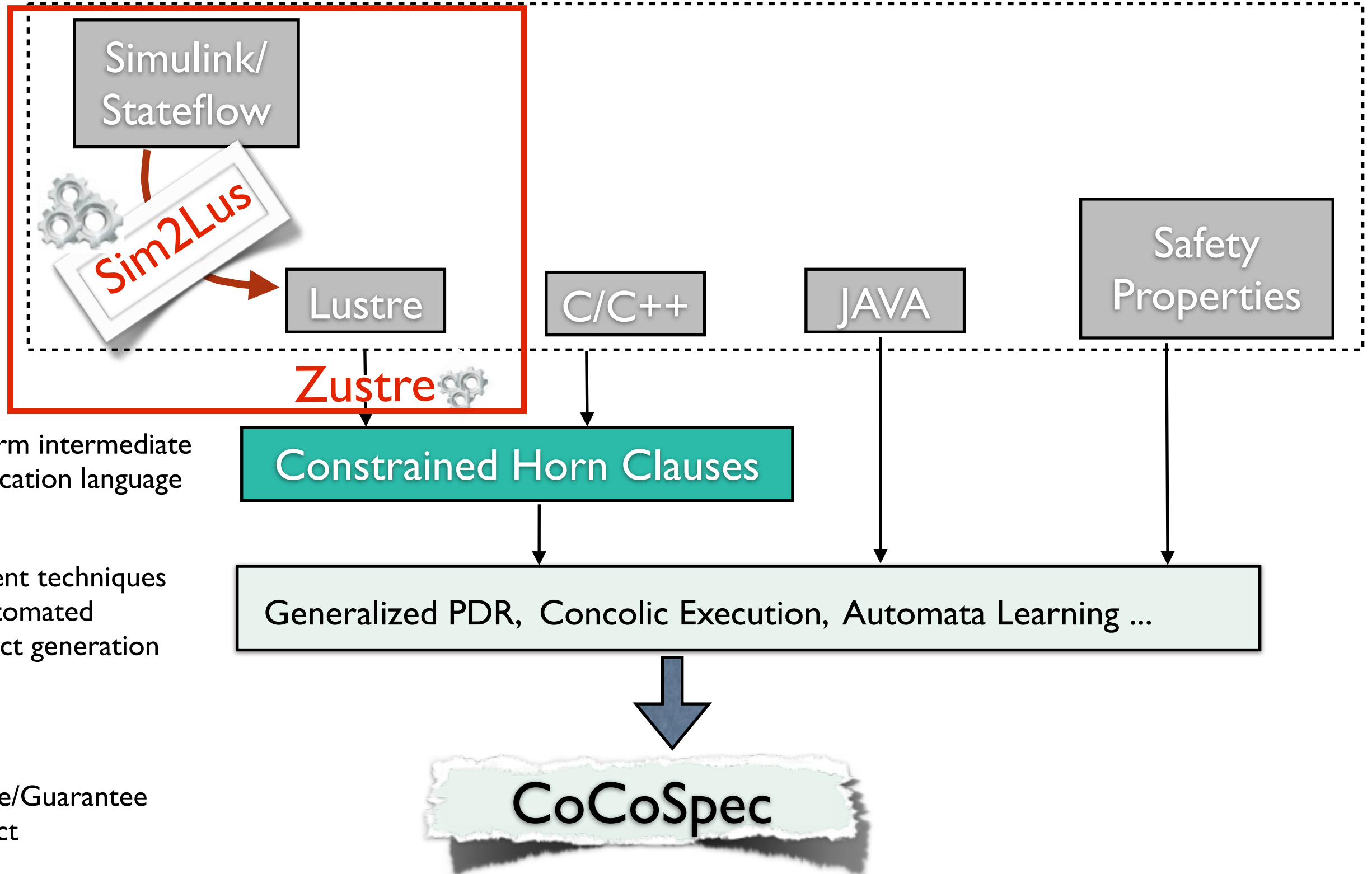
Our current approach:



Pre-delivery Verification Stage

Our current approach:

Integrated Analysis



Integrated Analysis Framework



```
MATLAB R2013a
HOME PLOTS APPS
New Script New Open Compare Import Save New Variable Analyze Code
Data Workspace Clear Workspace Run and Time Simulink Library
FILE VARIABLE CODE SIMULINK ENVIRONMENT RESOURCES
Search Documentation
/Users/teme/Documents/BitBucket/coco-simulink/tools/gac
>> coco(' ../../test/gac/properties/property_3_test.mdl')
(Info)[genecode] Welcome to the CoCo -- Contract generation and verification of Simulink models
MATLAB Sim2PreludeLustre is free software: you can redistribute it
and/or modify it under the terms of the GNU General Public License
as published by the Free Software Foundation, either version 3 of
the License, or (at your option) any later version.
MATLAB Sim2PreludeLustre is distributed in the hope that it will be
useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
General Public License for more details.
You should have received a copy of the GNU General Public License.
(Info)[genecode] Generating Lustre code from Simulink model: ../../test/gac/properties/property_3_test.mdl
(Info)[genecode] Internal representation building
(Info)[genecode] Printing original dataflow model
(Info)[genecode] Flattening of virtual SubSystems
(Info)[genecode] Printing flattened dataflow model
(Info)[genecode] Internal representation browsing for implicit data type conversions detection
(Info)[genecode] Printing flattened-type-converted dataflow model
(Info)[genecode] Code printing
(Warning)[write_code] A Terminator block have been found. No code will be generated for it:
property_3_test/Terminator
(Info)[genecode] End of code generation
(Info)[genecode] Cleaning temporary files
(Info)[Traceability] Traceability data generated in file: ../../test/gac/properties/src_property_3_test/property_3_test.trace
(Info)[Generation result] Lustre code generated in file: ../../test/gac/properties/src_property_3_test/property_3_test.lus
(Info)[Safety] Running Zustre

zustre =

/Users/teme/Documents/GitHub/zustre/src/

(Info)[Zustre property checking] Zustre result for property node [property_3_test_observer]: SAFE
fz >> |
```

Integrated Analysis Framework



The image displays the MATLAB R2013a environment. The left pane shows the Command Window with the following text:

```
>> coco('.../test/gac/properties/property_3_test.mdl')
(Info)[genecode] Welcome to the CoCo -- Contract generation and verification of Simulink mod
MATLAB Sim2PreludeLustre is free software: you can redistribute it
and/or modify it under the terms of the GNU General Public License
as published by the Free Software Foundation, either version 3 of
the License, or (at your option) any later version.
MATLAB Sim2PreludeLustre is distributed in the hope that it will be
useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
General Public License for more details.
You should have received a copy of the GNU General Public License.
(Info)[genecode] Generating Lustre code from Simulink model: .../test/gac/properties/prope
(Info)[genecode] Internal representation building
(Info)[genecode] Printing original dataflow model
(Info)[genecode] Flattening of virtual SubSystems
(Info)[genecode] Printing flattened dataflow model
(Info)[genecode] Internal representation browsing for implicit data type conversions detecti
(Info)[genecode] Printing flattened-type-converted dataflow model
(Info)[genecode] Code printing
(Warning)[write_code] A Terminator block have been found. No code will be generated for it:
property_3_test/Terminator
(Info)[genecode] End of code generation
(Info)[genecode] Cleaning temporary files
(Info)[Traceability] Traceability data generated in file: .../test/gac/properties/src_prop
(Info)[Generation result] Lustre code generated in file: .../test/gac/properties/src_prope
(Info)[Safety] Running Zustre

zustre =

/Users/teme/Documents/GitHub/zustre/src/

(Info)[Zustre property checking] Zustre result for property node [property_3_test_observer]: SAFE
fz >> |
```

The right pane shows a Simulink model diagram for 'property_3_test'. The diagram includes two input ports labeled 'In1' and 'In2'. 'In1' is connected to an 'Observer Block' (labeled 'observer') and a 'Relational Operator' block. 'In2' is connected to the 'Relational Operator' block. The output of the 'Relational Operator' block goes to a 'Switch' block. The 'Switch' block has two outputs: one goes to 'Out1' and the other goes to an 'Abs' block. The 'Observer Block' also has an output that goes to a 'Terminator' block. The 'Abs' block has an output labeled 'Out2'. The status bar at the bottom indicates 'Ready', '100%', and 'VariableStepDiscrete'.

*Specify safety properties
using synchronous observers*

Integrated Analysis Framework



The image displays the MATLAB R2013a environment. The left pane shows the command window with the following text:

```
>> coco(' ../../test/gac/properties/property_3_test.mdl')
(Info)[genecode] Welcome to the coco -- Contract generation and verification of Simulink mod
MATLAB Sim2PreludeLustre is free software: you can redistribute it
and/or modify it under the terms of the GNU General Public License
as published by the Free Software Foundation, either version 3 of
the License, or (at your option) any later version.
MATLAB Sim2PreludeLustre is distributed in the hope that it will be
useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
General Public License for more details.
You should have received a copy of the GNU General Public License.
(Info)[genecode] Generating Lustre code from Simulink model: ../../test/gac/properties/prope
(Info)[genecode] Internal representation building
(Info)[genecode] Printing original dataflow model
(Info)[genecode] Flattening of virtual SubSystems
(Info)[genecode] Printing flattened dataflow model
(Info)[genecode] Internal representation browsing for implicit data type conversions detecti
(Info)[genecode] Printing flattened-type-converted dataflow model
(Info)[genecode] Code printing
(Warning)[write_code] A Terminator block have been found. No code will be generated for it:
property_3_test/Terminator
(Info)[genecode] End of code generation
(Info)[genecode] Cleaning temporary files
(Info)[Traceability] Traceability data generated in file: ../../test/gac/properties/src_prop
(Info)[Generation result] Lustre code generated in file: ../../test/gac/properties/src_prope
(Info)[Safety] Running Zustre

zustre =

/Users/teme/Documents/GitHub/zustre/src/

(Info)[Zustre property checking] Zustre result for property node [property_3_test_observer]: SAFE
fz >> |
```

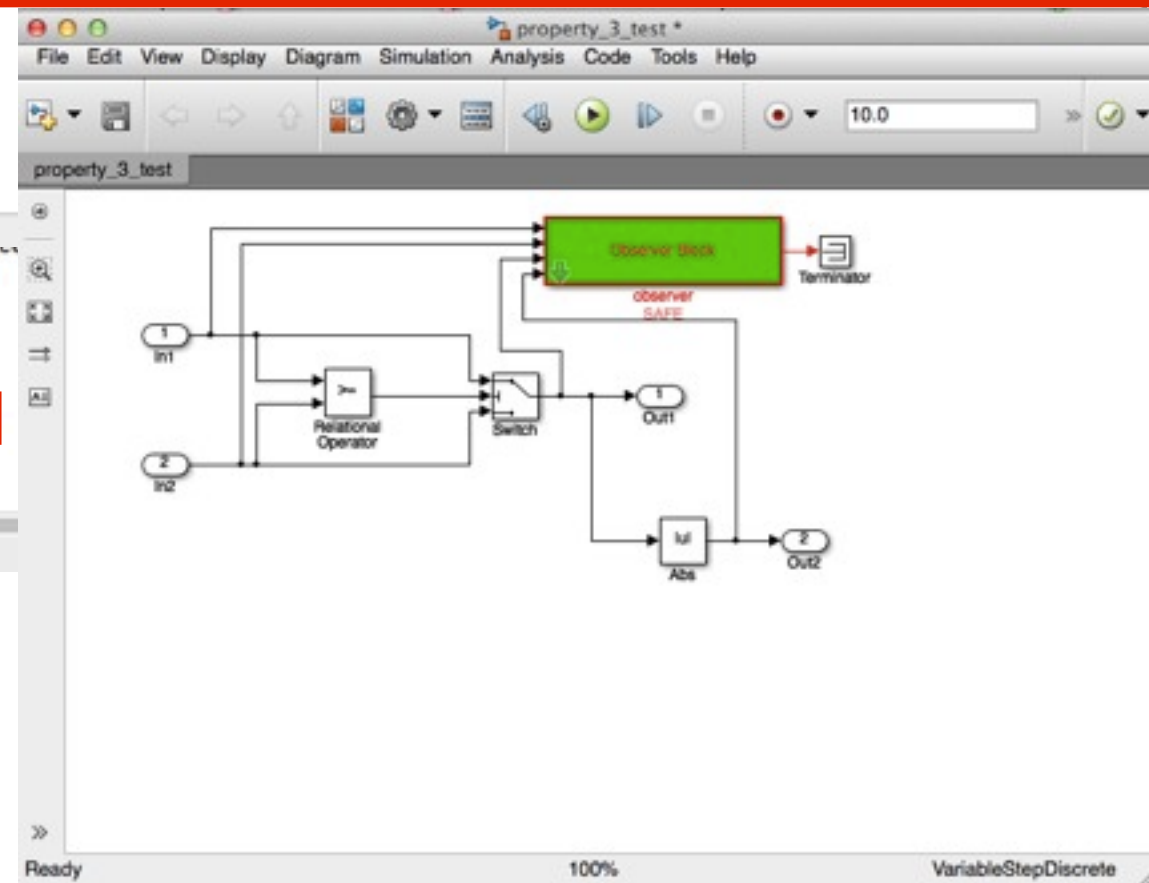
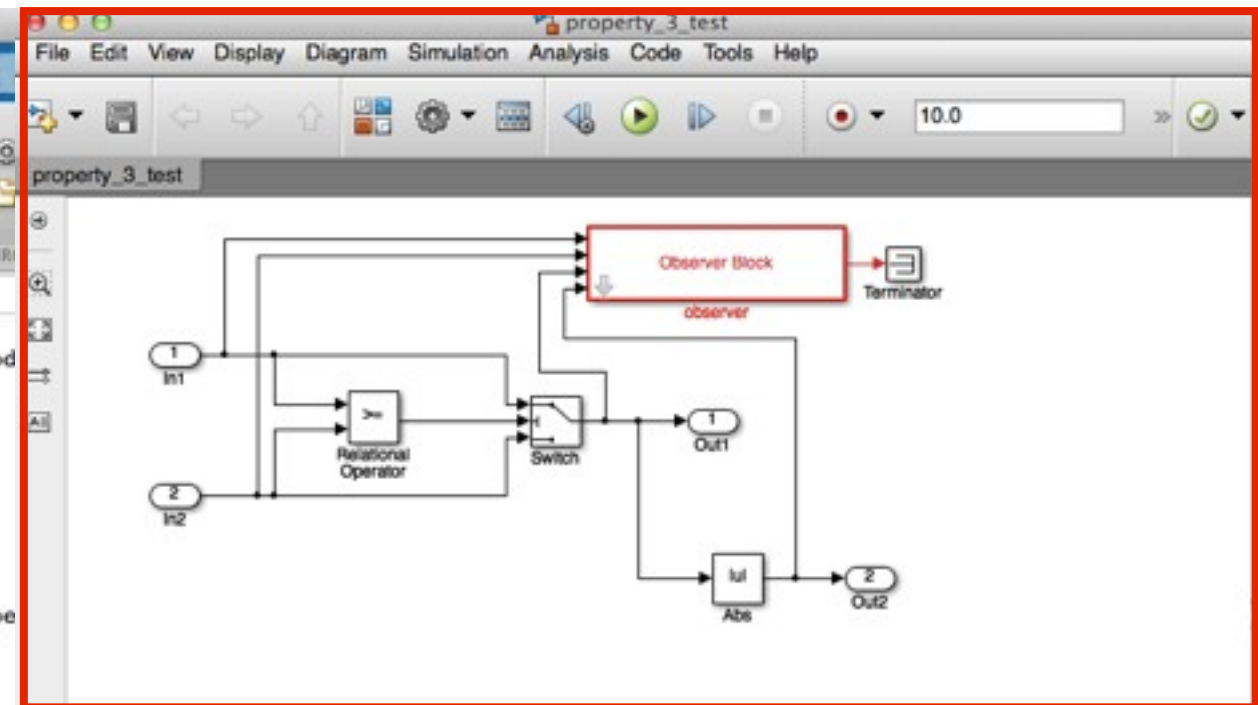
The right pane shows a Simulink model diagram for 'property_3_test'. The diagram includes two input ports labeled 'In1' and 'In2'. 'In1' is connected to a 'Relational Operator' block. 'In2' is connected to a 'Switch' block. The output of the 'Relational Operator' is connected to the 'Switch' block. The 'Switch' block has two outputs: one goes to an 'Observer Block' (labeled 'observer') and the other goes to an 'Abs' block. The 'Observer Block' is connected to a 'Terminator' block. The 'Abs' block has two outputs: 'Out1' and 'Out2'. The 'Observer Block' is highlighted with a red border.

*Specify safety properties
using synchronous observers*

Integrated Analysis Framework



```
MATLAB R2013a
HOME PLOTS APPS
New Script New Open Compare Import Data Save Workspace Clear Workspace Analyze Code Run and Time Clear Commands Simulink Library Layout
FILE VARIABLE CODE SIMULINK ENVIR
/Users/teme/Documents/BitBucket/coco-simulink/tools/gac
>> coco(' ../../test/gac/properties/property_3_test.mdl')
(Info)[genocode] Welcome to the coco -- Contract generation and verification of Simulink models
(Info)[genocode] MATLAB Sim2PreludeLustre is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.
(Info)[genocode] MATLAB Sim2PreludeLustre is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
(Info)[genocode] You should have received a copy of the GNU General Public License.
(Info)[genocode] Generating Lustre code from Simulink model: ../../test/gac/properties/property_3_test.mdl
(Info)[genocode] Internal representation building
(Info)[genocode] Printing original dataflow model
(Info)[genocode] Flattening of virtual SubSystems
(Info)[genocode] Printing flattened dataflow model
(Info)[genocode] Internal representation browsing for implicit data type conversions detected
(Info)[genocode] Printing flattened-type-converted dataflow model
(Info)[genocode] Code printing
(Warning)[write_code] A Terminator block have been found. No code will be generated for it: property_3_test/Terminator
(Info)[genocode] End of code generation
(Info)[genocode] Cleaning temporary files
(Info)[Traceability] Traceability data generated in file: ../../test/gac/properties/src_property_3_test/property_3_test_observer.lustre
(Info)[Generation result] Lustre code generated in file: ../../test/gac/properties/src_property_3_test/property_3_test_observer.lustre
(Info)[Safety] Running Zustre
zustre =
/Users/teme/Documents/GitHub/zustre/src/
(Info)[Zustre property checking] Zustre result for property node [property_3 test observer]: SAFE
>>
```

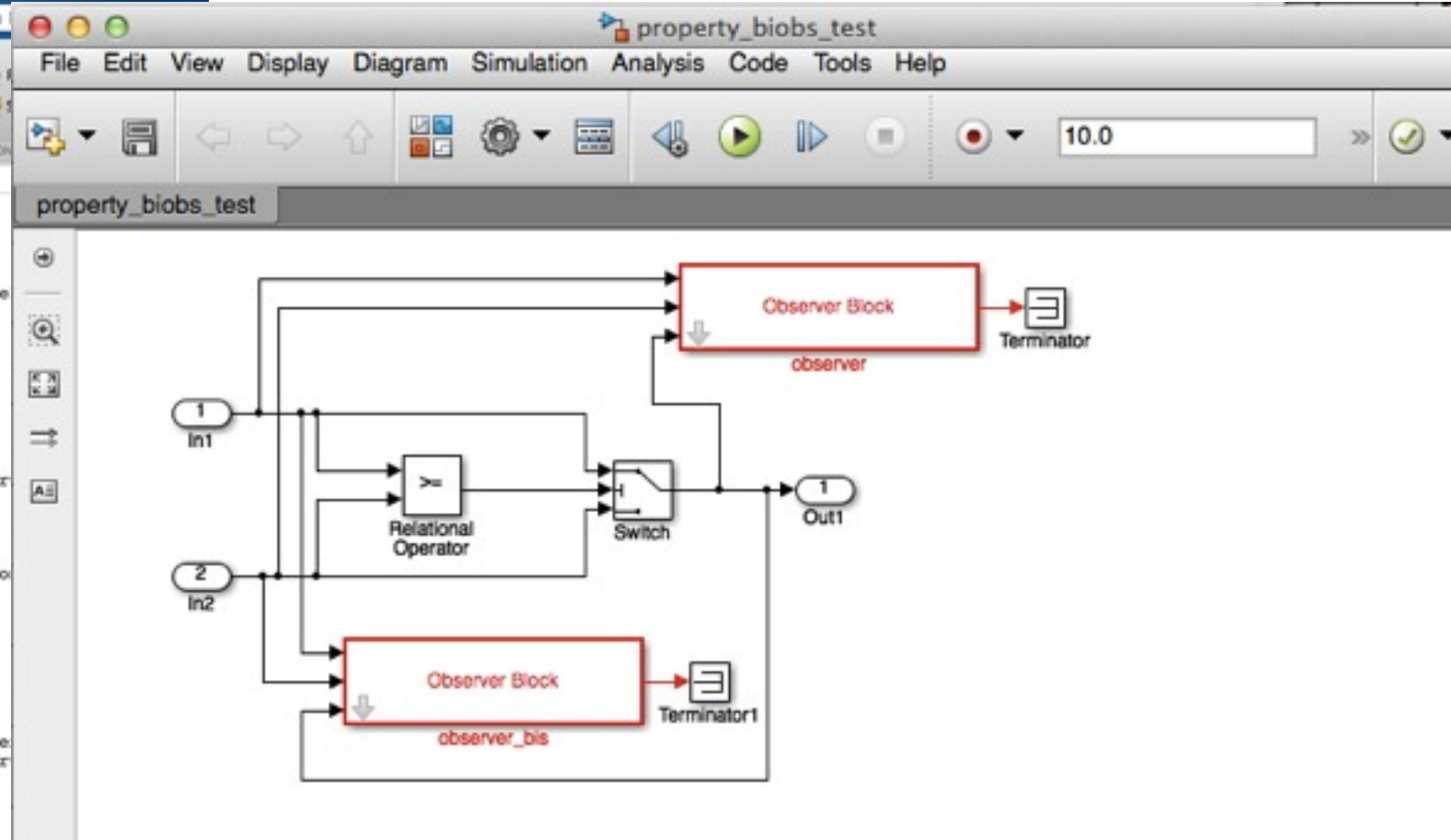


Specify safety properties using synchronous observers

Integrated Analysis Framework



```
MATLAB R2013a
HOME PLOTS APPS
New Script New Open Compare Import Data Save Workspace New Variable Open Variable Analyze Code Run and Time Simulink Library Layout
/Users/teme/Documents/BitBucket/coco-simulink/tools/gac
Warning: property_biobs_test.mdl, line 1491: line does not have a parameter named 'ZOrder'
In general/private/openmdl at 13
In open at 159
>> coco('.../test/gac/properties/property_biobs_test.mdl')
(Info)[genecode] Welcome to the CoCo -- Contract generation and verification of Simulink mode
MATLAB Sim2PreludeLustre is free software: you can redistribute it
and/or modify it under the terms of the GNU General Public License
as published by the Free Software Foundation, either version 3 of
the License, or (at your option) any later version.
MATLAB Sim2PreludeLustre is distributed in the hope that it will be
useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
General Public License for more details.
You should have received a copy of the GNU General Public License.
(Info)[genecode] Generating Lustre code from Simulink model: .../test/gac/properties/propert
(Info)[genecode] Internal representation building
(Info)[genecode] Printing original dataflow model
(Info)[genecode] Flattening of virtual SubSystems
(Info)[genecode] Printing flattened dataflow model
(Info)[genecode] Internal representation browsing for implicit data type conversions detectio
(Info)[genecode] Printing flattened-type-converted dataflow model
(Info)[genecode] Code printing
(Warning)[write_code] A Terminator block have been found. No code will be generated for it:
property_biobs_test/Terminator
(Warning)[write_code] A Terminator block have been found. No code will be generated for it:
property_biobs_test/Terminator1
(Info)[genecode] End of code generation
(Info)[genecode] Cleaning temporary files
(Info)[Traceability] Traceability data generated in file: .../test/gac/properties/src_prope
(Info)[Generation result] Lustre code generated in file: .../test/gac/properties/src_proper
(Info)[Safety] Running Zustre
zustre =
/Users/teme/Documents/GitHub/zustre/src/
(Info)[Zustre property checking] Zustre result for property node [property_biobs_test_observer]: SAFE
(Info)[Zustre property checking] Zustre result for property node [property_biobs_test_observer_bis]: CEX
>> |
```



Integrated Analysis Framework



The image displays the MATLAB R2013a environment with two windows. The left window shows the Command Window with the following text:

```
In open at 159
Warning: property_biobs_test.mdl, line 1491: line does not have a parameter named 'ZOrder'
In general/private/openmdl at 13
In open at 159
>> cocol('.../test/gac/properties/property_biobs_test.mdl')
(Info)[genecode] Welcome to the CoCo -- Contract generation and verification of Simulink mode
MATLAB Sim2PreludeLustre is free software: you can redistribute it
and/or modify it under the terms of the GNU General Public License
as published by the Free Software Foundation, either version 3 of
the License, or (at your option) any later version.
MATLAB Sim2PreludeLustre is distributed in the hope that it will be
useful, but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
General Public License for more details.
You should have received a copy of the GNU General Public License.
(Info)[genecode] Generating Lustre code from Simulink model: .../test/gac/properties/propert
(Info)[genecode] Internal representation building
(Info)[genecode] Printing original dataflow model
(Info)[genecode] Flattening of virtual SubSystems
(Info)[genecode] Printing flattened dataflow model
(Info)[genecode] Internal representation browsing for implicit data type conversions detectio
(Info)[genecode] Printing flattened-type-converted dataflow model
(Info)[genecode] Code printing
(Warning)[write_code] A Terminator block have been found. No code will be generated for it:
property_biobs_test/Terminator
(Warning)[write_code] A Terminator block have been found. No code will be generated for it:
property_biobs_test/Terminator1
(Info)[genecode] End of code generation
(Info)[genecode] Cleaning temporary files
(Info)[Traceability] Traceability data generated in file: .../test/gac/properties/src_prope
(Info)[Generation result] Lustre code generated in file: .../test/gac/properties/src_proper
(Info)[Safety] Running Zustre

zustre =
/Users/teme/Documents/GitHub/zustre/src/

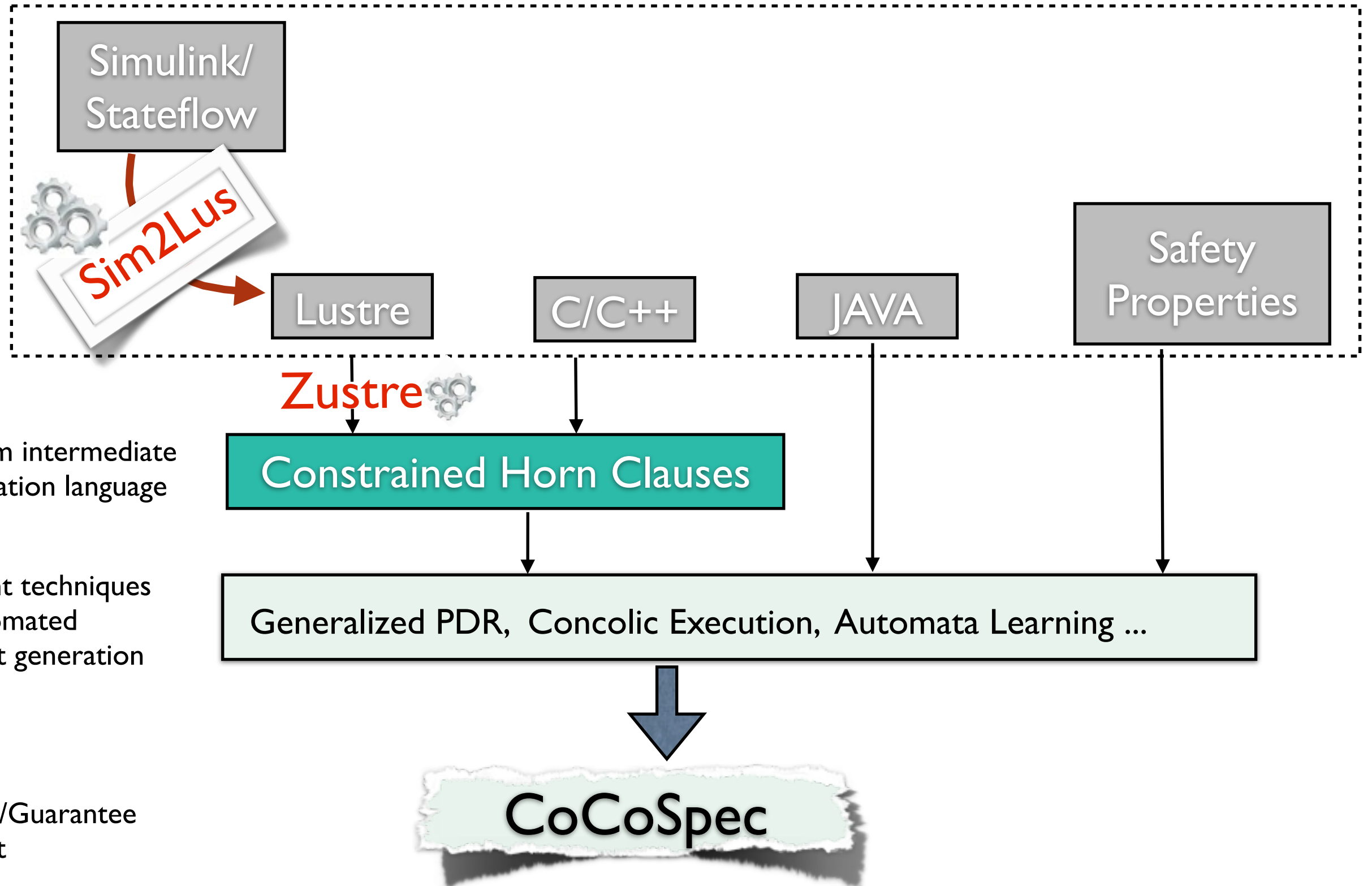
(Info)[Zustre property checking] Zustre result for property node [property_biobs_test_observer]: SAFE
(Info)[Zustre property checking] Zustre result for property node [property_biobs_test_observer_bis]: CEX
fx >> |
```

The right window shows the Simulink model 'property_biobs_test'. The top diagram shows the initial state with two 'Observer Block' components: 'observer' and 'observer_bis'. The 'observer' block is connected to a 'Terminator' block. The 'observer_bis' block is connected to a 'Terminator1' block. The diagram includes inputs 'In1' and 'In2', a 'Relational Operator' block, a 'Switch' block, and an output 'Out1'. A large blue arrow points from the top diagram to the bottom diagram.

The bottom diagram shows the same Simulink model after analysis. The 'observer' block is now highlighted in green and labeled 'Property: observer SAFE'. The 'observer_bis' block is highlighted in red and labeled 'Property: observer_bis CEX'. The 'Terminator' and 'Terminator1' blocks are also present.

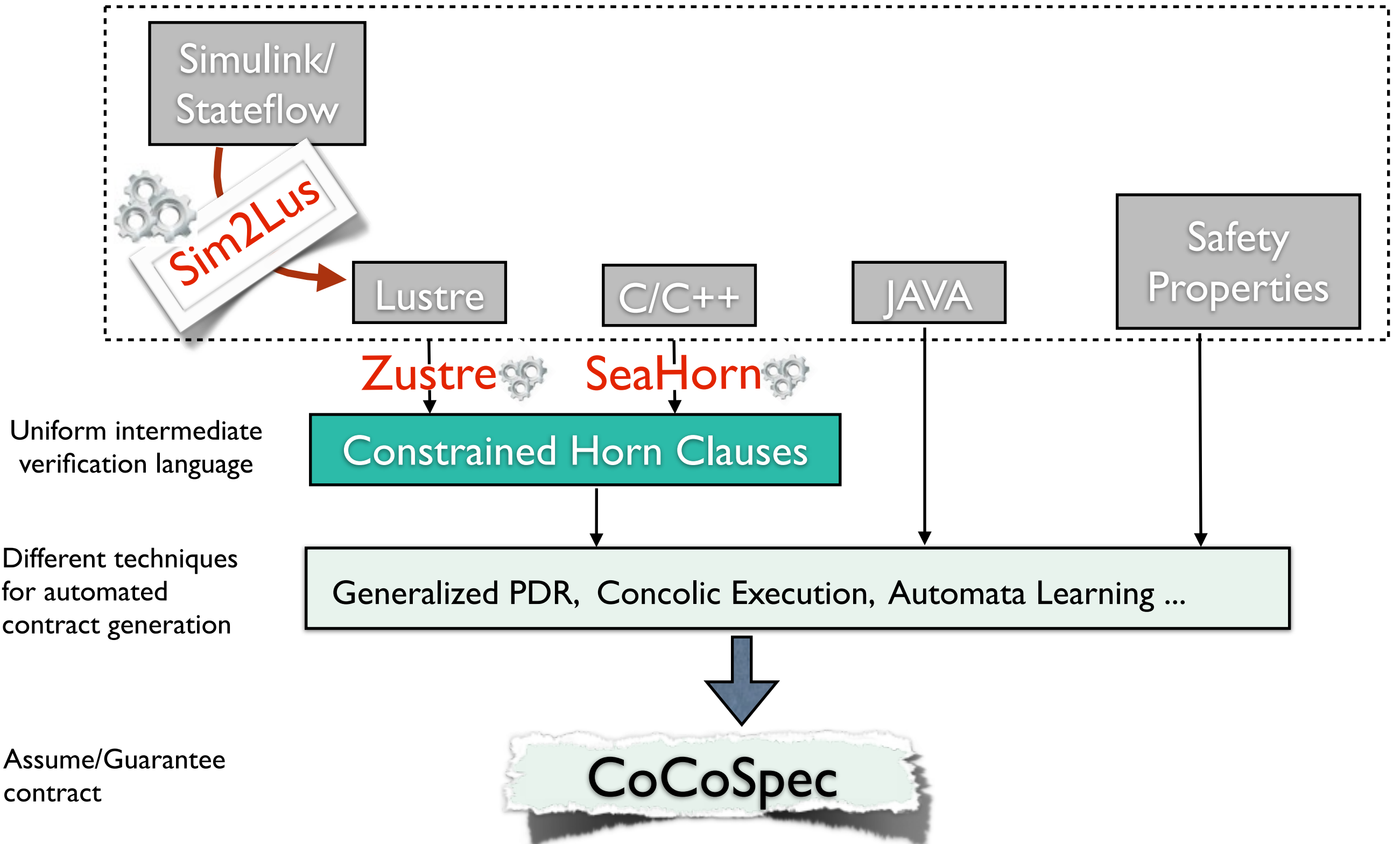
Pre-delivery Verification Stage

Our current approach:



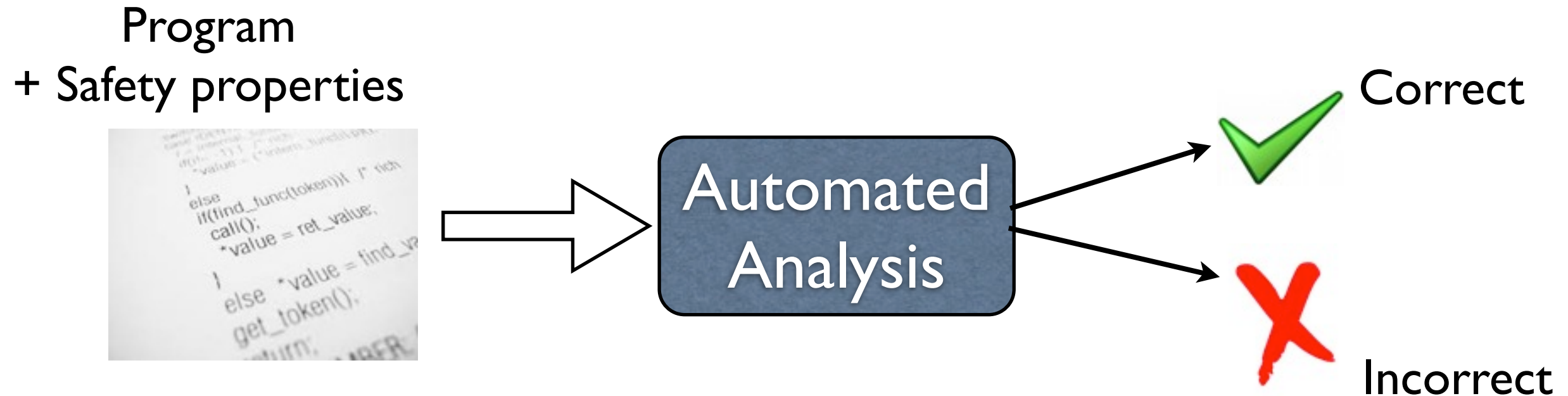
Pre-delivery Verification Stage

Our current approach:



SeaHorn

A framework for verifying LLVM-based programs



NB. (i) *Current version targets C programs*
(ii) *and does not generate CoCoSpec*

A. Gurfinkel, T. Kahsai, J. Navas, :“*Algorithmic Logic-based verification*”. In ACM-SIGLOG, April 2015.

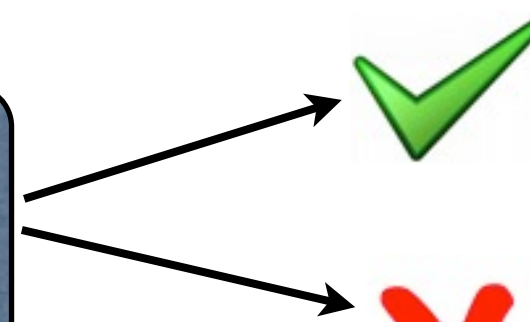
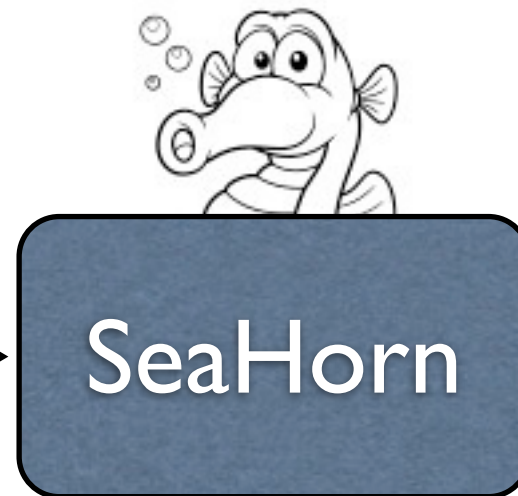
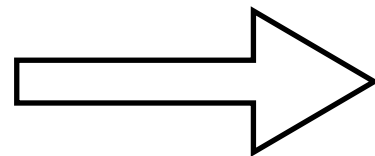
A. Gurfinkel, T. Kahsai, J. Navas, :“*SeaHorn: A framework for verifying C programs (competition contribution)*”. In SVCOMP (TACAS-2015).

A. Gurfinkel ,T. Kahsai, A. Komuravelli, J. Navas, :“*The SeaHorn Verification Framework*”. In CAV 2015.

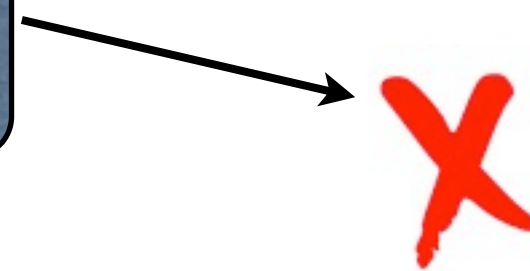
SeaHorn

A framework for verifying LLVM-based programs

Program
+ Safety properties



SAFE
+ Certificate



UNSAFE
+ CEX

NB. (i) *Current version targets C programs*
(ii) *and does not generate CoCoSpec*

A. Gurfinkel, T. Kahsai, J. Navas, :“*Algorithmic Logic-based verification*”. In ACM-SIGLOG, April 2015.

A. Gurfinkel, T. Kahsai, J. Navas, :“*SeaHorn: A framework for verifying C programs (competition contribution)*”. In SVCOMP (TACAS-2015).

A. Gurfinkel ,T. Kahsai, A. Komuravelli, J. Navas, :“*The SeaHorn Verification Framework*”. In CAV 2015.

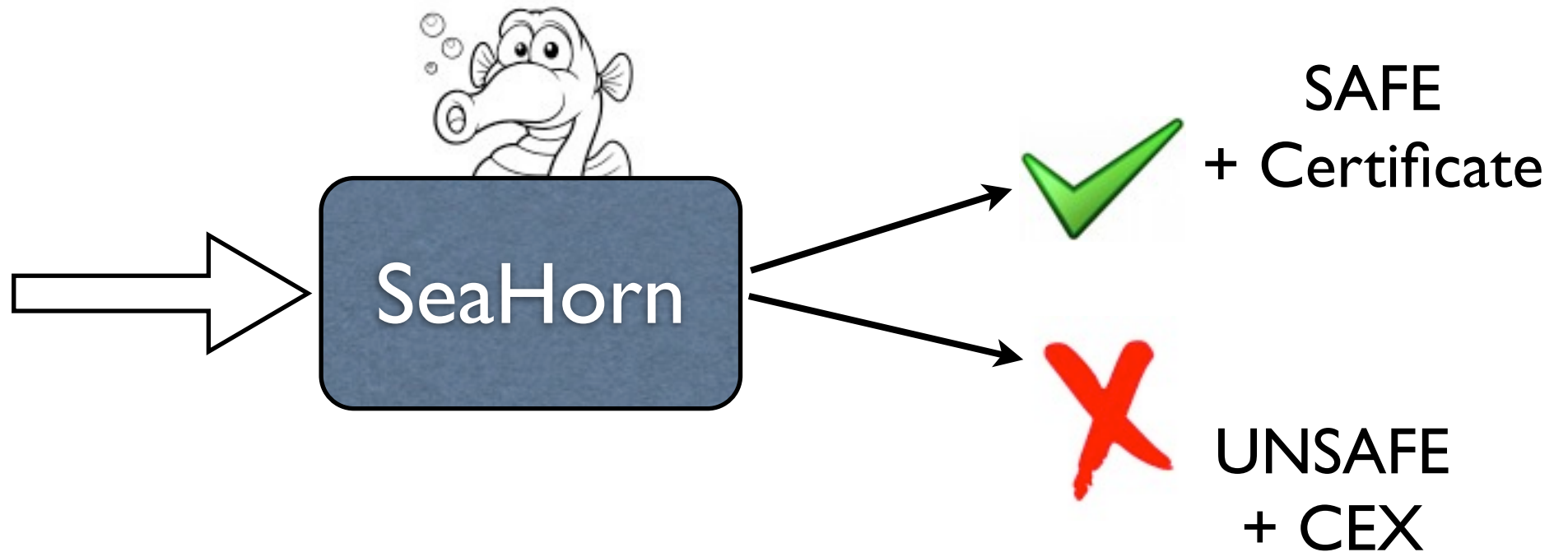
SeaHorn



Dr. Arie Gurfinkel
(SEI / CMU)

A framework for verifying LLVM-based programs

Program
+ Safety properties



NB. (i) *Current version targets C programs*
(ii) *and does not generate CoCoSpec*

A. Gurfinkel, T. Kahsai, J. Navas, :“*Algorithmic Logic-based verification*”. In ACM-SIGLOG, April 2015.

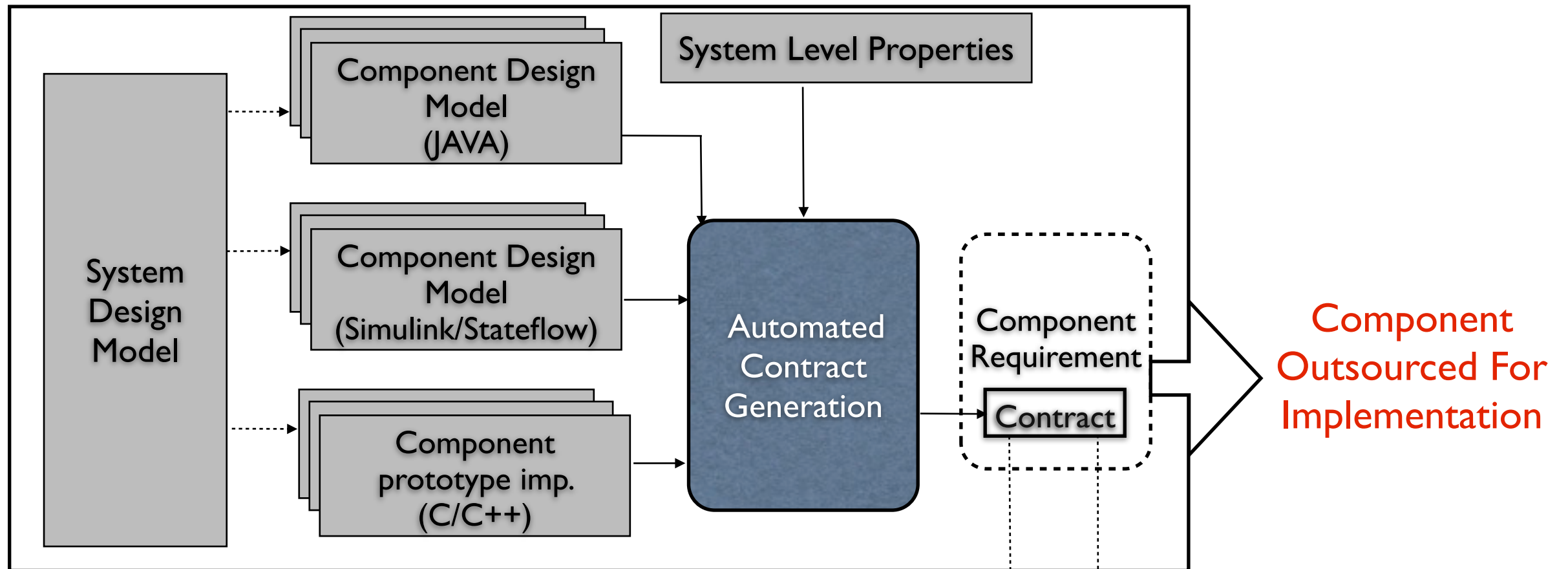
A. Gurfinkel, T. Kahsai, J. Navas, :“*SeaHorn: A framework for verifying C programs (competition contribution)*”. In SVCOMP (TACAS-2015).

A. Gurfinkel ,T. Kahsai, A. Komuravelli, J. Navas, :“*The SeaHorn Verification Framework*”. In CAV 2015.

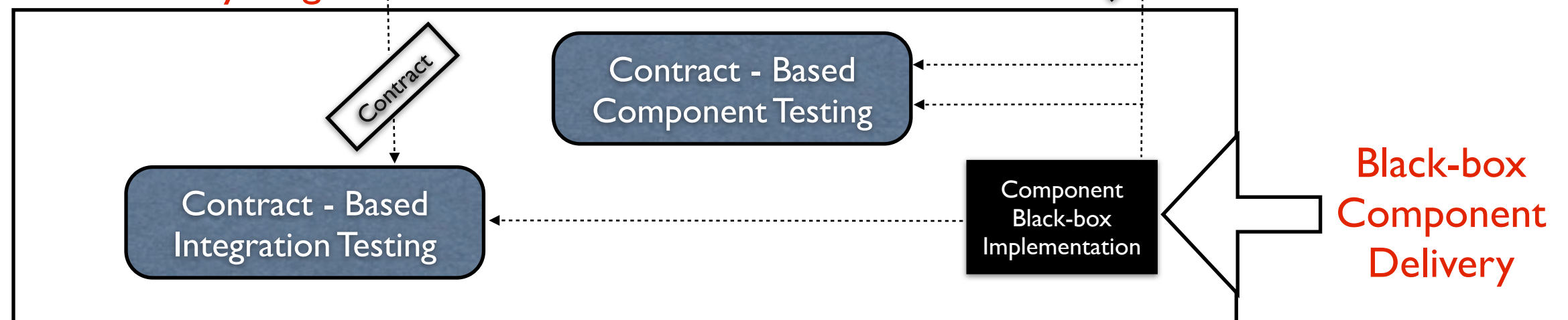
Post-delivery verification stage

Two Stage solution for virtual integration

Pre-Delivery Stage

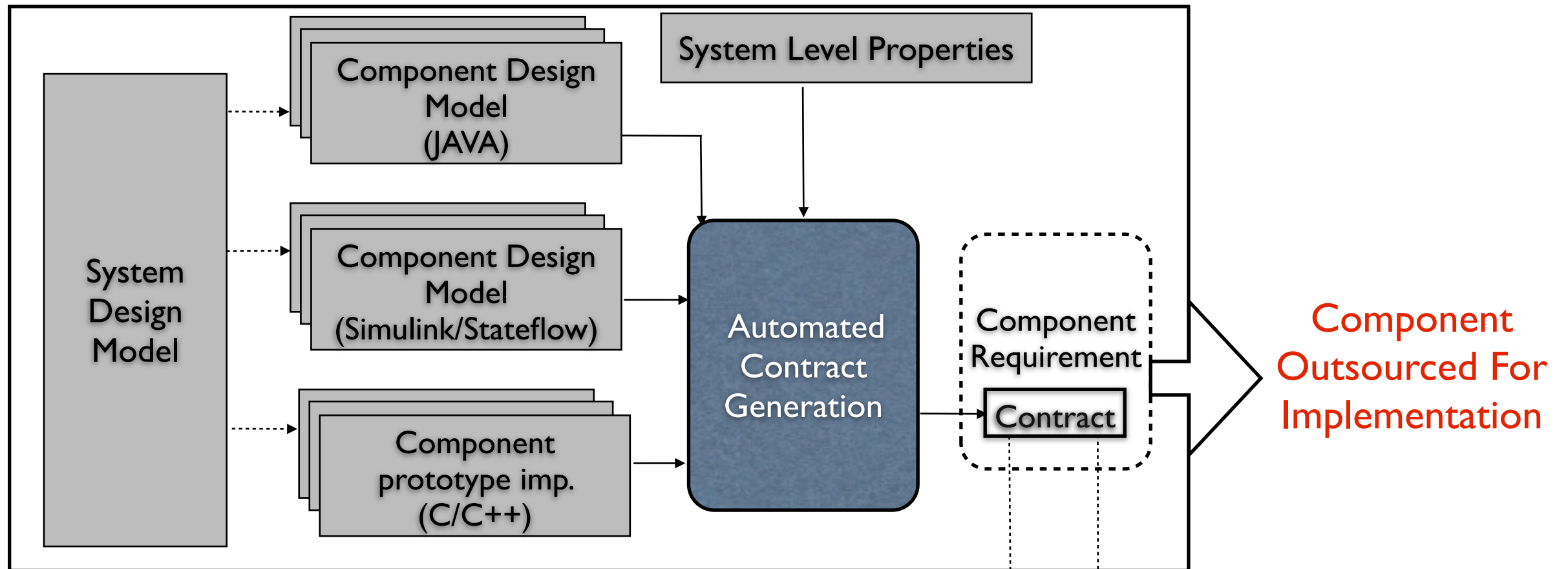


Post-Delivery stage

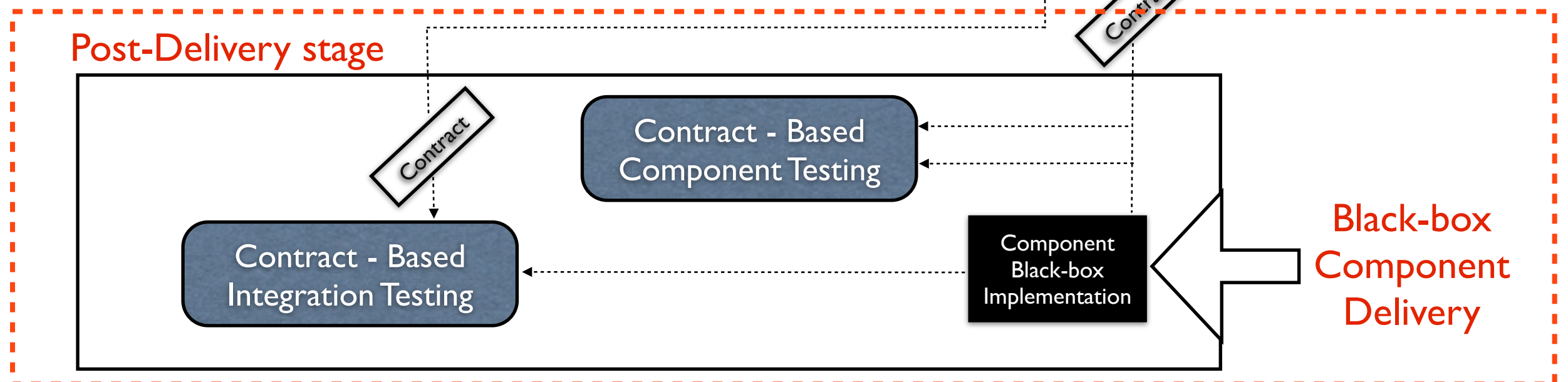


Two Stage solution for virtual integration

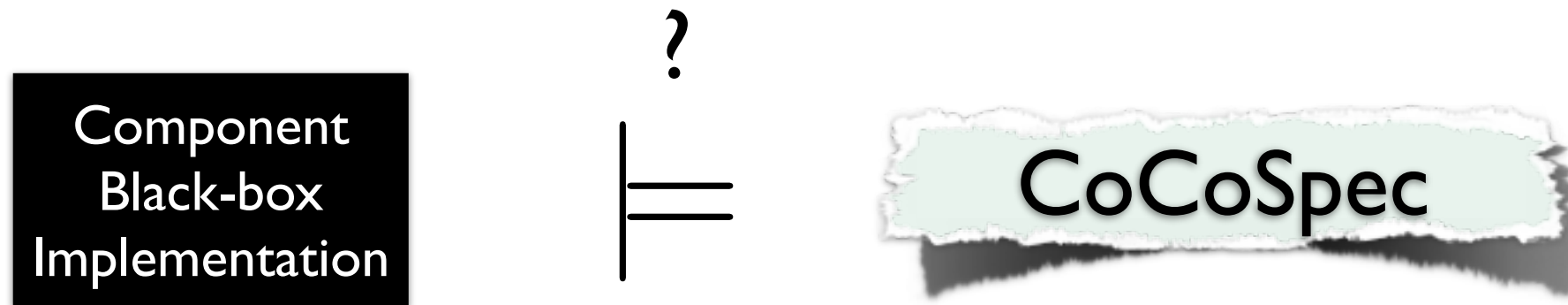
Pre-Delivery Stage



Post-Delivery stage



Post-delivery Verification Stage



Contract-based test generation

- Test generation via Bounded Model Checking
- **Coverage** and **mutation** oriented
- **TestEAS**: test execution and analysis system

Test generation via BMC

Components are represented as **transition systems**:

- s is the vector of state variables of the system
- $\mathcal{I}(s_0)$ is the **init predicate**, true if s_0 is initial
- $\mathcal{T}(s_i, s_{i+1})$ is the **transition predicate**, true if s_{i+1} is a successor of s_i

Given a **test objective** $\mathcal{O}(s)$, we can query an *SMT solver* for a trace of k states leading to it:

$$\mathcal{I}(s_0) \wedge \mathcal{T}(s_0, s_1) \wedge \cdots \wedge \mathcal{T}(s_{k-2}, s_{k-1}) \wedge \mathcal{O}(s_{k-1})$$

Test generation via BMC

Components are represented as **transition systems**:

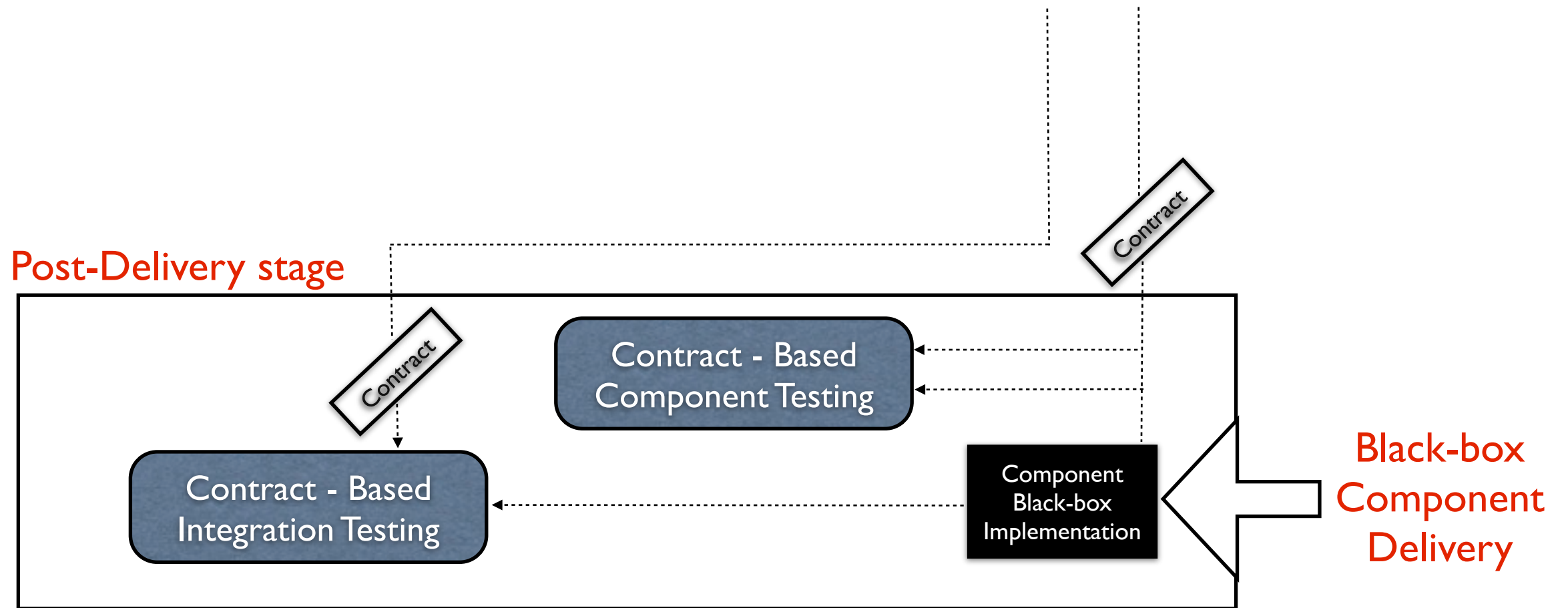
- s is the vector of state variables of the system
- $\mathcal{I}(s_0)$ is the **init predicate**, true if s_0 is initial
- $\mathcal{T}(s_i, s_{i+1})$ is the **transition predicate**, true if s_{i+1} is a successor of s_i

Given a **test objective** $\mathcal{O}(s)$, we can query an *SMT solver* for a trace of k states leading to it:

$$\mathcal{I}(s_0) \wedge \mathcal{T}(s_0, s_1) \wedge \cdots \wedge \mathcal{T}(s_{k-2}, s_{k-1}) \wedge \mathcal{O}(s_{k-1})$$

- **Coverage-oriented**: the set of test cases are generated to realize some coverage criterion on the source file, e.g. (O)MC/DC.
- **Mutation-based**: alter the syntax of the source code and generate test cases failing on (*killing*) the mutants.

Post-delivery Integration testing



A. Cimatti et al :*“A property-based proof system for contract based design”*. In SEAA 2012.

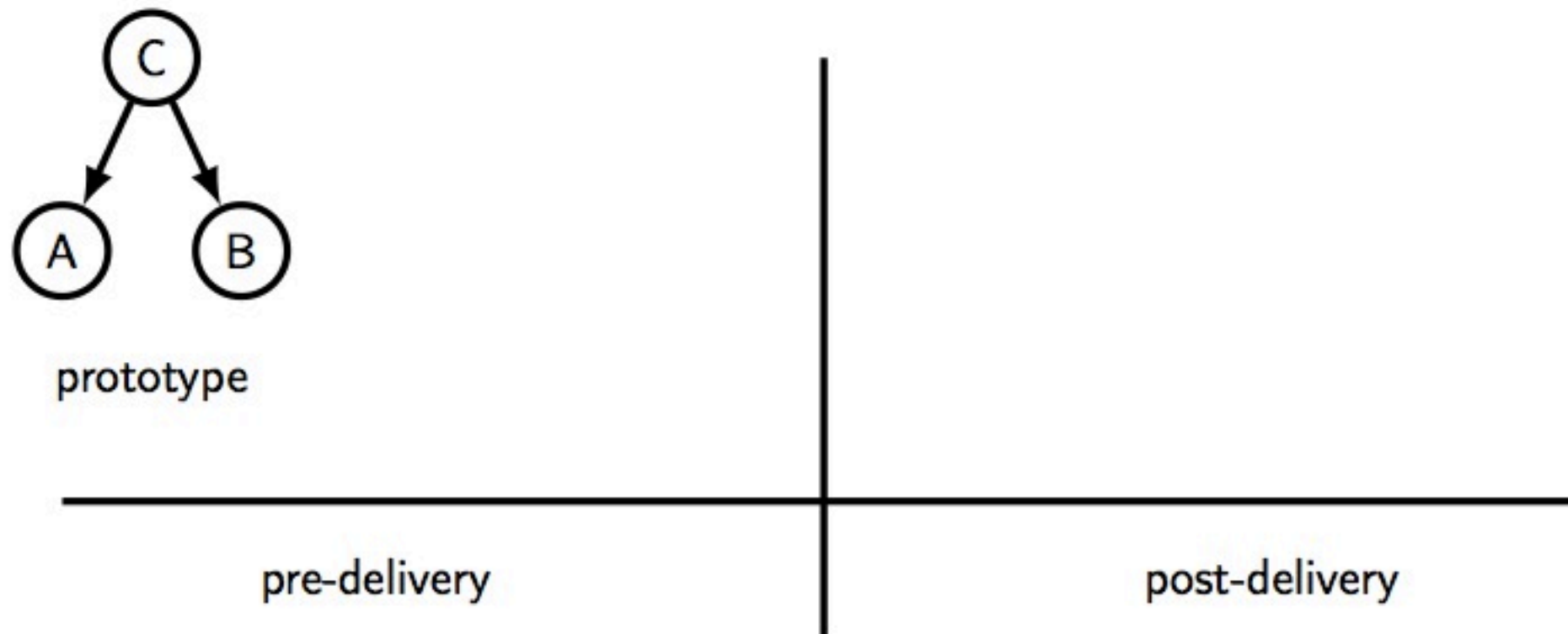
W. Damm et al :*“Using contract-based component specifications for virtual integration and architecture design”*. In DATE 2011.

E. Kessler et al :*“Assessing COTS software in a certifiable safety-critical domain”*. In Information Systems Journal 2008.

A. Benveniste et al :*“Multiple Viewpoint Contract-based Specification and Design”*. In FMCO 2007.

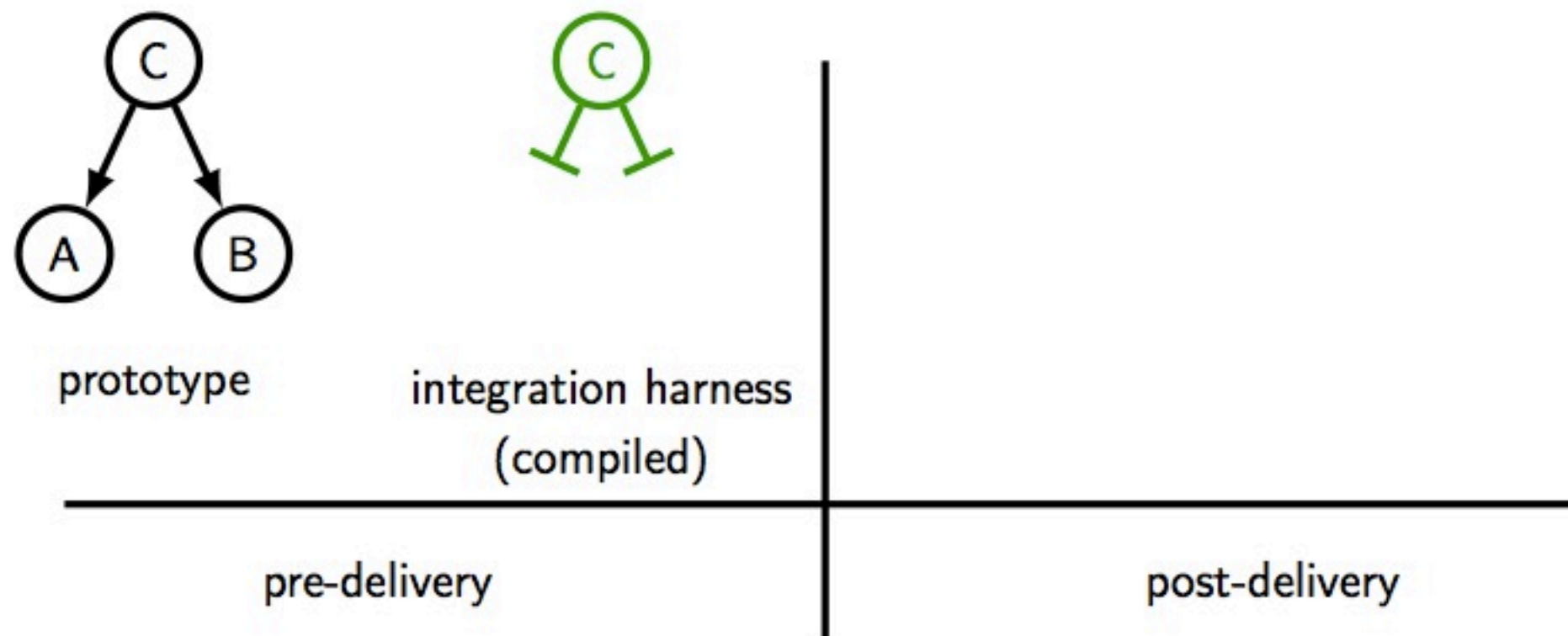
Post-delivery integration testing

- pre-delivery:
 - contract-based test generation for all components,



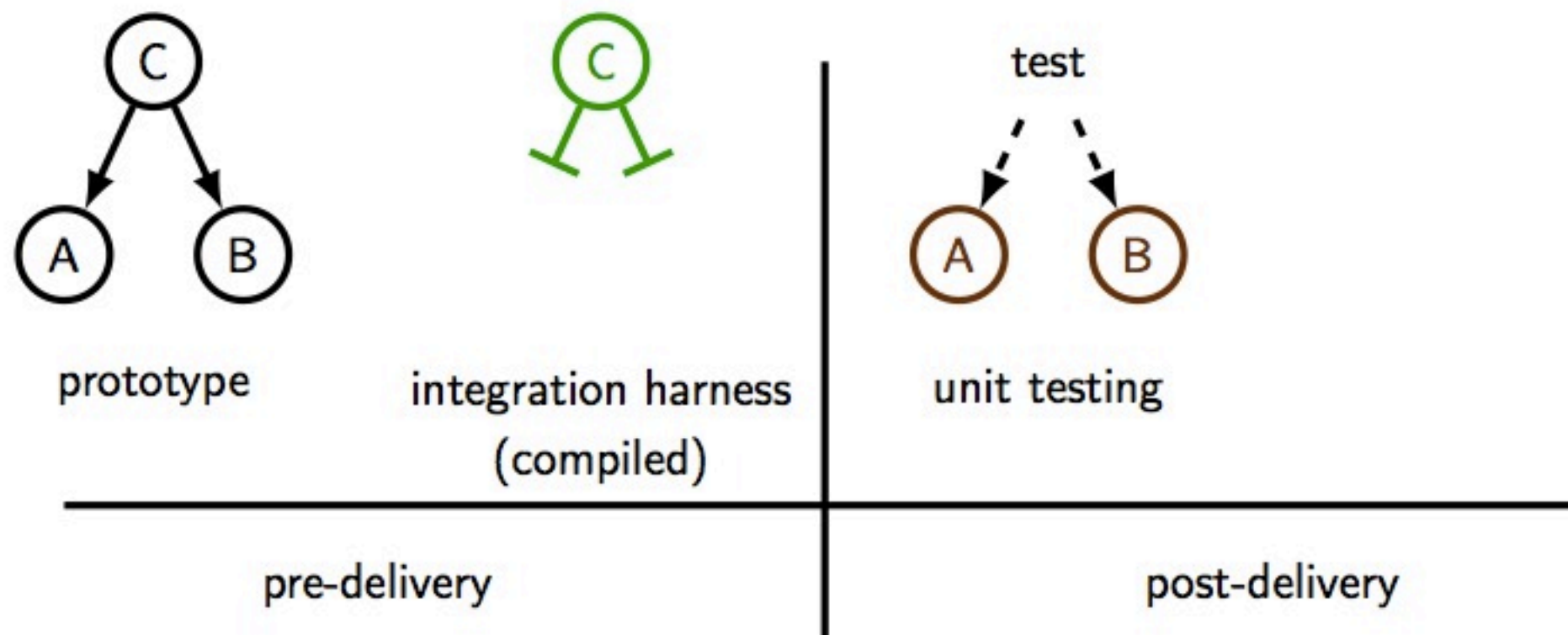
Post-delivery integration testing

- pre-delivery:
 - contract-based test generation for all components,
 - compile complex components without their subcomponents,



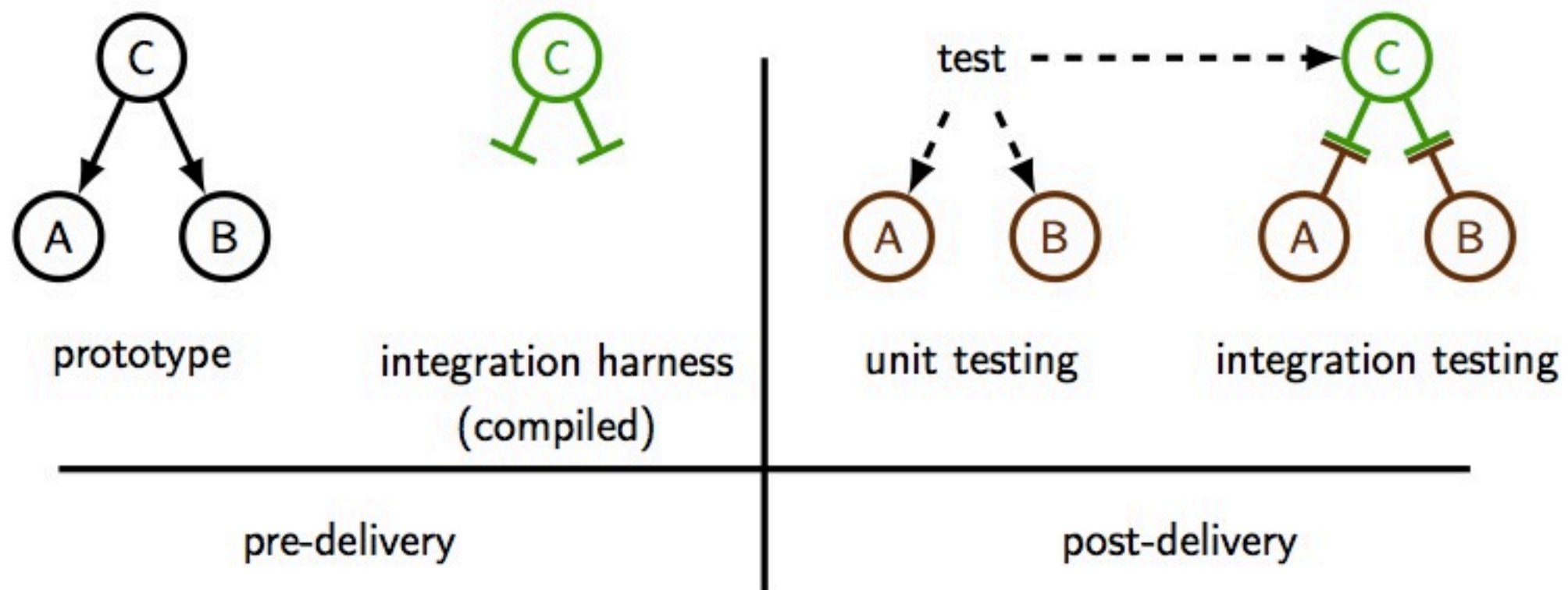
Post-delivery integration testing

- pre-delivery:
 - contract-based test generation for all components,
 - compile complex components without their subcomponents,
- post-delivery:
 - unit testing of the binaries,



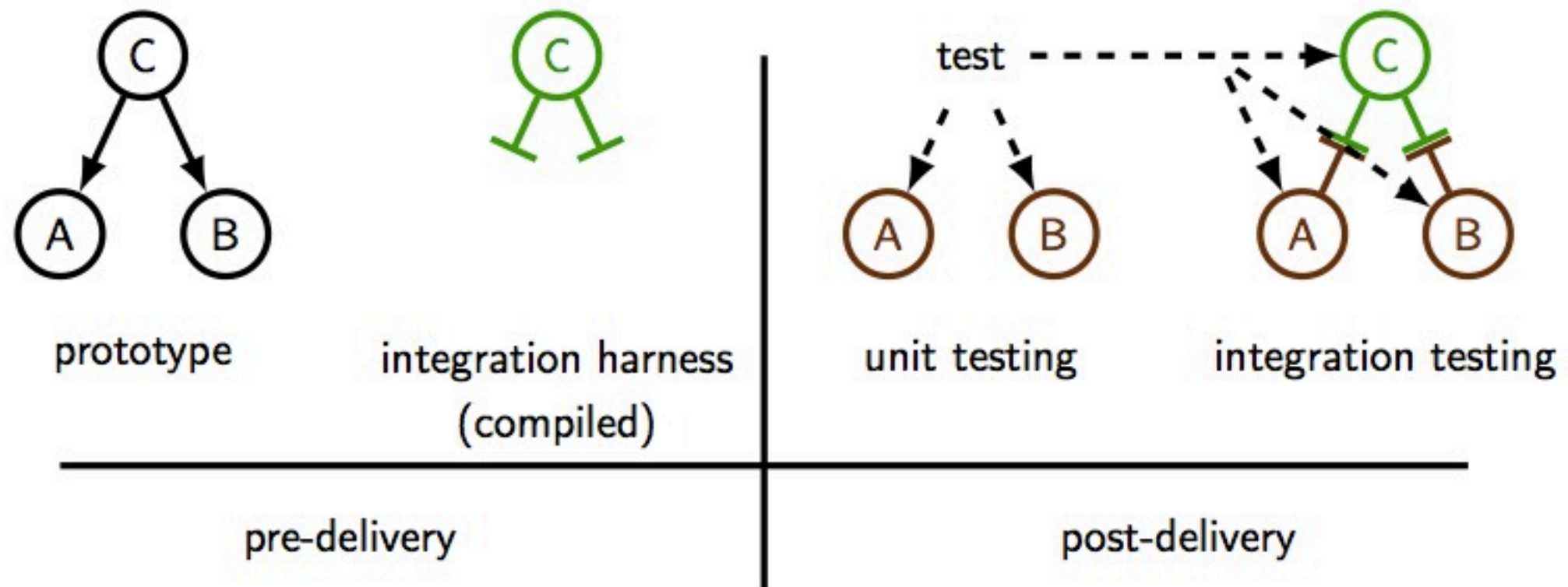
Post-delivery integration testing

- pre-delivery:
 - contract-based test generation for all components,
 - compile complex components without their subcomponents,
- post-delivery:
 - unit testing of the binaries,
 - integration testing using the compiled prototype component.



Post-delivery integration testing

- pre-delivery:
 - contract-based test generation for all components,
 - compile complex components without their subcomponents,
- post-delivery:
 - unit testing of the binaries,
 - integration testing using the compiled prototype component.



FCS Case Studies

Transport Class Model (TCM)



The TCM – a twin-engine tube and wings configuration aircraft simulation, scaled up from the **Generic Transport Model (GTM)**.

Transport Class Model (TCM)



The TCM – a twin-engine tube and wings configuration aircraft simulation, scaled up from the **Generic Transport Model (GTM)**.

UAV-sized (wingspan ~6ft) version of a plane with geometry similar to a transport -class aircraft

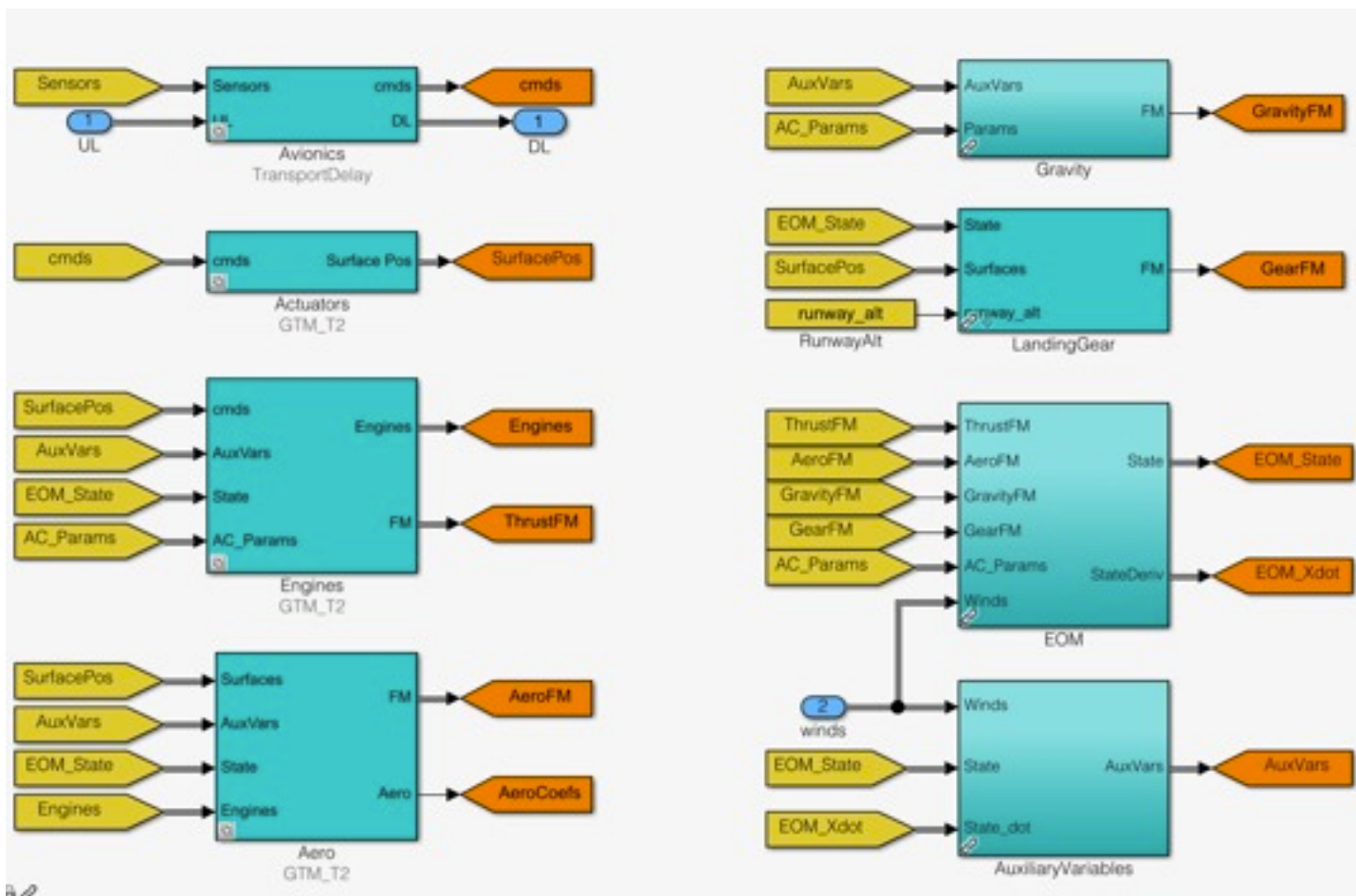
Intended as an experimental platform for controls and health management system

Transport Class Model (TCM)



The TCM – a twin-engine tube and wings configuration aircraft simulation, scaled up from the **Generic Transport Model (GTM)**.

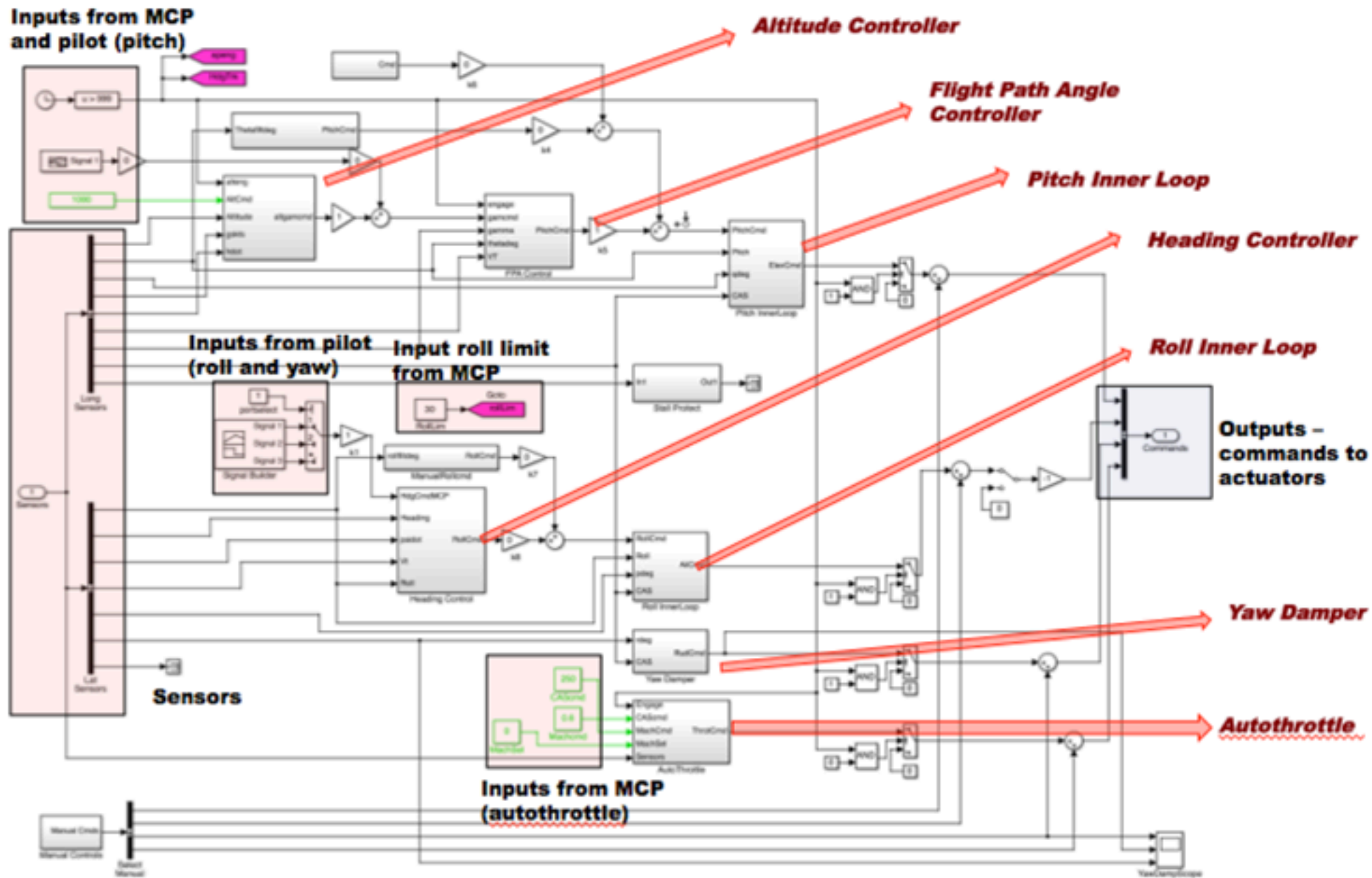
UAV-sized (wingspan ~6ft) version of a plane with geometry similar to a transport -class aircraft



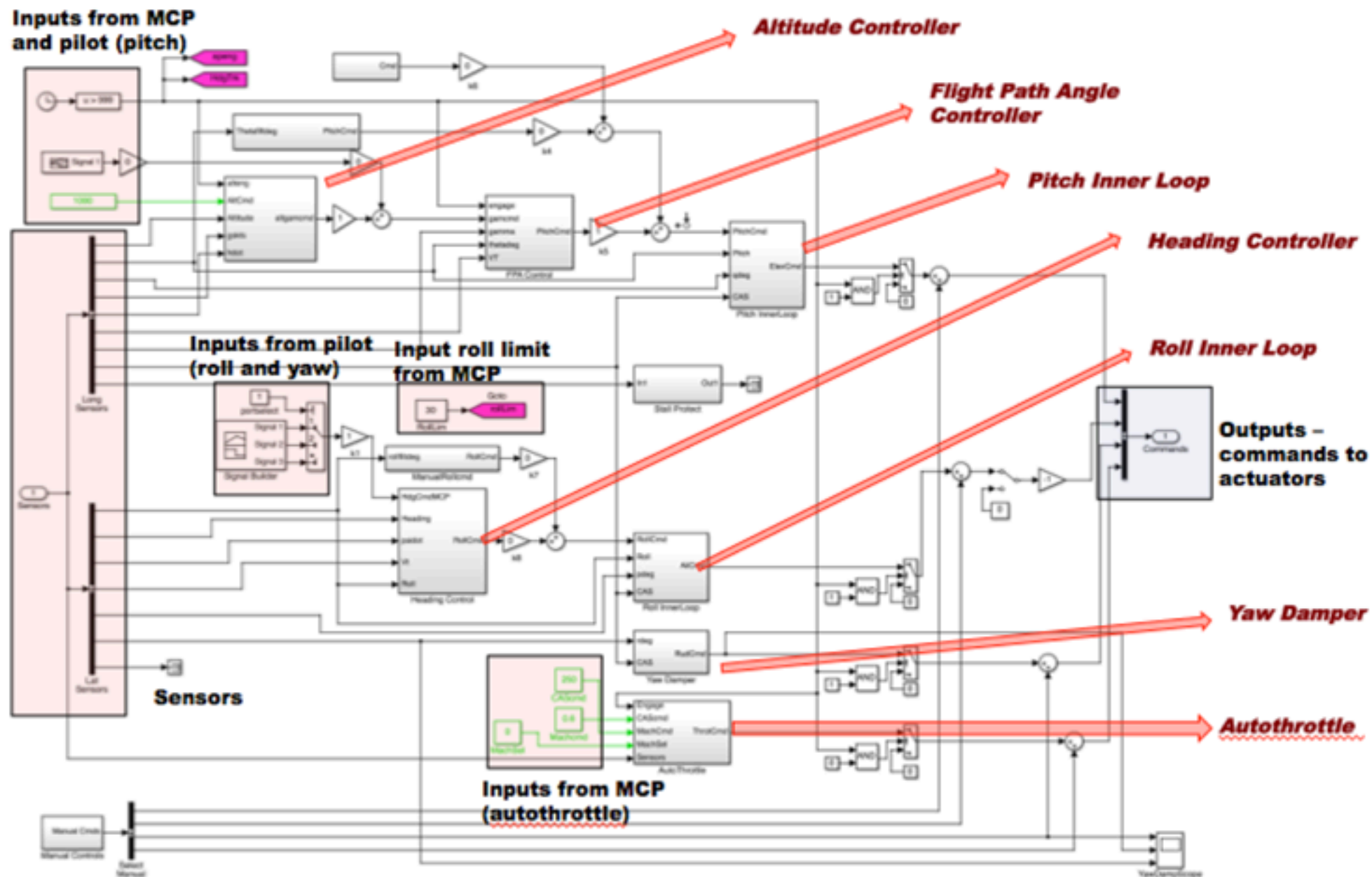
Intended as an experimental platform for controls and health management system

Simulink simulator for the **avionics** (transport delay), **actuators**, **engines**, **landing gear**, **aero**, **sensors** (including noise) ...

TCM Autopilot



TCM Autopilot

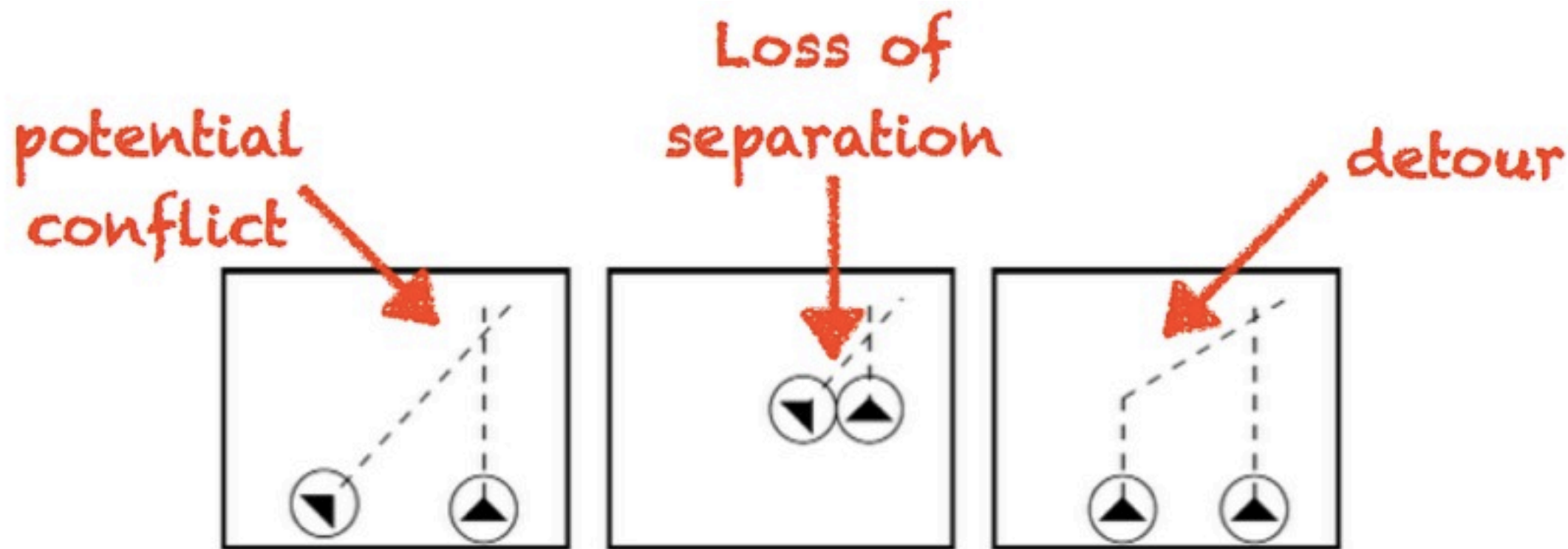


11

Kahsai et. al. "Verifying the safety of a flight critical software". FM'15.

- Safety verification via model checking
- Manual decomposition of 'hard' safety properties

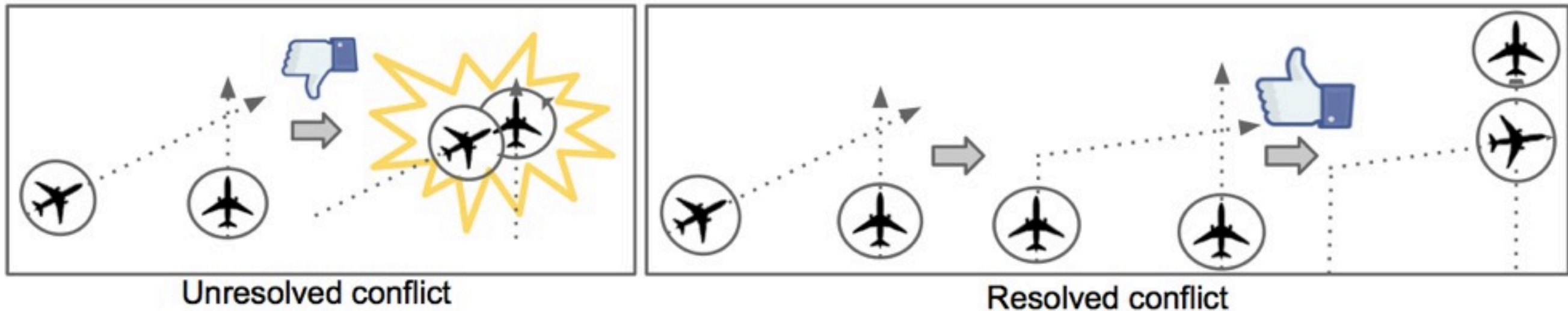
NextGen Air-Traffic Control



- **NextGen.** New national airspace system in the US.
- **Air-Traffic Control.** Separation assurance: resolution of potential future conflicts between aircrafts.
- **Loss of Separation.** Two airplanes come closer than a specified safe distance (horizontally or vertically)

—

NextGen Air-Traffic Control



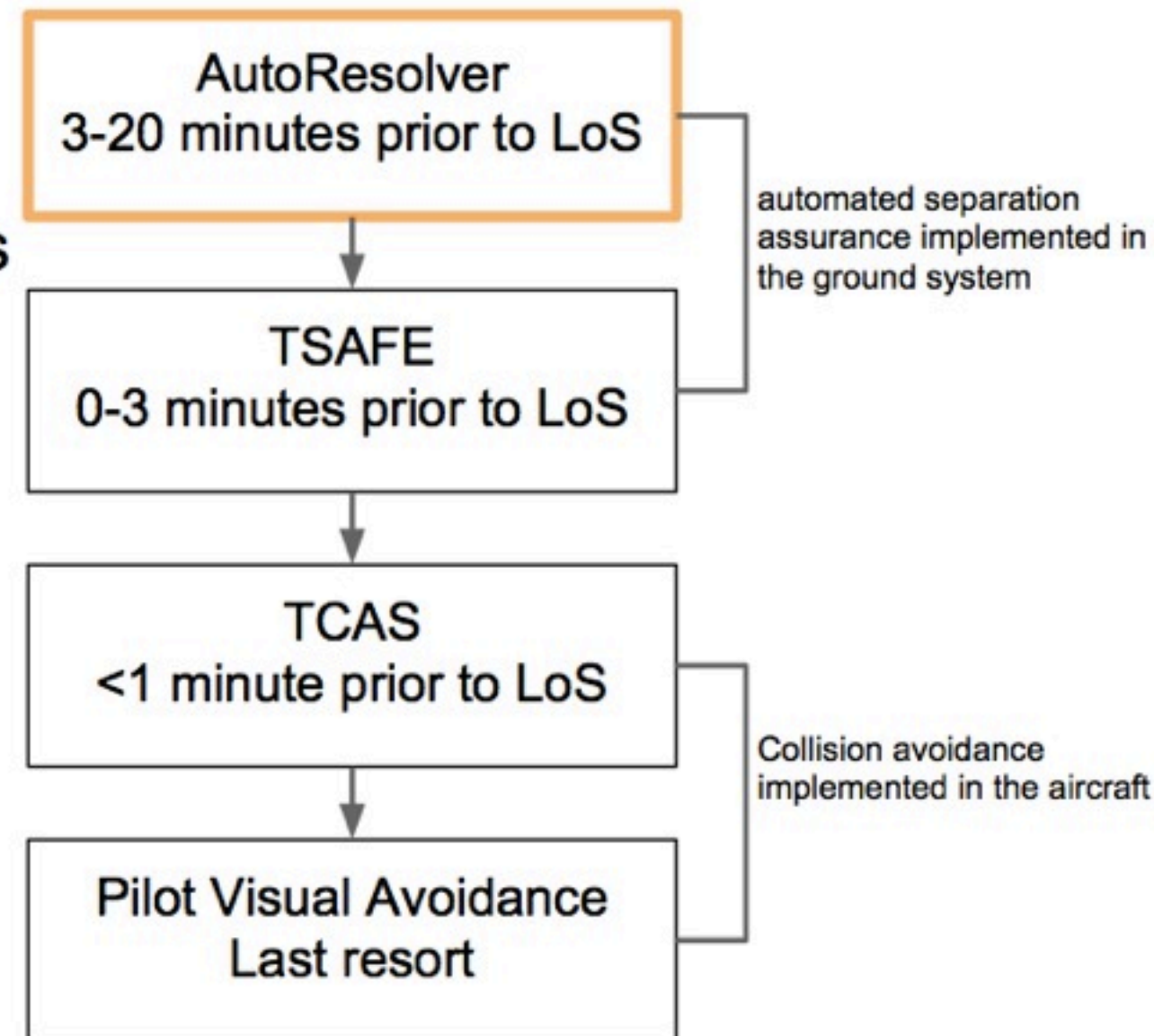
- Air-traffic control. Provides separation assurance by resolving potential future conflicts between aircraft
- Loss of separation. Airplanes come closer than a specified safe distance (horizontally and vertically)

NextGen Air-Traffic Control

NextGen component. 3-20 min time horizon

Java prototype developed at NASA Ames

- 2,500 classes, 150kloc (w/ ACES)
- 150 classes, 65kloc (w/o ACES)
- (+ NASA Worldwind, etc.)



Summary

This talk

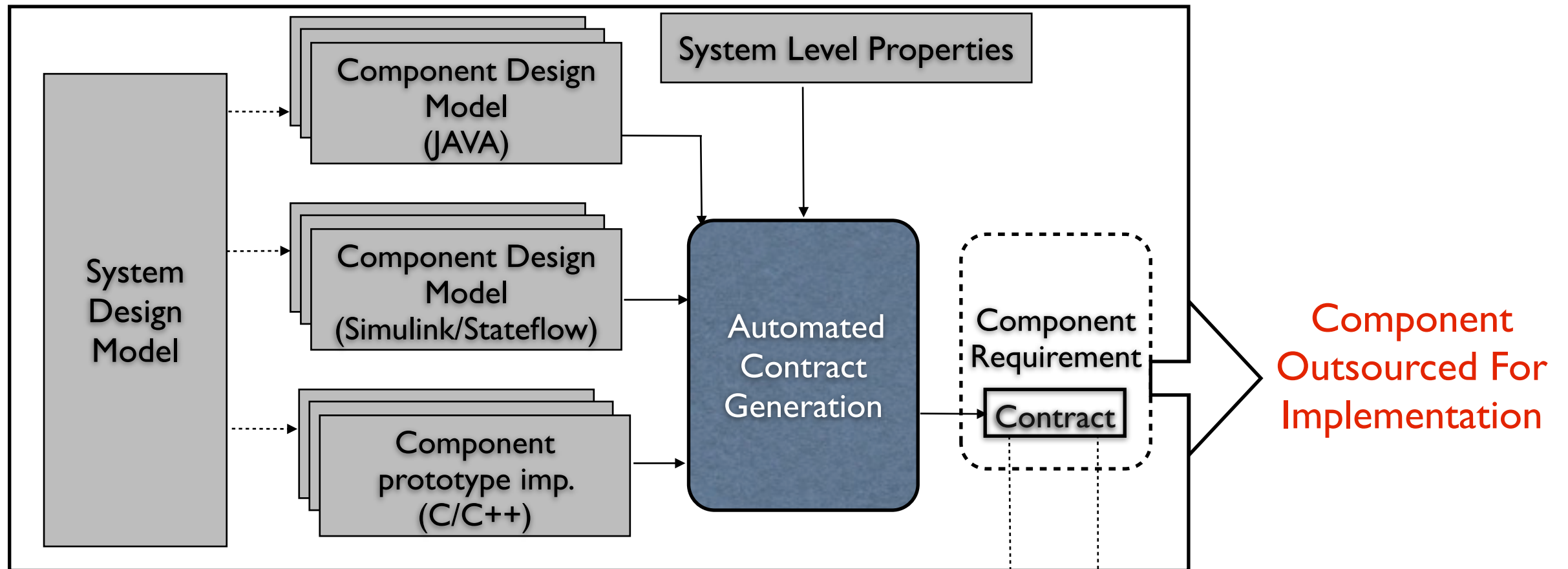


.... outsourcing in flight critical software

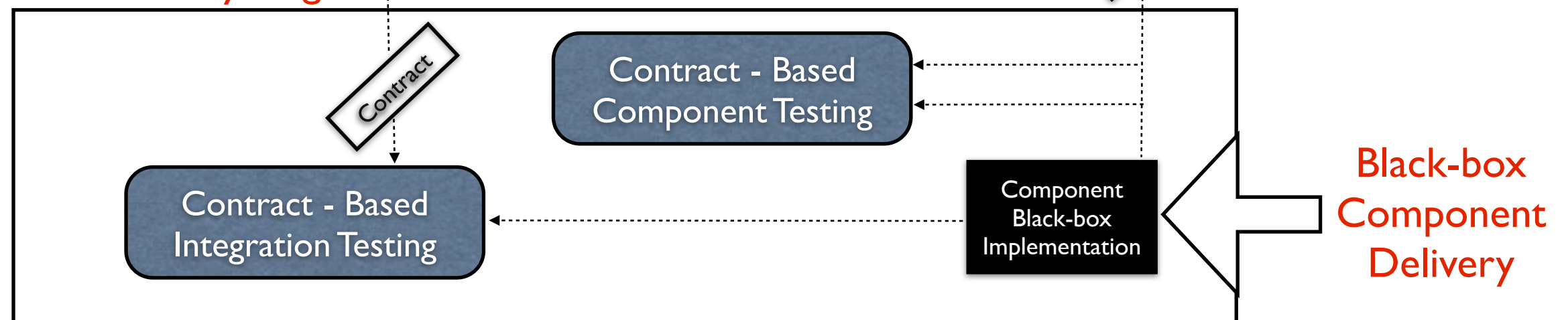
.... virtual integration of outsourced components

Two Stage solution for virtual integration

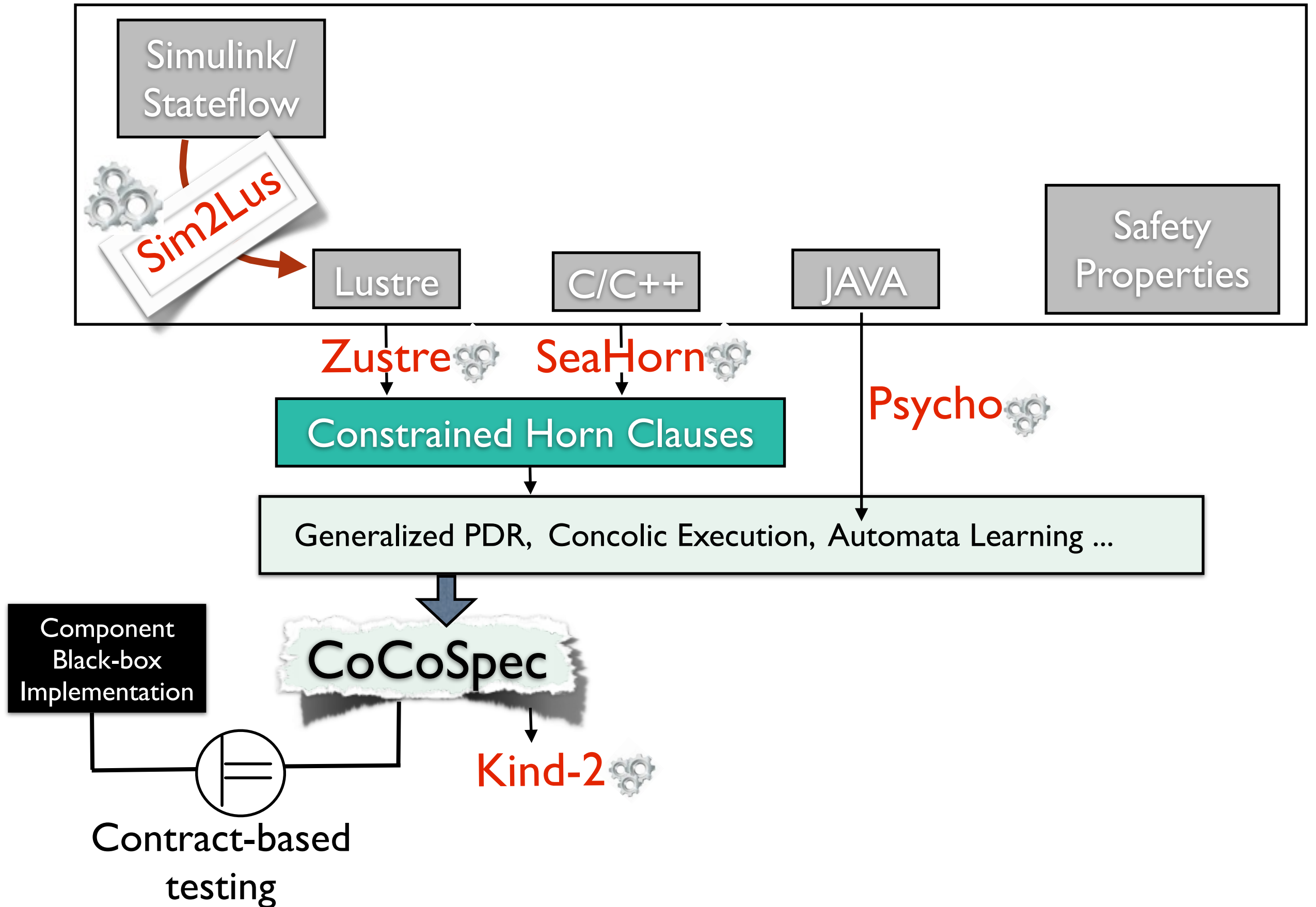
Pre-Delivery Stage



Post-Delivery stage



tools tools and tools



Thank you

Contact information

Temesghen Kahsai

Research Scientist @ RSE (Code TI) NASA Ames / CMU

email: temesghen.kahsaiazene@nasa.gov

web (work): <http://ti.arc.nasa.gov/profile/tkahsaia/>

web (personal): <http://www.lememta.info/>