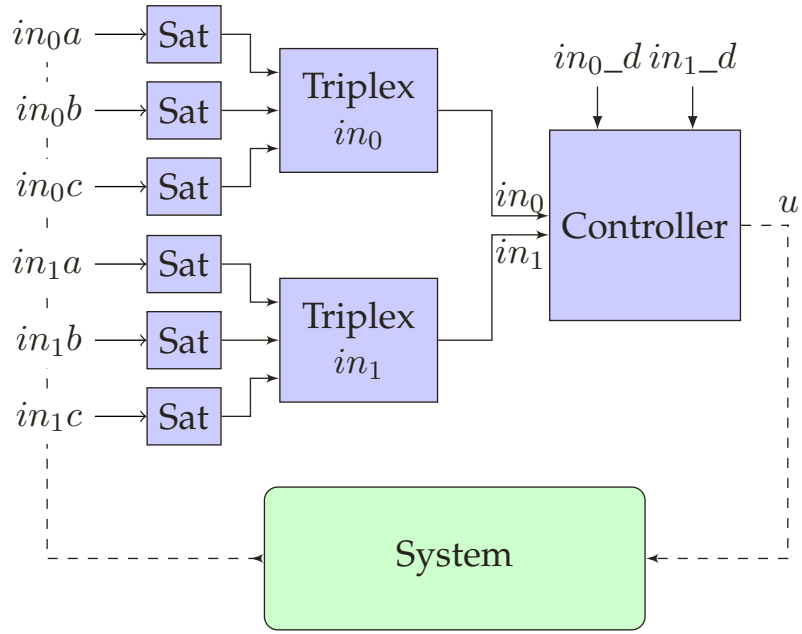
CocoSim: an integration hub for compilation and formal verification of Simulink/Stateflow

H. Bourbouh, G. Brat, P.-L. Garoche

ESA MBSE Workshop | September 28th-29th 2020 | Virtual

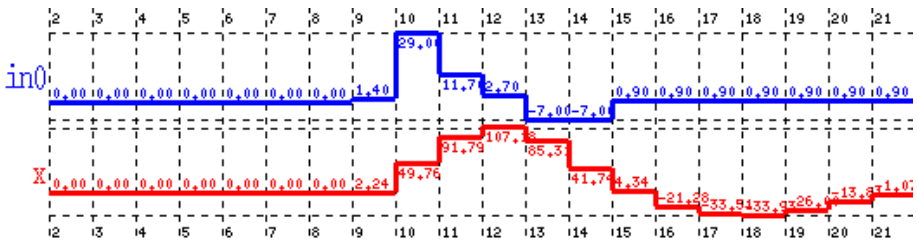
CONTEXT: CRITICAL EMBEDDED CONTROLLERS

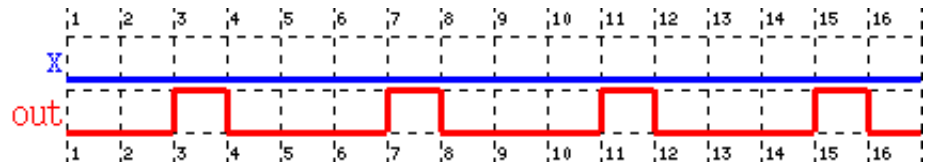
- Core elements of runtime systems
- Designed with dataflow models
  - validation through simulation/test
  - code generation
- Infinite behavior: endless loop



Designed by local composition:

- a linear controller
- combined with safety constructs





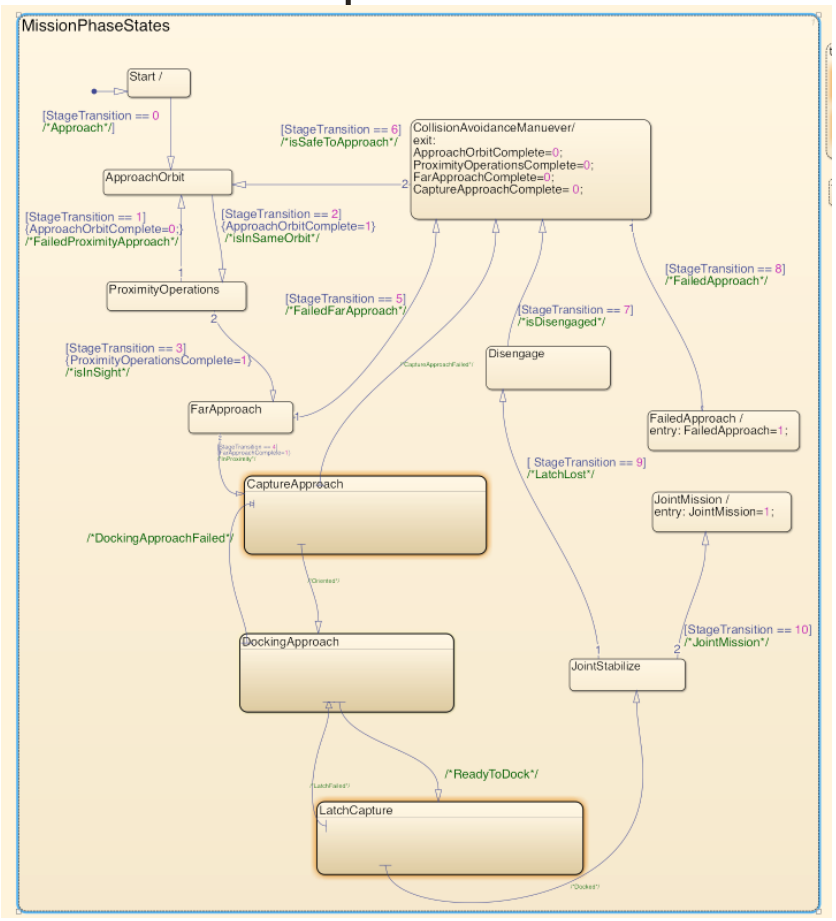
Most properties are analyzed locally.

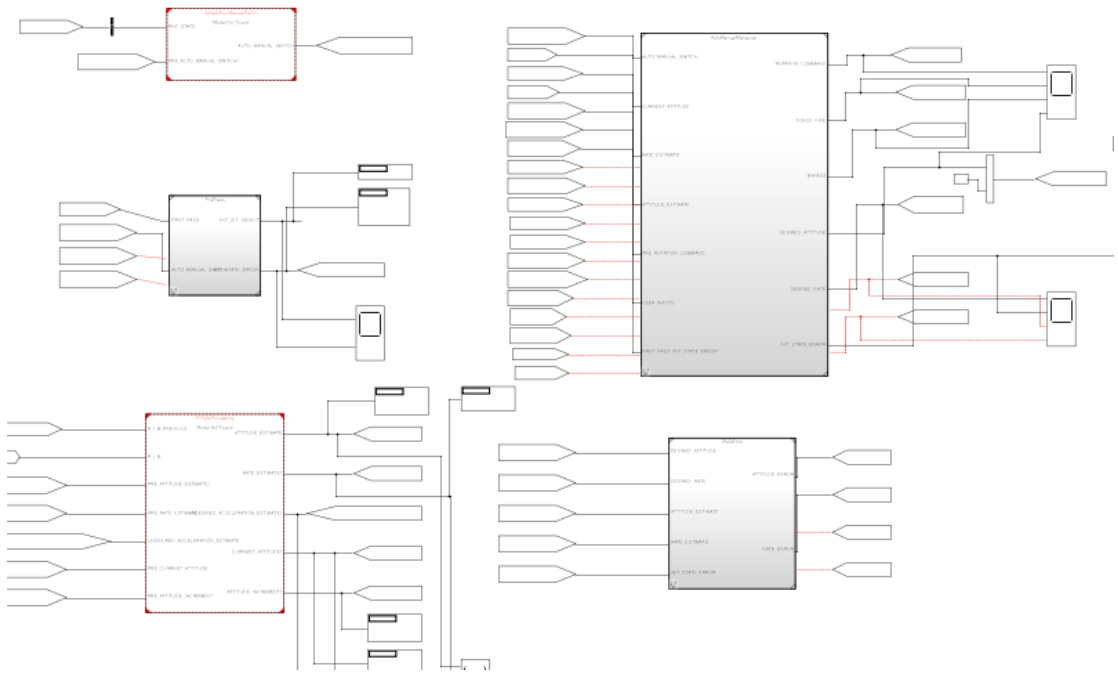
Requirements $\rightarrow$ Verification

Models and semantics – Which models to design, compile and verify critical control systems?

Different models for differents uses:

- mode changes, discrete transitions: state machines, automata
- time-sampled systems, eg. controllers, discrete dynamical systems: synchronous data flow languages (n-synchronous if need for complex or distributed systems)

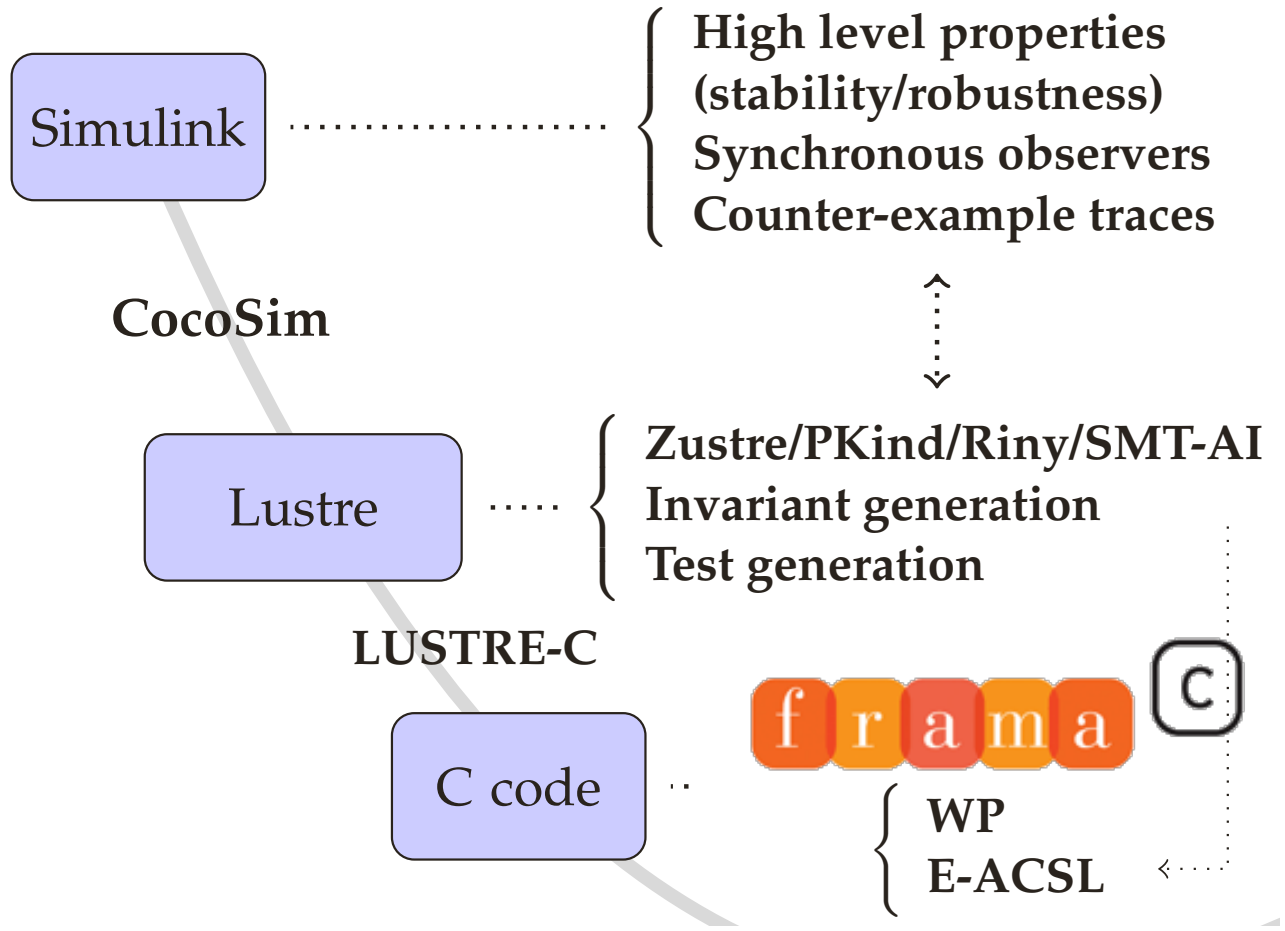




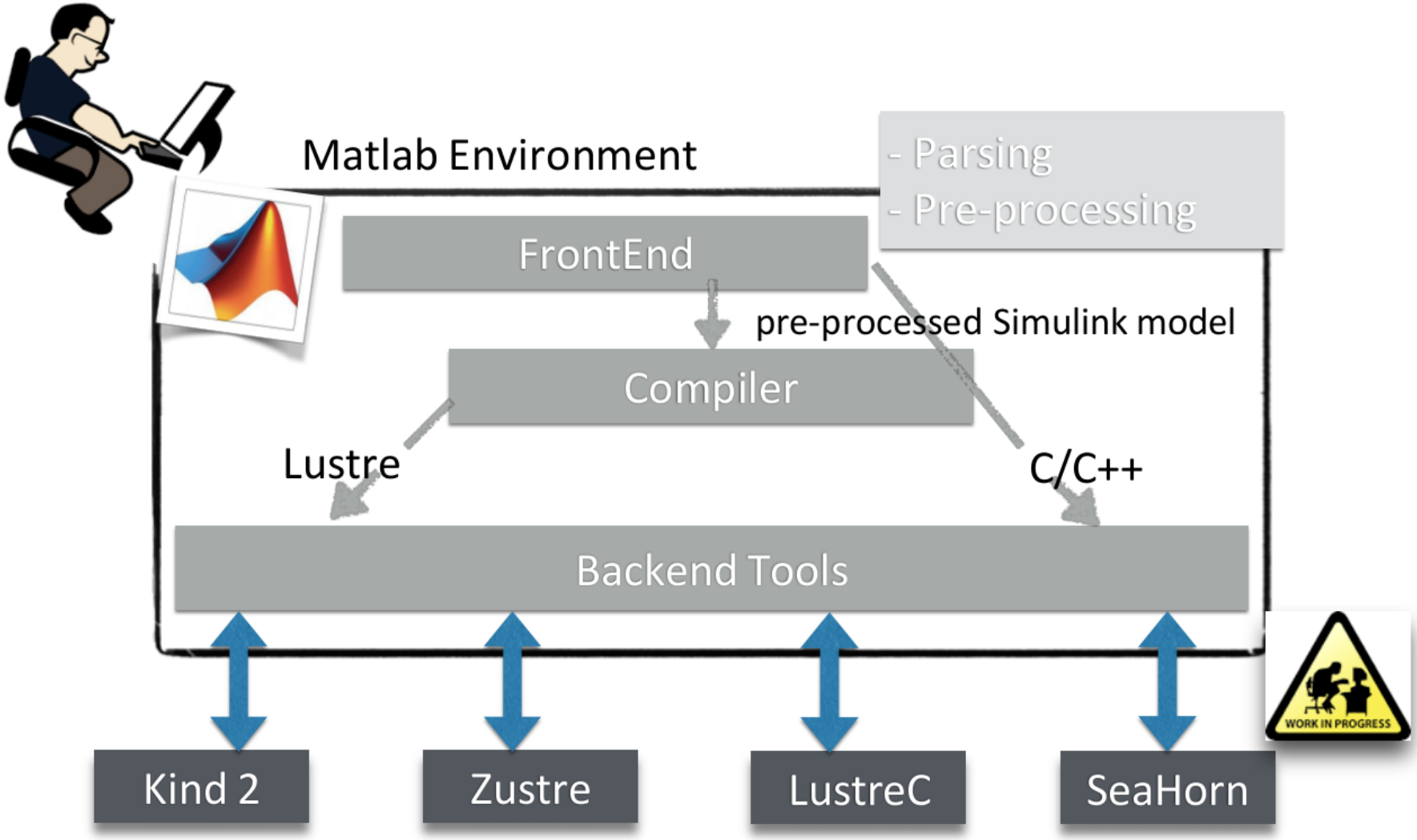
VISION: INTEGRATE FORMAL METHODS IN THE DEV. CYCLE

KEY ELEMENTS – OUR PHILOSOPHY : CODE GENERATION – SEMANTICS LAYER

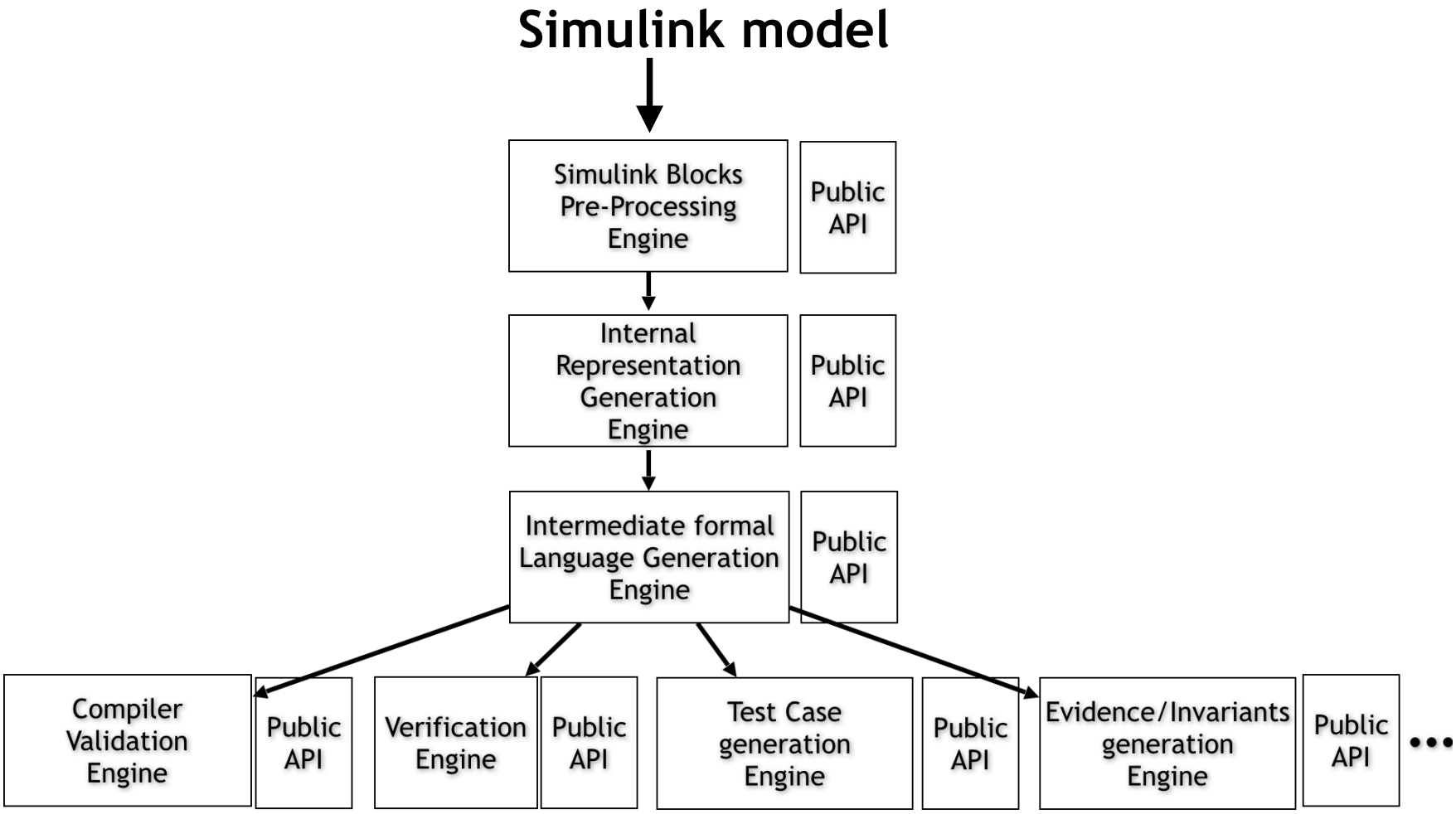
- Model is the source – Its semantics shall be preserved and lead to final code
- Requirements are formal and drive the analyses



CocoSim Architecture

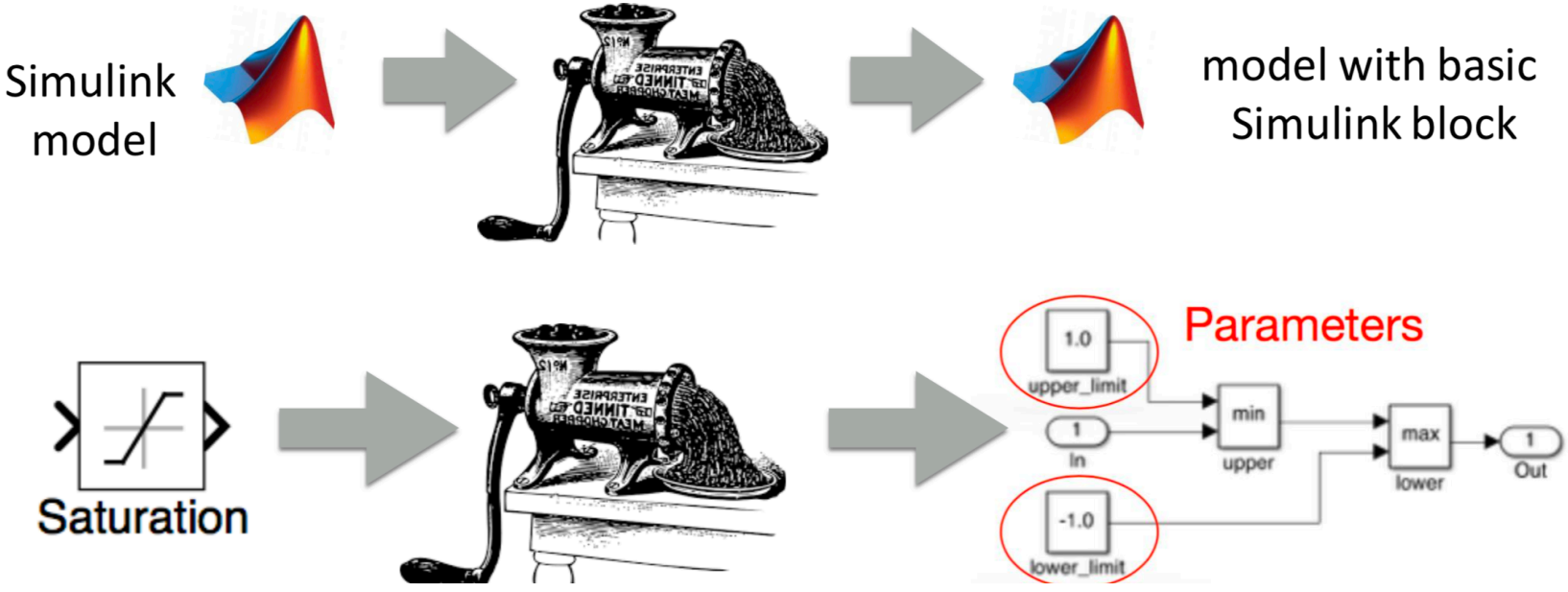


CocoSim 2 – An open system



FRONTEND: PREPROCESSING

Transform a Simulink model with 'fancy' blocks to one that uses basic Simulink block



$$\frac{z^2 + 5z + 6}{z^3 + 6z^2 + 8z + 1}$$

Discrete Transfer Fcn

$$X_n = AX_{n-1} + BU_{n-1}$$

$$Y_n = CX_n + DU_n$$

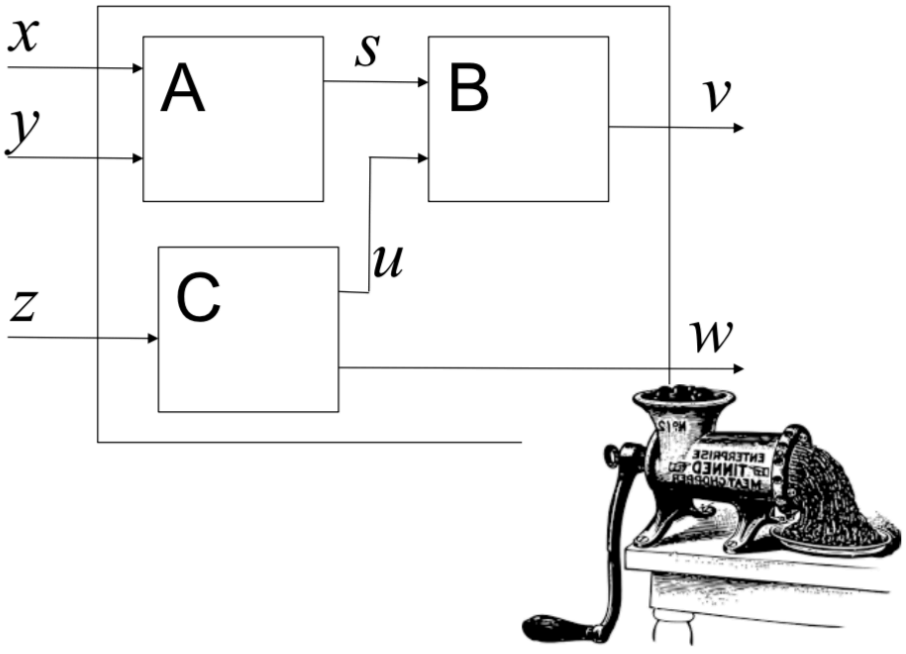
Discrete State Space Representation

Middle-end: compilation to Lustre

- **Translate discrete-time non-ambiguous part of Simulink (the controller).**

- **Goal:** preserve semantics of Simulink.

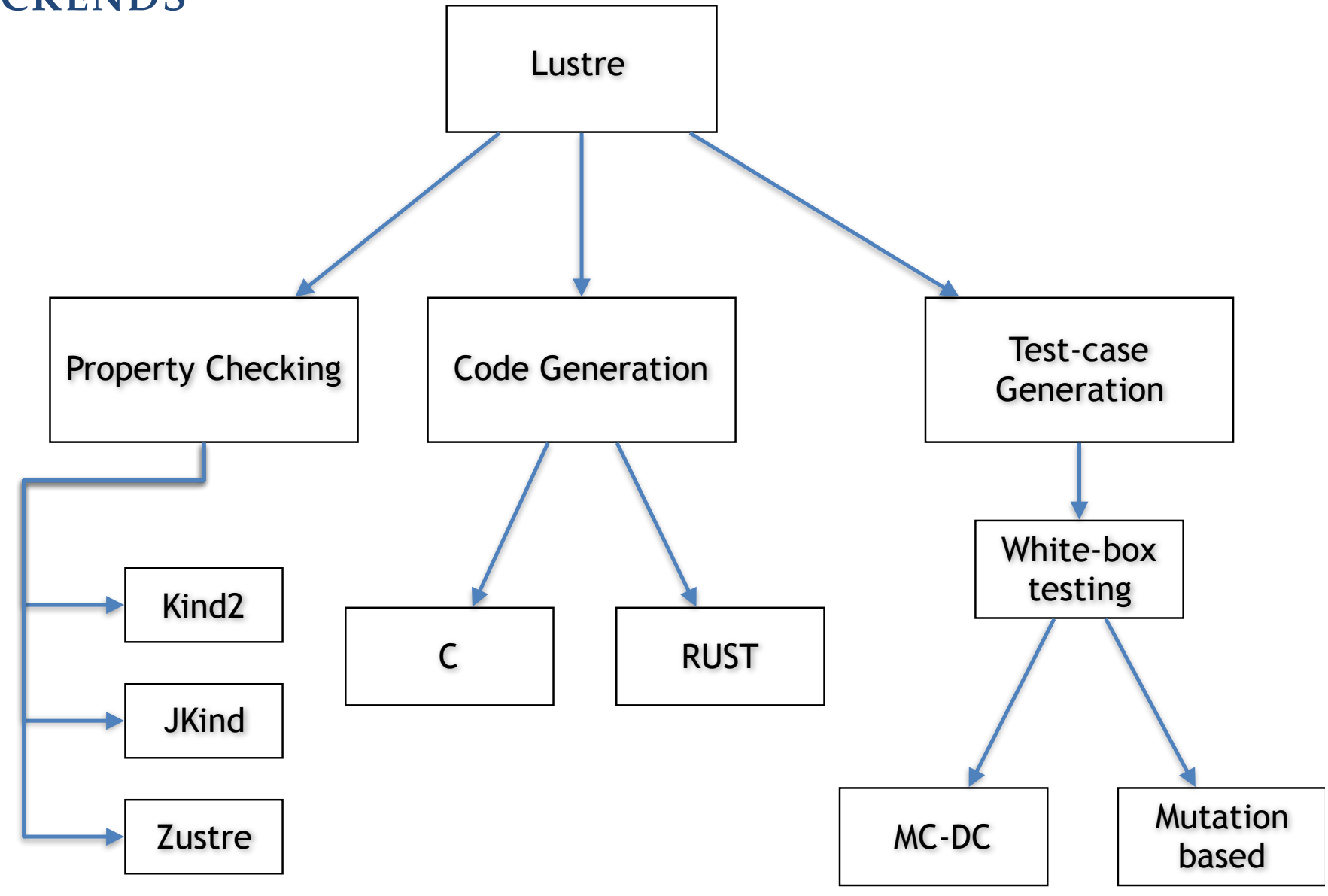
Simulink model



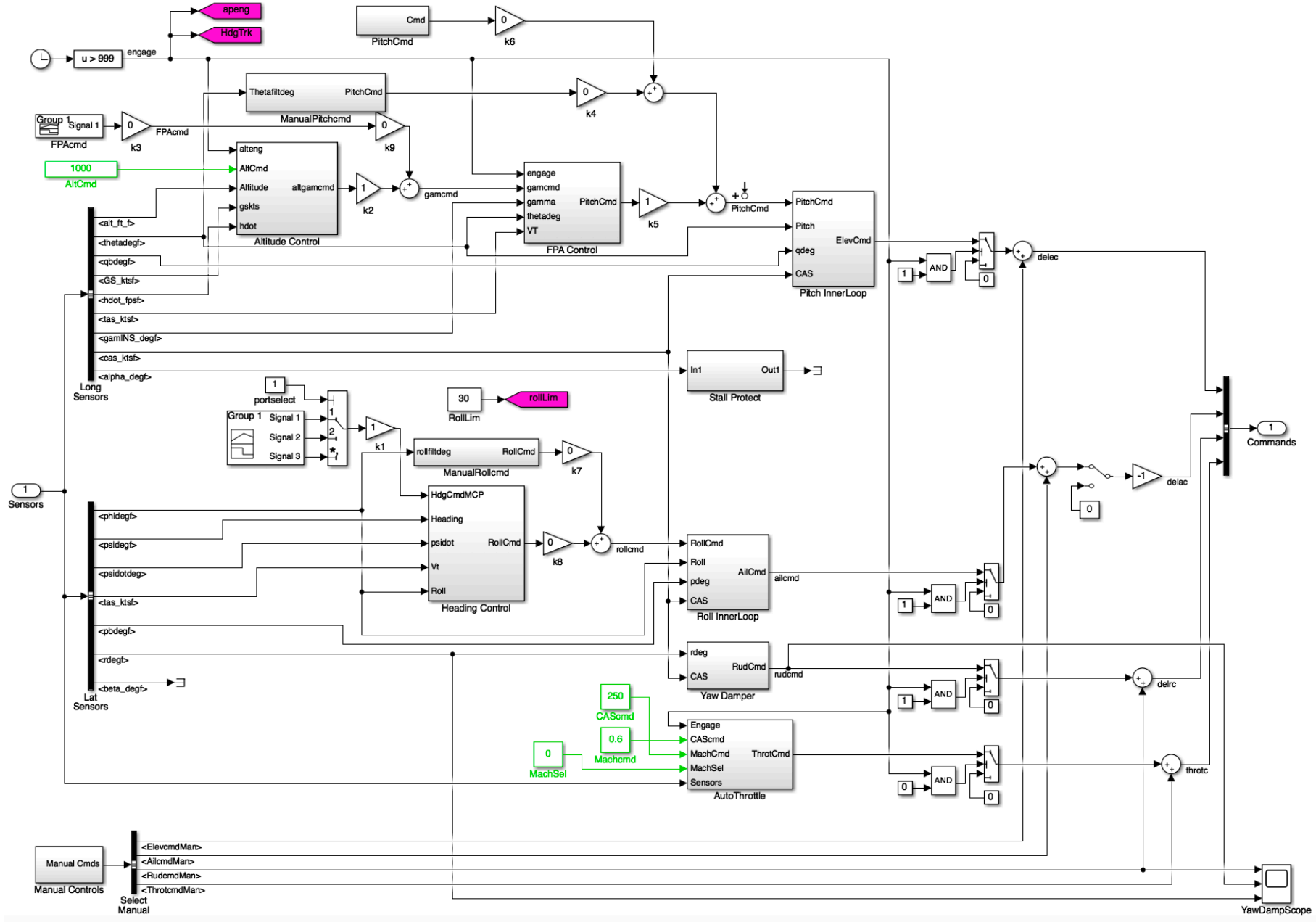
Lustre program

```
node A(x,y) returns(s);
 let ... tel

node B(s,u) returns(v);
 let ... tel

node C(z) returns(u,w);
 let ... tel

node Root(x,y,z) returns(v,w);
  var s, u;
 let
   s      = A(x,y);
   v      = B(s,u);
  (u,w)   = C(z);
 tel
```

BACKENDS


Lustre
Property Checking
Code Generation
Test-case Generation
Kind2
JKind
Zustre
C
RUST
White-box testing
MC-DC
Mutation based

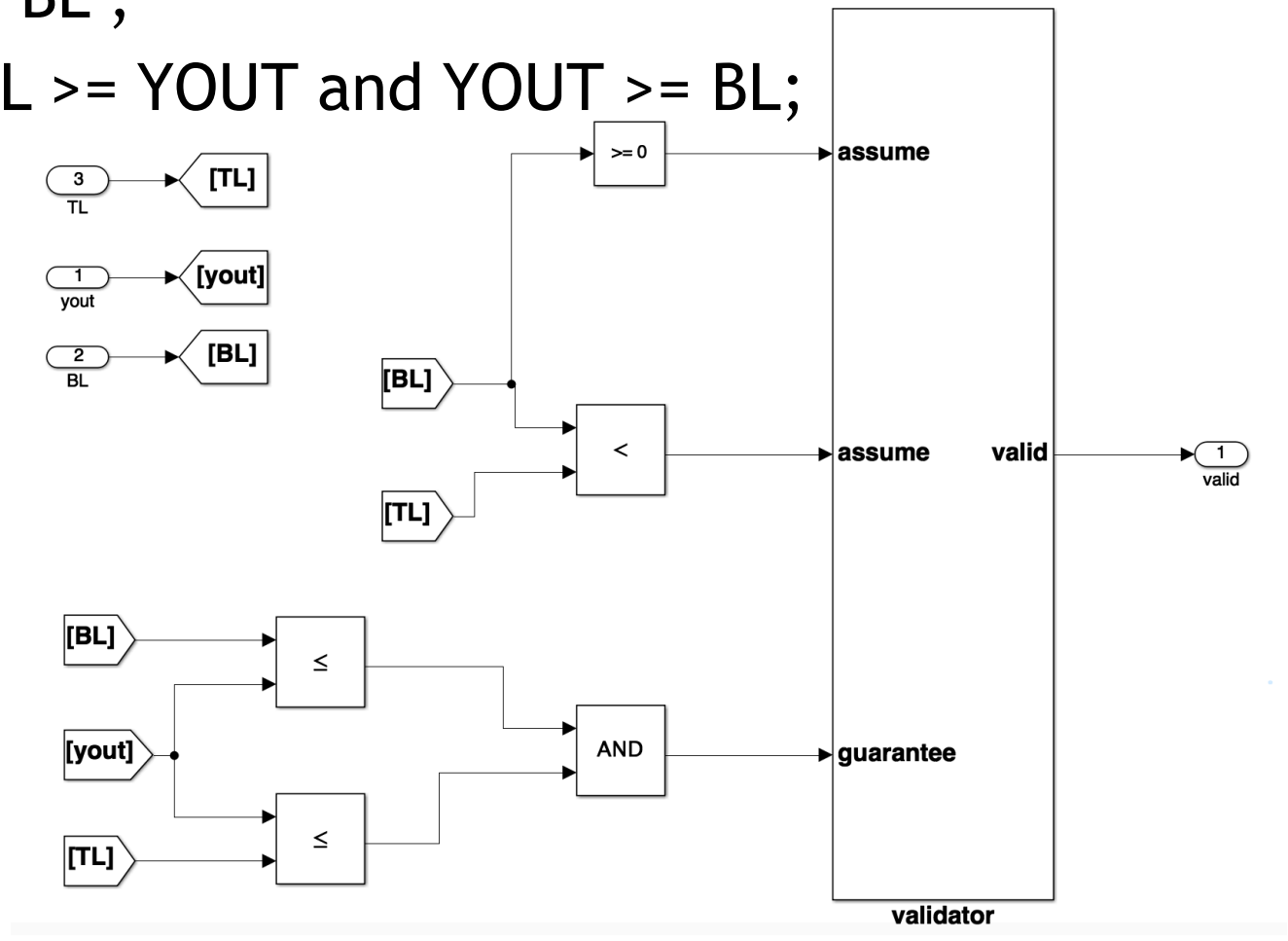
SIMULINK GUIDANCE AND CONTROLS SYSTEM FOR THE TCM



Requirements as synchronous observers 1/2

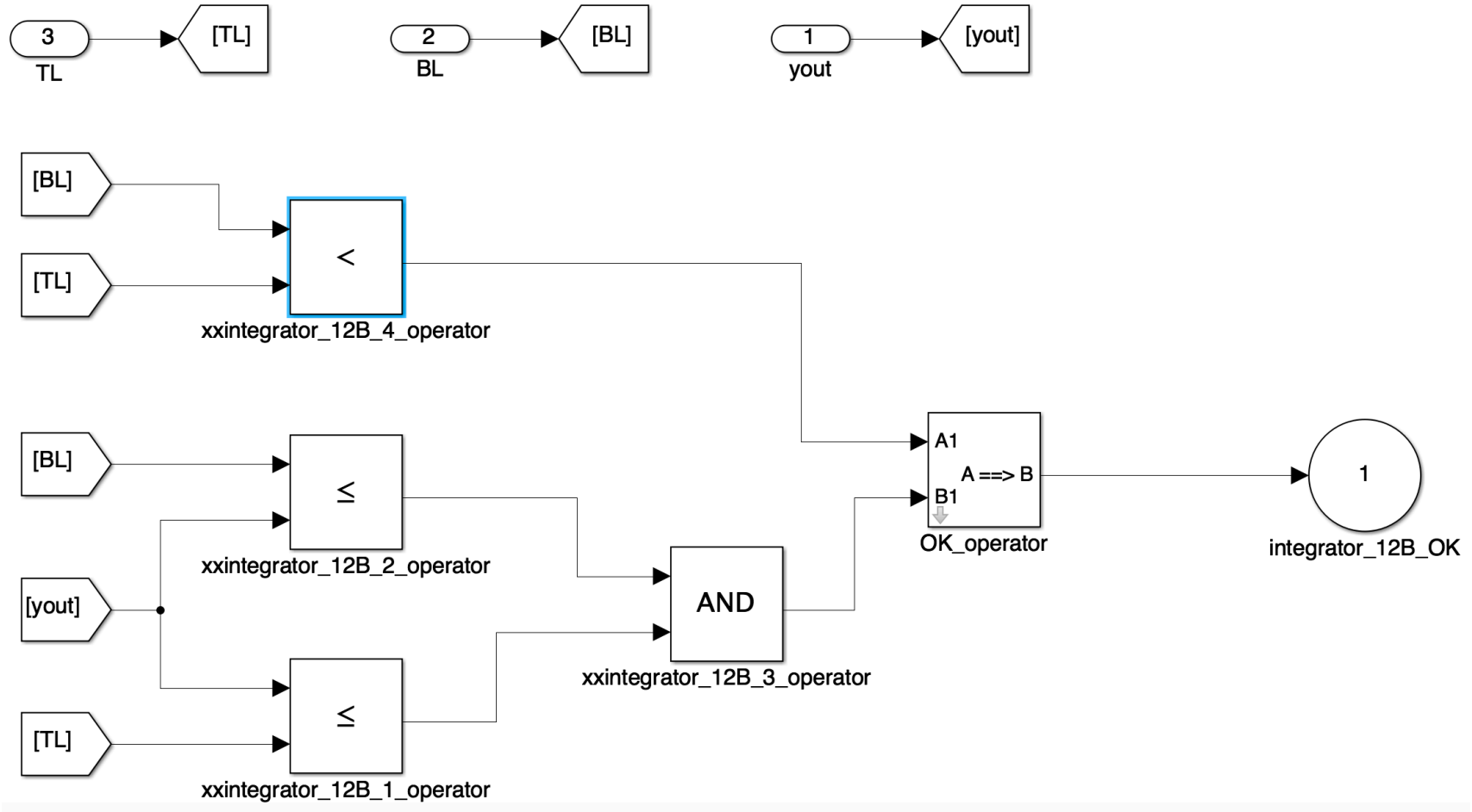
As a boolean predicate

- **assume** BL >= 0.0;
- **assume** TL > BL ;
- **guarantee** TL >= YOUT and YOUT >= BL;



REQUIREMENTS AS SYNCHRONOUS OBSERVERS 2/2

ASSUME-GUARANTEE CONTRACTS



(BL < TL) => (BL <= YOUT <= TL)

Features / Available tools

- Code generation
  - ADA, C, Rust
- Model-checking
  - Zustre (LustreC + Z3/Spacer): PDR
  - Kind2: PDR + k-induction
- Test generation
  - enumerating coverage condition and searching for models
- Requirements elicitation with FRET
- Static Analysis (not integrated yet)
  - computation of non linear invariants
    - ellipsoids, semi-algebraic sets
  - templates / support functions / policy iterations
- Semantic preservation – Proof revalidation at code level (ongoing)
- Numerical accuracy (ongoing)
  - Floating point analyses (zonotopes)
  - Floating point optimization
- Simulation / Hybrid system analyses (ongoing)

Conclusion

- Simulink is used and widespread
  $\rightarrow$ a good target to support MBSE VV
- Model semantics shall be precise enough to be executable
- CocoSim provides
  * expression of requirements as model components
  * test generation
  * State-of-the-art formal methods
- Future work: extensions to hybrid systems

https://github.com/NASA-SW-VnV/CoCoSim

Conclusion

- Simulink is used and widespread
  $\rightarrow$ a good target to support MBSE VV
- Model semantics shall be precise enough to be executable
- CocoSim provides
  * expression of requirements as model components
  * test generation
  * State-of-the-art formal methods
- Future work: extensions to hybrid systems

https://github.com/NASA-SW-VnV/CoCoSim

Thank you! Questions ?