Formal Modeling and Assurance of Cyber-Physical Systems

Sam Procter, Gabriel Moreno, Jerome Hugues
Presenter: Dionisio de Niz

Assuring Cyber-Physical Systems

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM23-0046

Assuring Cyber-Physical Systems

Multiple Formalisms

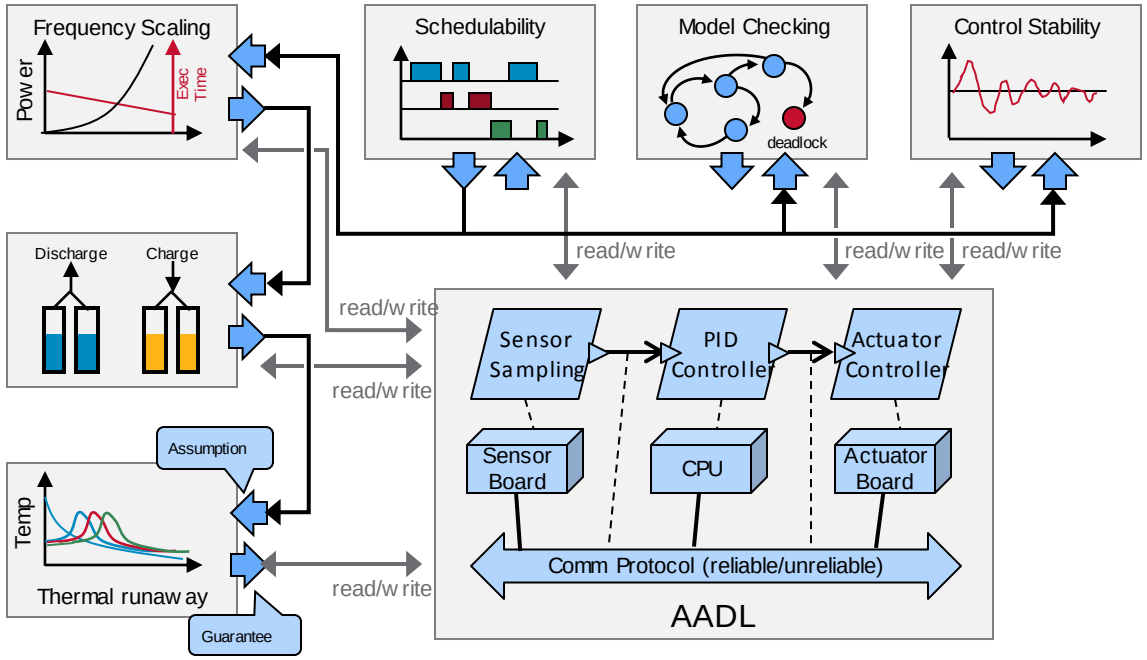
- Different Abstractions
  - Ignore different things

Multiple Analyses

- Optimized algorithms
- Different communities

Multiple Assurance Claims

- For complete assurance



CPS Analyses

Faster Assurance

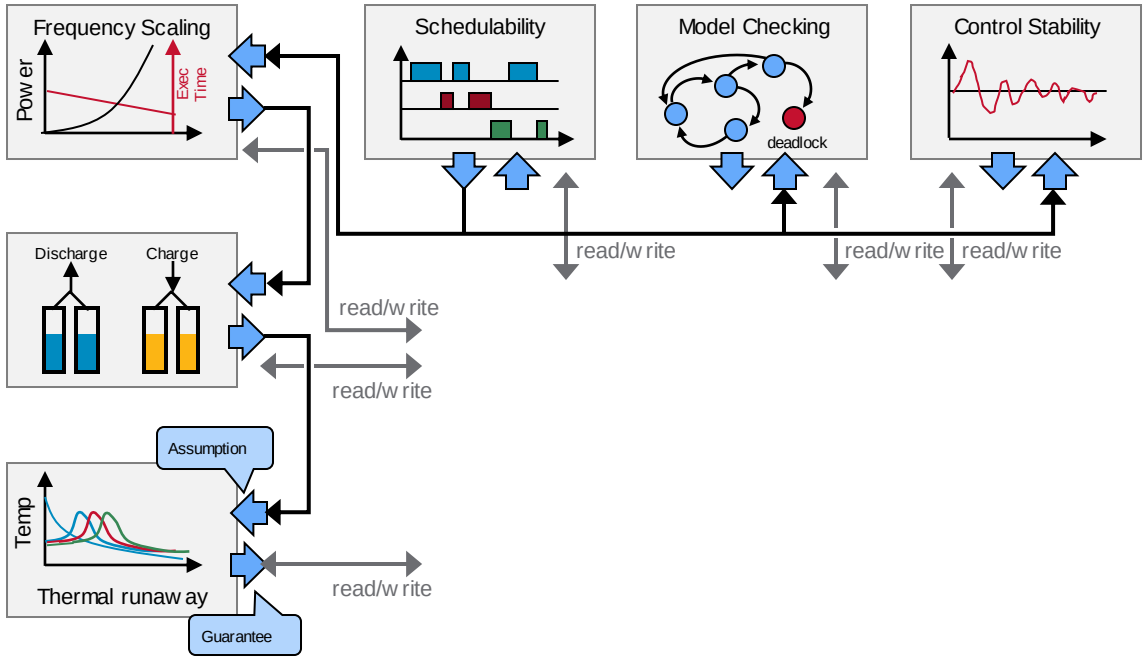
- Minimize verification
- Preserve soundness

Across Multiple Domains

- Timing
- Control
- Security

Mechanisms + Analysis

- Prevent unsafe unverified behavior
- Verify critical behavior



Enforcement-Based Verification Methods

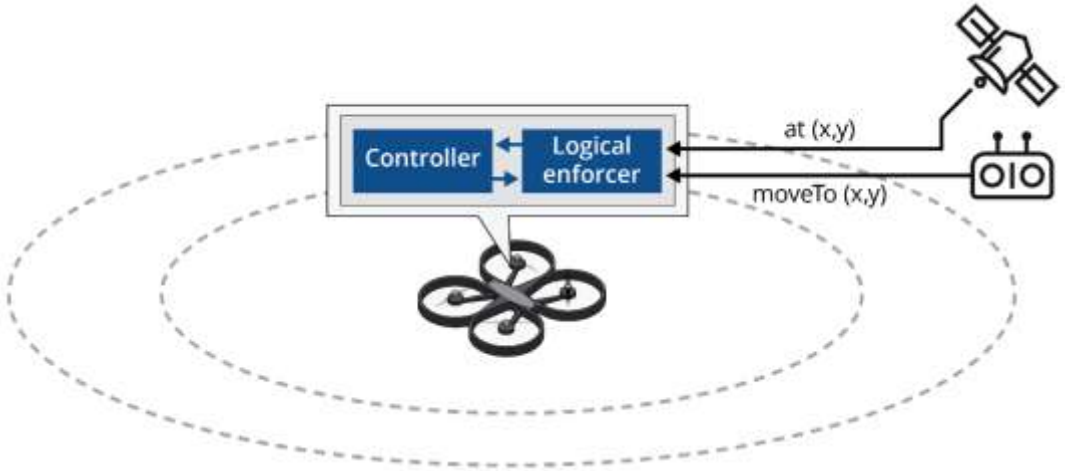
Most code unverified

Add safety enforcer

- Detects unsafe outputs
- Replace them for save ones

Verify enforcer

- Logic: safe values
- Timing: at right time
- Physics: safe physical effect



Verifying Physics

Recoverable Set: $\varepsilon_{SCj}(1)$

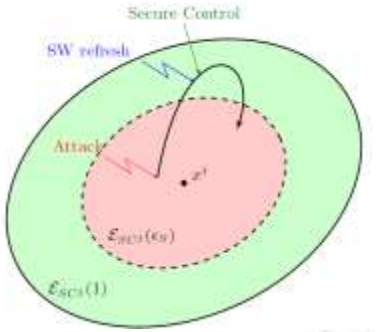
Safety Set: $\varepsilon_{SCj}(\epsilon_s) \triangleq \epsilon_s\, \varepsilon_{SCj}(1)$

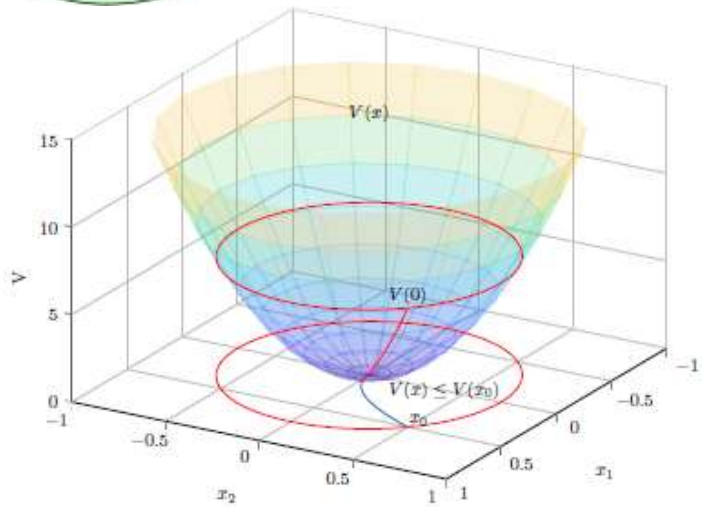
Controlled System: $\dot{x} = f_\varphi(x) \triangleq f(x, \varphi(x))$
Lyapunov Function: $V_\varphi : \mathbb{R}^n \to \mathbb{R}$, $\mathcal{N}_{V_\varphi}(x_{eq}) \subseteq \mathcal{N}_\varphi(x_{eq})$,
$V_\phi(x_{eq}) = 0$ and $\forall x \in \mathcal{N}_{V_\varphi}(x_{eq}) - \{x_{eq}\}$: (i) $V_\varphi(x) > 0$,

$$\dot{V}_\varphi(x) = \frac{\partial V}{\partial x} \cdot f_\varphi(x) < 0$$

Lyapunov level set:For $\epsilon > 0$,

$$\mathcal{E}_\varphi(\epsilon) = \{x \in \mathcal{N}_{V_\varphi}(x_{eq}) | V_\varphi(x) \leq \epsilon\}. \qquad \epsilon \leq 1$$





R. Romagnoli, B. H. Krogh, B. Sinopoli. Design of Software Rejuvenation for CPS Security Using Invariant Sets. ACC 2019

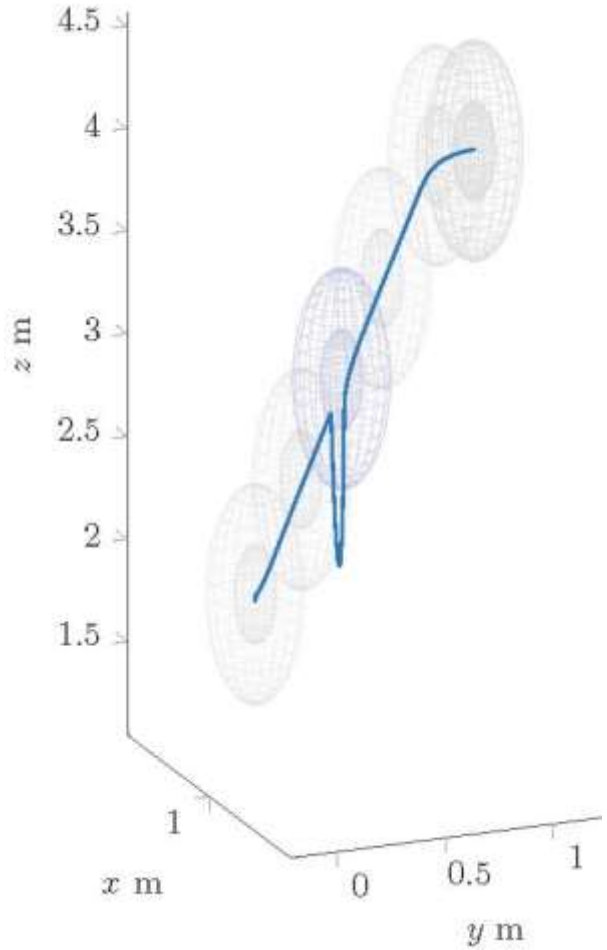
Analysis of Mission Progress Enforcing Unsafe Behavior

6 DOF $\Rightarrow$ 12 state variables

$$\ddot{p}_x = -\cos\phi \sin\theta \frac{F}{m}$$
$$\ddot{p}_y = \sin\phi \frac{F}{m}$$
$$\ddot{p}_z = g - \cos\phi\cos\theta \frac{F}{m}$$
$$\ddot{\phi} = \frac{1}{J_x}\tau_\phi$$
$$\ddot{\theta} = \frac{1}{J_y}\tau_\theta$$
$$\ddot{\psi} = \frac{1}{J_z}\tau_\psi.$$

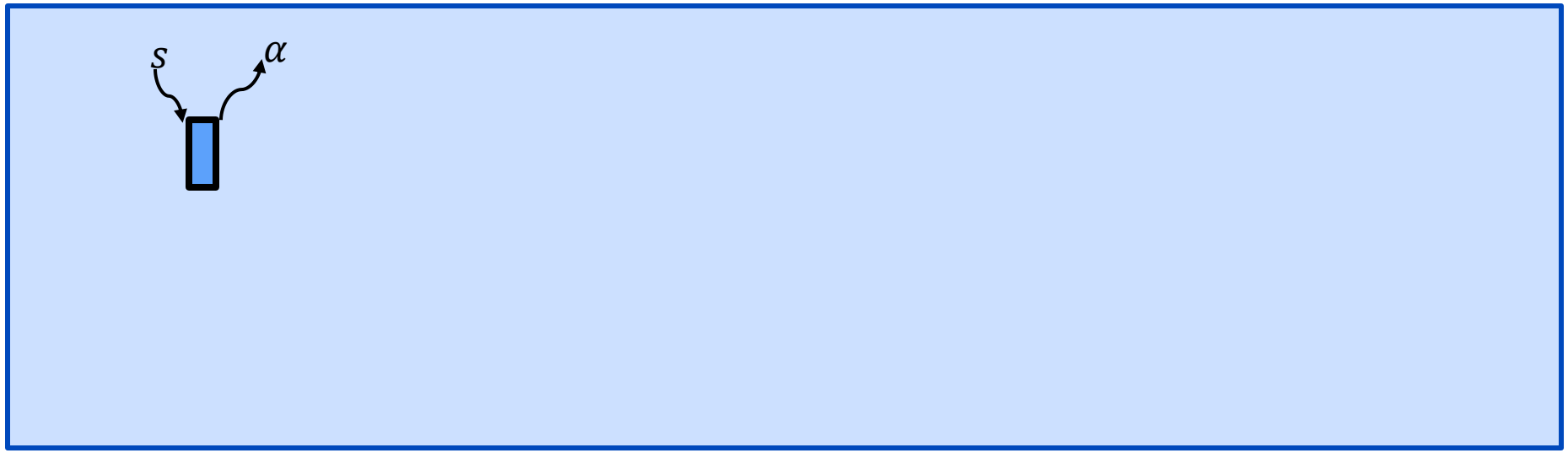
Linear design:

- linearize at equilibrium
- assume full state available
- LQ state feedback design
- reference points = equilibrium states

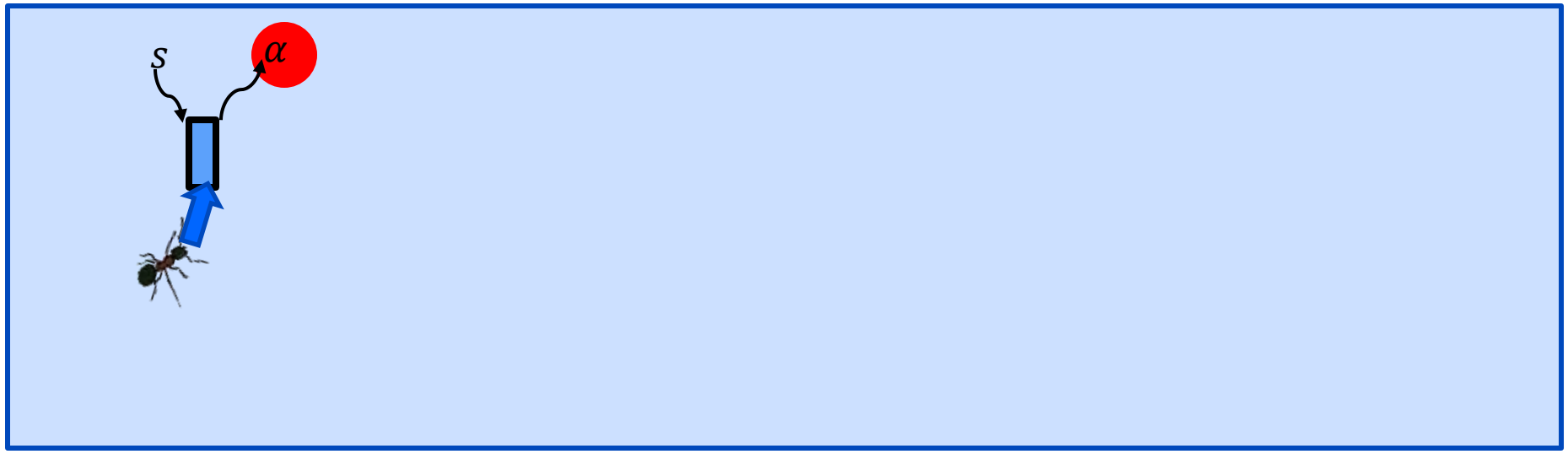


Drone Experiment

Enforcing Unverified Components

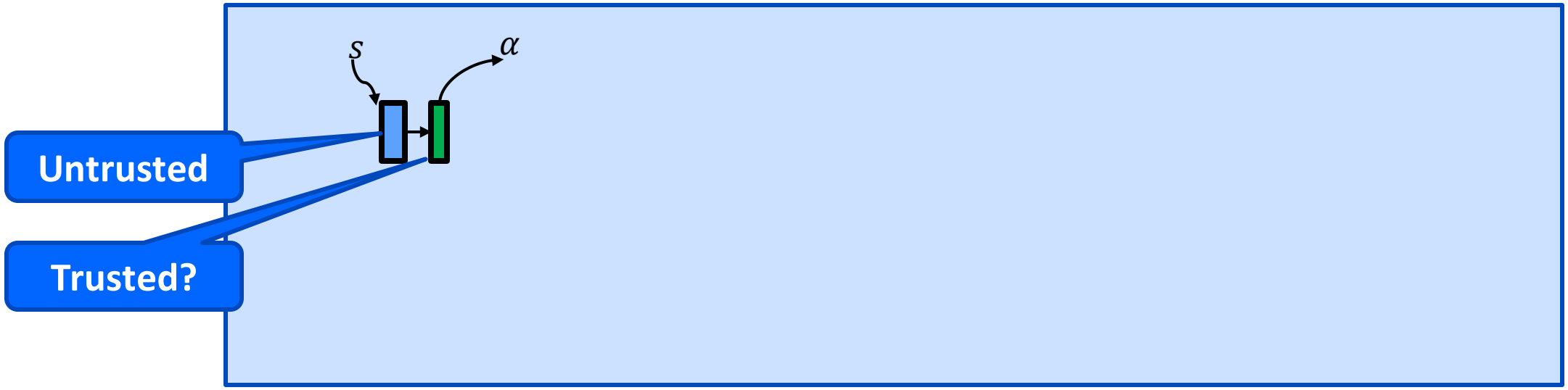


Enforcing Unverified Components

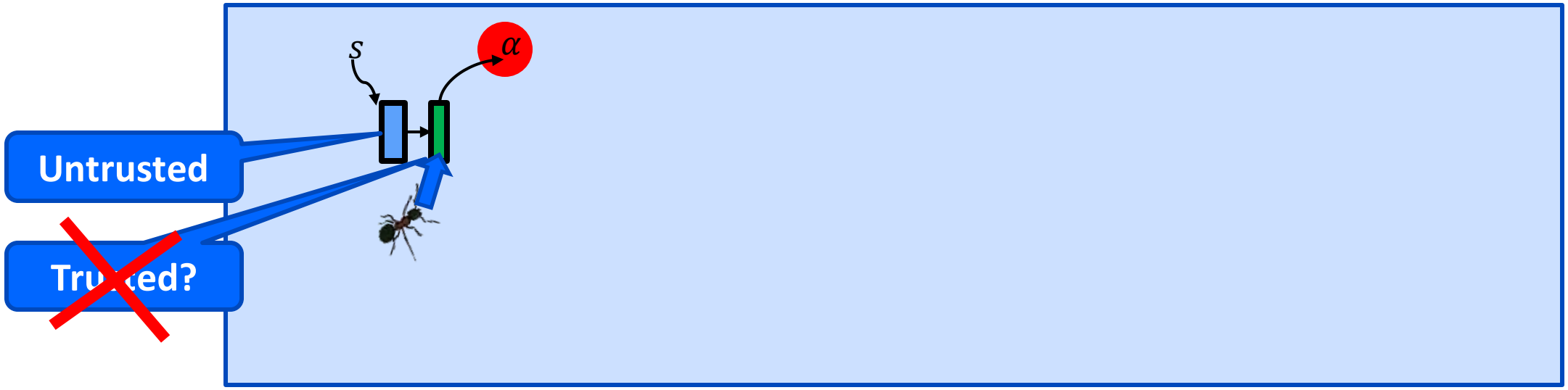


Ant illustration by Jan Gillbank, license by Creative Commons Attribution 3.0 Unported

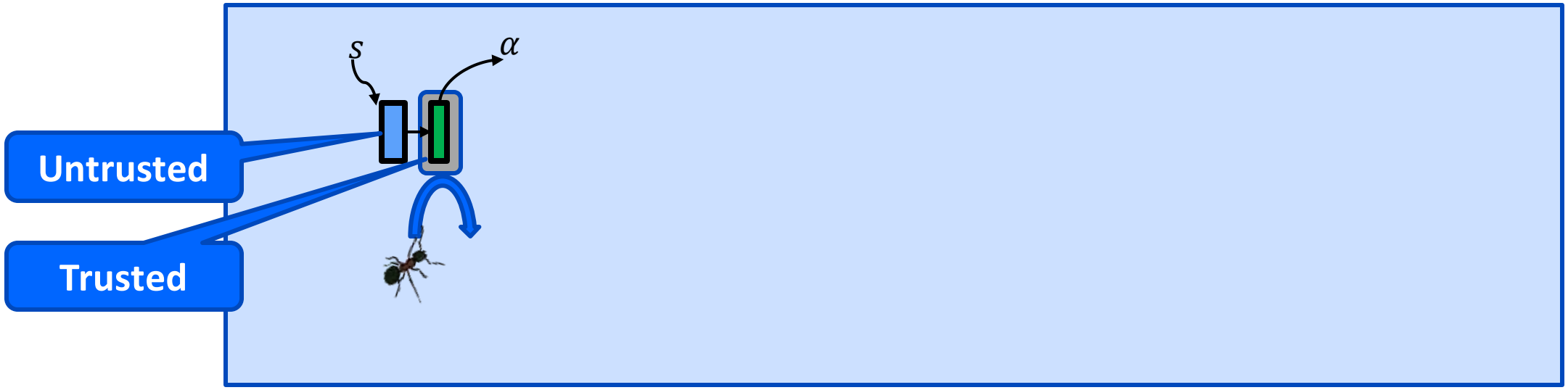
Enforcing Unverified Components



But enforcer can be corrupted (bug or cyber attack)

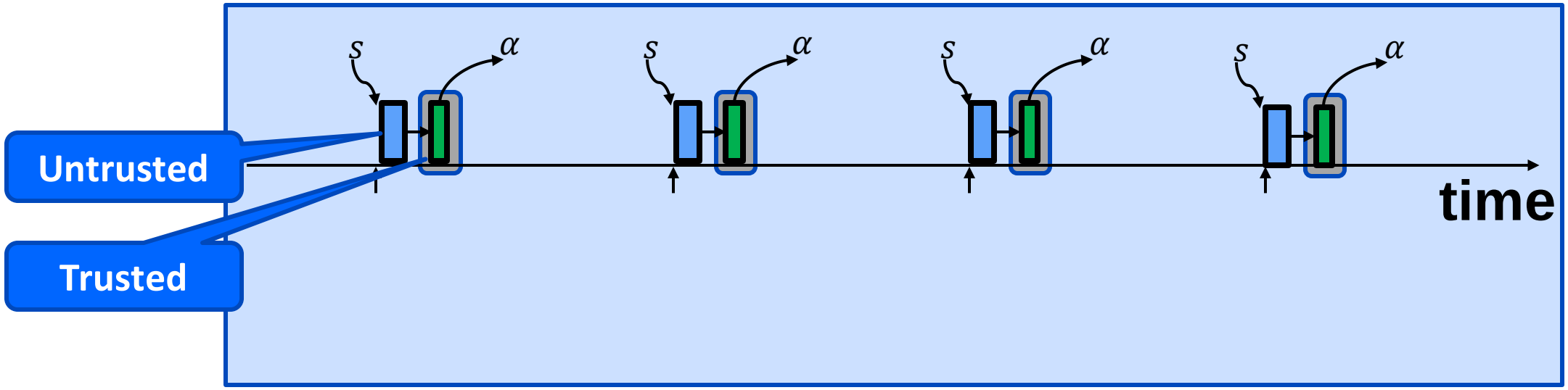


Add Memory Protection

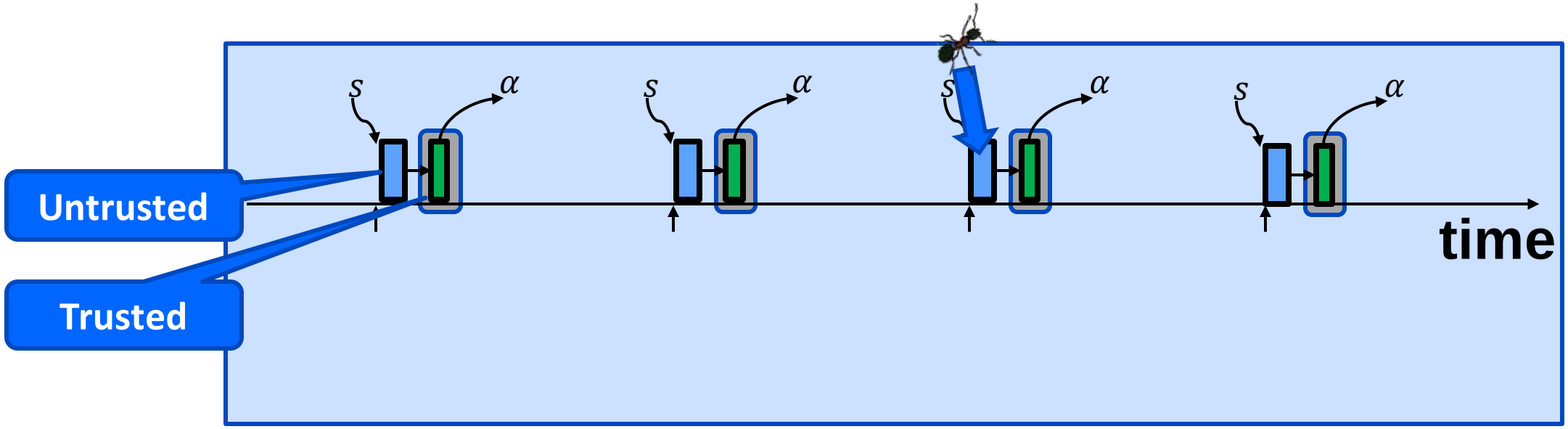


Trusted = Verified & Protected

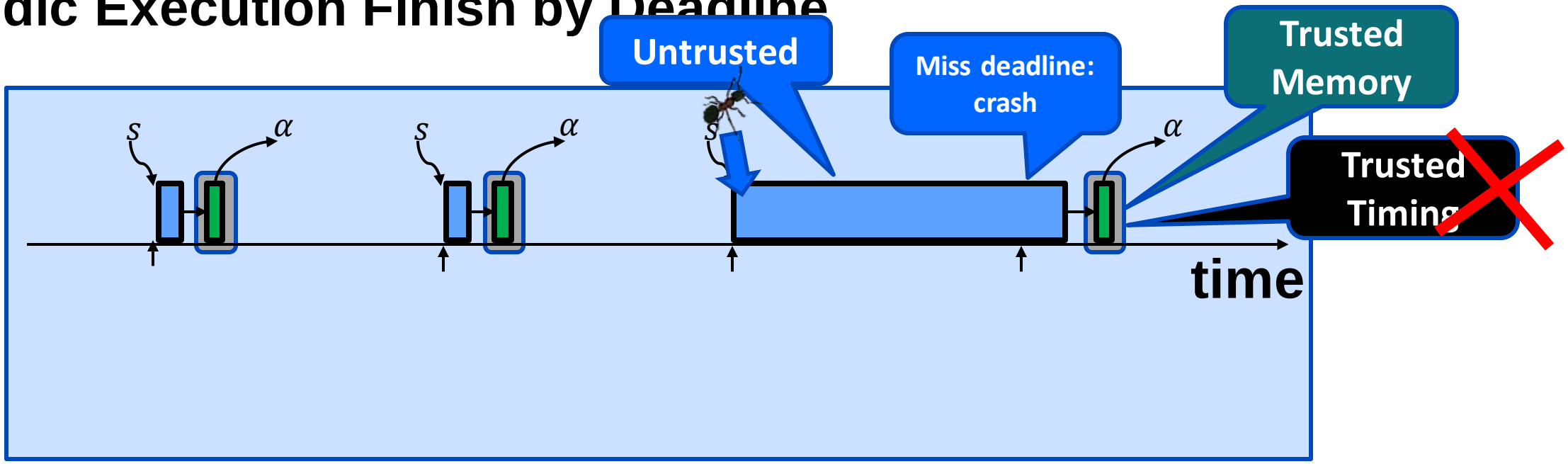
Periodic Execution Must Finish by Deadline



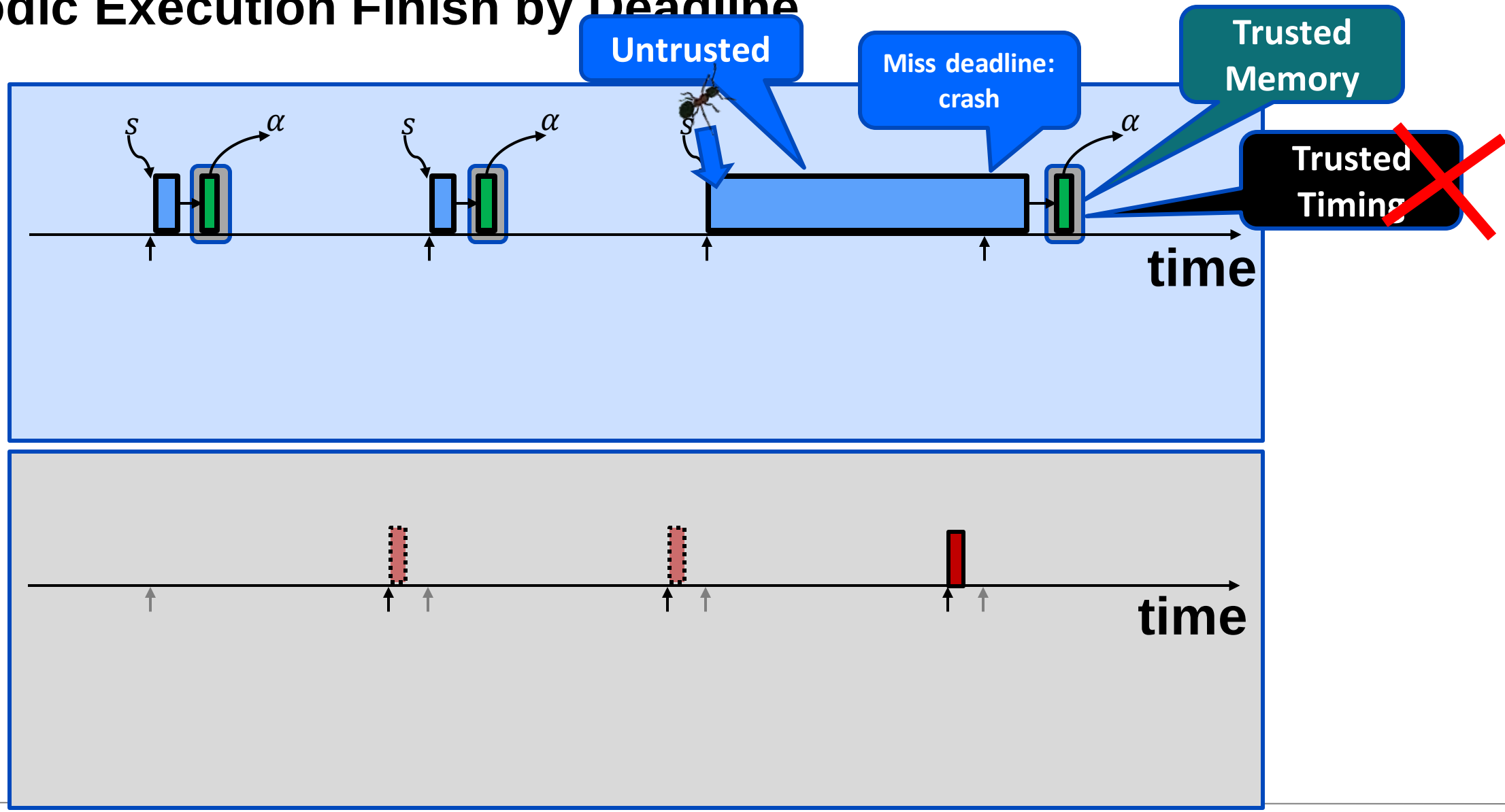
Periodic Execution Must Finish by Deadline



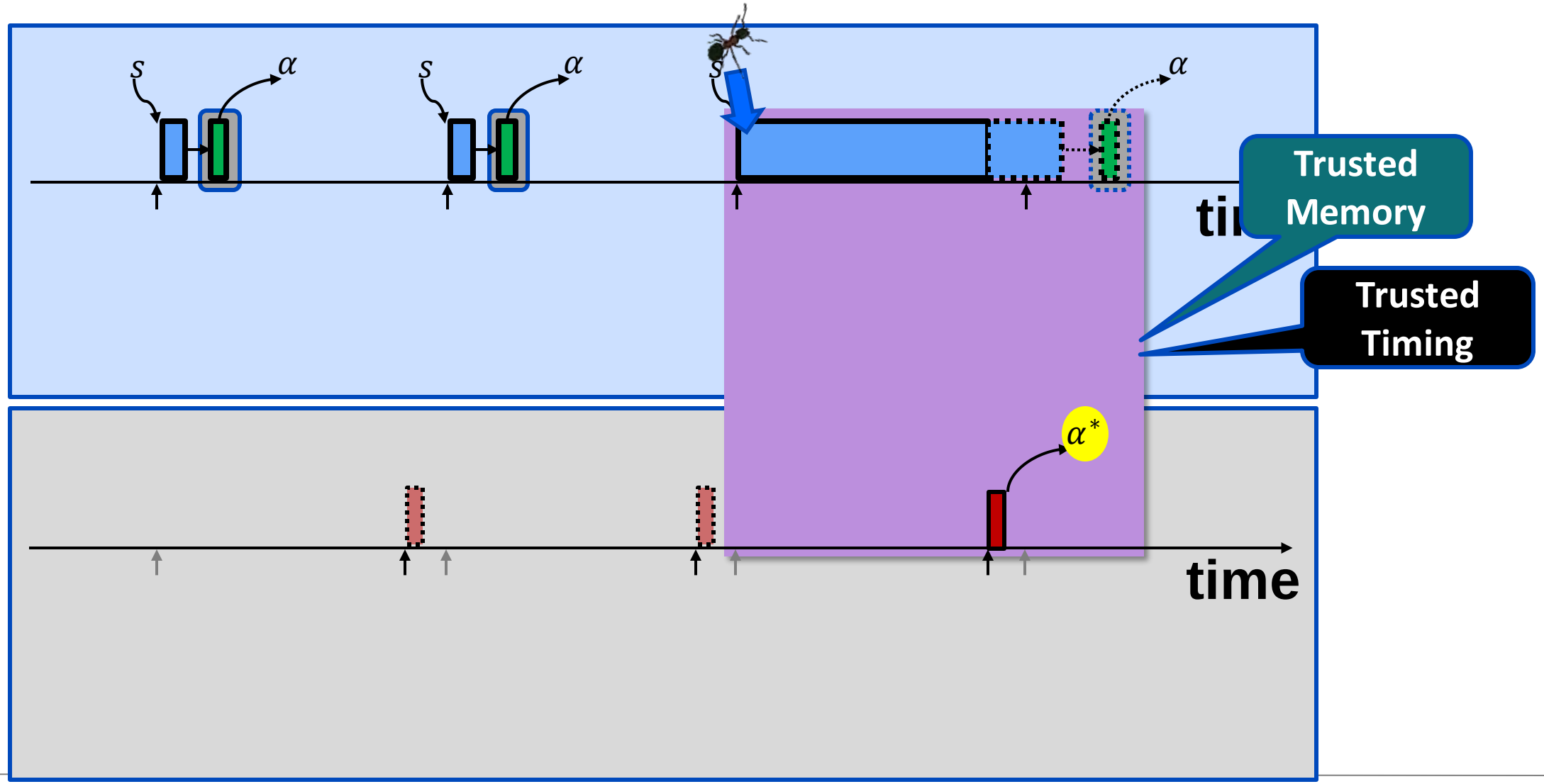
Periodic Execution Finish by Deadline



Periodic Execution Finish by Deadline



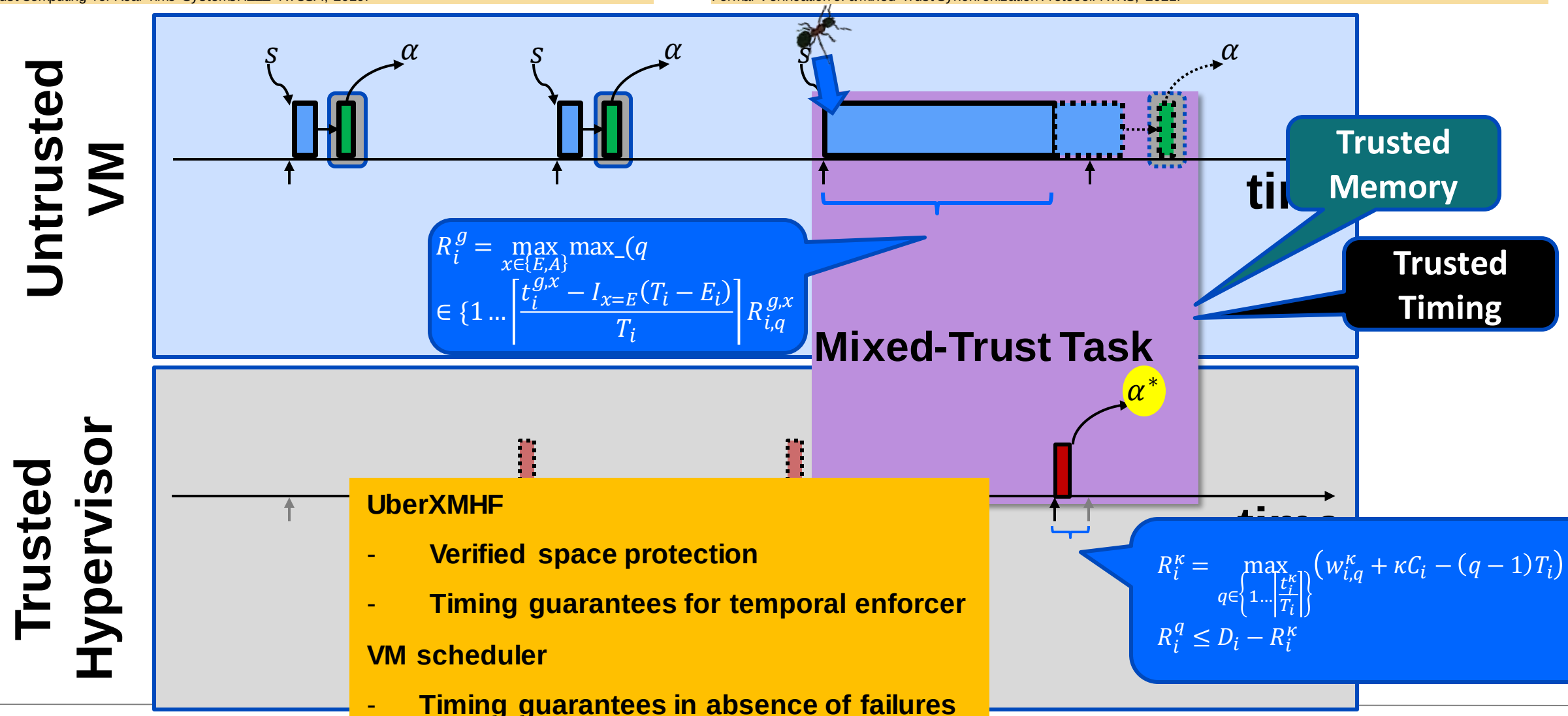
Periodic Execution Finish by Deadline



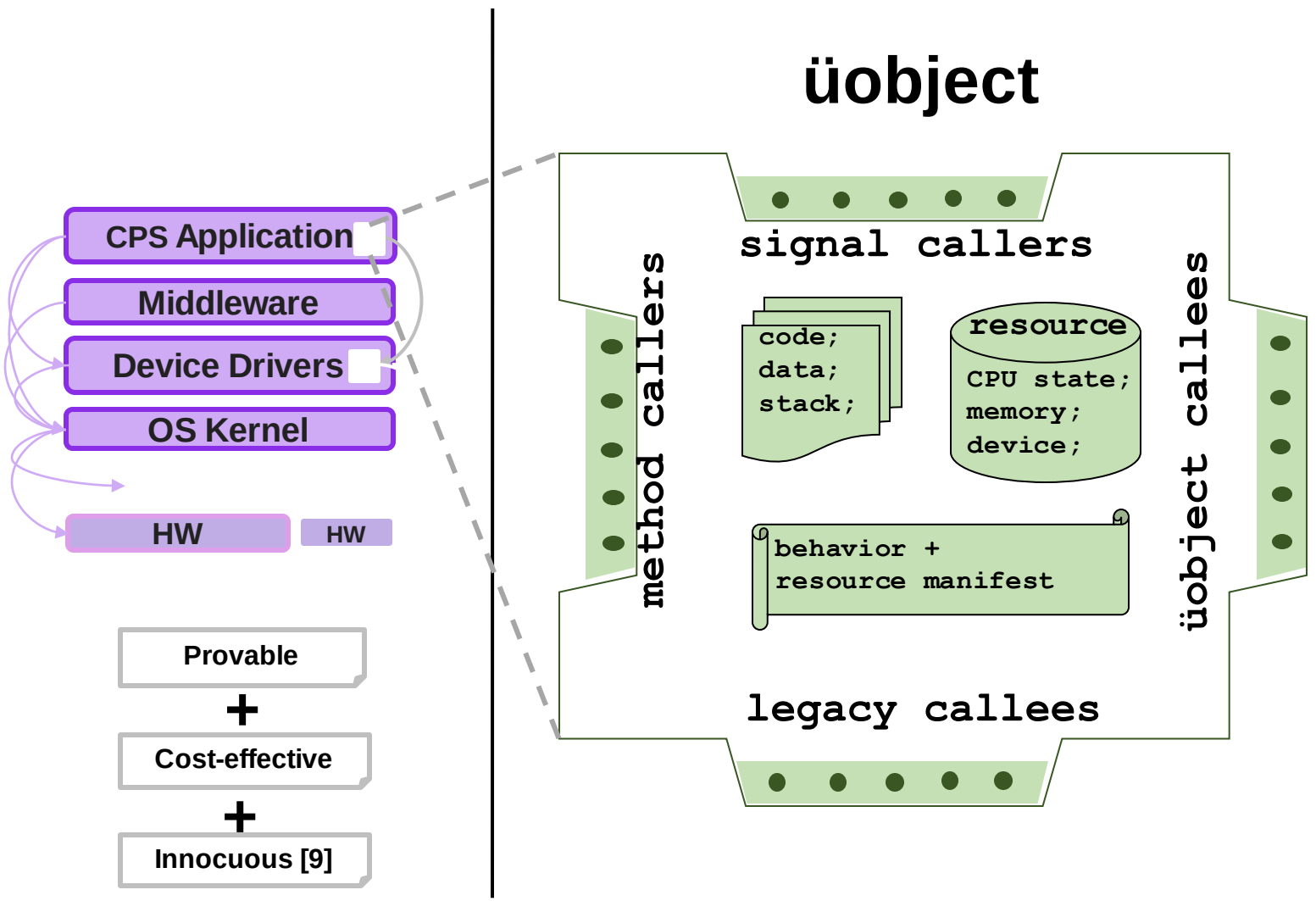
Real-Time Mixed-Trust Computation

D. de Niz, B. Andersson, M. Klein, J. Lehoczky, A. Vasudevan, H. Kim, & G. Moreno.
Mixed-Trust Computing for Real-Time Systems. IEEE RTCSA, 2019.

R.Martins, M. McCall, D. de Niz, A. Vasudevan, B. Andersson, M. Klein, J. Lehoczky, and H. Kim.
Formal Verification of a Mixed-Trust Synchronization Protocol. RTNS, 2021.



Verified Protection at Hypervisor, Kernel, Application



- Singleton object guarding exclusive indivisible system resource
- Principled entry, interruption, legacy code invocations and üobject invocations
  - execution trace respecting program control-flow enables use of state-of-the-art program verification tools
  - facilitate AG reasoning and composition
- Call-return Interfacing
  - Handle various CHIC programming idioms
- Resource Interface Confinement
  - Resource protection and access control
  - Support Shared memory concurrency -> multi-threaded execution and reasoning

A. Vasudevan, P. Maniatis, R.Martins, S. Chaki. Practical, Provable, End-to-End Guarantees at the Edge. USENIX Workshop on Hot Topics in Edge Computing 2020

M. McCormack, A. Vasudevan, G. Liu, V. Sekar
Formalizing an Architectural Model of a Trustw orthy Edge IoT Security Gatew ay
RTCSA 2021

Design for Verification – Verifying Design

Analyze Early Design

- Avoid late, COSTLY discovery / correction of errors

Integrate Optimized Multi-domain Analyses

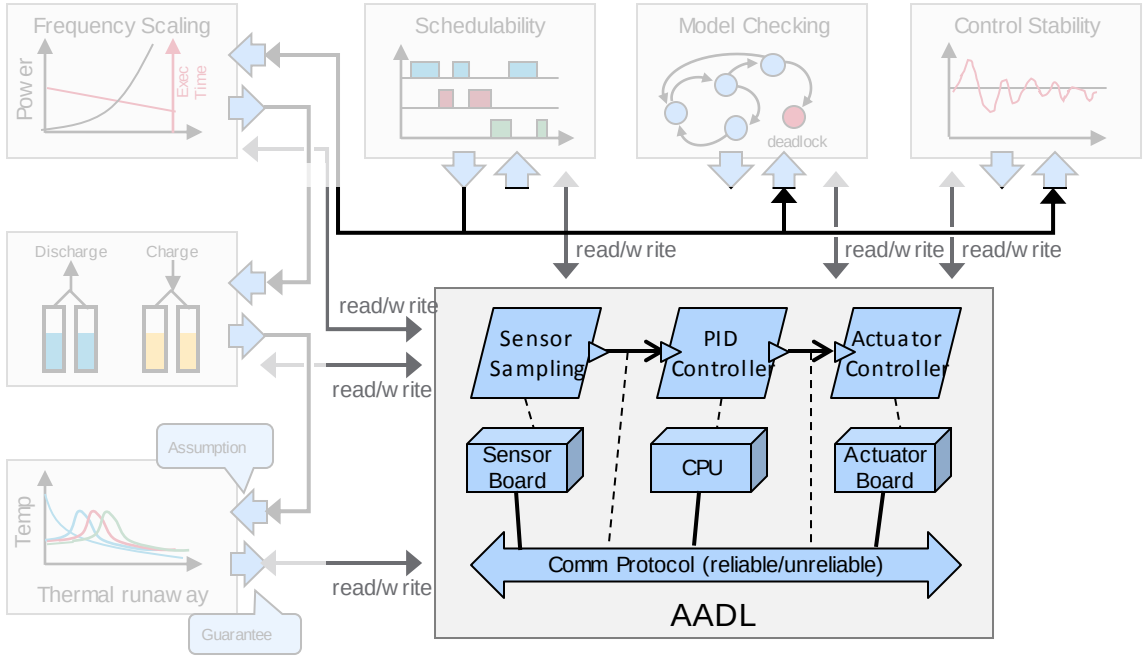
- Discover/resolve conflicts
- Preserve proofs soundness

Verify Analysis Assumptions

- Through refinement
  - Track proof obligations
- Down to the metal

Engineering Practice

- Analysis-Based Processes



Verifying Early Architectural Models

Late Discovery of Design Errors is very costly.

Architecture analysis can **detect** design **error early**

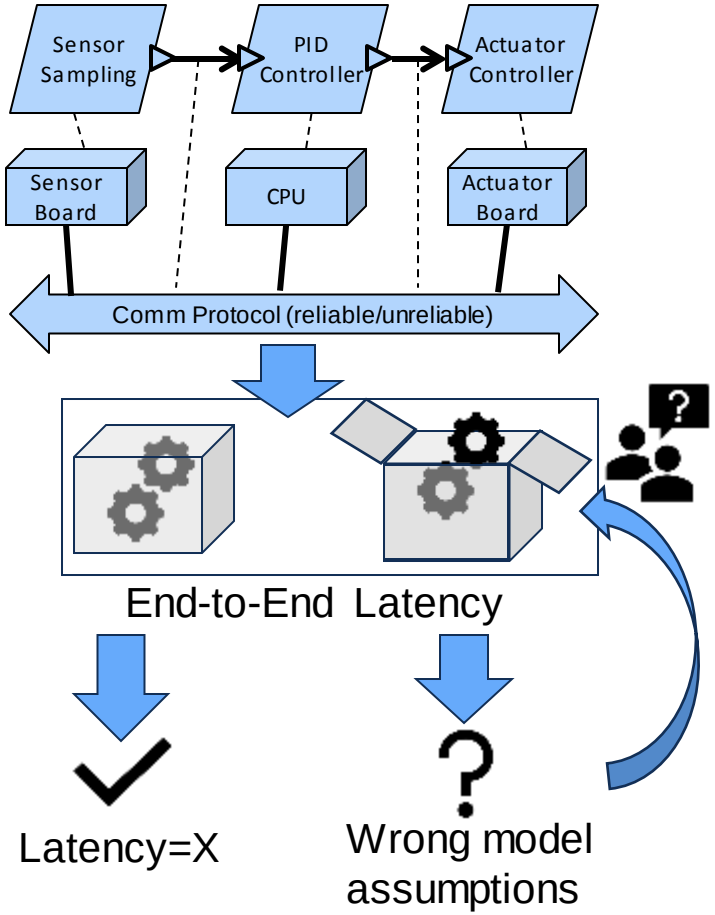
BUT:

Analysis assumptions are often implicit

if **assumptions not met**:

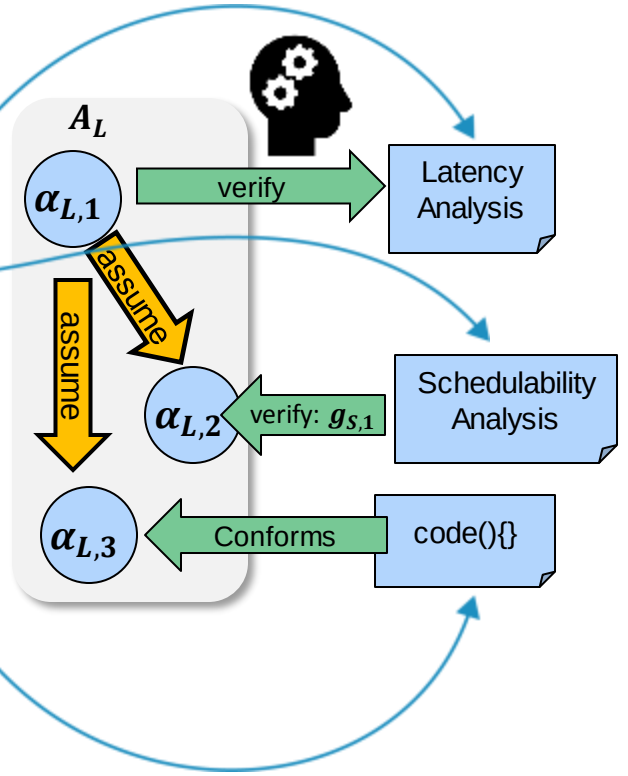
- analyses break down
- reasons not clear to users of analysis tools.

E.g., e2e Latency Assumption: periods multiple of each other (harmonic)



Analysis Contract: Tracking Assumptions and Guarantees

```
contract{
  inputs:
        E2ELatancies
  assumptions:
        areConnectionsDelayed()
        areDeadlinesConstrained()
        areTasksSchedulable()
        areAllThreadsPeriodic()
  analysis:
        meetEndToEndLatencies()
  guarantee:
        [E2EResponses[i] <= E2ELatencies[i]
          for i in range(len(Responses))]
}
```



Shift Left And Down to the Metal

Early Analysis

- Evaluate design decisions with partial information
- E.g., latency analysis before periods
  - periods of tasks must be multiples of each other

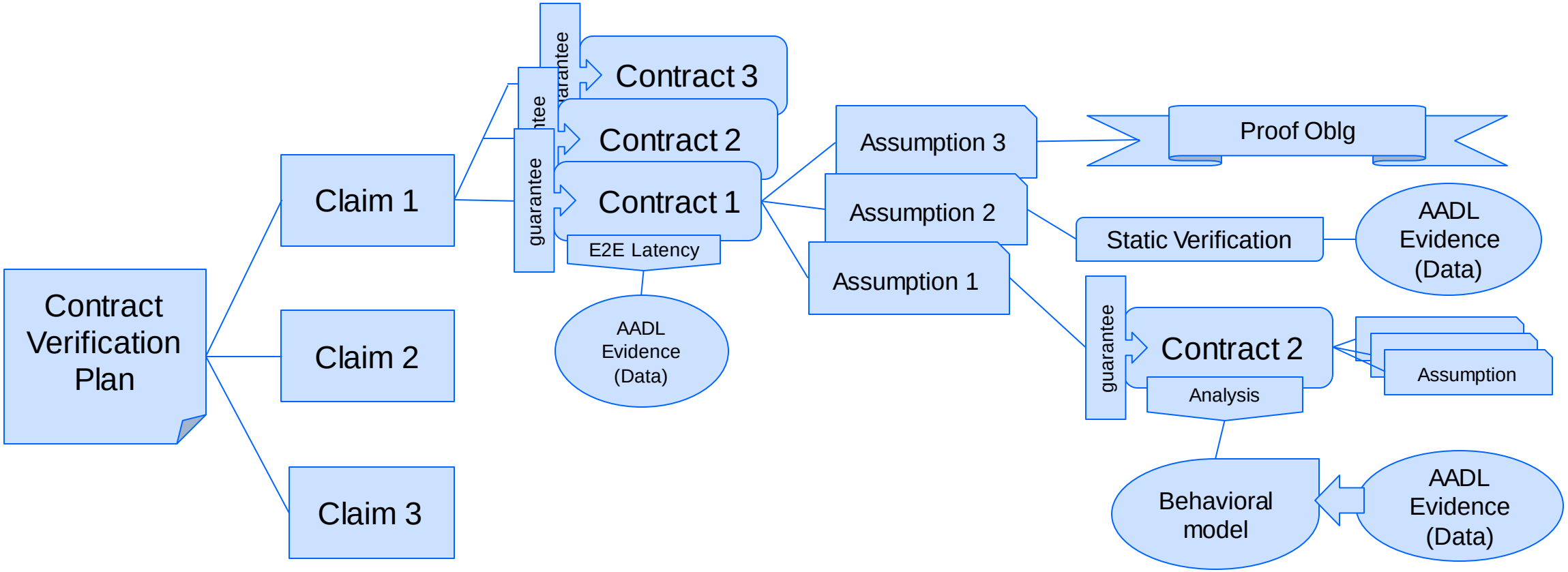
Refinement

- Track pending information
  - periods
- Track and execute pending verification
  - Schedulability

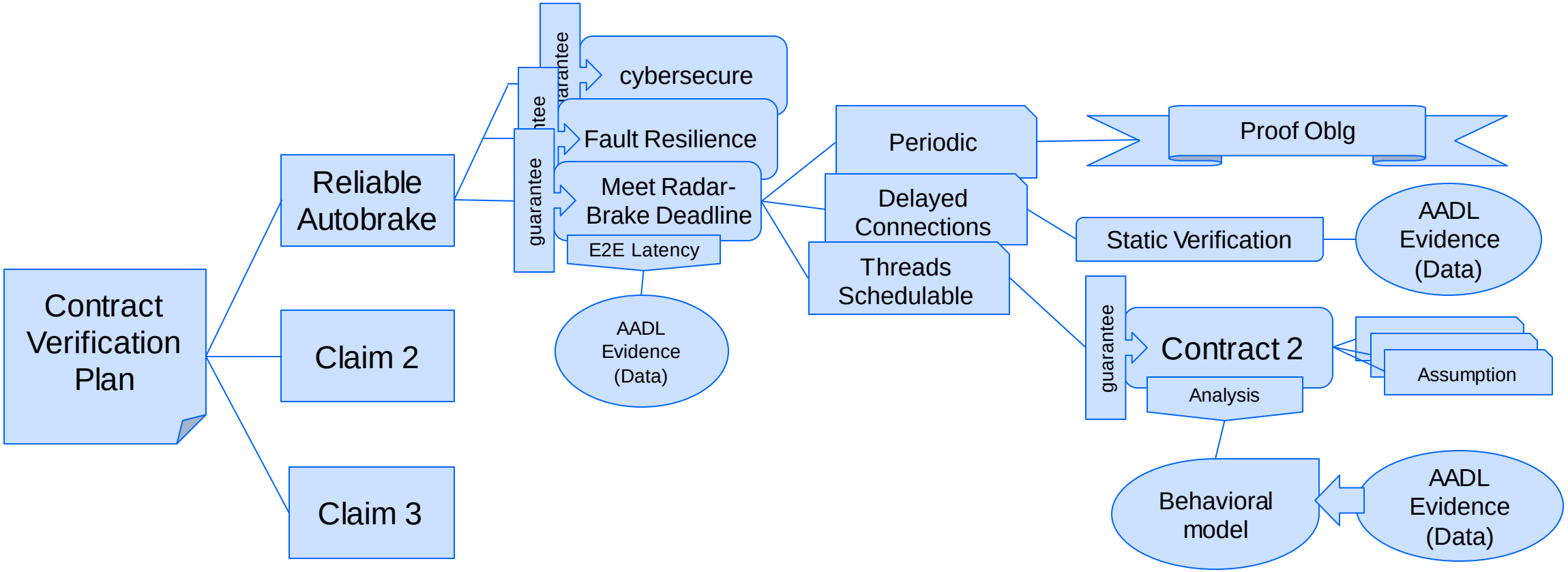
Conformance

- Track implementation assumption
- Verify implementation conformance
  - Task executed strictly periodic $\alpha_{L,3}$

Assurance Contract Argumentation (1)



Assurance Contract Argumentation (2)



Symbolic Contract Argumentation

Assumptions

- Constraints that must be satisfied for a valid analysis

Analysis

- Evaluate whether the guarantee can be discharged

Guarantee

- Assertion presented as a true fact on model

Implementation

- Constraint Satisfaction Solver (Satisfiability Modulo Theories – Z3)
- Implements contract argumentation
  - Evaluate whether constrains can be satisfied with facts from analysis guarantees
- Validate assumptions
  - Proof obligations: lack of constraints allow any value that satisfy assumption (e.g., RM priorities)

Contract Argumentation Scalability

Exploit Knowledge from Scientific Domain

- Efficient algorithms from specialized domains
  - E.g., greedy worst-case response time in real-time theory
  - Implemented in imperative languages

Assume correctness of analysis

- When validating the contract argumentation
- Enables connection with other lower-level verification results
  - E.g., PROSA: coq (theorem prover) verification of real-time theory

Correctness of implementation

- Exploit proven properties of runtime mechanisms: e.g., schedulers, hypervisors
- Exploit code generation
- Deferred code verification to conform to assumptions

Contract Argumentation in Development Lifecycle

Integrity of Analysis

- Verify assumptions
  - Detect violation
  - Suggest repairs
- Offer alternative analysis that satisfy assumptions

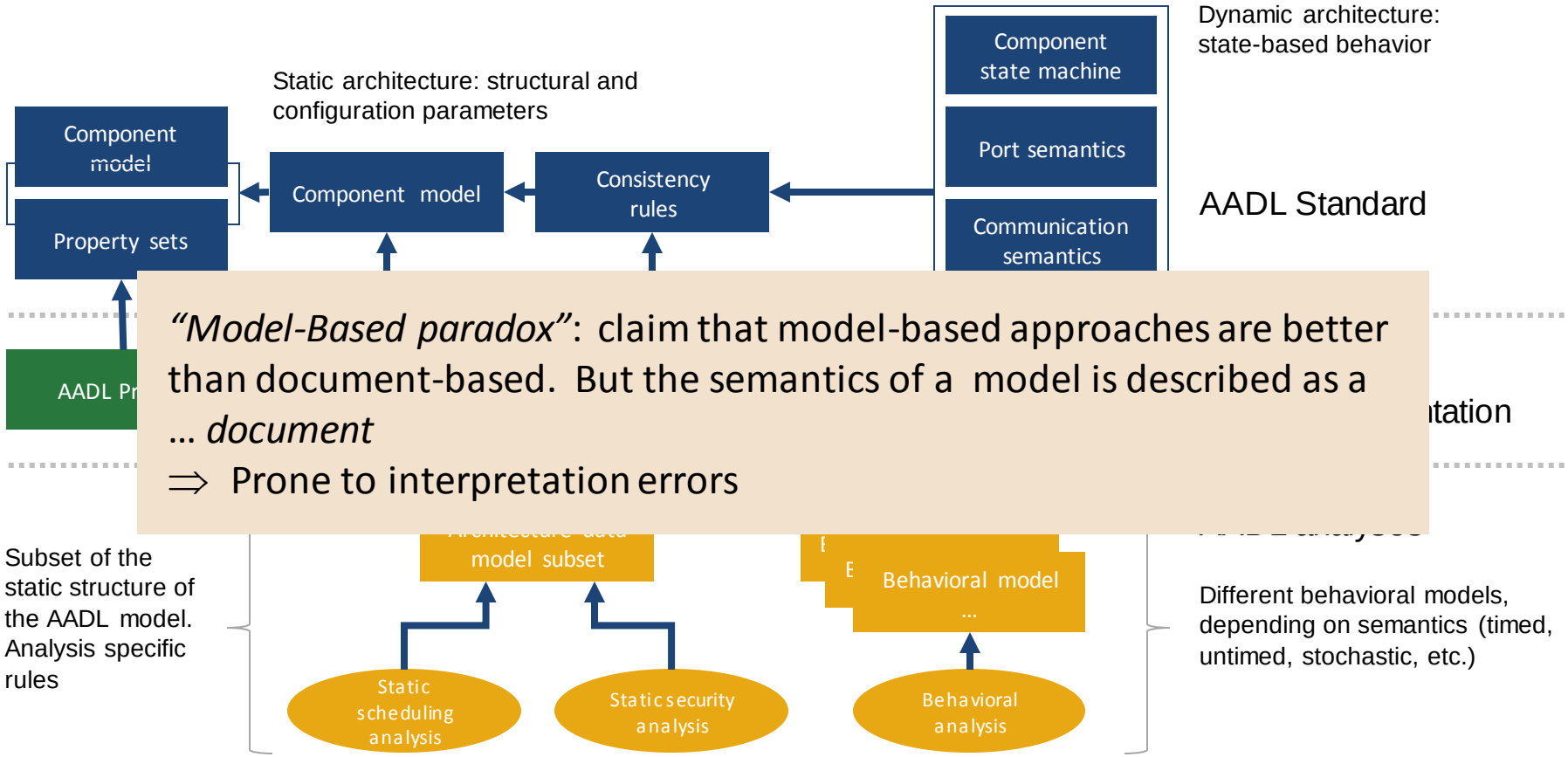
Refine design

- When enough new data for new analysis
- When new data affects proof obligations

Argument reusability

- Self-contained modular analysis contracts

AADL Model Semantics



AADL to Coq – encoding the grammar

Coq inductive types provide the foundation to encode an AST as a Coq type

```
<category> implementation foo.i [extends <bar>.i]
subcomponents
  -- internal elements
connections
  -- from external interface to
  -- internal subcomponents
properties
  -- list of properties
end foo.i;
```

```
Inductive component :=
| Component : identifier →
    ComponentCategory → (* category *)
    fq_name →
    list feature →
    list component →
    list property_association → component
(* .. *)
```

Coq typing rules restricts the construction of model elements, e.g. components

AADL to Coq – legality rules

Legality rules define the correctness of some syntactic statements,

e.g. well-formedness of an AADL component, as a proposition

```
Definition Well_Formed_Component (c : component) :
Prop :=
    Well_Formed_Component_Id (c) /\
    Well_Formed_Component_Classifier (c) /\
    Well_Formed_Component_Features (c) /\
    Rule_4_5_N1 (c).
```

A decidable proposition (in `Prop`) denotes a statement that can be proved as either true or false.

(so far) implemented rules are decidable => they can be implemented as Boolean-returning functions

Note: some (minor) reformulations in the standard required to remove ambiguities in order of evaluation for typing rules

Integrating Coq Proofs

Schedulability is one facet of the correctness of a CPS

PROSA supports abstract Response-Time Analysis in Coq

- Data structure lemmas to check schedulability
- Axioms on the system (mono-core, fixed priority) not visible

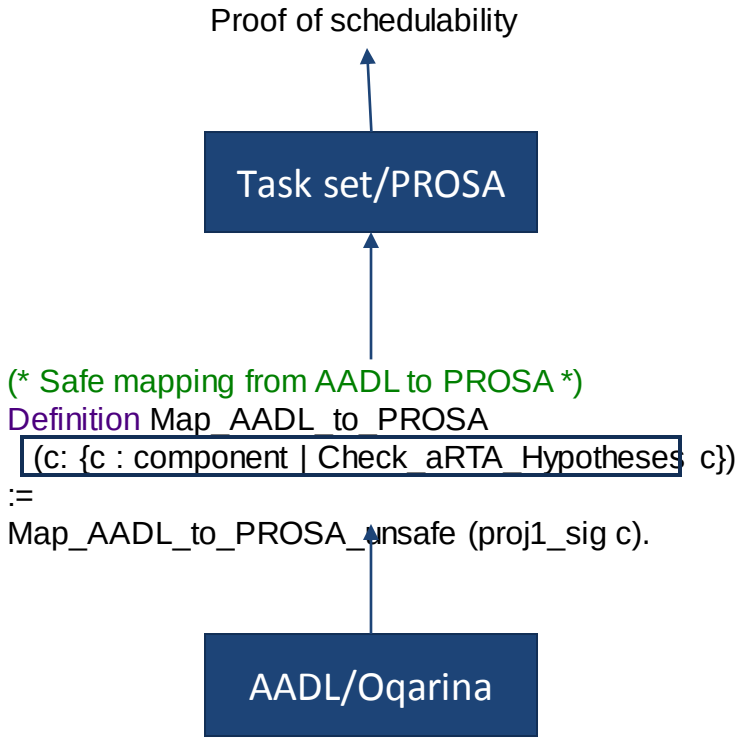
PROSA axioms are decidable properties of AADL models

- Expressed using Resolute

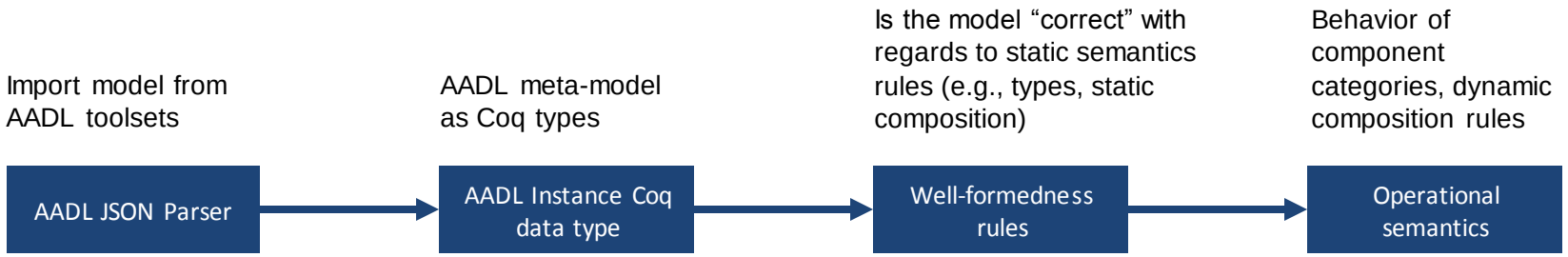
Mapping from AADL to PROSA taskset definition

- translation of concepts (task -> job, priority, WCET, …), guarded by a proof the AADL model is correct

Proof of schedulability using PROSA lemmas



Oqarina
https://github.com/Oqarina/oqarina



Coq data types ⇔ AADL meta-model, typing rules, support for building a model

Well-formedness rules ⇔ AADL legality/consistency rules (i.e., model validity)

Operational semantics ⇔ how to "execute" a model (e.g., proof, model checking, simulation)

Features:

- User-defined propositions, Resolute
- mono-core scheduling analysis using the PROSA library
- simulation of an AADL model by mapping to the DEVS formalism *(not discussed today)*

Concluding Remarks

Verification of CPS

- Multiple Analyses from Multiple Domains

Scalable Verification

- Minimize verification : enforcement
- Optimized Algorithms (domain specific)

Analyses Integration

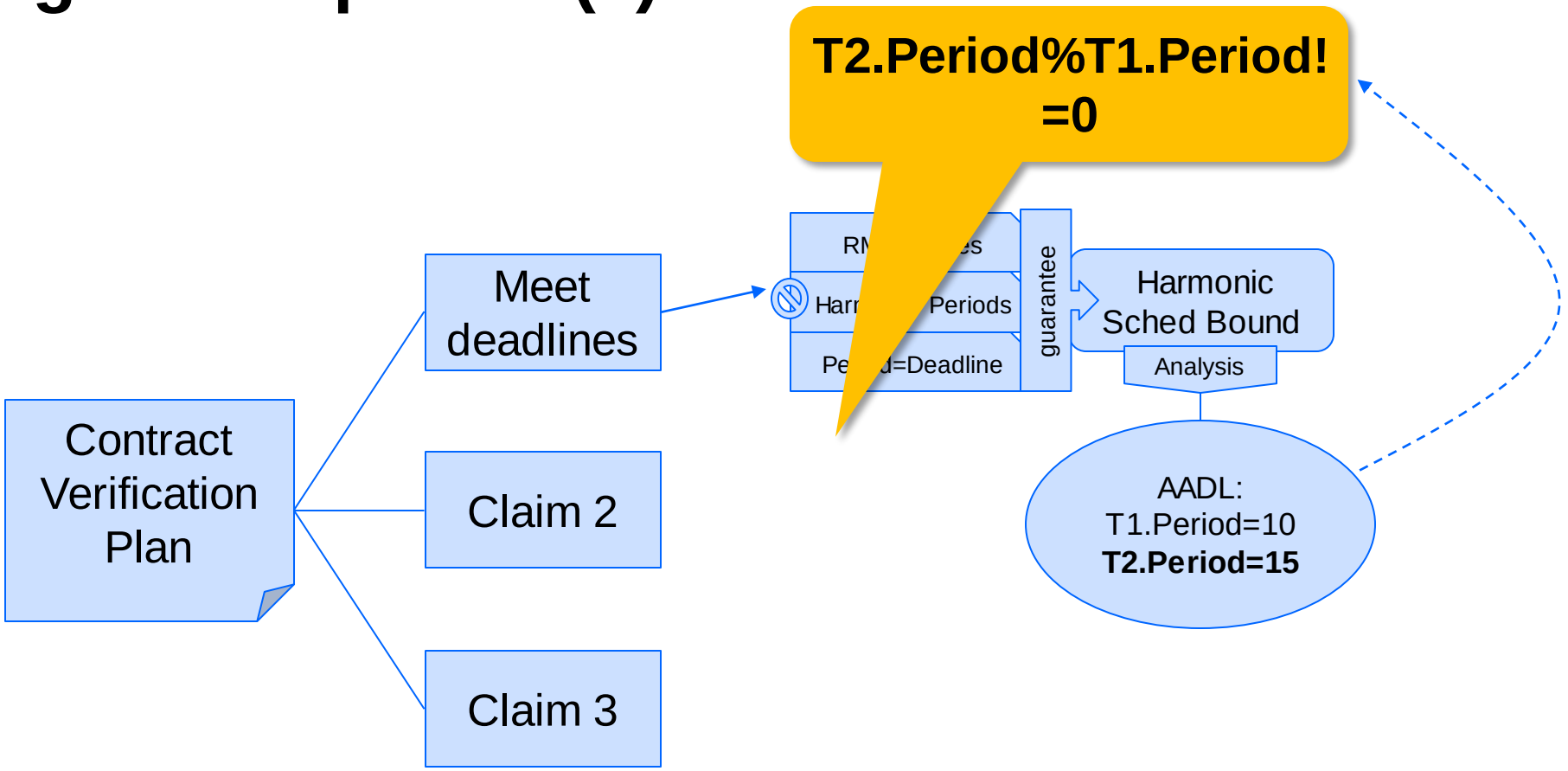
- Across domains

Preserve Soundness

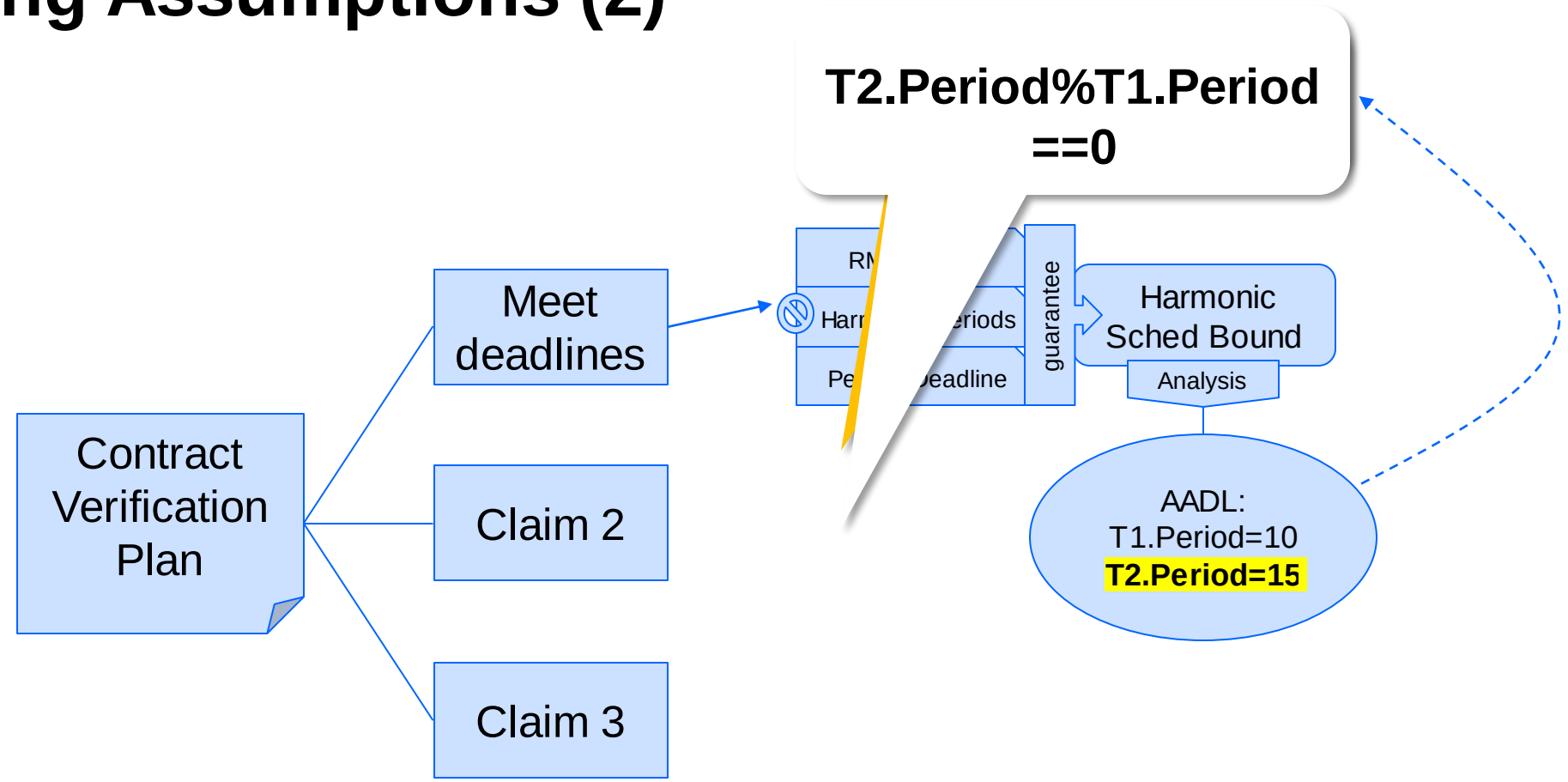
- Protect verified components from unverified ones
- Track assumptions to the metal
- Integrate proofs

BACKUP

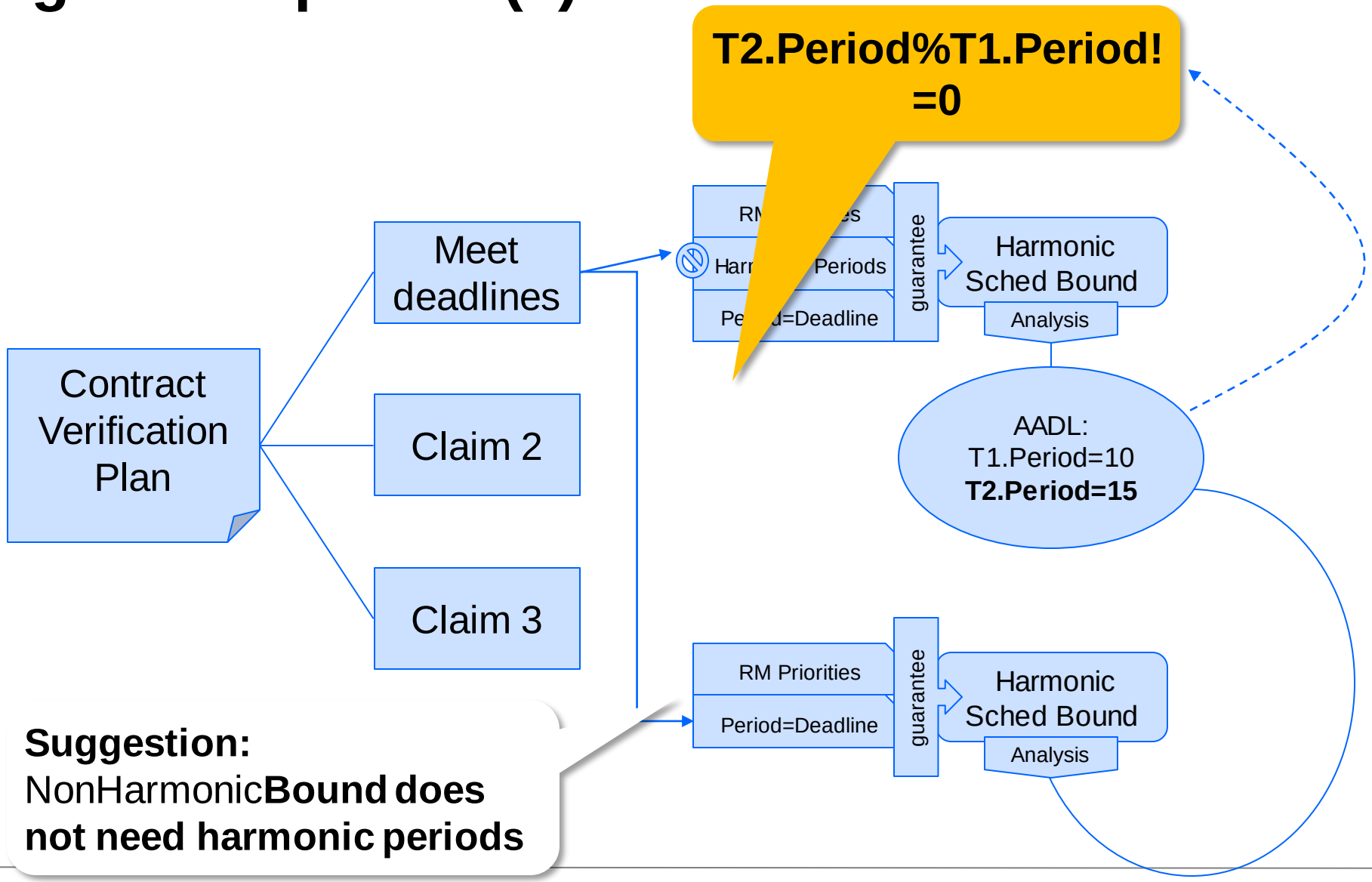
Repairing Assumptions (1)



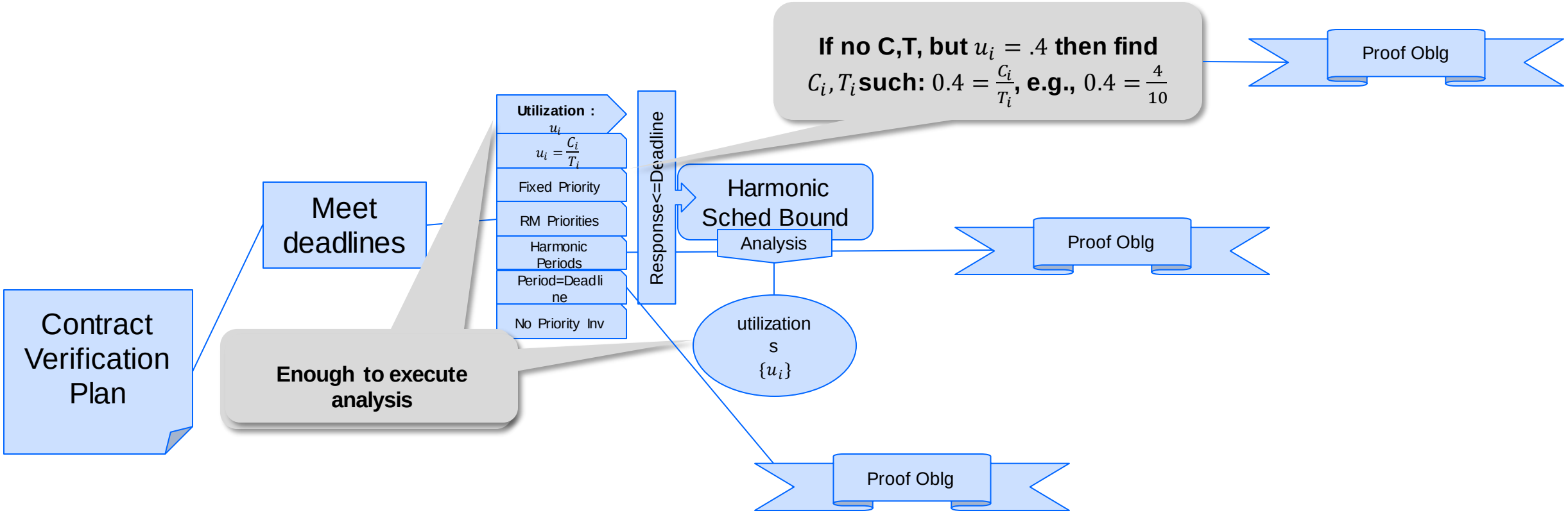
Repairing Assumptions (2)



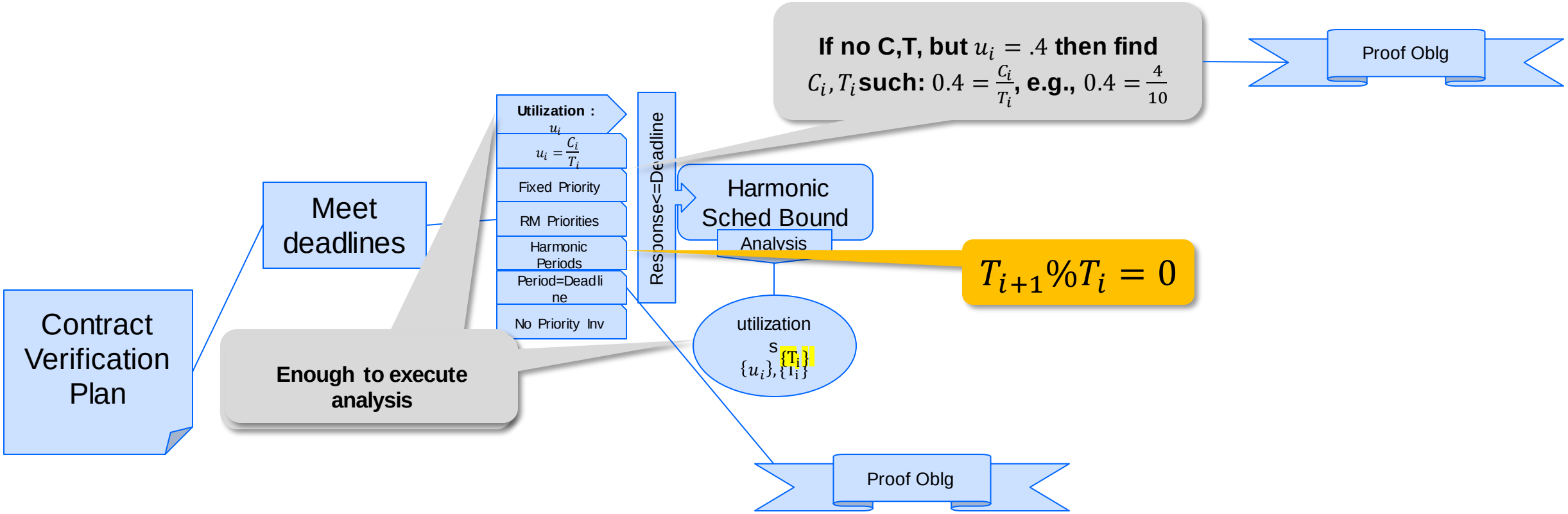
Repairing Assumptions (3)



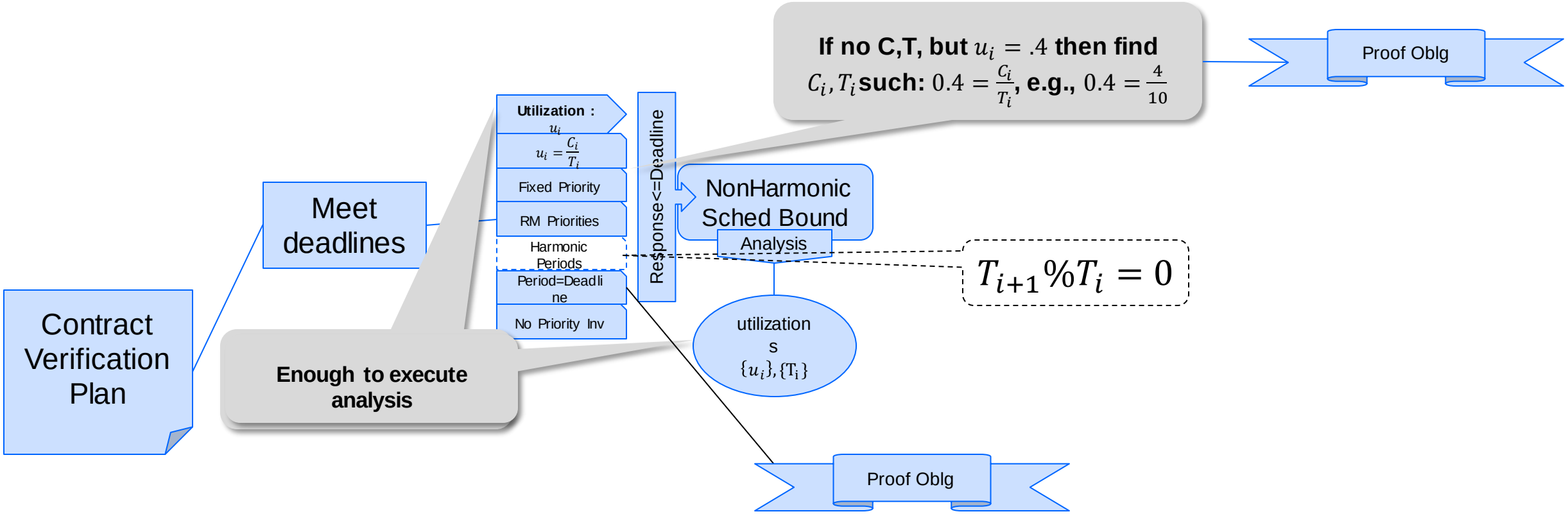
Refinement Throughout Development (1)



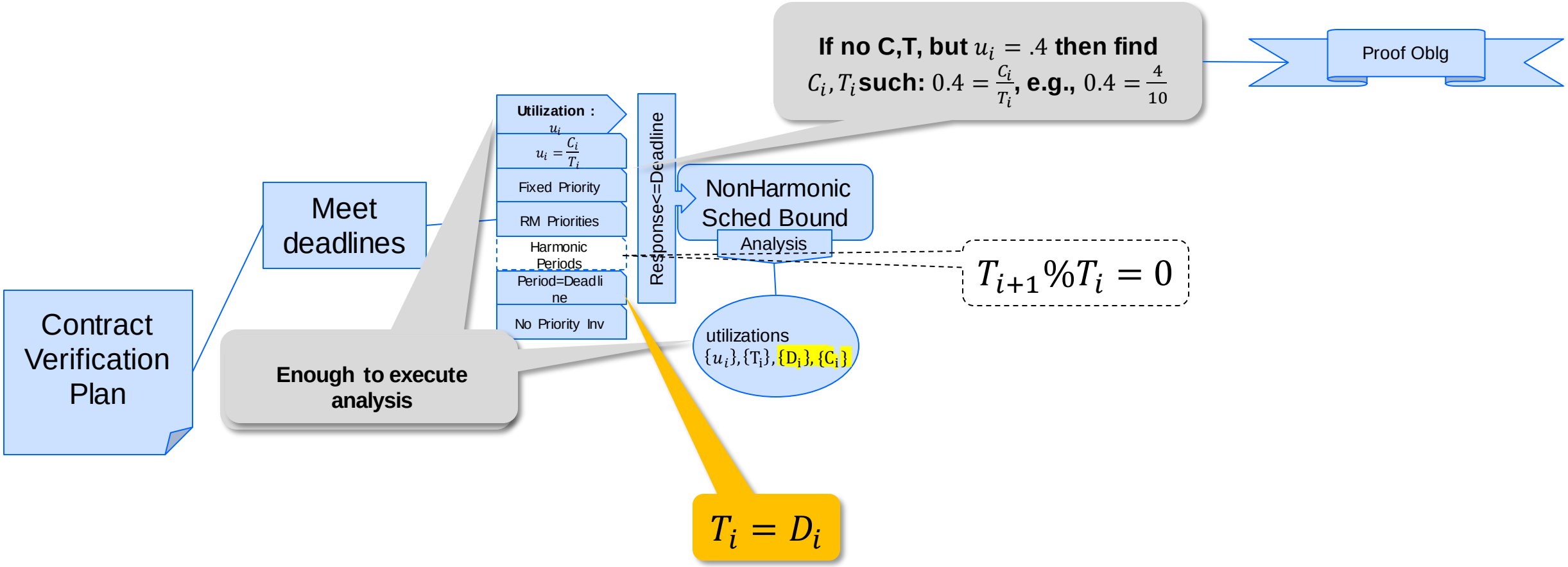
Refinement Throughout Development (2)



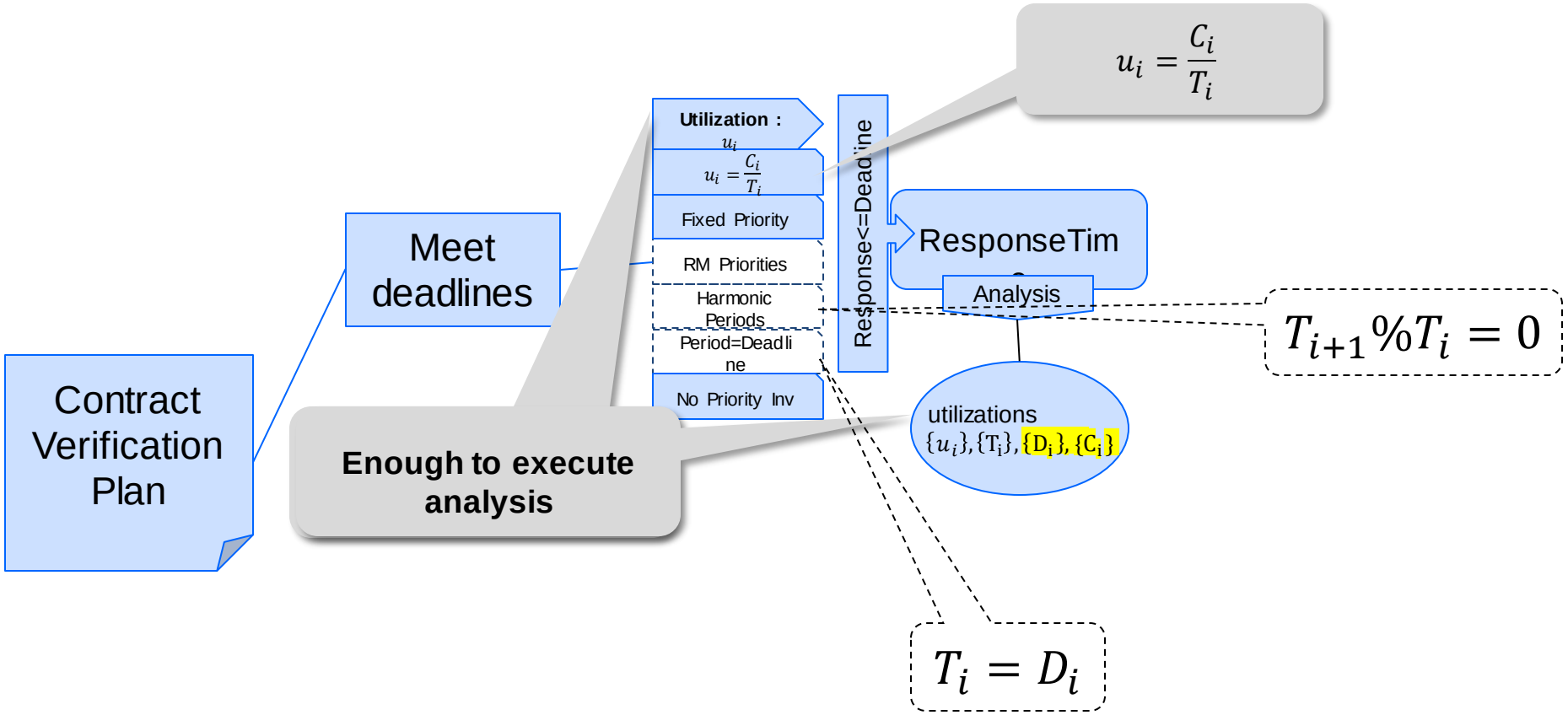
Refinement Throughout Development (3)



Refinement Throughout Development (4)



Refinement Throughout Development (5)



Argument Modularity