



Scaling Up Formal Verification

Dionisio de Niz, Ph.D.

Principal Researcher & Technical Director
Assuring Cyber Physical Systems Directorate



Copyright 2020 Carnegie Mellon University.

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.

The view, opinions, and/or findings contained in this material are those of the author(s) and should not be construed as an official Government position, policy, or decision, unless designated by other documentation.

NO WARRANTY. THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

[DISTRIBUTION STATEMENT A] This material has been approved for public release and unlimited distribution. Please see Copyright notice for non-US Government use and distribution.

This material may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

Carnegie Mellon® is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

DM20-0651



Formal Verification of Complex Software

Formal Verification Does not Scale

Hence: Minimize what must be verified

Prevent Unverified Software to Behave Unsafe

- Enforcer: Detect & Correct Unsafe Behavior
- Verify Enforcer



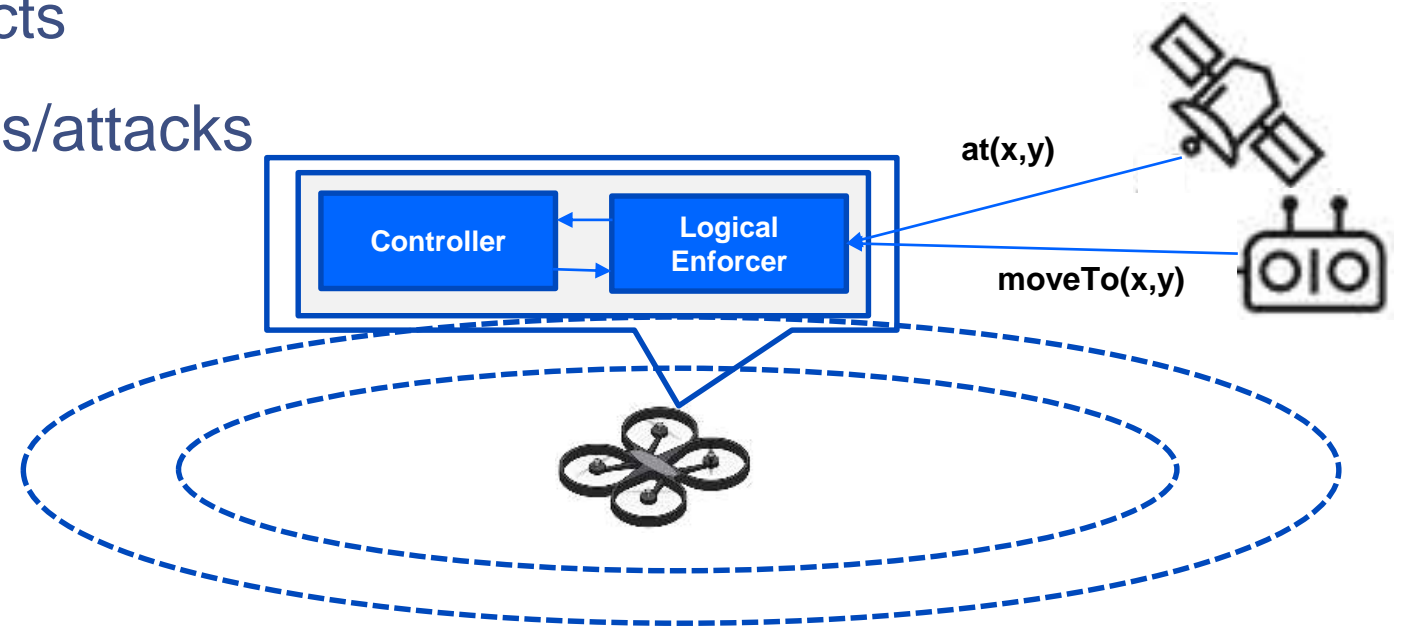
Enforcement-based Verification

Add **simpler (verifiable)** runtime enforcer to make algorithms predictable

Formally: specify, verify, and compose multiple enforcers

- Logic: Enforcer **intercepts/replaces** unsafe action
- Timing: at **right time**
- Physics: verified physical effects

Protect enforcers against failures/attacks



Verifying Physics (Control Theory)

Recoverable Set: $\mathcal{E}_{SCj}(1)$

Safety Set: $\mathcal{E}_{SCj}(\epsilon_s) \triangleq \epsilon_s \mathcal{E}_{SCj}(1)$

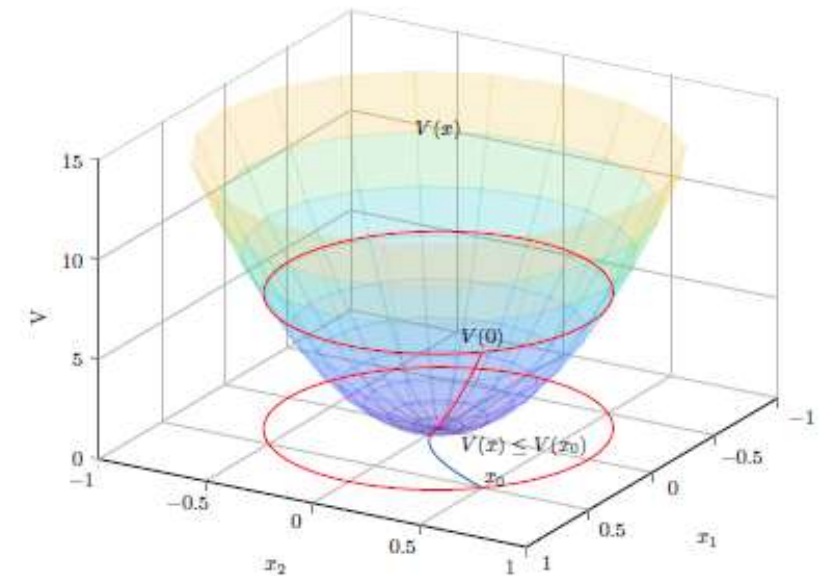
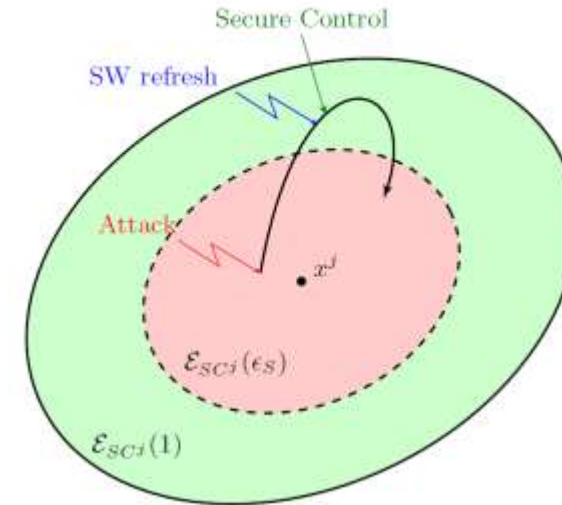
Controlled System: $\dot{x} = f_\varphi(x) \triangleq f(x, \varphi(x))$

Lyapunov Function: $V_\varphi : \mathbb{R}^n \rightarrow \mathbb{R}$, $\mathcal{N}_{V_\varphi}(x_{eq}) \subseteq \mathcal{N}_\varphi(x_{eq})$,
 $V_\varphi(x_{eq}) = 0$ 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 $\epsilon > 0$,

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



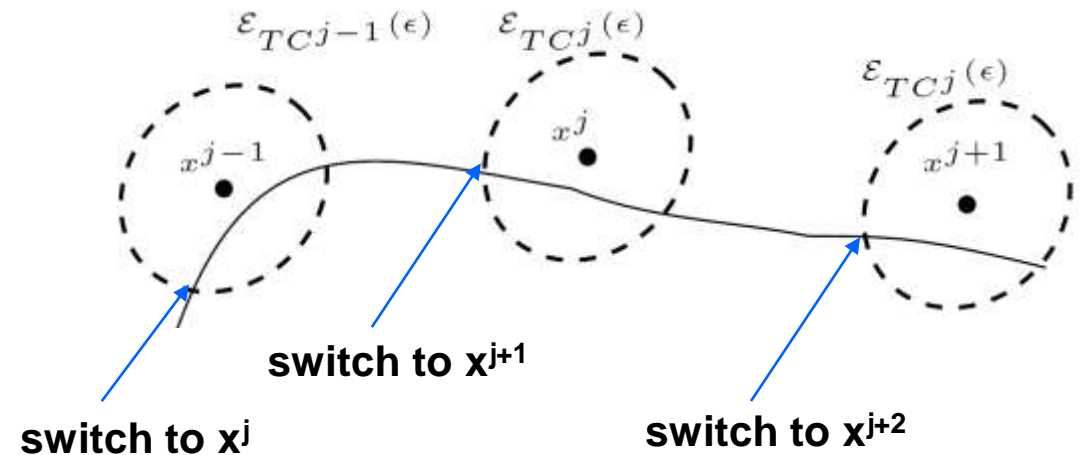
Analysis of Mission Progress

Idea:

Provide a sequence of waypoints that represent a sequence of equilibrium points around which we define the Safe Set.

Goal:

- Safety transition from one waypoint to the next one.
- Liveness (in the case of no errors)



Analysis of Mission Progress Enforcing Unsafe Behavior

6 DOF \Rightarrow 12 state variables

$$\ddot{p}_x = -\cos\phi \sin\theta \frac{F}{m}$$

$$\ddot{p}_y = \sin\phi \frac{F}{m}$$

$$\ddot{p}_z = g - \cos\phi \cos\theta \frac{F}{m}$$

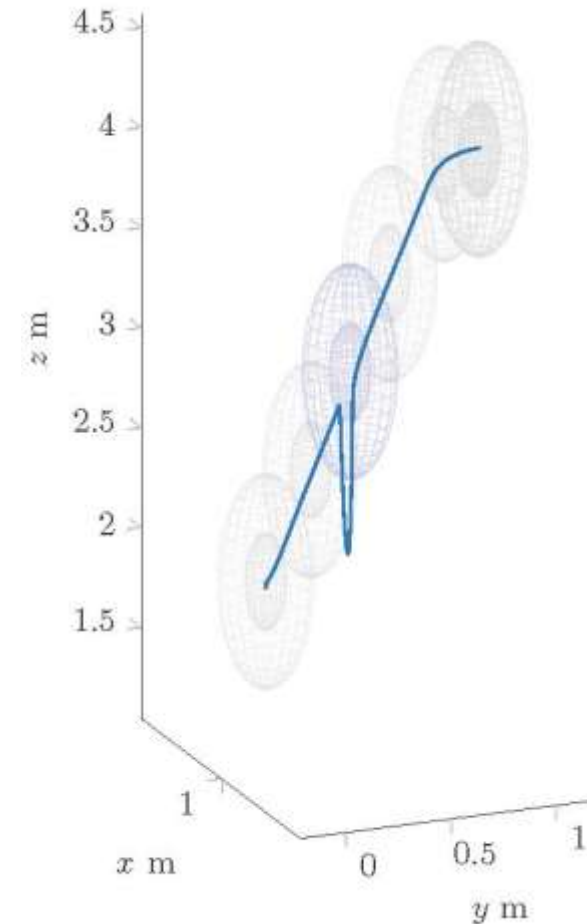
$$\ddot{\phi} = \frac{1}{J_x} \tau_\phi$$

$$\ddot{\theta} = \frac{1}{J_y} \tau_\theta$$

$$\ddot{\psi} = \frac{1}{J_z} \tau_\psi$$

Linear design:

- linearize at equilibrium
- assume full state available
- LQ state feedback design



Drone Experiment



Are We Done Yet?

Scalable Verification

- Only verify safety-critical components
- Guarding unverified one

Trust

- Protect verified components
- Against attacks or bugs from unverified components



Enforcing Unverified Components

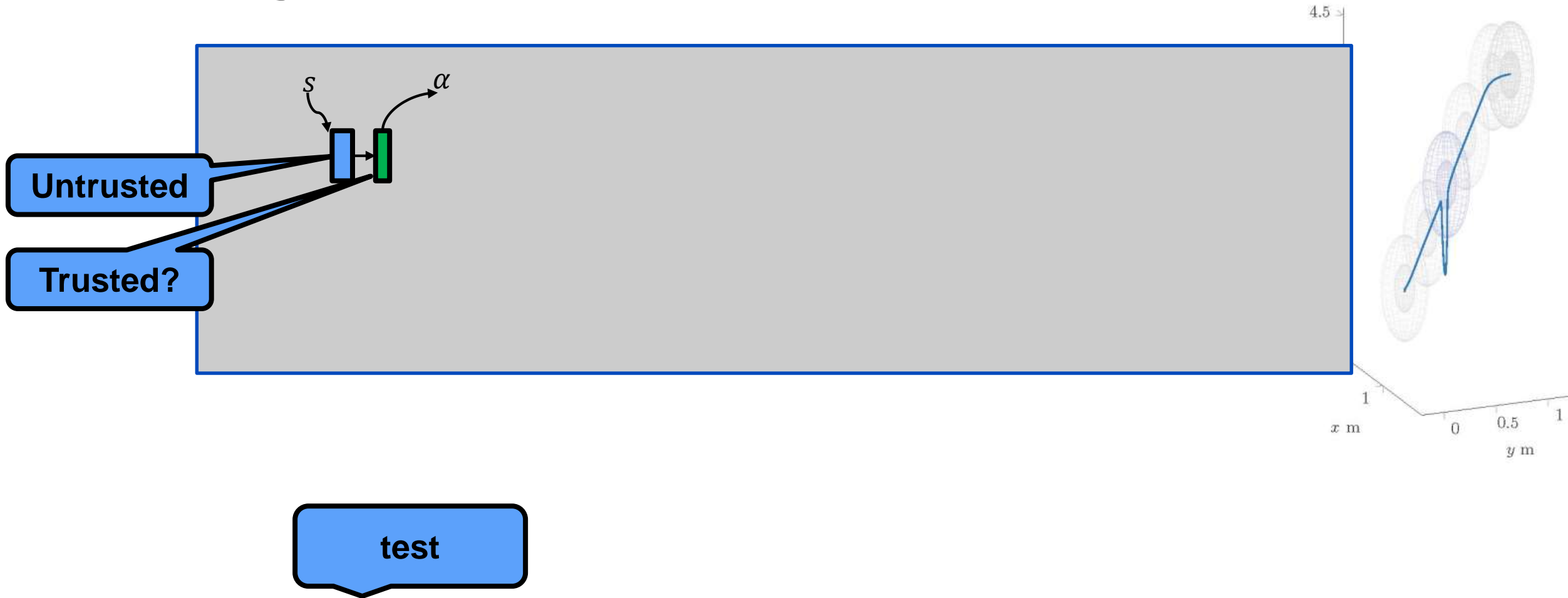


Enforcing Unverified Components



Look for ant picture attribution in : https://commons.wikimedia.org/wiki/File:Ant_illustration.jpg

Enforcing Unverified Components



But enforcer can be corrupted (bug or cyber attack)



Look for ant picture attribution in : https://commons.wikimedia.org/wiki/File:Ant_illustration.jpg



Add Memory Protection



Trusted = Verified & Protected



Are We Done Yet?

Timing can still be corrupted

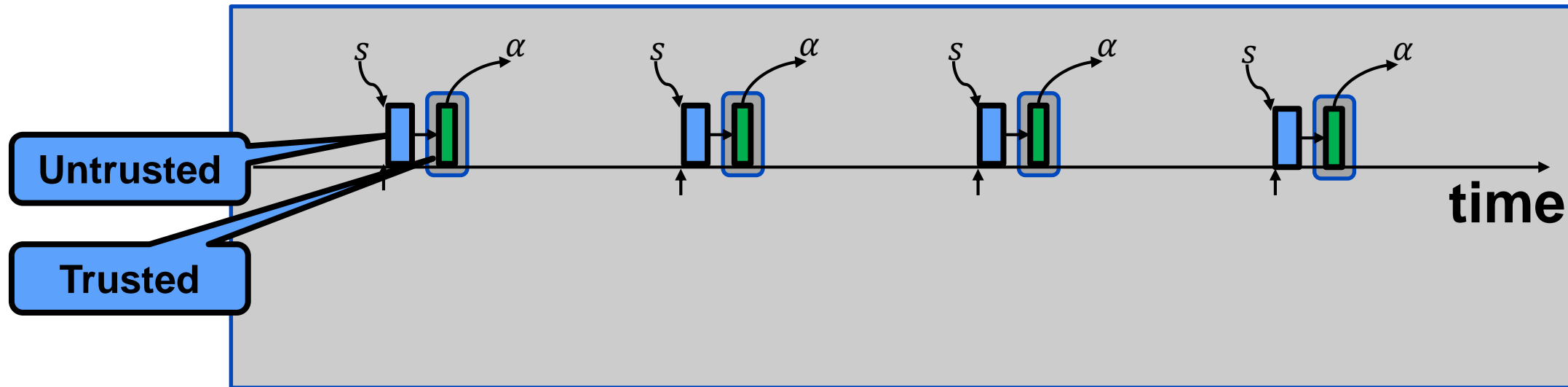
- Guaranteed correct value
- BUT potentially at wrong time

Trusted timely actuation

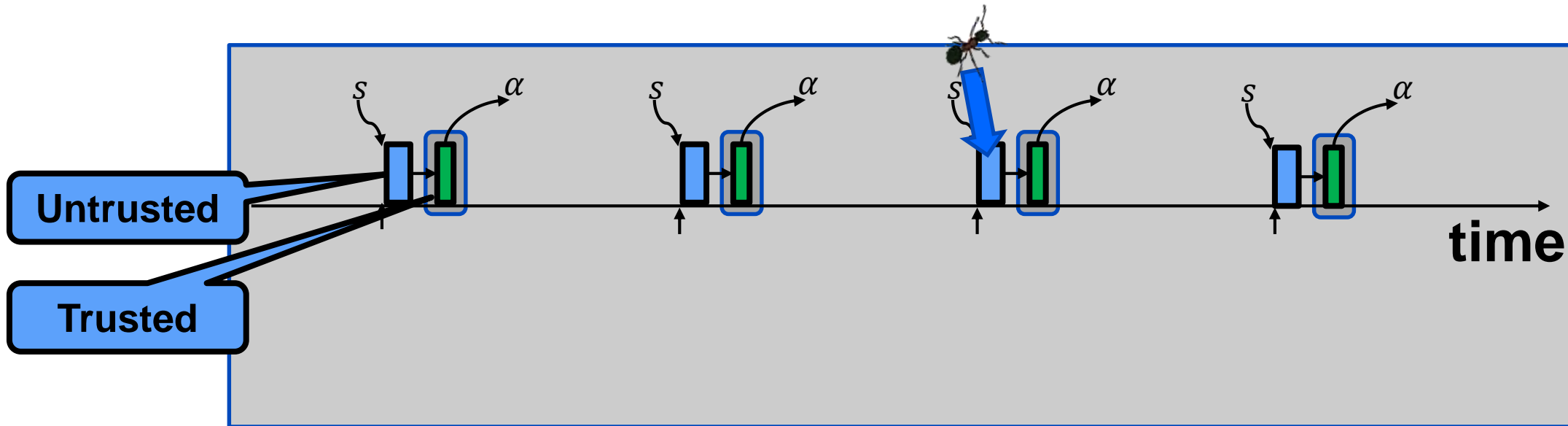
- Tamper-proof time-triggering mechanism
- In sync with periodic controller
- In sync with expected untrusted



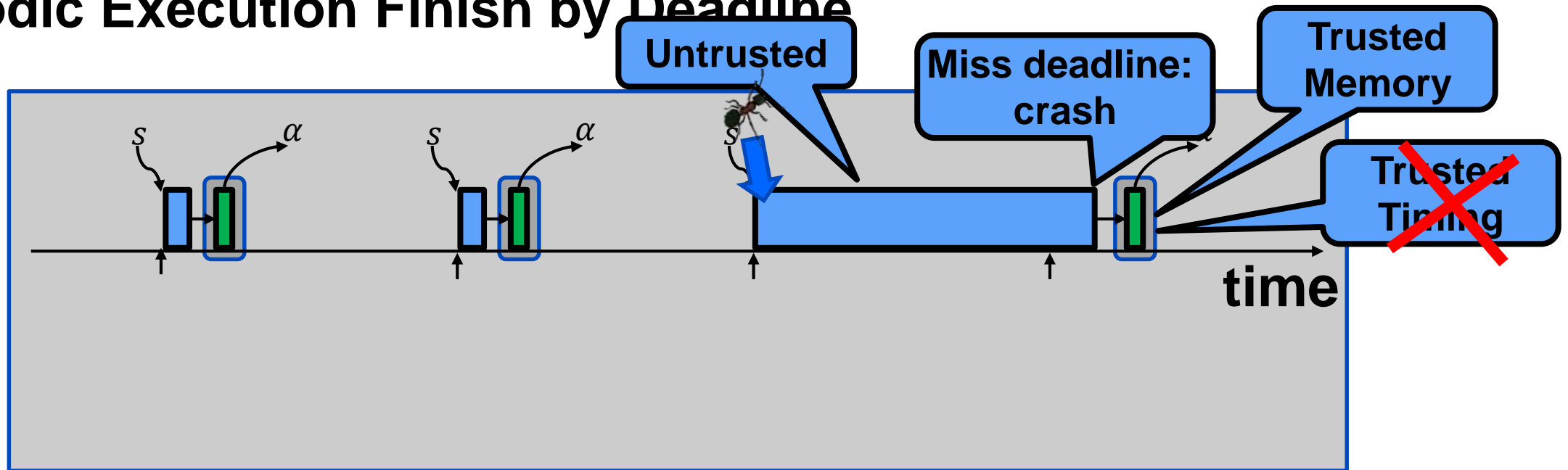
Periodic Execution Must Finish by Deadline



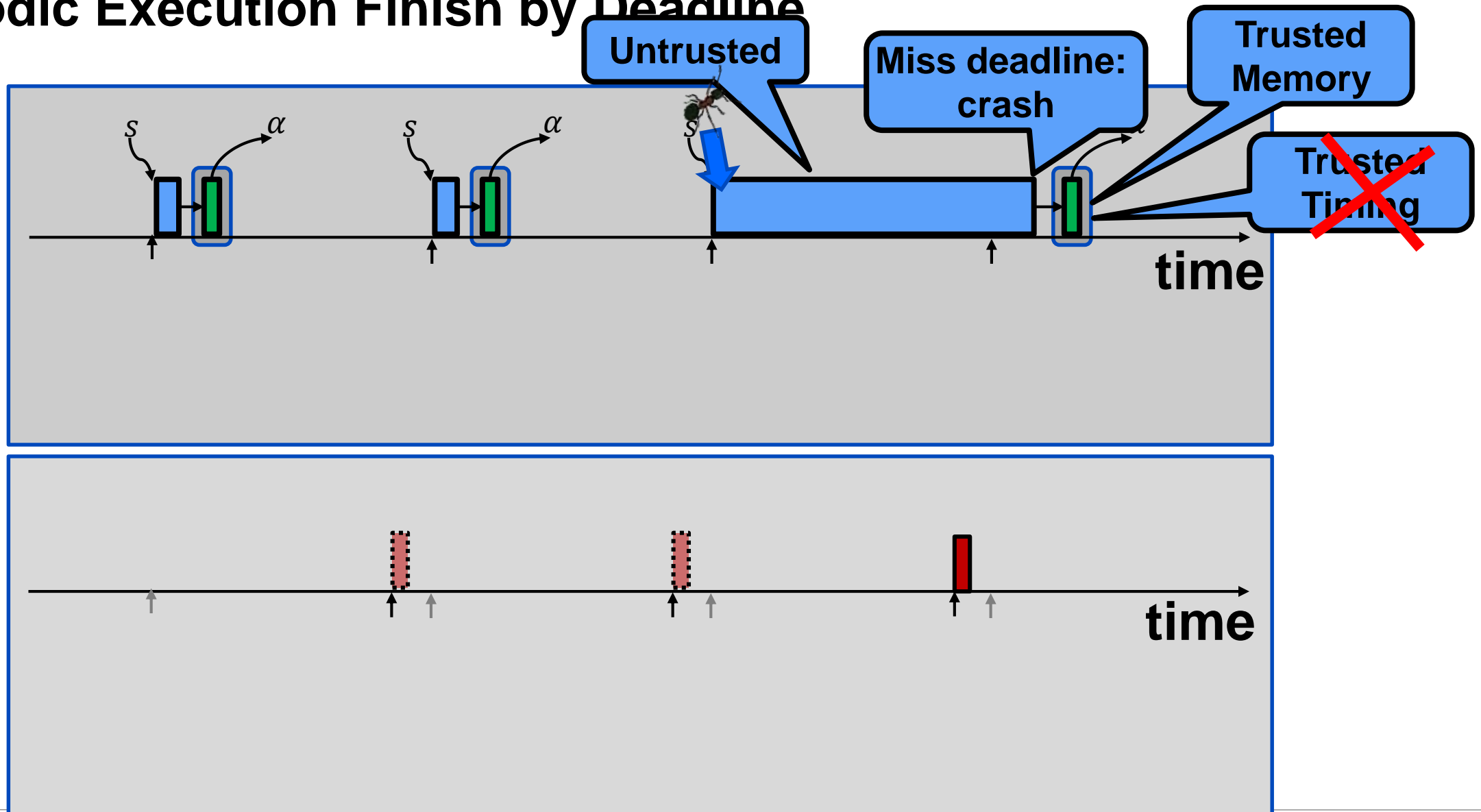
Periodic Execution Must Finish by Deadline



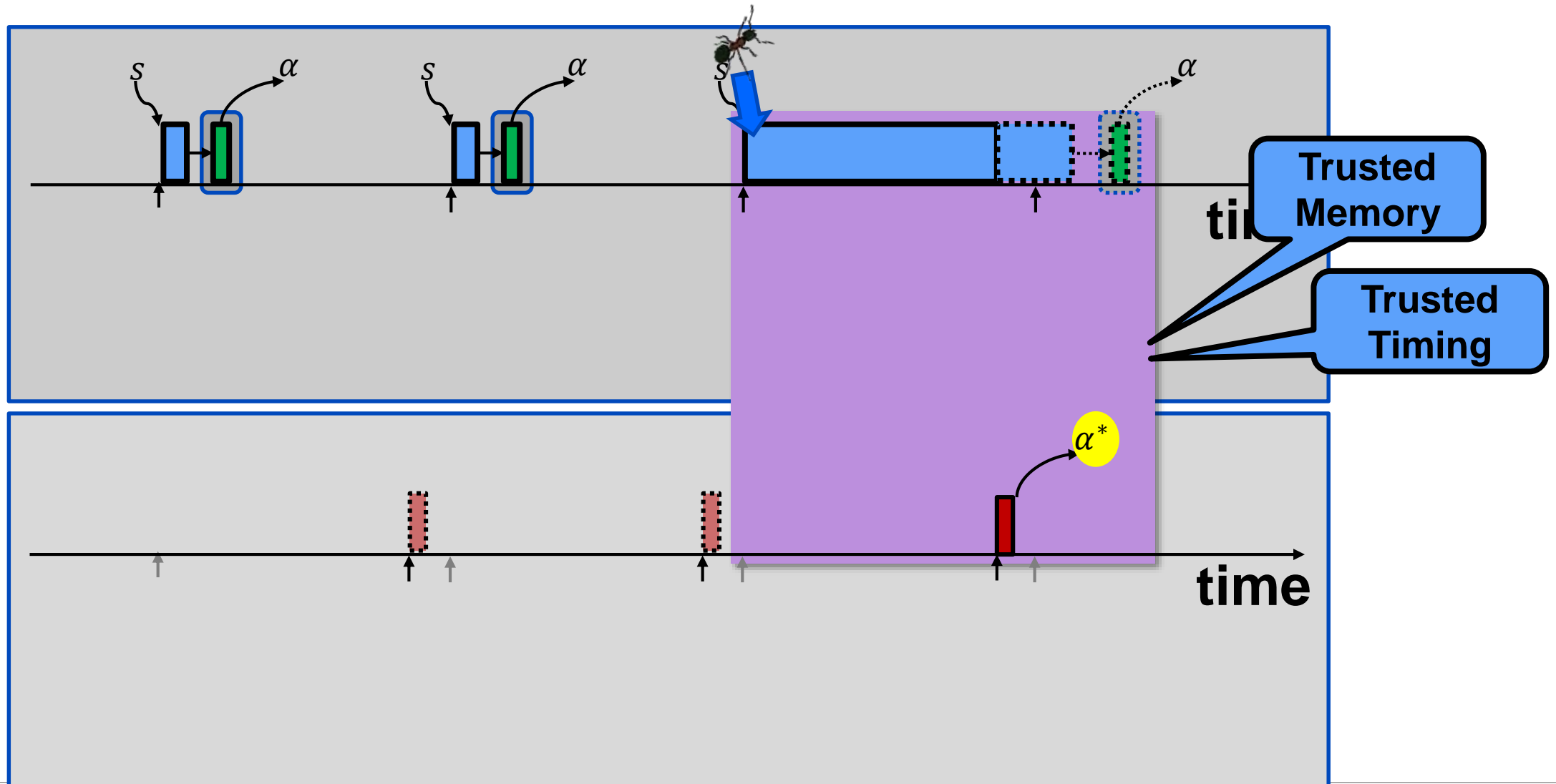
Periodic Execution Finish by Deadline



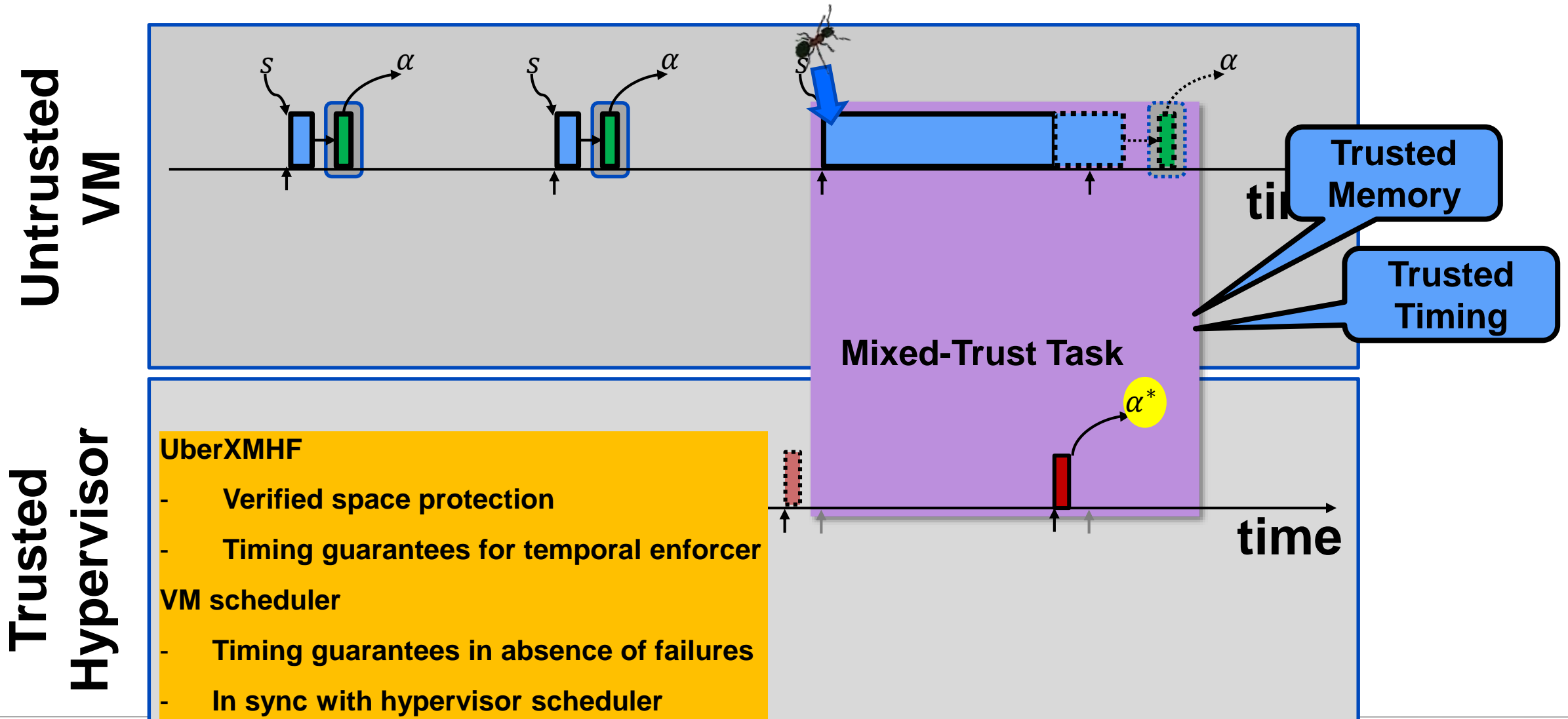
Periodic Execution Finish by Deadline



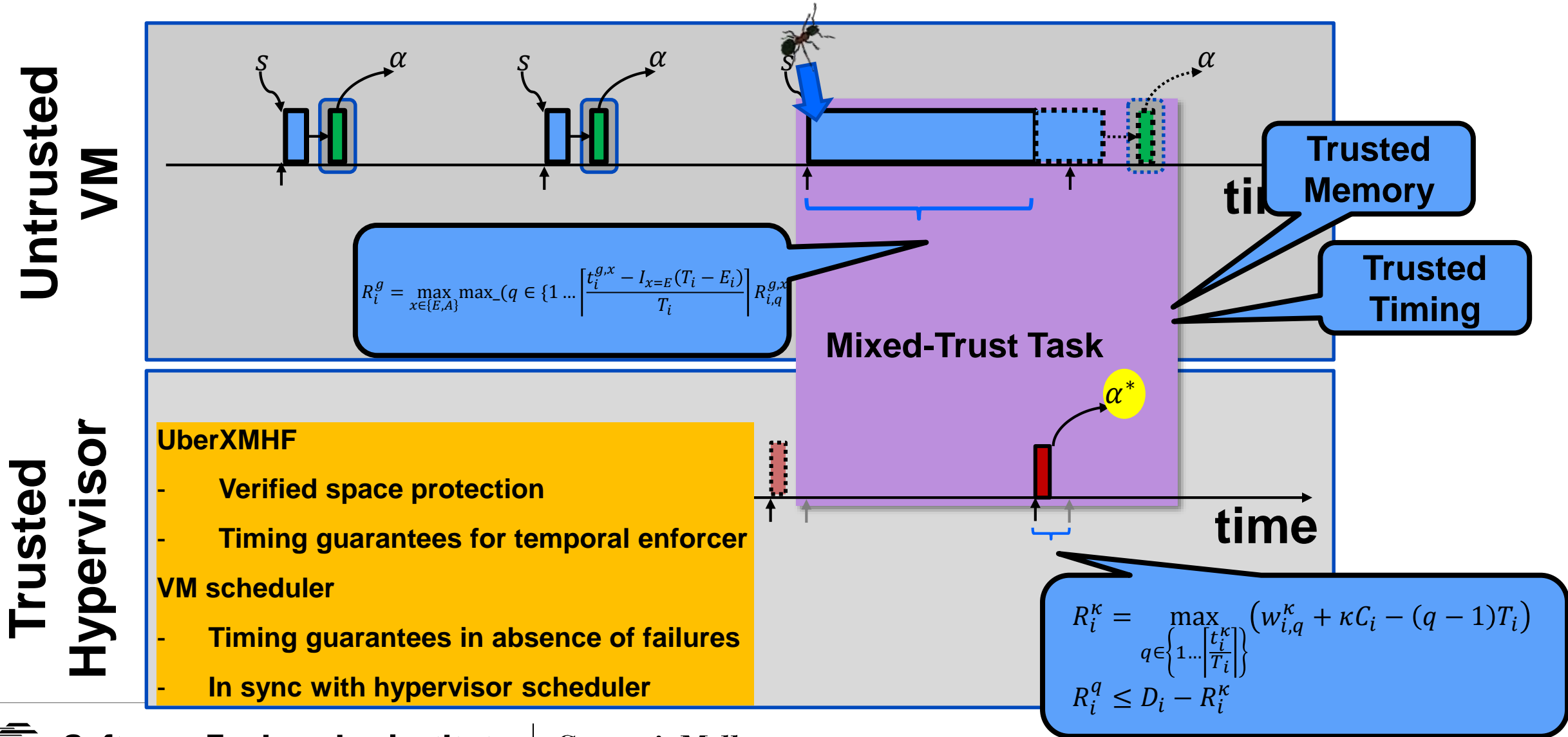
Periodic Execution Finish by Deadline



Real-Time Mixed-Trust Computation



Real-Time Mixed-Trust Computation



Concluding Remarks

Scaling up:

- Minimize Code to Verify
- Enforcers

Focus on key properties:

- Safety

Combined Relevant Scientific Domains

- Timing
- Logic
- Physics (Control)

Verification only effective if protected!

- Verified Protection: Hypervisor

