# AFRL and SEI: Past, Present, and Future

Dionisio de Niz
Technical Director and Principal Researcher
Assuring Cyber-Physical Systems / Software Solutions Division

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

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.



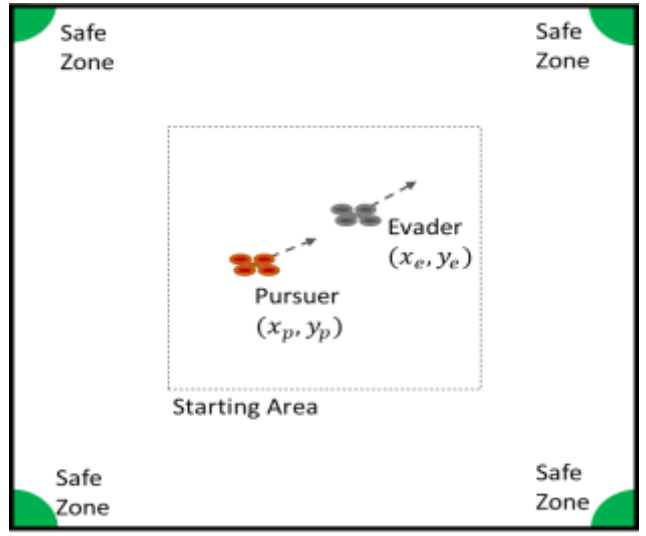
DM21-0489

# Summer of Innovation 2017

Statistical Model Checking for Missions

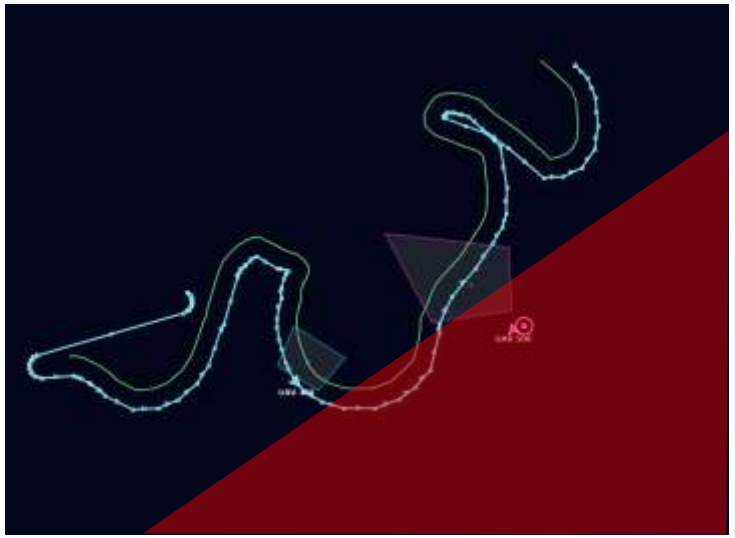
- Simulation in UxAS:
- Pursuer/Evader mission.
- The evader's goal is to reach one of the two "safe" zones located in the corners before the pursuer drone catches it.

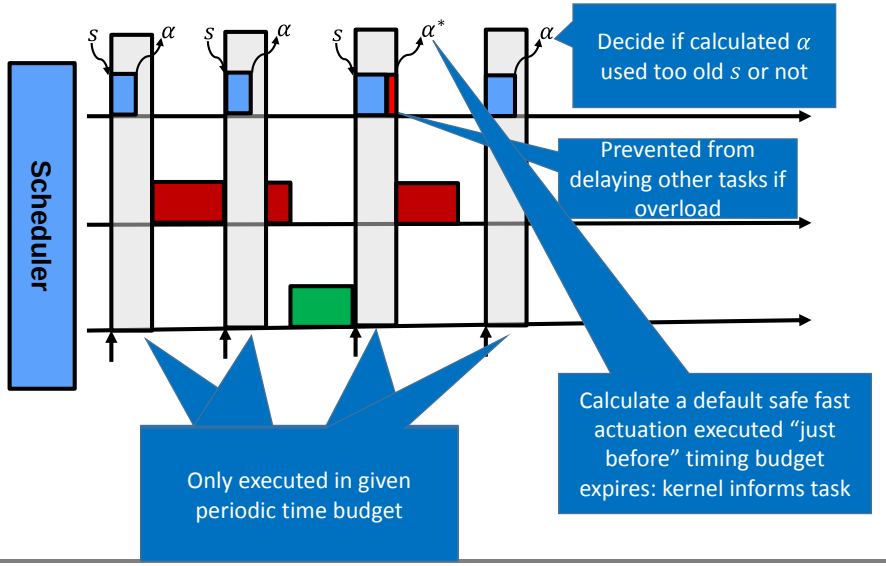
$$\hat{p} = \frac{1}{N}\sum_{i=1}^{N} I_{\mathcal{M}\vDash\Phi}(x_i)$$




Runtime Assurance

- Logic
  - No-fly zone at runtime
  - Enforcer monitors and modifies trajectory
- Timing
  - If program takes too long force loitering





# Teaming-Enabled Architectures for Manned-Unmanned Systems (TEAMS)

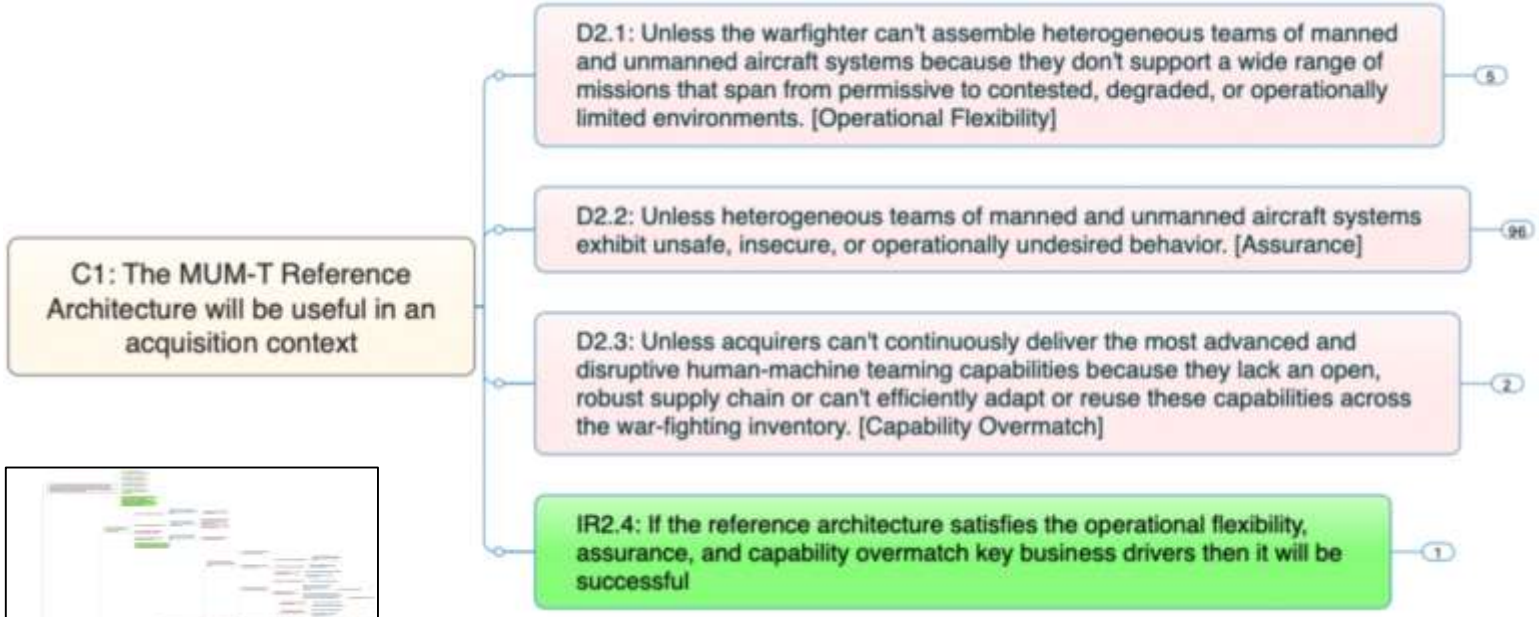
SEI provided support for technical risk reduction to the AFRL program office.

One form of risk reduction consisted on *technical excursions*, addressing topics important to AFRL but out of the scope for the prime contractor:

- Development of confidence arguments with the Method of Doubts (MOD) for the claims:
  - *the TEAMS Reference Architecture will be useful in an acquisition context*
  - *the TEAMS missions exhibit effective teamwork*
- Analysis and prototyping of runtime adaptation patterns in aircraft teaming

# TEAMS: Assurance Cases

- Key business drivers (KBD) expressed as doubts
- Shows (early) what evidence would ultimately be required to provide some level of confidence
  - Allows directing resources to obtain needed evidence
  - Helps determine if the KBDs will be met




Weinstock, C. & Wallnau, K. *Assuring Key Business Drivers and the Big Five in Teamwork.* CMU/SEI-2021-SR-010.

# Runtime Adaptation in Aircraft Teaming

Systems that operate in uncertain and unknown environment must be able to adapt to environment and systems changes to carry out their missions. Teams of aircraft are such kind of systems.

It's not possible to know a priori when this changes will occur and what the state will be at the time, so it is necessary for the system to detect these situations and decide how to adapt at run time.

In this technical excursion, the SEI

- evaluated the applicability of decentralized adaptation patterns to teams of aircraft:
- prototyped and evaluated the patterns in the DARPA CODE autonomy framework
- proposed changes to the TEAMS Reference Architecture to better support adaptation

Moreno, G. *Runtime Adaptation in Aircraft Teaming.* CMU/SEI-2021-SR-008.

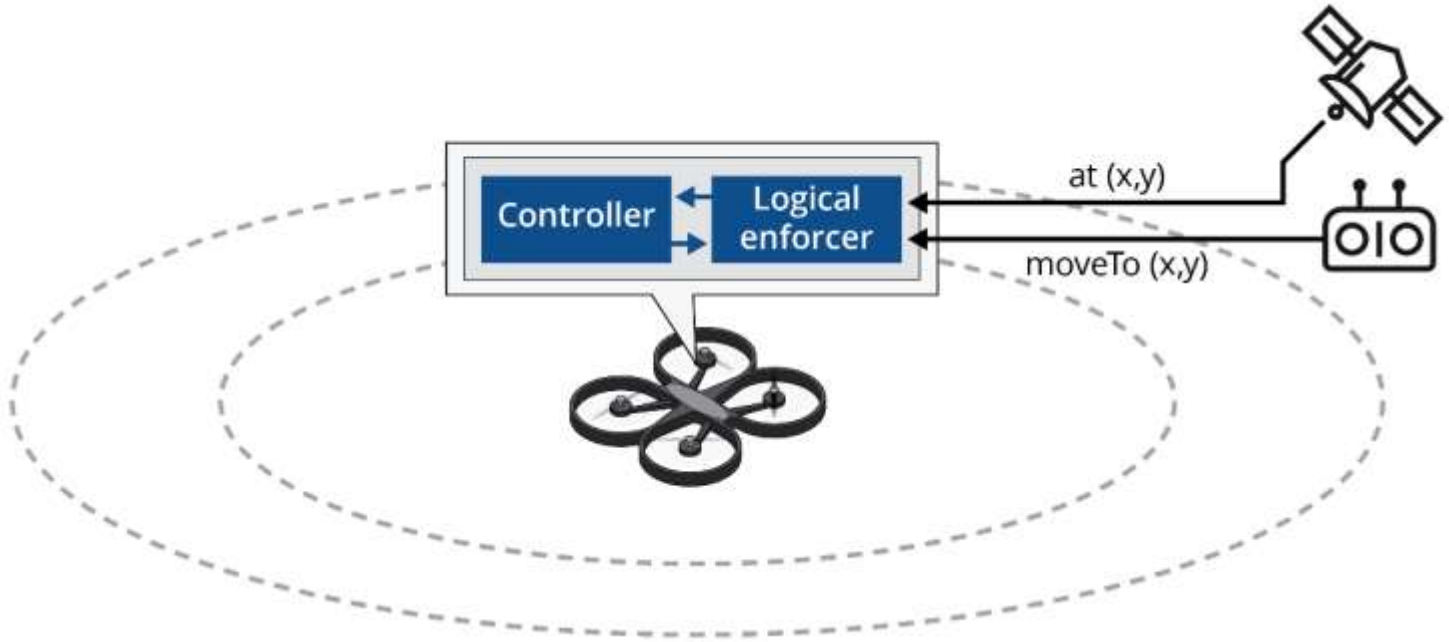
# Enforcement-based Verification of CPS

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

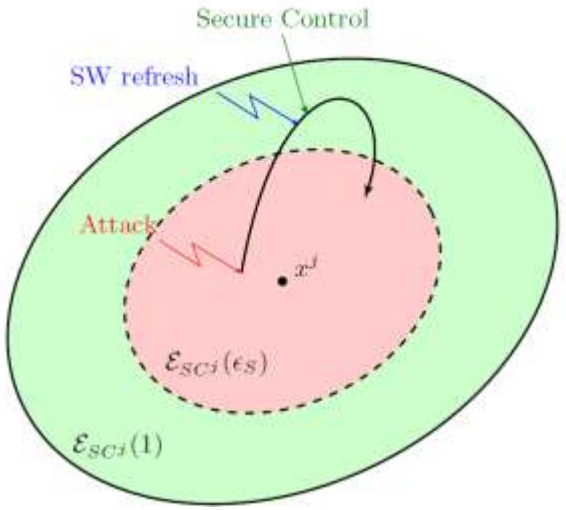
Protect enforcers against failures/attacks



# Verifying Physics (Control Theory)

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

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




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

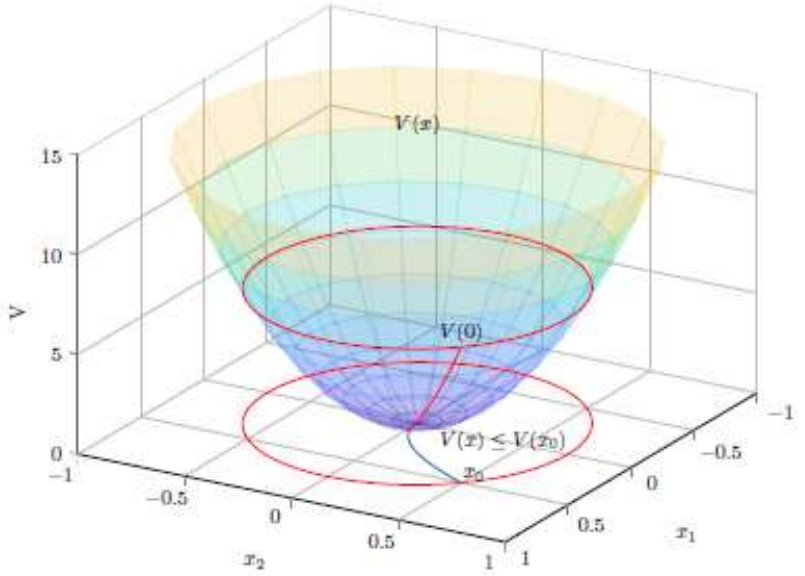
**Lyapunov Function**: $V_\varphi : \mathbb{R}^n \to \mathbb{R}$, $\mathcal{N}_{V_\varphi}(x_{eq}) \subseteq \mathcal{N}_\varphi(x_{eq})$,

$V_\phi(x_{eq}) = 0$ 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 $\epsilon > 0$,

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



# Analysis of Mission Progress Enforcing Unsafe Behavior

6 DOF $\Rightarrow$ 12 state variables

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

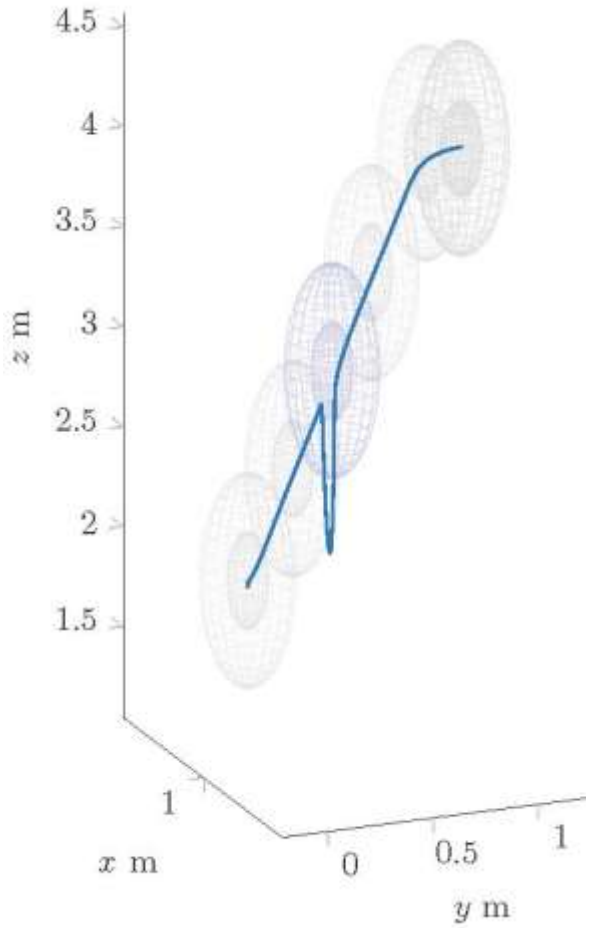
$$\ddot{p}_y = \sin\phi \frac{F}{m}$$

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

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

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

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




Linear design:

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

# Drone Experiment

# Are We Done Yet?

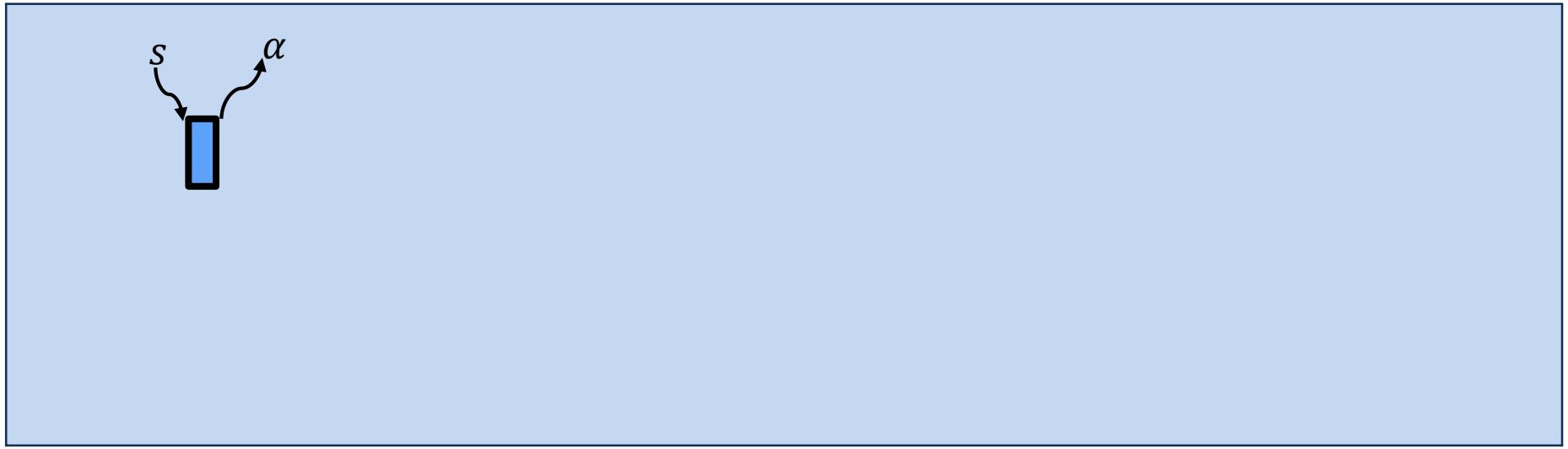
Scalable Verification

- Only verify safety-critical components
- Guarding unverified components

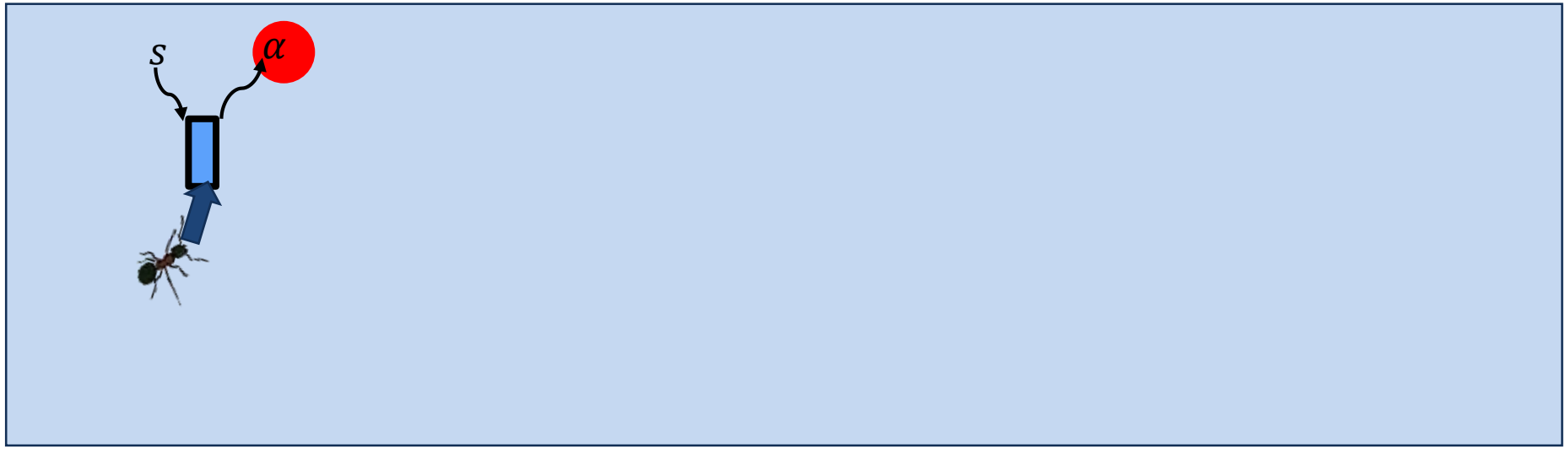
Trust

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

# Enforcing Unverified Components



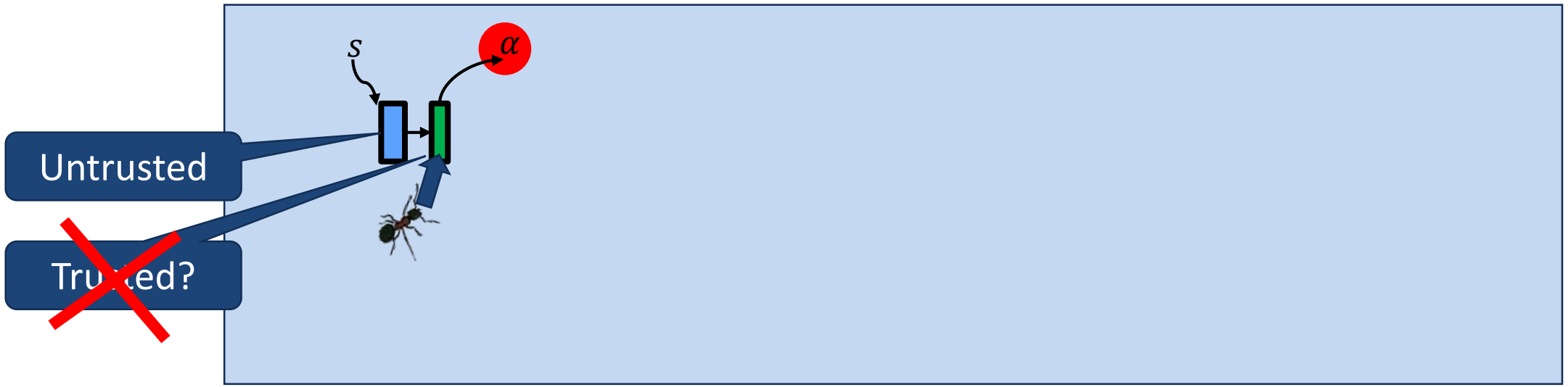
# Enforcing Unverified Components



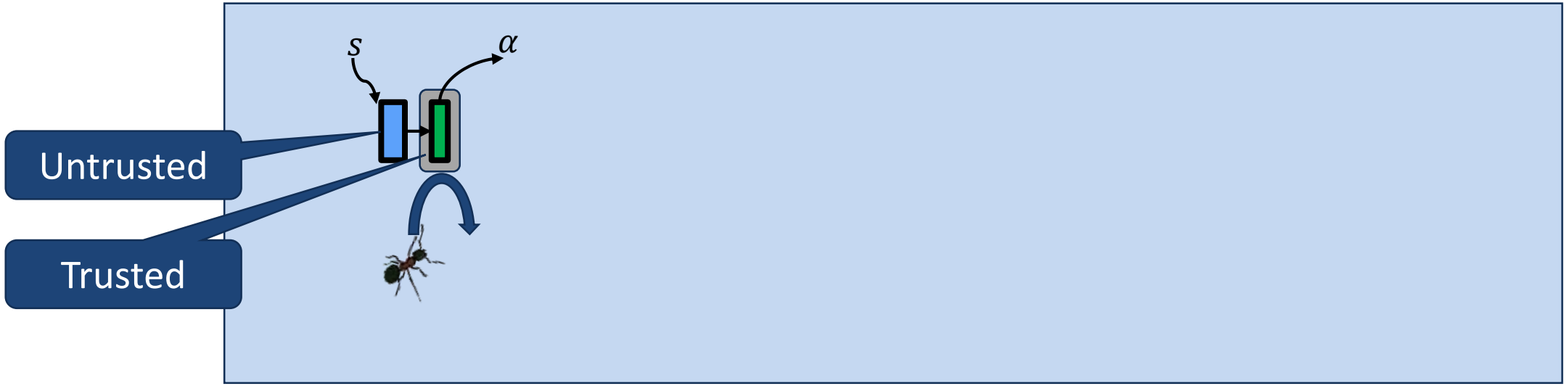

Ant illustration by Jan Gillbank, license by Creative Commons Attribution 3.0 Unported

# Enforcing Unverified Components

# But enforcer can be corrupted (bug or cyber attack)

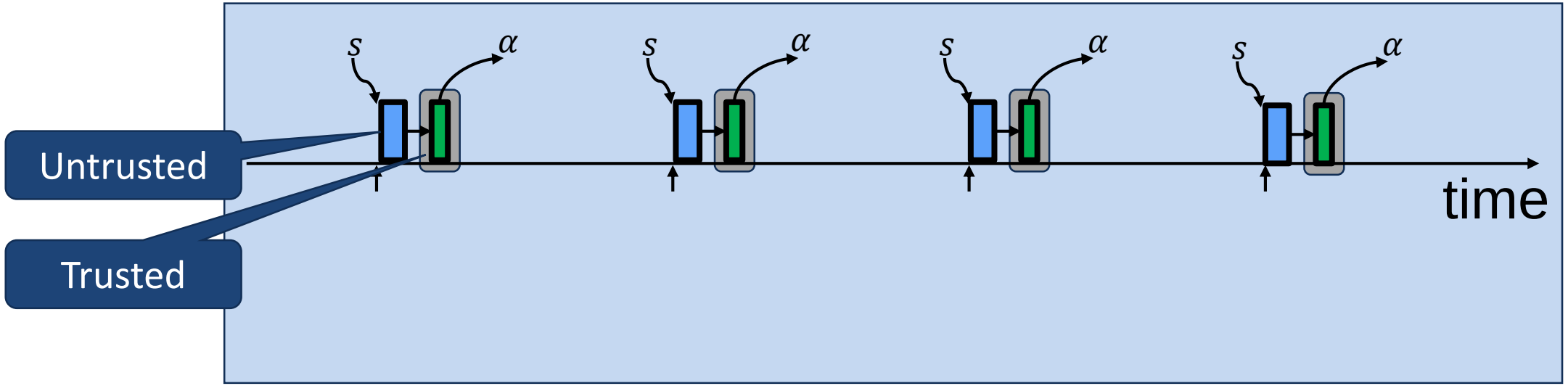


# Add Memory Protection

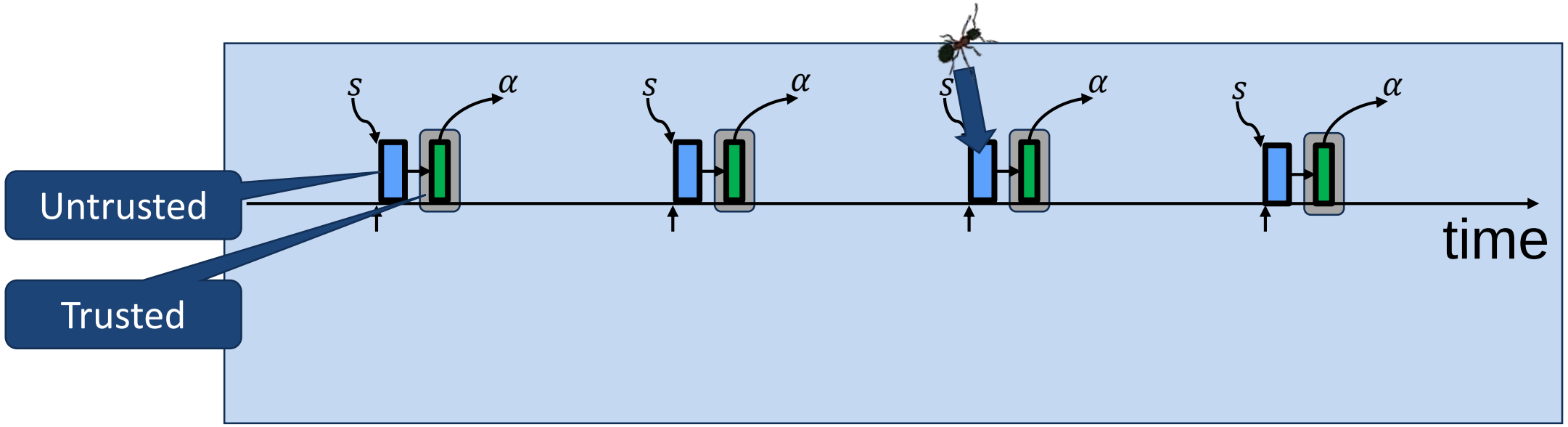



Trusted = Verified & Protected

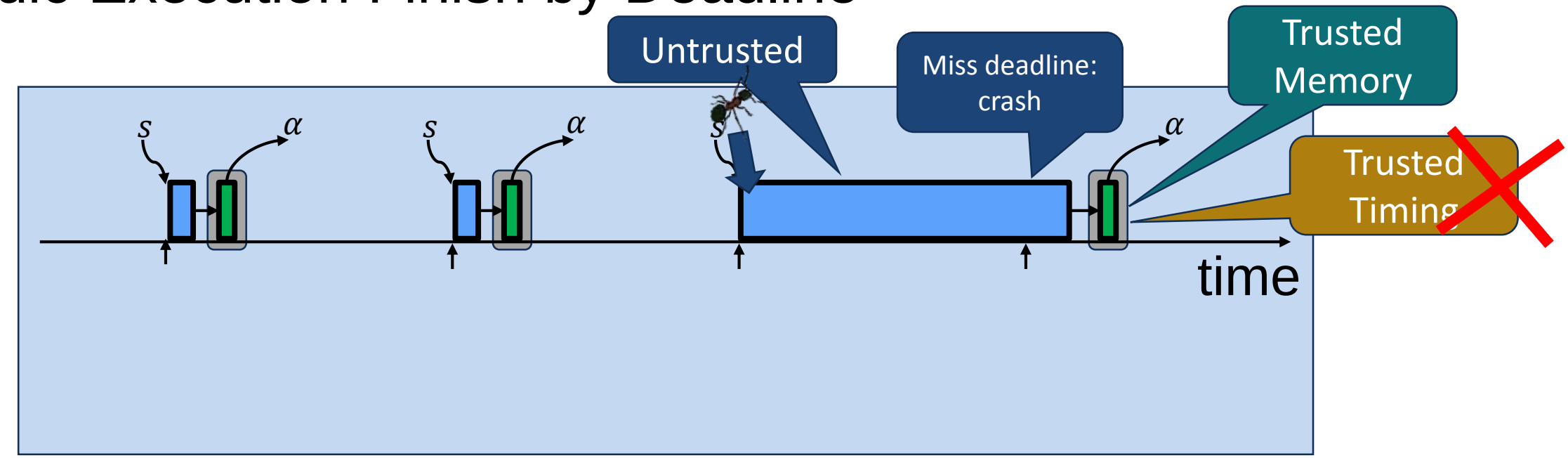
# Periodic Execution Must Finish by Deadline



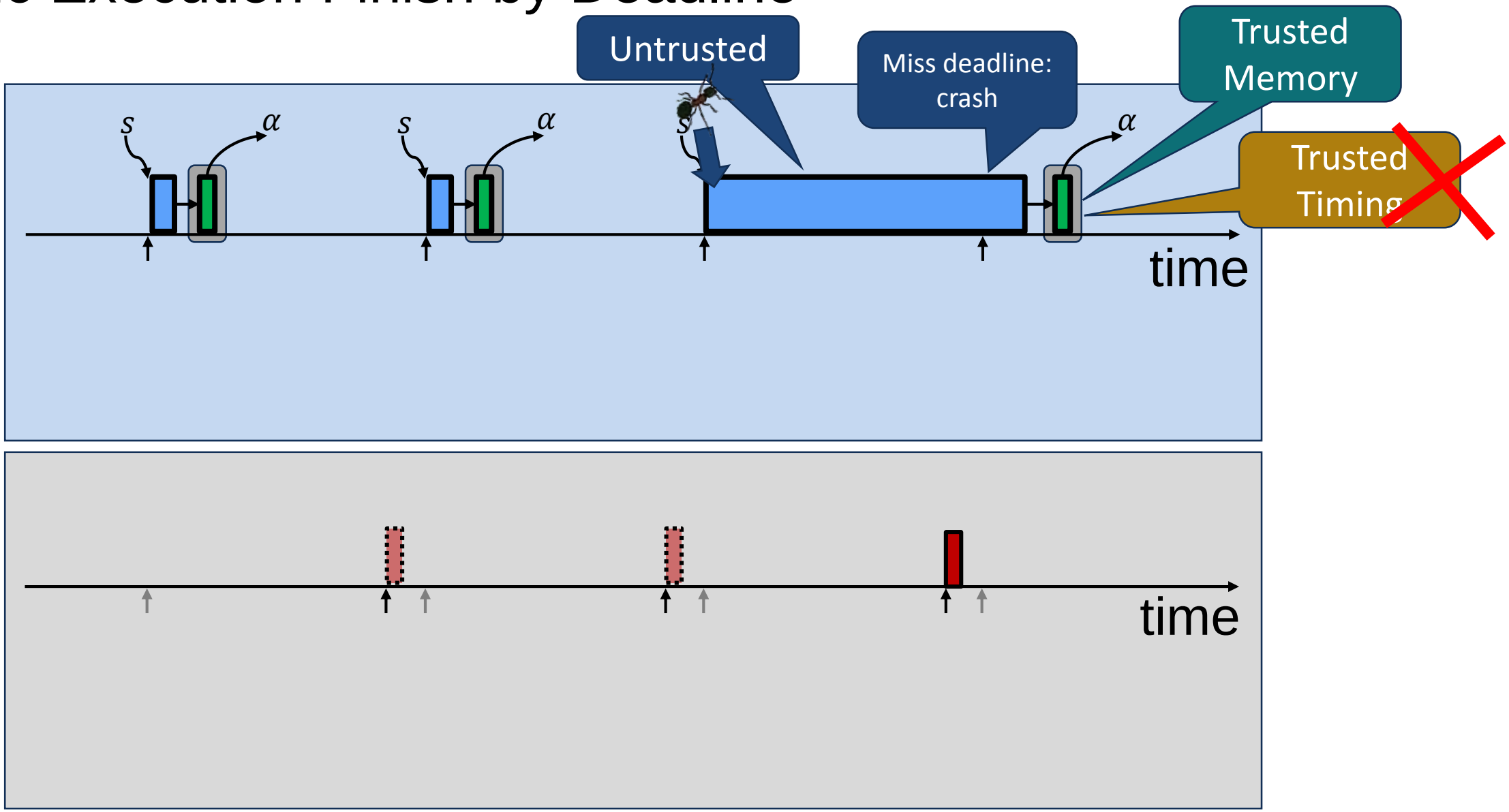
# Periodic Execution Must Finish by Deadline



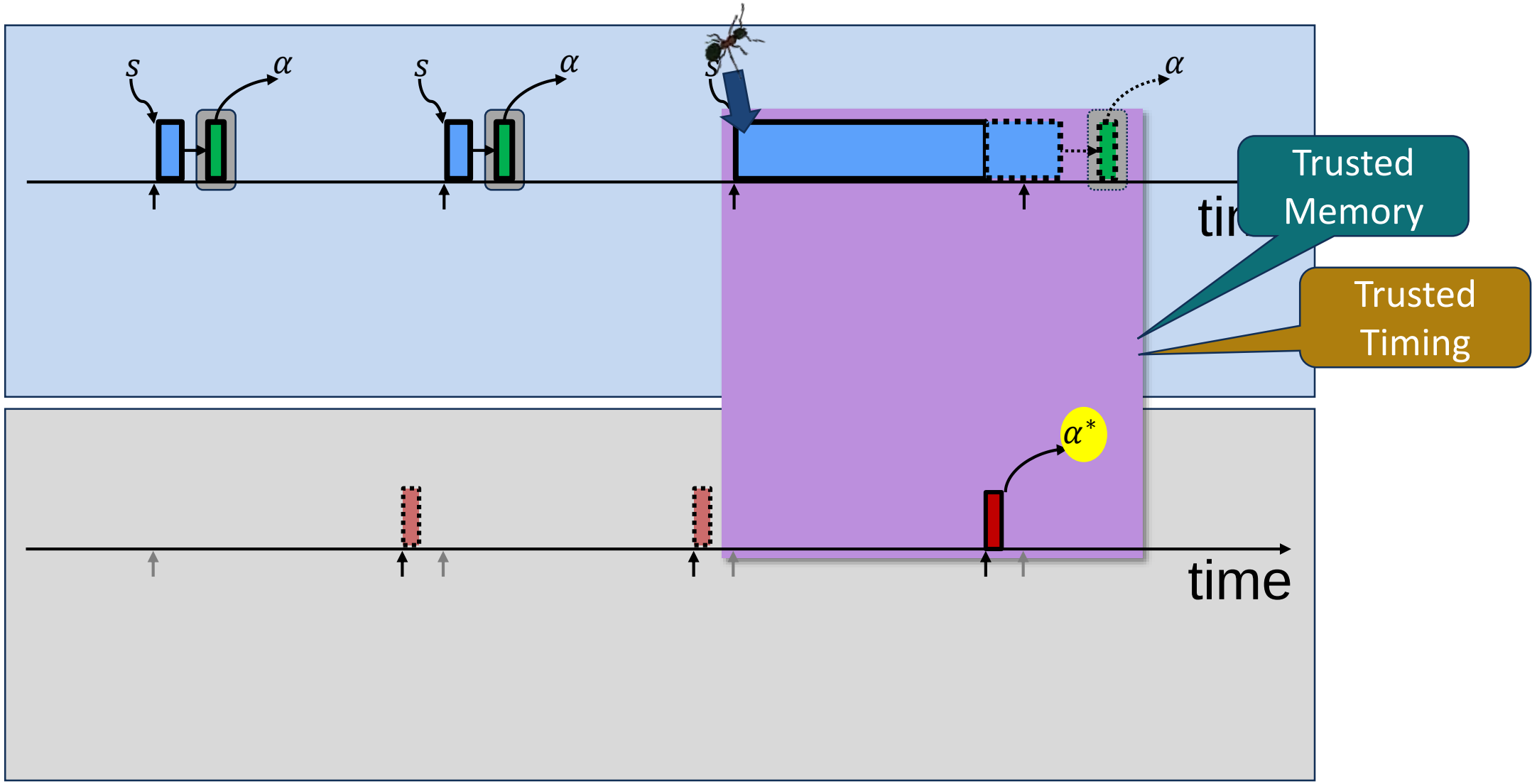
# Periodic Execution Finish by Deadline



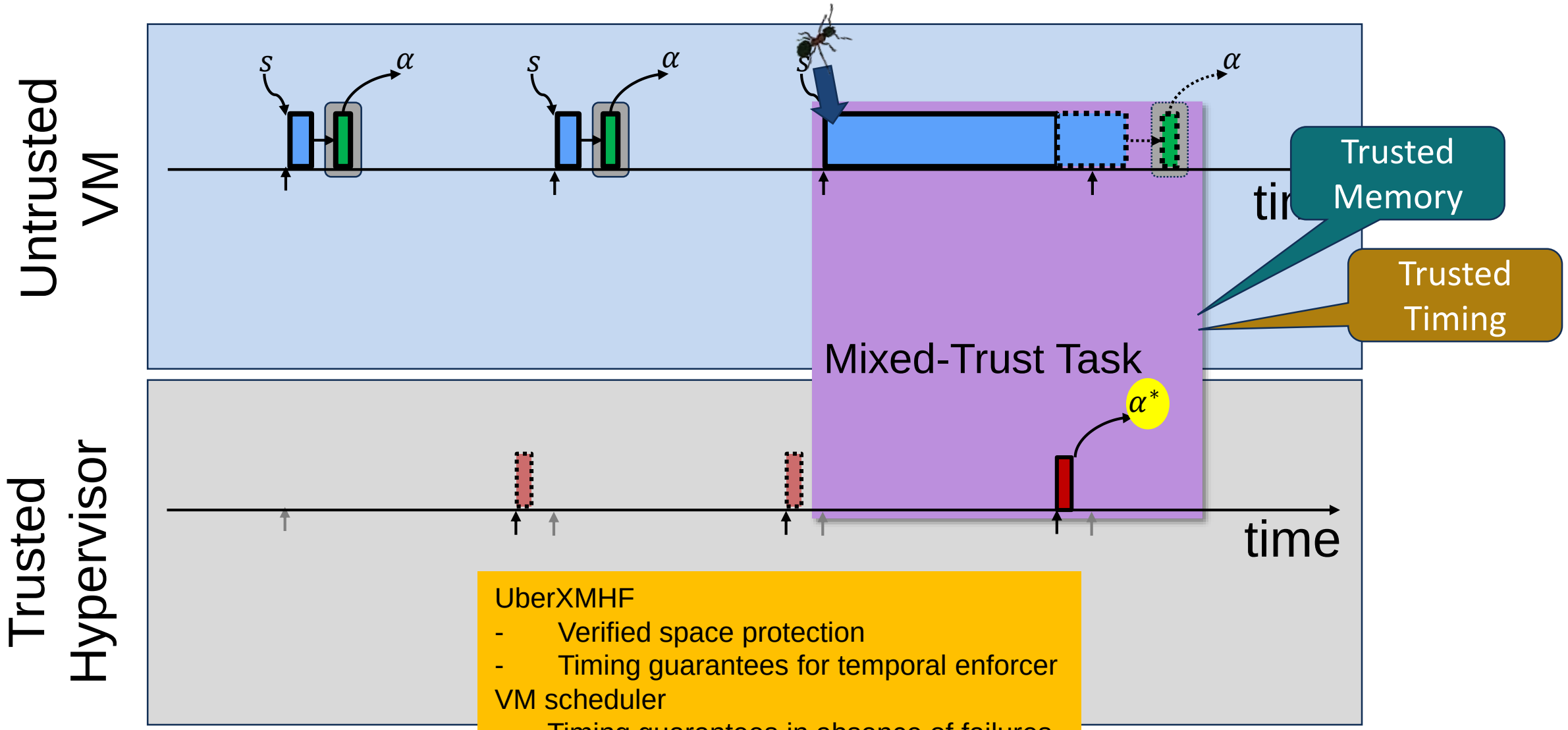
# Periodic Execution Finish by Deadline



# Periodic Execution Finish by Deadline



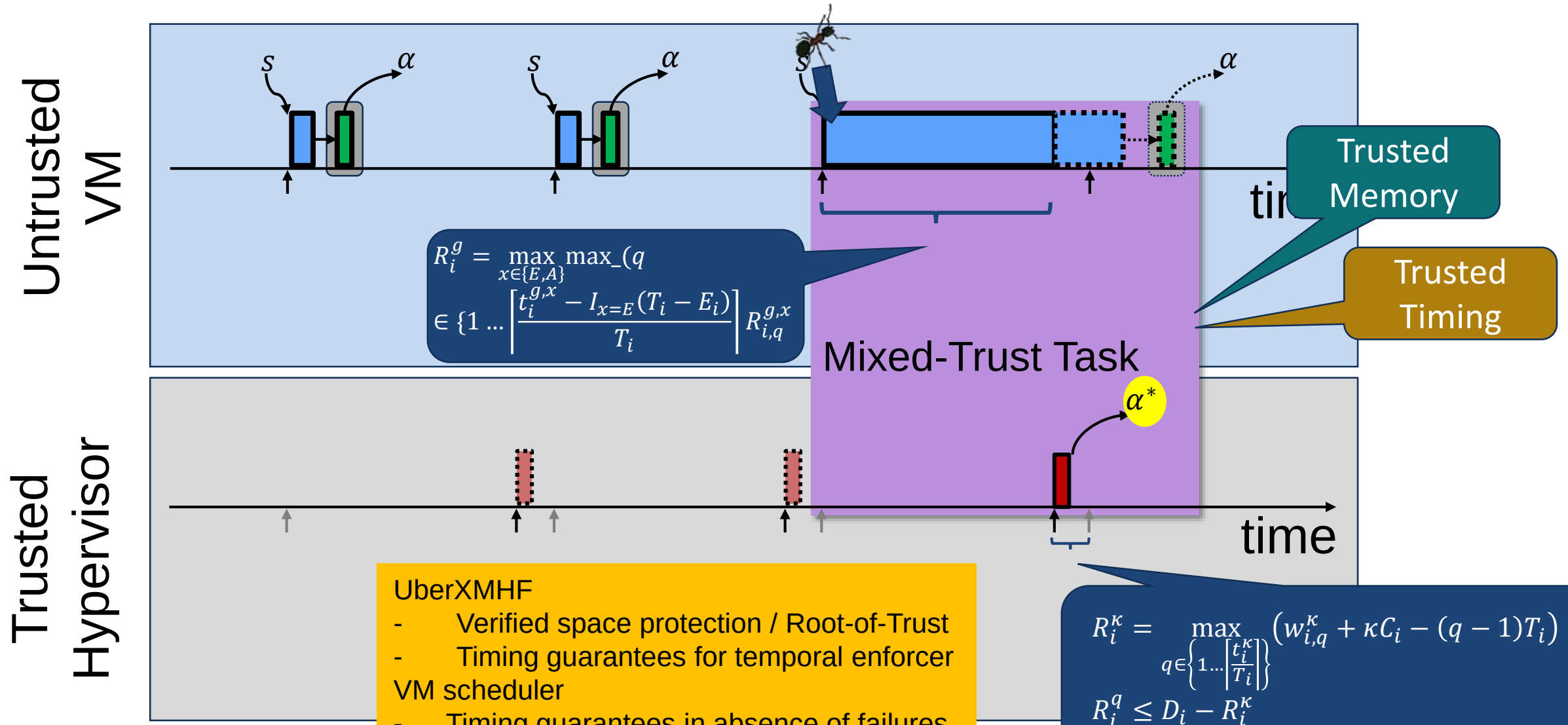
# Real-Time Mixed-Trust Computation




UberXMHF
- Verified space protection
- Timing guarantees for temporal enforcer

VM scheduler
- Timing guarantees in absence of failures
- In sync with hypervisor scheduler

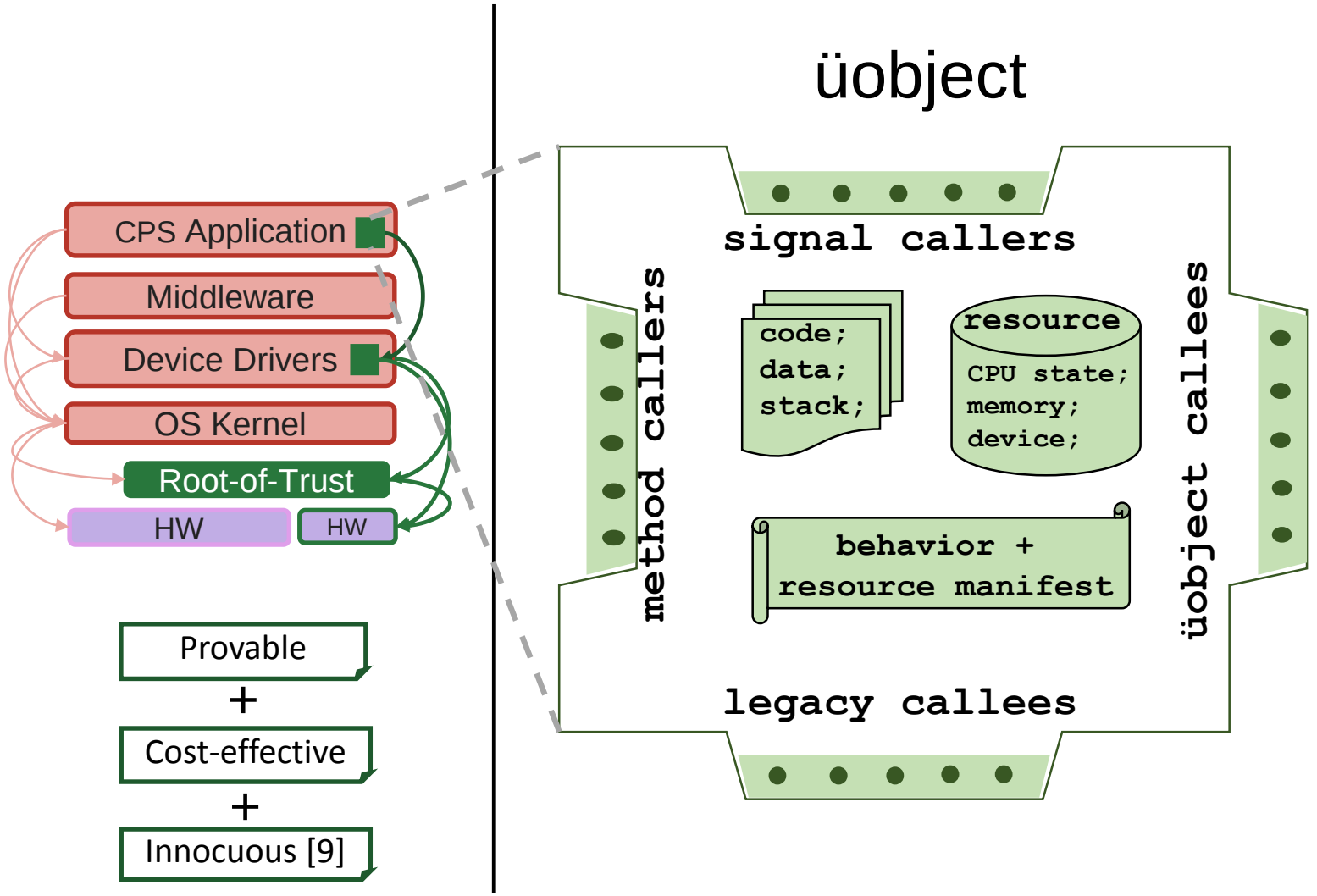
# Real-Time Mixed-Trust Computation




UberXMHF
- Verified space protection / Root-of-Trust
- Timing guarantees for temporal enforcer

VM scheduler
- Timing guarantees in absence of failures
- In sync with hypervisor scheduler

# 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

# Verified Protection:
# Discussion on Cyber-Security with Skyborg Team

Hypervisor (uberXMHF) for verified Root-of-Trust

- Host verified+trusted components
- Verified cyber-security protection

Skyborg contacts

- Maj Felix Abeyta (Program Manager, Advanced Aircraft Division)
- Matt Duquette, AFRL Skyborg team senior engineer
- Maj Rajan Pal

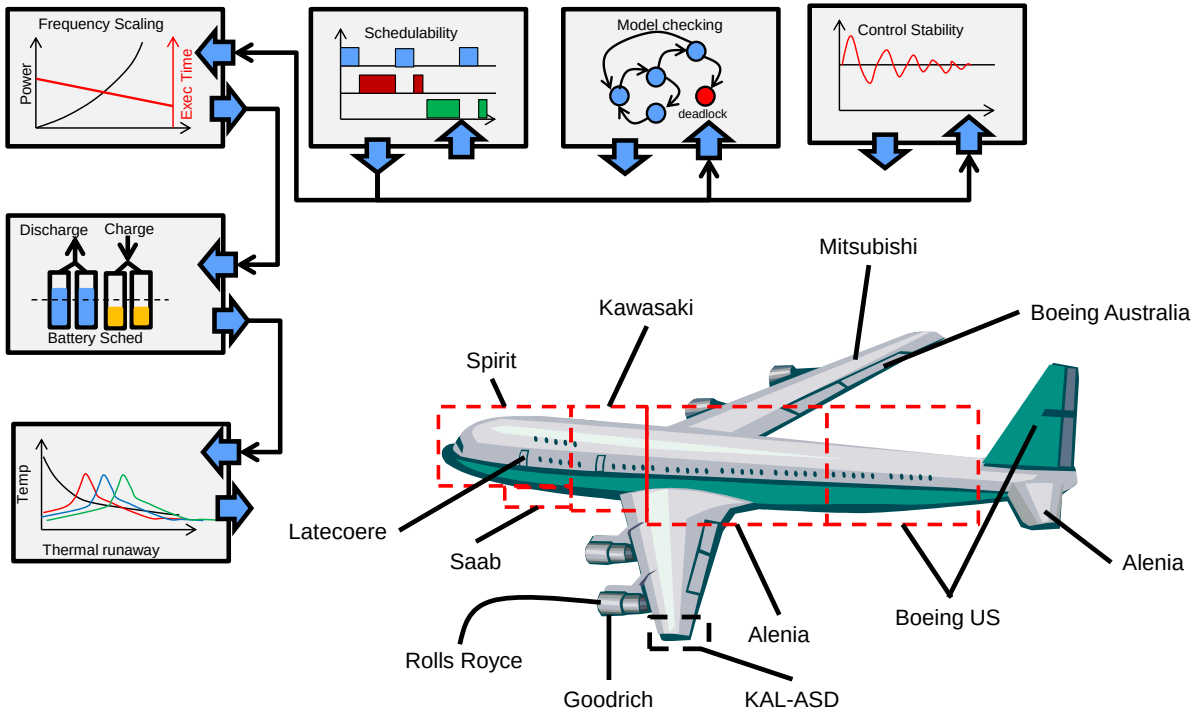
# Resilience in AI-Enabled Autonomy

**Problem:** The accuracy of AI components (AICs) can be affected by differences between training and operational contexts, compromising the system operation. Current mitigations focus on safety, blocking unsafe actions, but don't take action to recover from the degraded functionality, which is needed for the system to be resilient.

**Solution:** (i) add a meta-control loop to the system to monitor the accuracy of AICs and how they affect system properties, and if needed, modify its architecture using alternative versions of AICs to minimize functionality and performance degradation; (ii) guide run-time decisions and modifications with cyber-physical properties that describe the system required behavior in terms of time, logic and continuous physical variables.

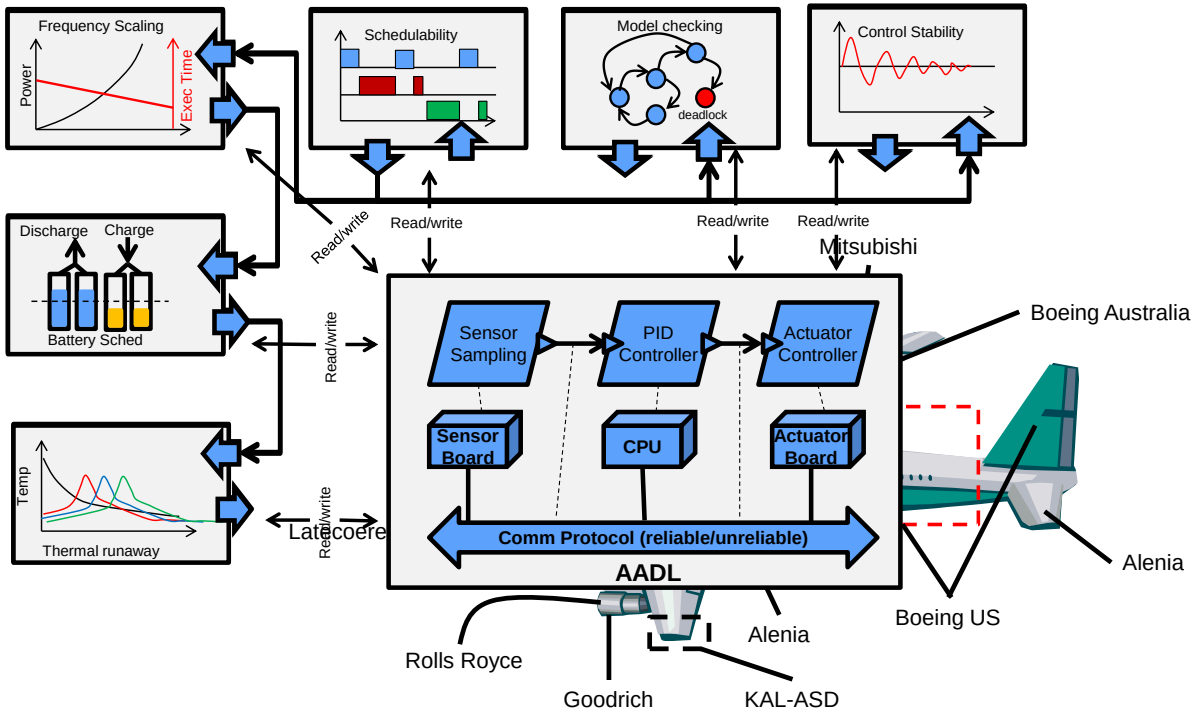
**Approach:** (1) use existing approaches to quantify confidence on AIC output, and monitor other properties of the system; (2) express required system properties in signal temporal logic (STL); (3) use run-time analysis of architecture model that uses (1) and (2) to select best component replacement; (4) integrate (1), (2), and (3) into meta-control loop; (5) use existing metric to measure improvement of resilience

# Architecture Analysis and Design Language (AADL) Virtual Integration




Early design errors are very expensive to remove if discovered late

# Architecture Analysis and Design Language (AADL) Virtual Integration




Early design errors are very expensive to remove if discovered late

Virtual Integration: Integrate model, analyze, then implement

Project with Apache Flight Management Computer Obsolescence (FMCO) software 16 architectural design issues were discovered avoiding estimated correction cost of $3M[1]





[1]Peter H. Feiler, John Hudak, and B. Craig Meyers. An Architecture-Centric Analysis of the Apache FMCO: Final Report. September 2012 SPECIAL REPORT. CMU/SEI-2012-SR-012.

# Concluding Remarks

History of CPS Verification Projects with AFRL

New Capabilities and Opportunities

- Scalable verification
- With Verified Protection

Opportunities for early design

- Virtual Integration

Opportunities for verification

- Early
- Down to the metal