

Research Review 2017

# Certifiable Distributed Runtime Assurance

Dionisio de Niz  
Principal Researcher

Copyright 2017 Carnegie Mellon University. All Rights Reserved.

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](mailto:permission@sei.cmu.edu).

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

DM17-0776

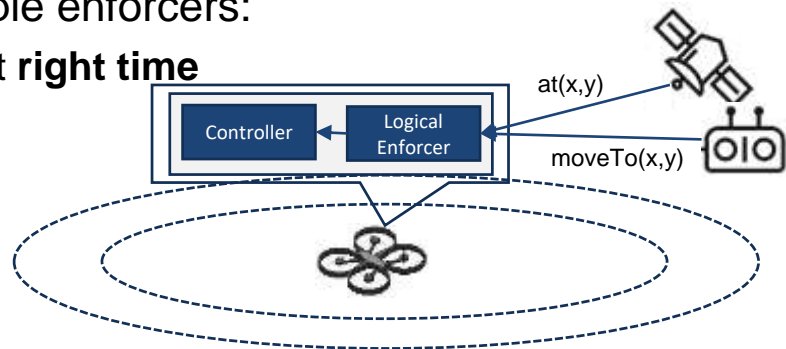
# Certifiable Distributed Runtime Assurance

**Challenge:** Assure Safety of Distributed Cyber-Physical Systems

- Unpredictable Algorithms (Machine Learning)
- Multi-Vehicle (distributed) coordinating to achieve mission

**Solution:**

- Add **simpler (verifiable)** runtime enforcer to make algorithms predictable
- Formally: specify, verify, and compose multiple enforcers:
  - Enforcer **intercepts/replaces** unsafe action at **right time**



# Formal Periodic Model: Representing Time-Aware Logic

## State of the system: values of variables

**State** variables:  $V_S$

**Action** variables:  $V_\Sigma$

Variable values from **domain**:  $D$

**Location** -- e.g.,  $(x, y)$  position  
**Movement** (move-to  $(x, y)$  position)  
**Domain** specific variables

**System state**  $\equiv$  assignment of values to state variables:  $s: V_S \mapsto D \in S$

**Action**  $\equiv$  assignment of values to action variables:  $\alpha: V_\Sigma \mapsto D$

**Behavior**  $\equiv$  state transitions given actuation every period  $P: R_P(\alpha) \subseteq S \times S$

Next state given action:  $R_P(\alpha, s) = \{s' \mid (s, s') \in R_P(\alpha)\}$

Add **values to quantify**  
position & move-to position

Account for **time &**  
actuations

Property to verify subset of all possible states:  $\phi \subseteq S$

Verify representative subset of ALL states  
 $(x, y)$  position within region

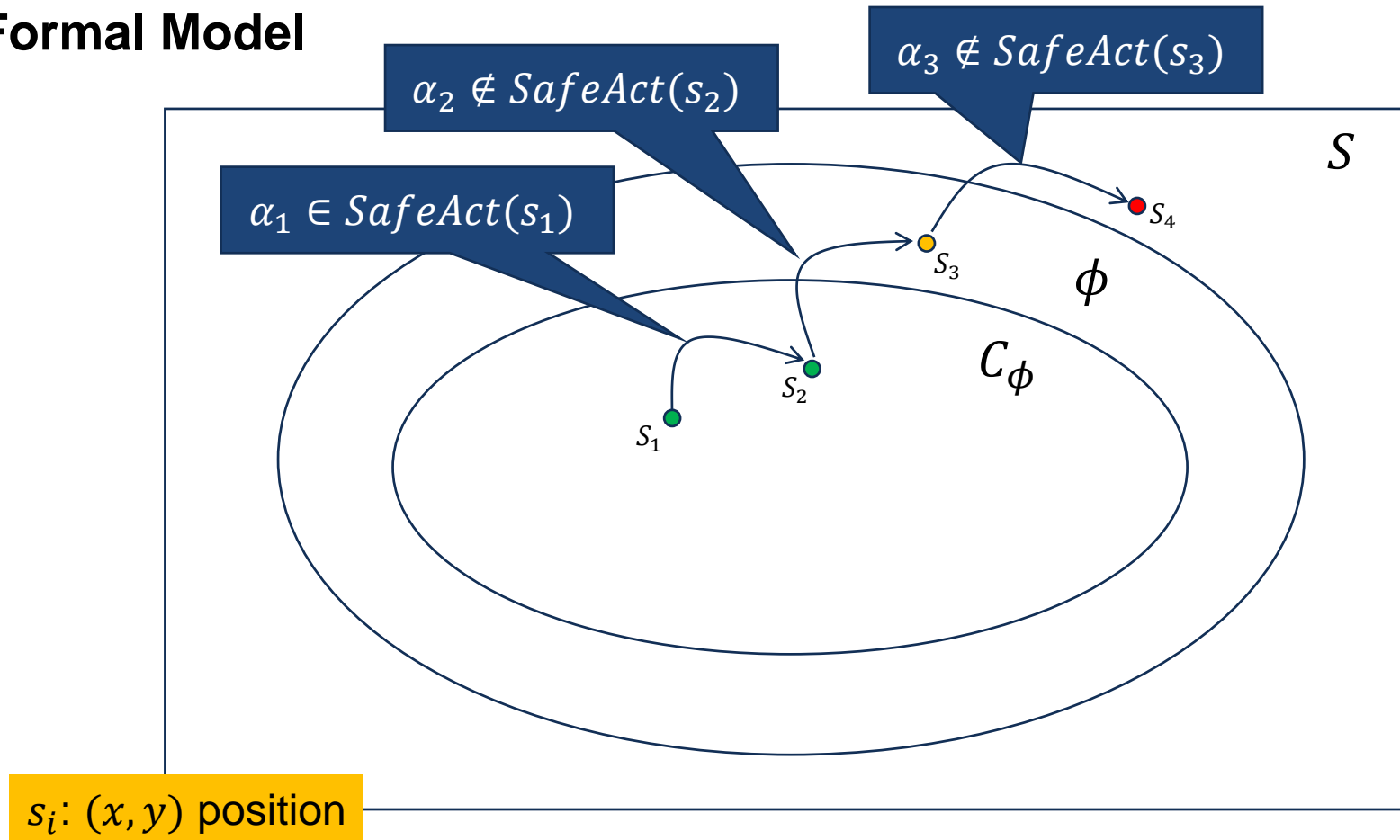
**Enforceable** state:  $C_\phi \subseteq \phi \wedge C_\phi = \{s \mid \exists \alpha \in \Sigma: R_P(\alpha, s) \in C_\phi\}$

Enforcement Mechanism  
 $(x, y)$  still prevent getting out

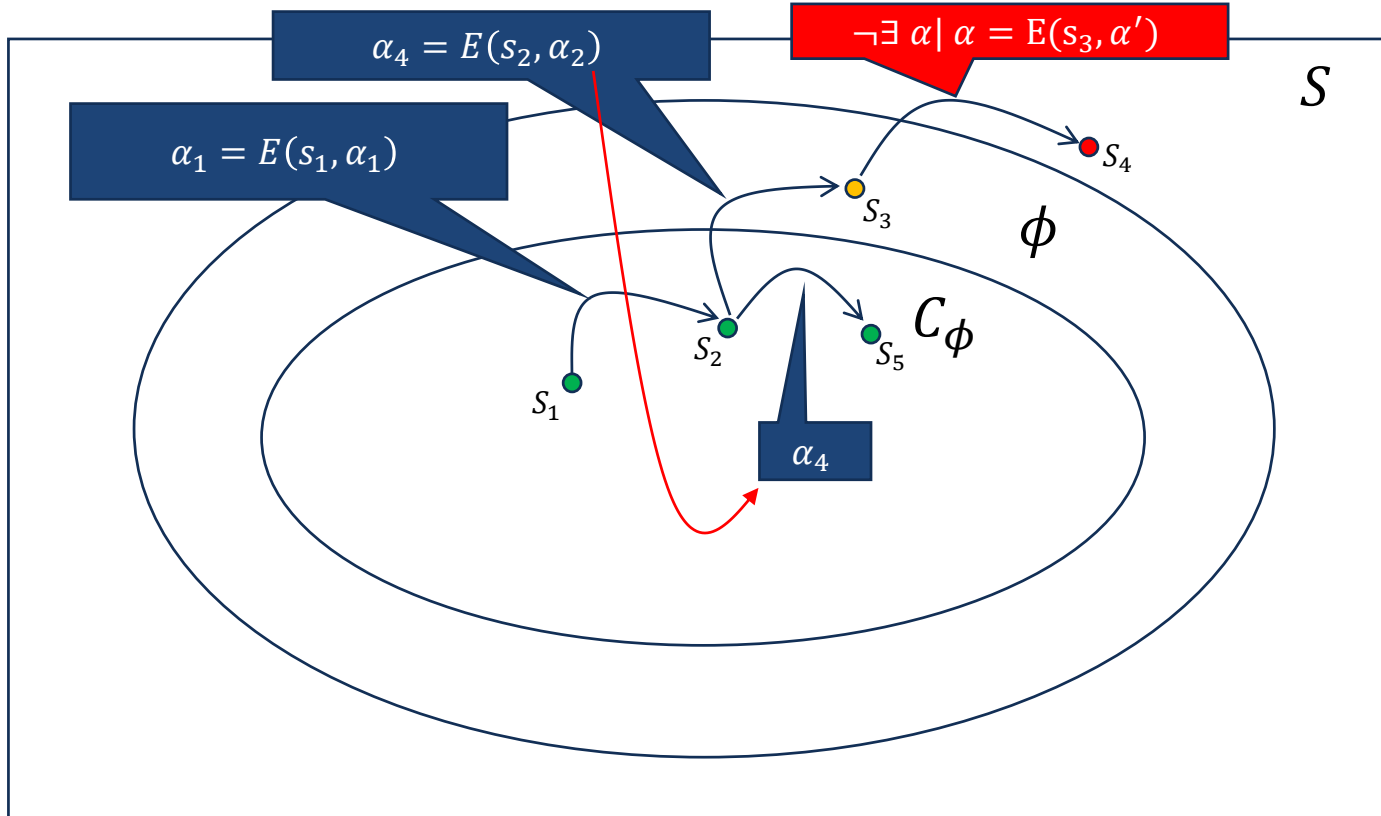
**Safe actuation** :  $SafeAct(s) = \{\alpha \mid R_P(\alpha, s) \in C_\phi\}$

Safe actuation *AHEAD* of enforcement

# Formal Model



# Enforcer $E(s, \alpha): \alpha \in \text{SafeAct}(s) ? \alpha : \alpha' \in \text{SafeAct}(s)$



# Composing Enforcers

Enforcer Details:  $E: (P, C_\phi, \mu, U)$

- $\forall s \in C_\phi: \mu(s) \subseteq \text{SafeAct}(s)$
- $U$ : utility

Composition without conflict

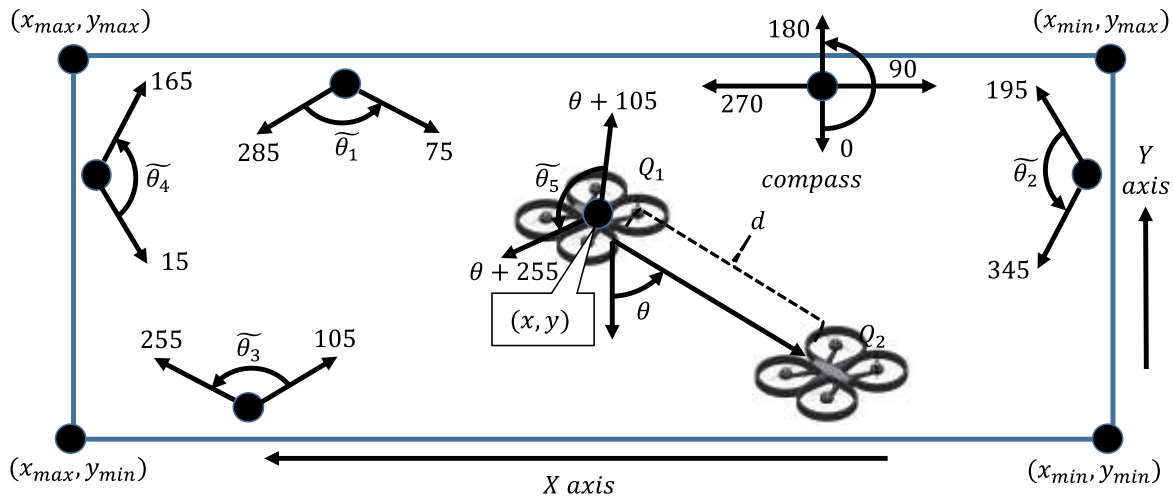
- $E_1: (P_1, C_{\phi_1}, \mu_1, U_1)$
- $E_2: (P_2, C_{\phi_2}, \mu_2, U_2)$
- $\mu_{1,2}: \mu_1 \cap \mu_2$

Conflicting: Priority:

- $\mu_{1,2}: \mu_1 \cap \mu_2 \neq \emptyset ? \mu_1 \cap \mu_2 : \mu_1$

Conflicting: Utility

- $\mu_{1,2}: \mu_1 \cap \mu_2 \neq \emptyset ? \text{argmax}_{\alpha \in \mu_1 \cap \mu_2} \sum U_i(s, \alpha') : \text{argmax}_{\alpha \in \mu_1} \sum U_i(s, \alpha')$



# Are We Done Yet?

## Timing Assumption:

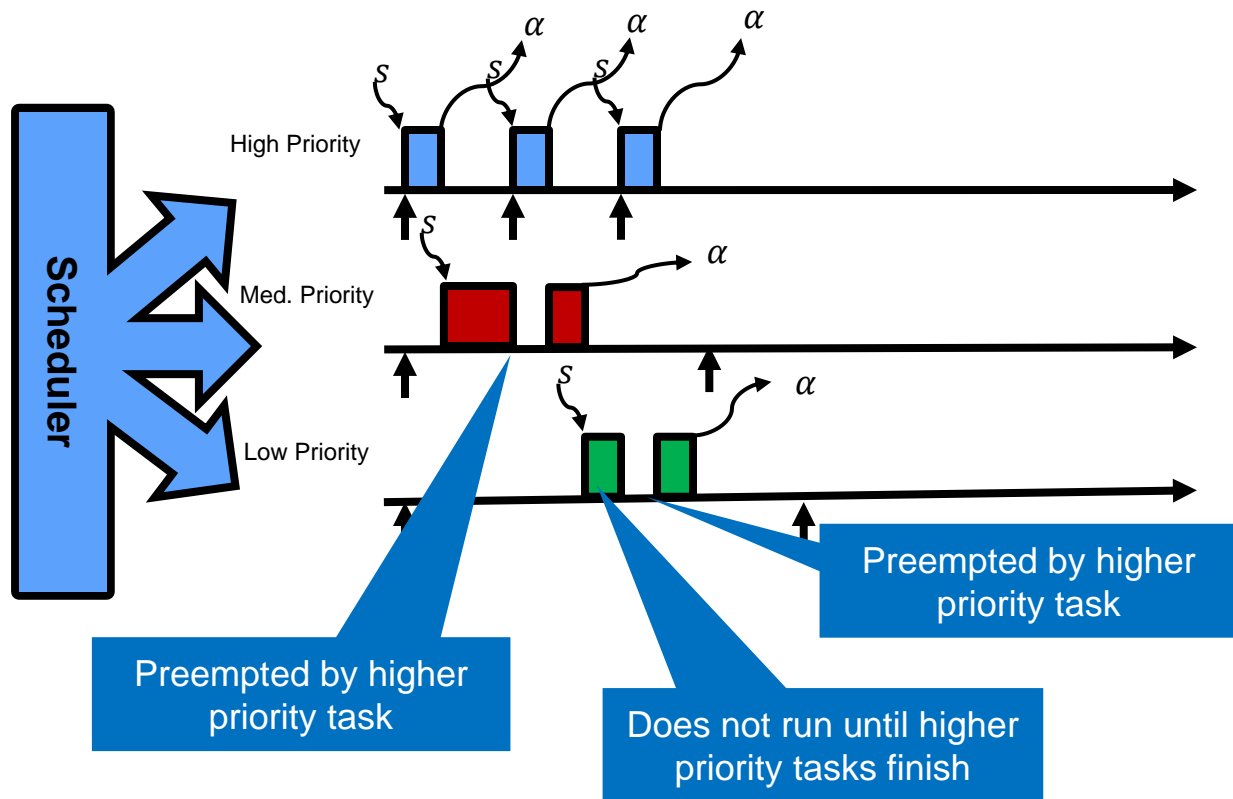
- Unverified software finishes execution and enforcer evaluates output every  $P$  period.
- Software is guaranteed to finish executing by the next period (schedulable)
  - Unverified software executes for less than its Worst-Case Execution Time (WCET)
  - Other software running also executes for less than its WCET
  - Schedulability analysis successful

## What can go wrong?

- Unverified software executes **A BIT** longer than WCET
  - Can make other software miss deadlines: late actions with old sensing
- Unverified software executes **A LOT** longer than WCET
  - Makes other miss deadline
  - Does **NOT** produce an output that can be evaluate by enforcer: late action + old sensing
    - **Inertia** takes it to **unsafe state**

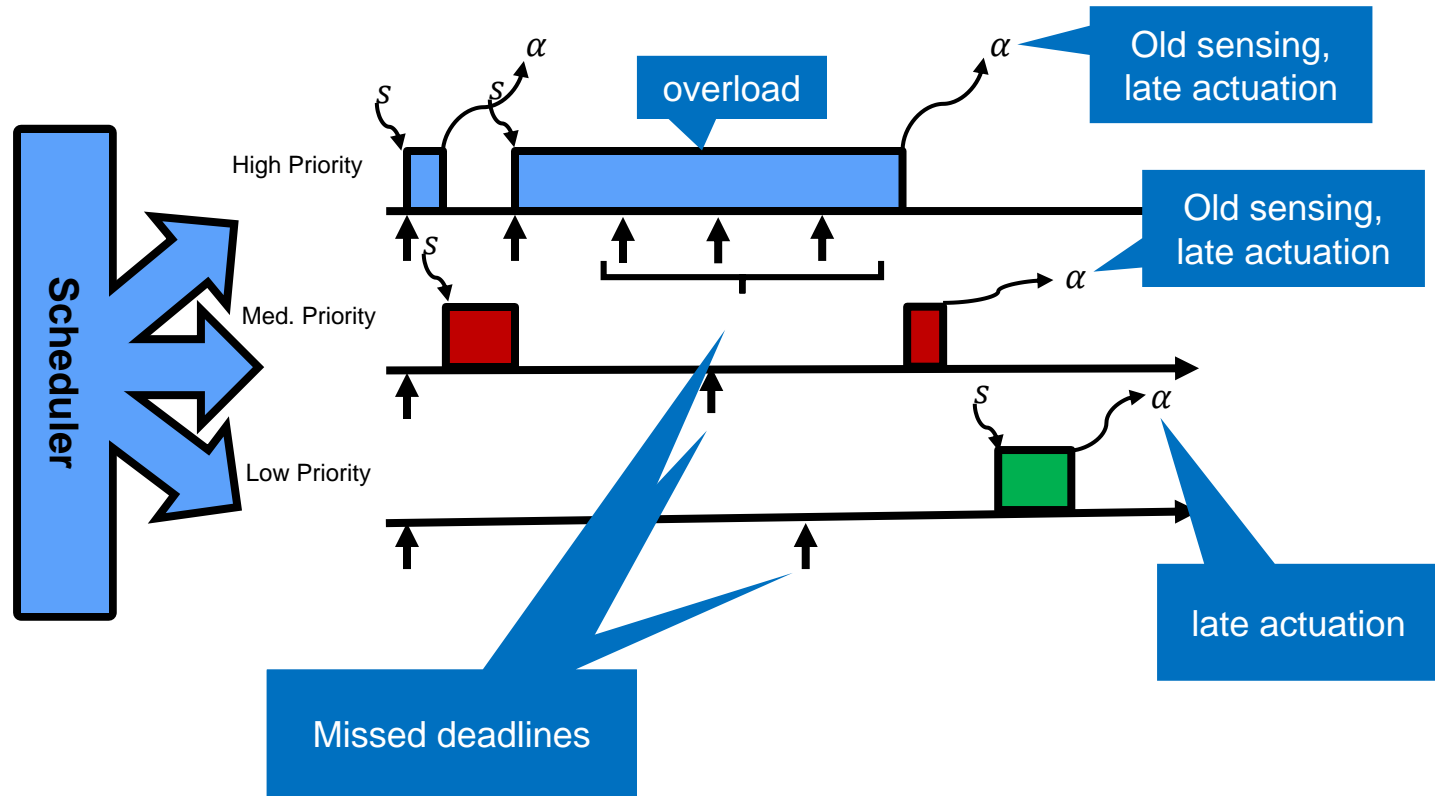


# Primer: Fixed-Priority Scheduling + Rate Monotonic



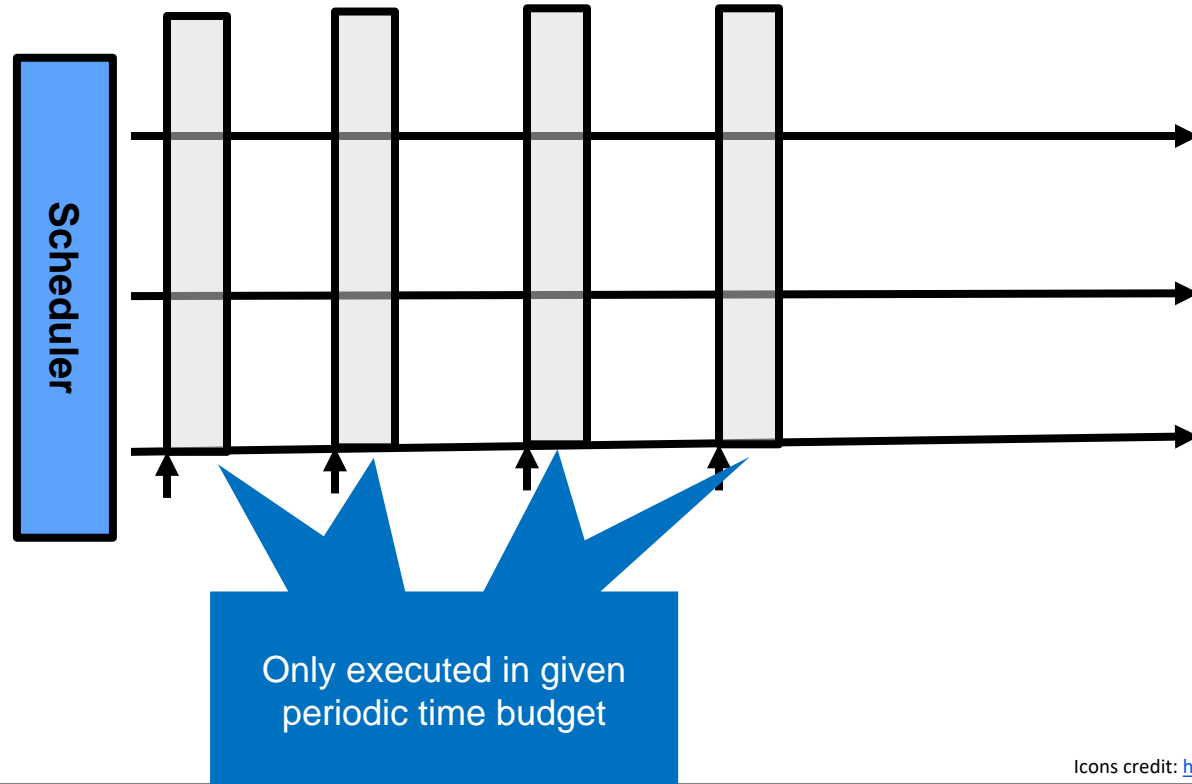
Icons credit: <http://www.doublejdesign.co.uk>

# Overload -> Old Sensed Data + Late Actuation

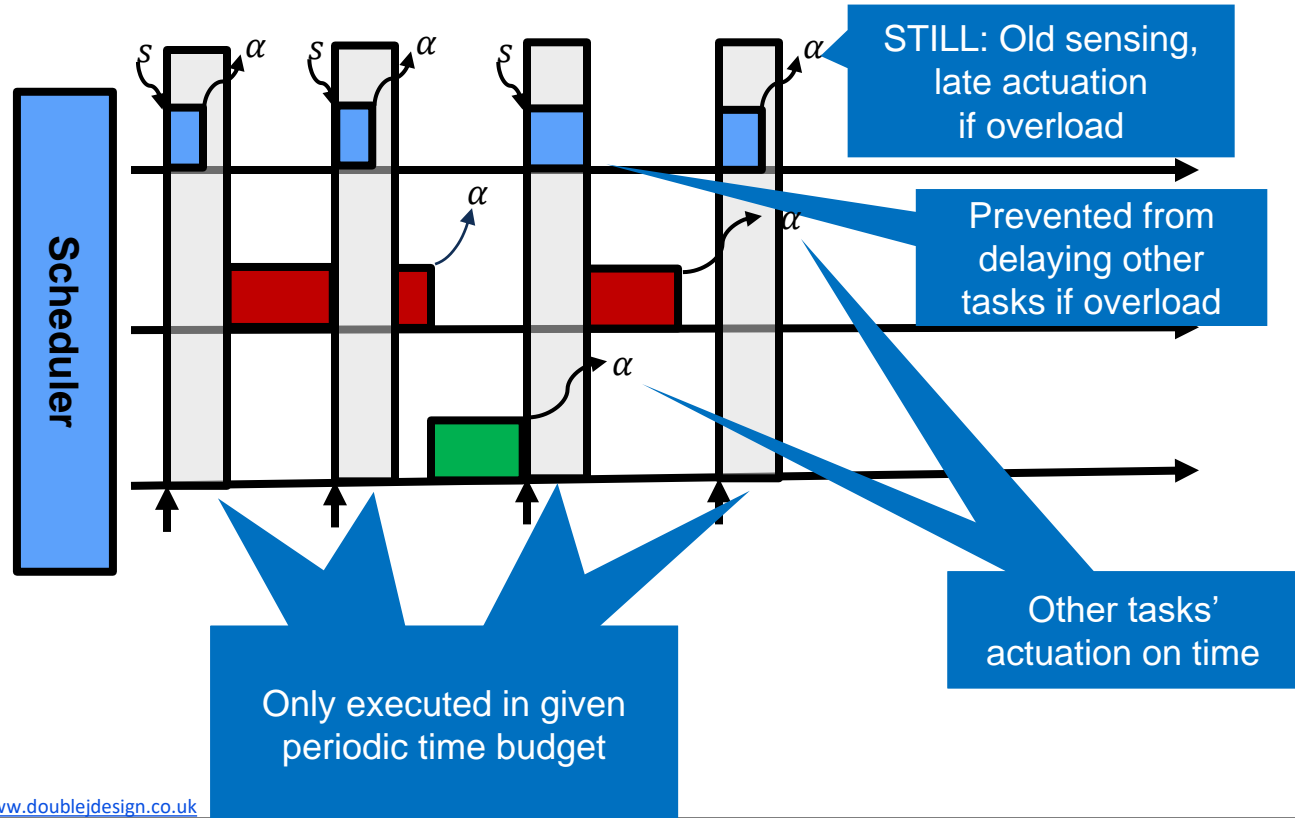


Icons credit: <http://www.doublejdesign.co.uk>

# Solution: Enforce Timing Budgets (Timing Enforcement)

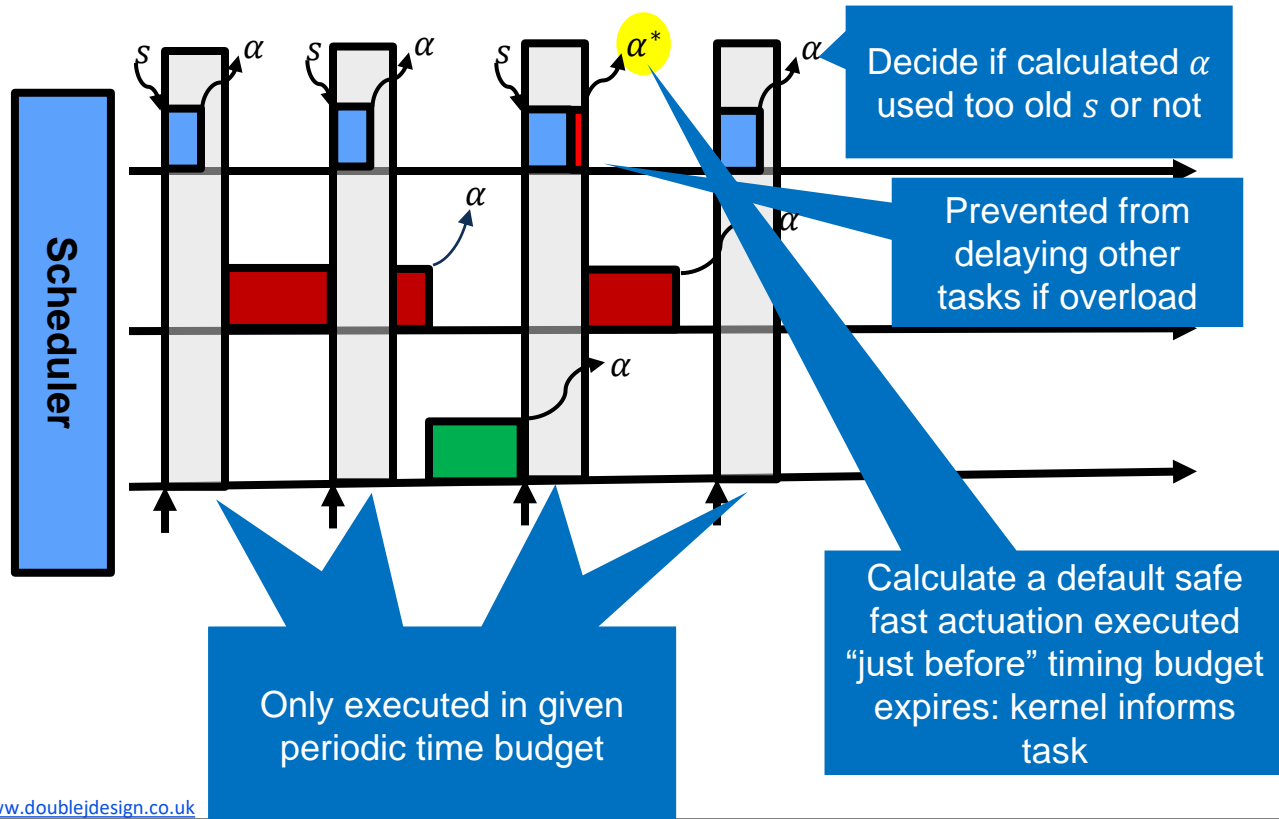


# Solution Step 1: Enforce Timing Budgets (Timing Enforcement)



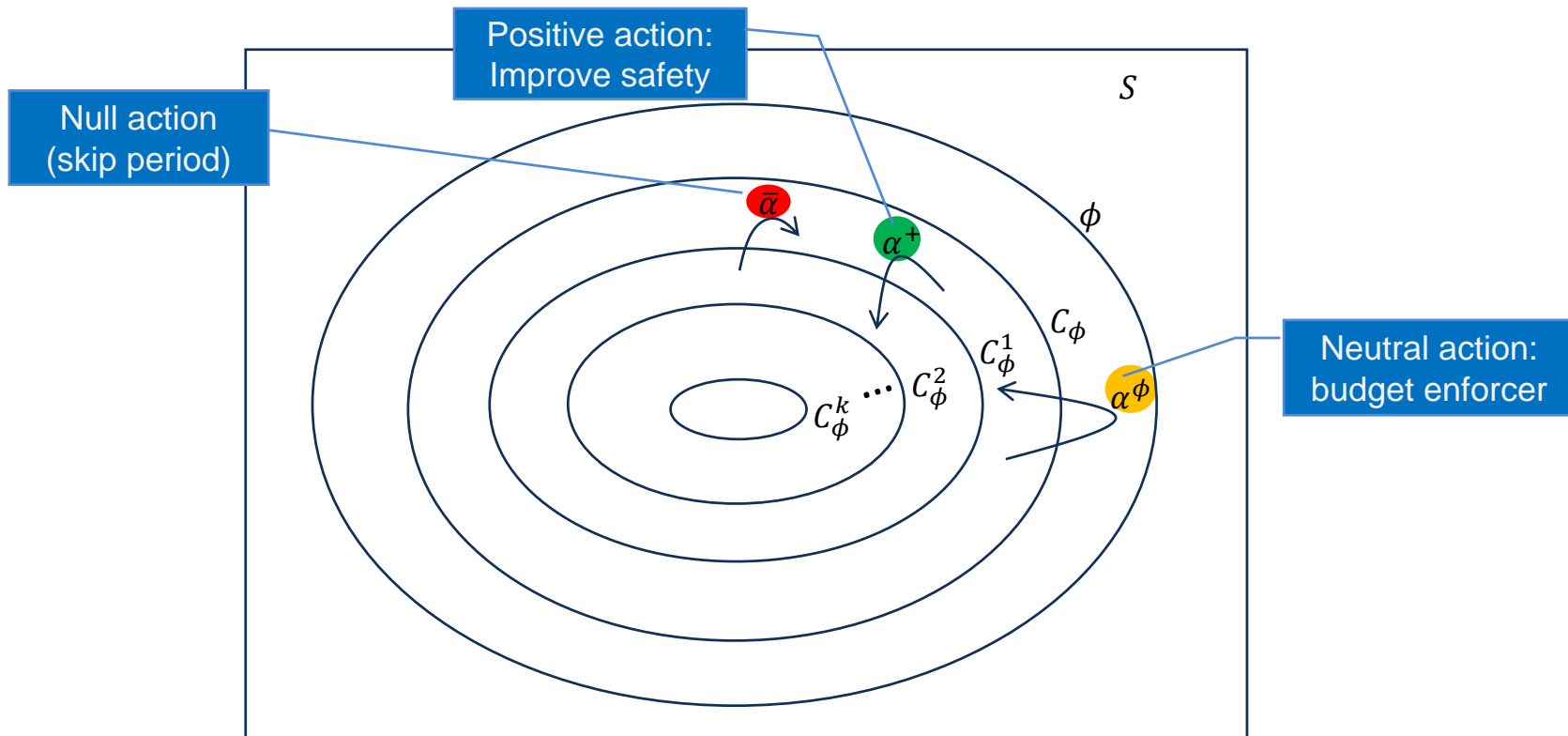
Icons credit: <http://www.doublejdesign.co.uk>

# Solution Step 2: Safe Actuation on Timing Enforcement

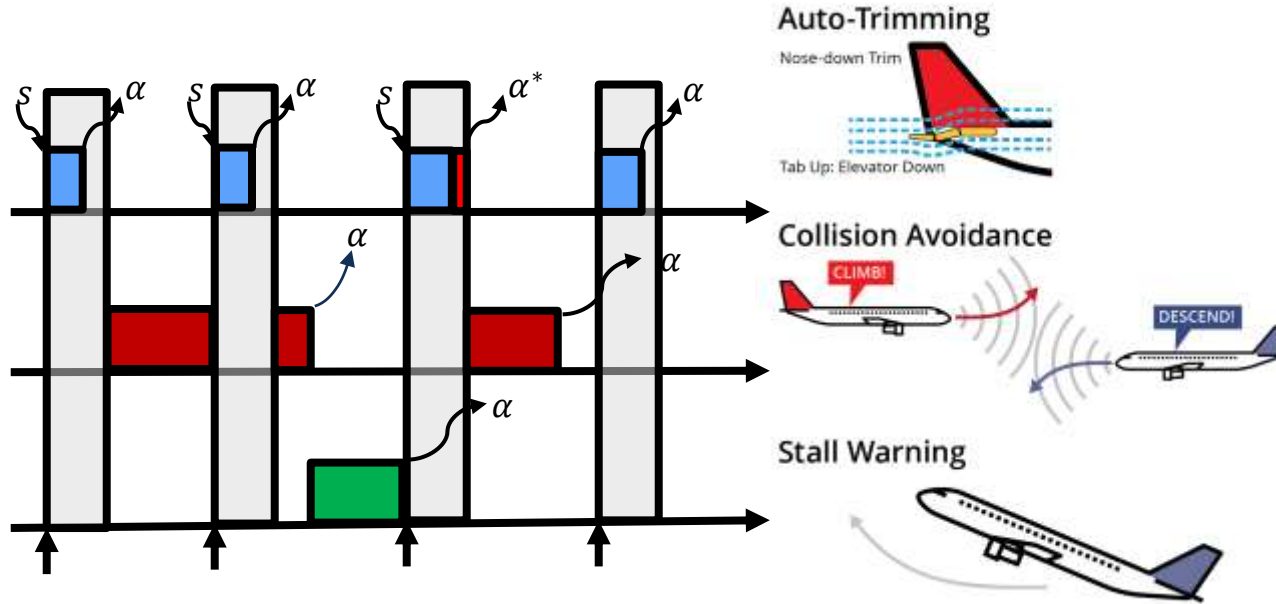


Icons credit: <http://www.doublejdesign.co.uk>

# Scheduling Resilience: Tolerance To Miss Deadlines

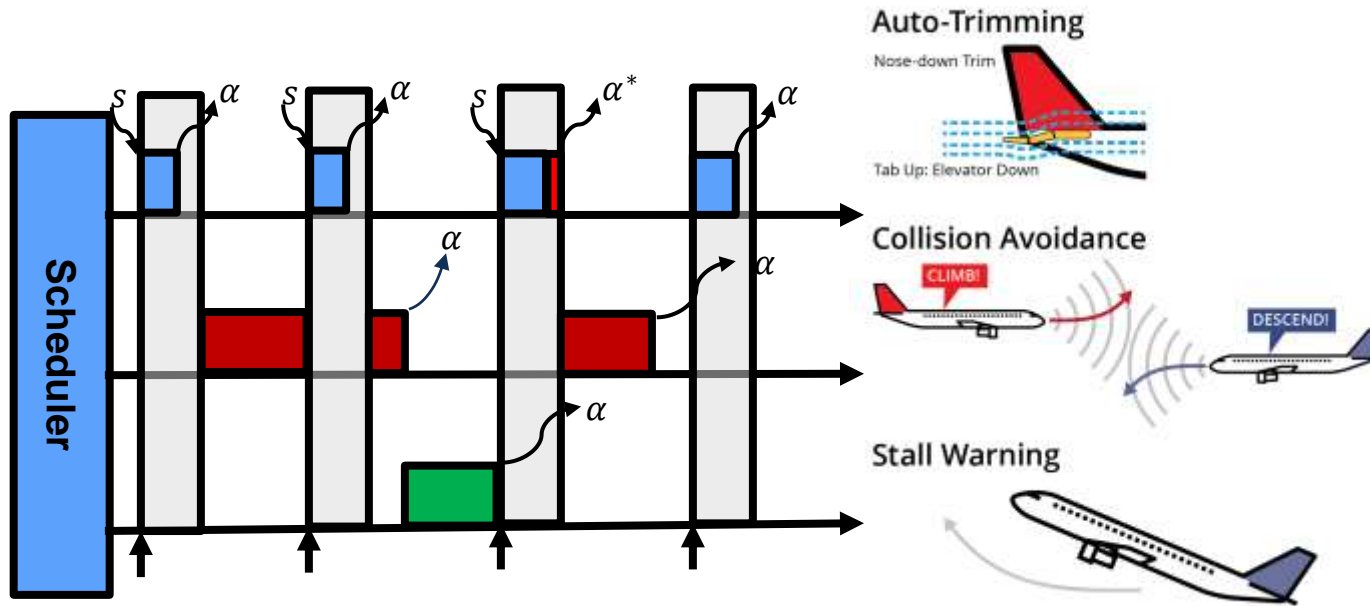


# Many Physical Processes – Many Threads



Icons credit: <http://www.doublejdesign.co.uk>

# Threads Share Single Processor

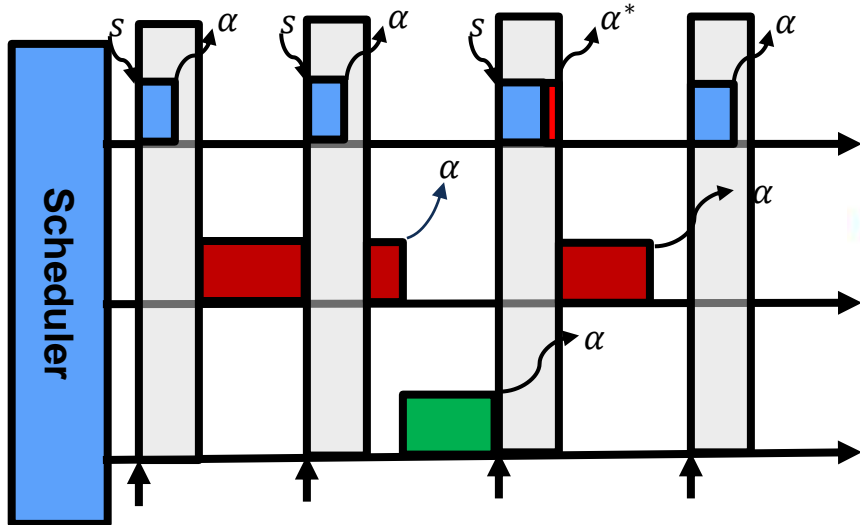


Analyze Resilience to Skip Actuations

Icons credit: <http://www.doublejdesign.co.uk>

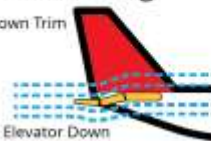


# Threads Share Single Processor



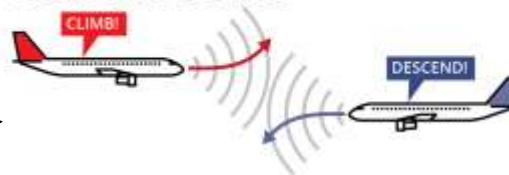
## Auto-Trimming

Nose-down Trim

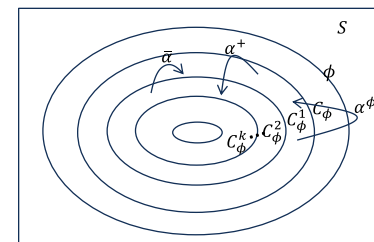
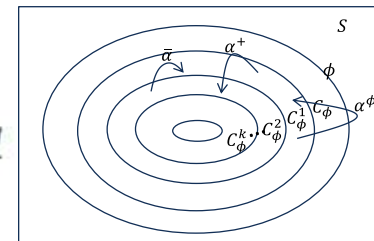
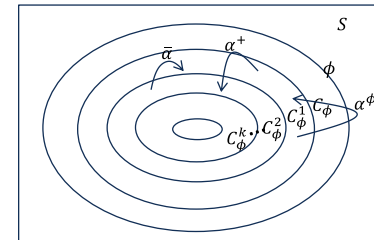


Tab Up: Elevator Down

## Collision Avoidance



## Stall Warning



## Analyze Resilience to Skip Actuations

Icons credit: <http://www.doublejdesign.co.uk>

# Hypervisor Porting

Porting of XMHF Hypervisor for Drone Demos

- Raspberry Pi 3
- New Timing Infrastructure to support integration with temporal enforcer

To Support Tamper-Proof Protection

# Results so Far (1)

Paper accepted on 17th International Conference on Runtime Verification 2017

- “Combining Symbolic Runtime Enforcers for Cyber-Physical Systems”  
Bjorn Andersson, Sagar Chaki, and Dionisio de Niz

Paper under submission

- “Analyzing Real-Time Scheduling of Cyber-Physical Resilience”  
Bjorn Andersson, Dionisio de Niz, and Sagar Chaki.

# Results So Far (2)

## Software Artifacts

- Temporal Enforcer Scheduler with default actuation
- SMT-Based Logical Enforcer Combination
- Porting of XMHF Hypervisor to Raspberry Pi 3 (to support drone demo)

## Demos

- SMT-Based Parrot Mini-Drone demos
  - Logical + Temporal Enforcer

## AFRL Summer of Innovation Transition

- Temporal (ZSRM) + Logical Enforcer into Drone Development Platform (UxAS)

ONR : Reuse of some core modeling ideas

# Future

## Second Year

- Integration of Hypervisor for Tamper-Proof Protection
  - Protect against compromised Virtual Machine
  - Coordinate temporal enforcer between hyper-visor and ZSRM
  - Logical Verification of Hypervisor Integration
- Logical Verification of Logical Enforcer and Default Actuation

## Long Term

- Minimize enforcement actions: allow riskier high reward actions BUT safely
  - Require deeper understanding of risky actions and application:
    - e.g., Autonomy and Machine Learning

# Contact Information

## Presenter / Point(s) of Contact

Dionisio de Niz

Principal Researcher

Email: [dionisio@sei.cmu.edu](mailto:dionisio@sei.cmu.edu)

Telephone: +1 412.268.9002

## Contributors

Bjorn Andersson

Sagar Chaki

James Edmonson

Jeffery Hansen

David Kyle

Gabriel Moreno