# Certifiable Distributed Runtime Assurance



## Challenge

Assure Safety of Distributed Cyber-Physical Systems

- Unpredictable Algorithms (Machine Learning)
- Coordinating multiple vehicles (distributed) to achieve mission

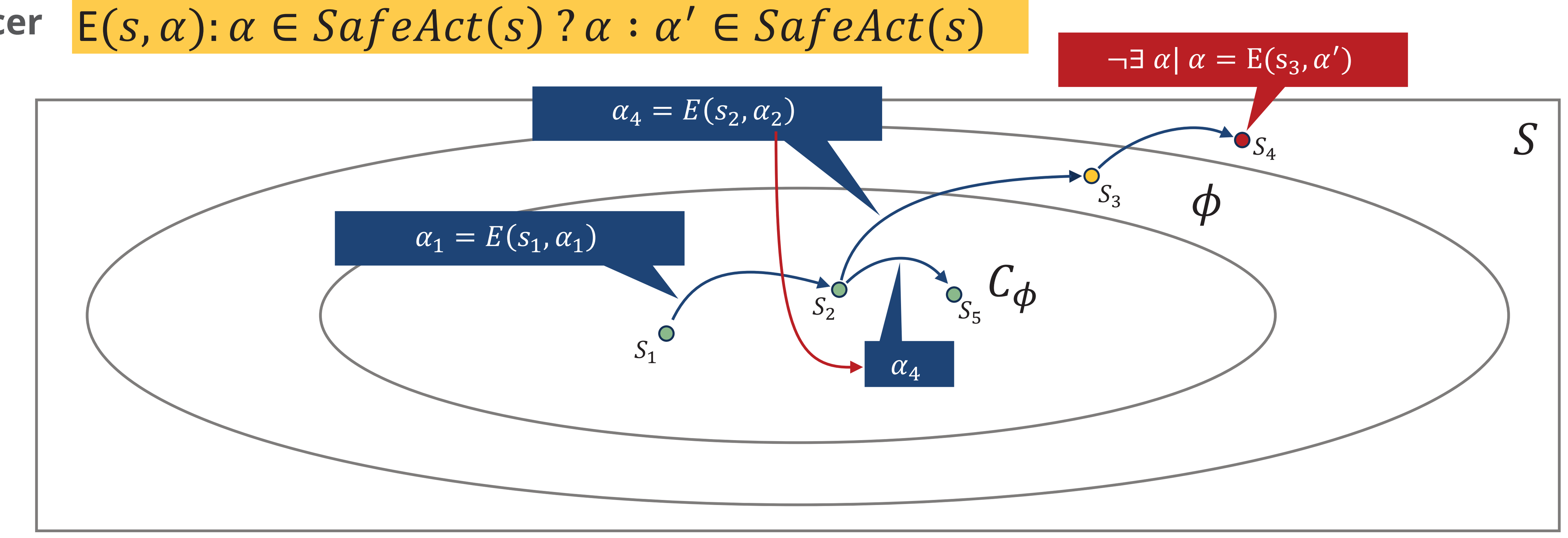
## Solution

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

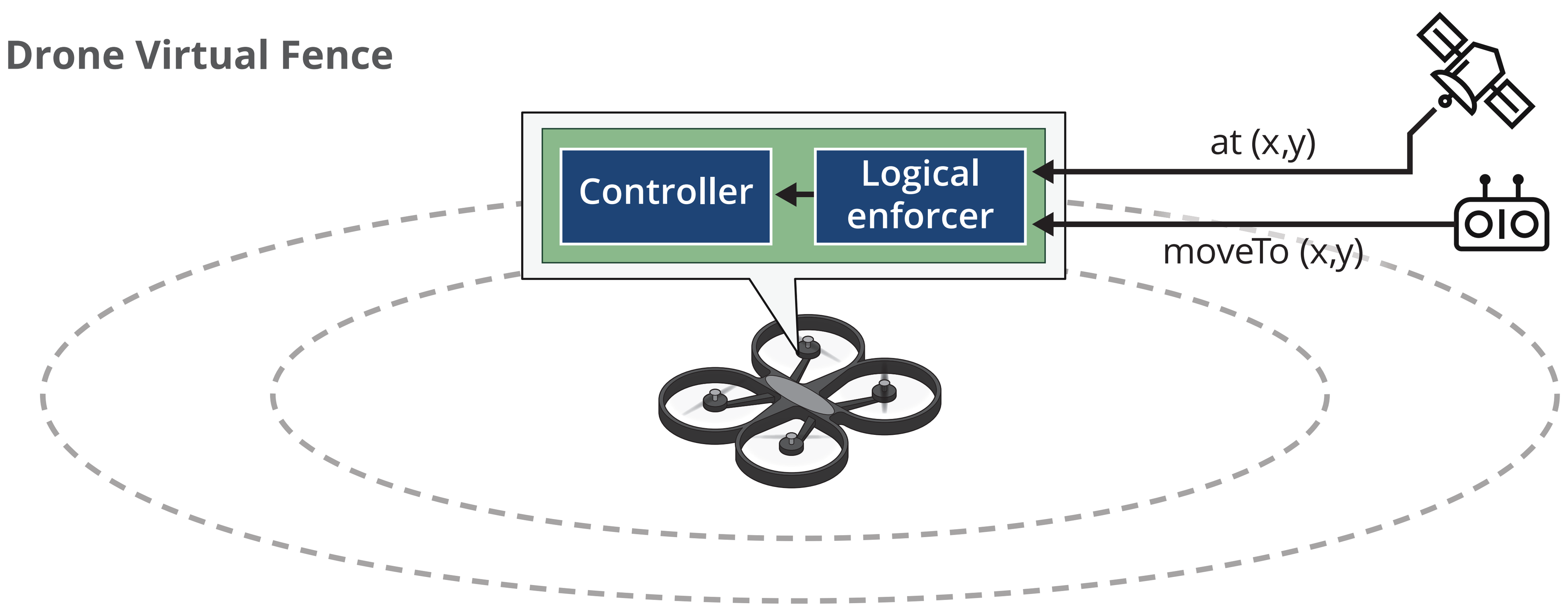
**Formalization (Time-Aware Logic)**
**State of system: variable values**

- State variables: $V_S$
  e.g., (x,y) position
- Action variables: $V_\Sigma$
  e.g., move-to(x,y) action
- System state: $s{:}V_S \mapsto D \in S$
- Actions: $\alpha{:}V_\Sigma \mapsto D$
- Behavior: periodic state transition
  $R_P(\alpha) \subseteq S{\times}S$
  $R_P(\alpha,s) = \{s' | (s,s') \in R_P(\alpha)\}$
- Safe state: $\phi \subseteq S$
- Enforceable state:
  $\phi \supseteq C_\phi = \{s \mid \exists\alpha \in \Sigma{:}R_P(\alpha,s) \in C_\phi\}$
- Safe Actuation
  $SafeAct(s) = \{\alpha | R_P(\alpha,s) \in C_\phi\}$

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




**Enforcing Drone Virtual Fence**




**Timing enforcement**

- Unverified software may never finish!
- => No action produced to be enforced!

**Temporal enforcer**

- Protect other tasks from bogus never-ending (or large) executions
- Produce default safe actuation if task takes too long

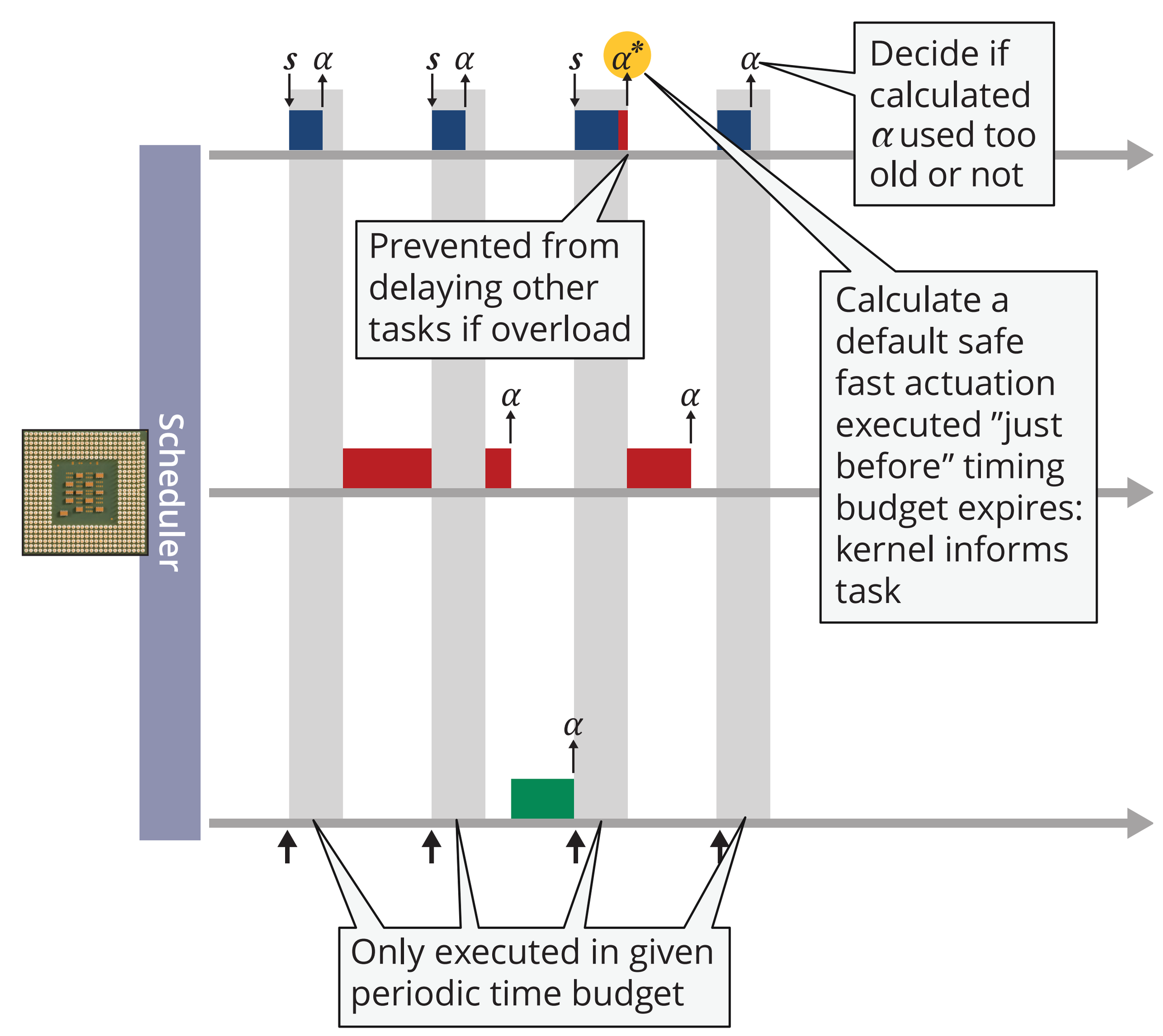
**How**

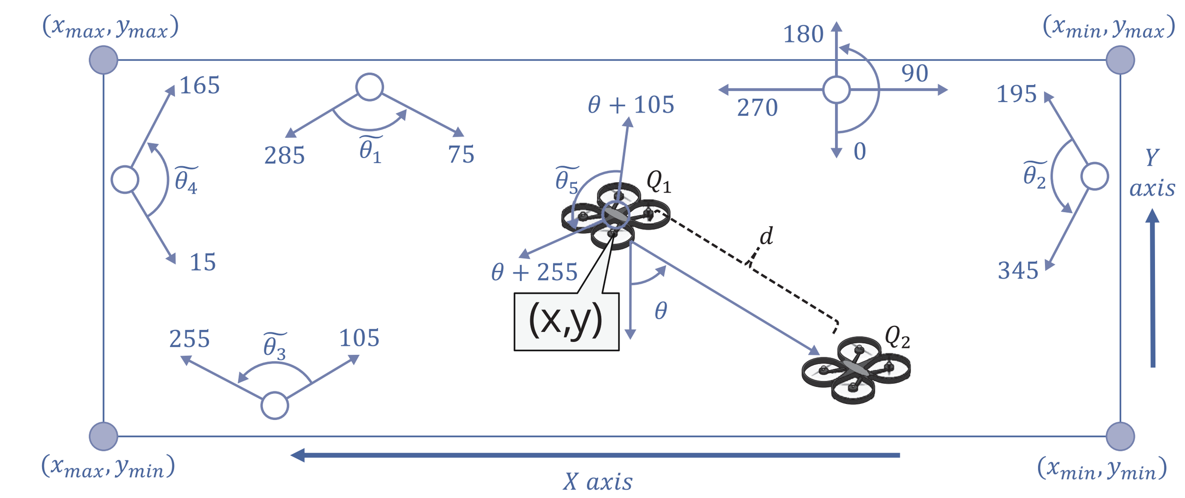
- Each task gets a CPU budget
  stop task if budget exceeded
- If task about to exceed budget
  execute safe action

**Timing guarantees**

- Never allow task to exceed budget
- Always execute actuation

**Safe actuation on timing enforcement**




**Composing enforcers**

- System with multiple enforcers: virtual fence+ collision avoidance
- Safe actions from different enforcers may conflict
  Drone at fence limit + other drone approaching
- Enforcer composition to resolve conflicts (1) priority based (2) utility maximization

**Formalization**

Enforcer: $\boldsymbol{E}{:}(\boldsymbol{P}, \boldsymbol{C_\phi}, \boldsymbol{\mu}, \boldsymbol{U})$
$\mu(s) \subseteq SafeAct(s)$; $U$: utility

No conflict:
$E_1(P_1, C_{\phi_1}, \mu_1, U_1), E_2(P_2, C_{\phi_2}, \mu_2, U_2)$
$\mu_{1,2}{:}\ \mu_1 \cap \mu_2$

Conflict: Priority resolution
$\mu_{1,2}{:}\ \mu_1 \cap \mu_1 \neq \emptyset\ ?\ \mu_1 \cap \mu_2 : \mu_1$

Conflict: Utility maximization
$\mu_{1,2}{:}\ \mu_1 \cap \mu_2 \neq \emptyset\ ?\ argmax_{\alpha\in\mu_1\cap\mu_2}\sum U_i(s,\alpha')$:
$argmax_{\alpha\in\mu_1}\sum U_i(s,\alpha')$

Enforcers allows verification of complex CPS: Autonomous Vehicles

- Limit misbehavior
- With Verifiable Enforcers
- Result: Verified whole system



Dio de Niz | dionisio@sei.cmu.edu

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.



DM17-0730
Certifiable Distributed Runtime Assurance