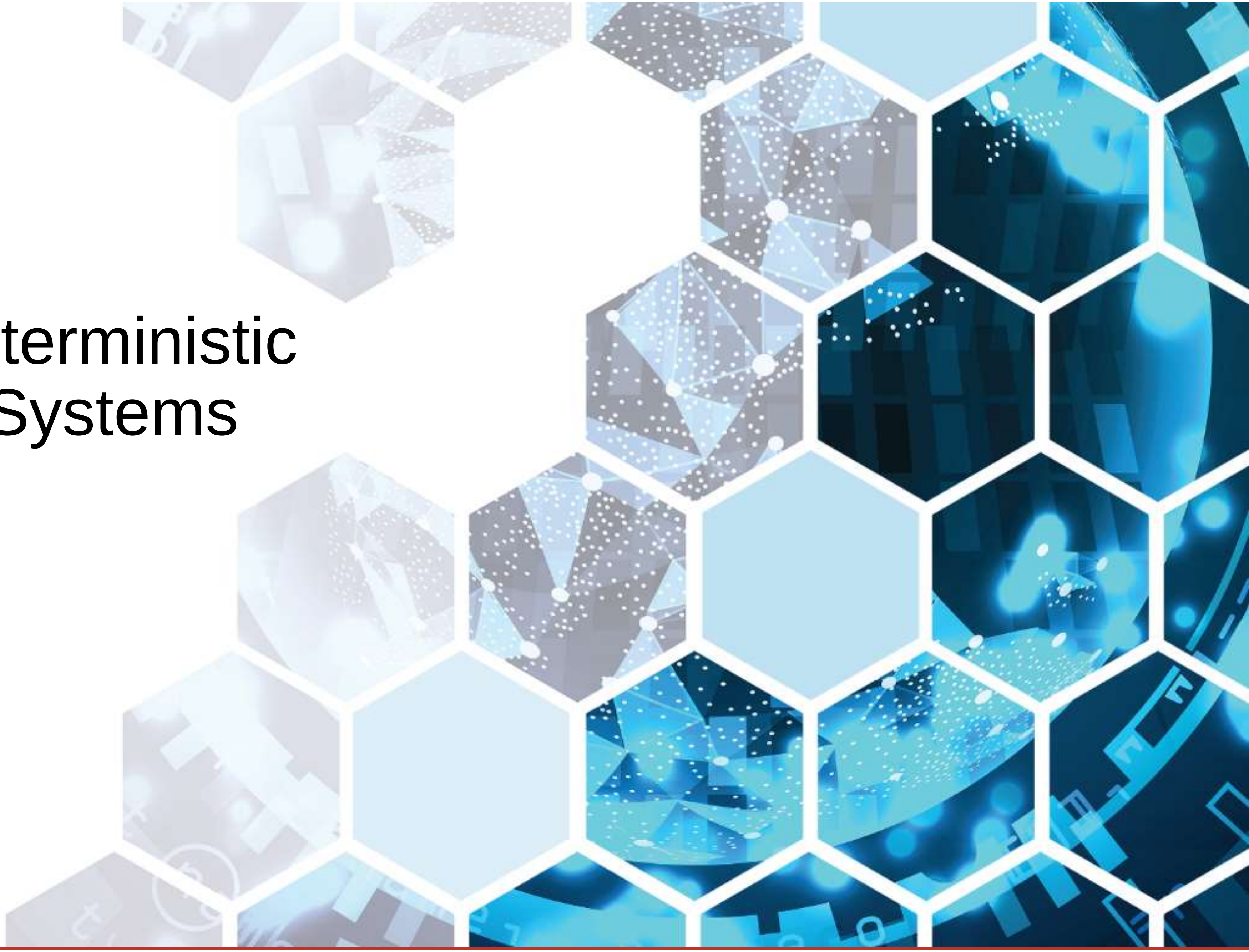
Research Review 2018



Assuring Non-Deterministic Software-Based Systems

Dionisio de Niz

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

Problem

Validation of behavior of new technologies is essential for its adoption

- Non-deterministic algorithms: e.g., Machine Learning (ML)
- Unpredictable environments: e.g., navigating unknown area

Assured Autonomy

- Enable ML to
  - Detect complex patterns (object recognition)
  - Determine actions to take in uncertain situations
- Interact with unknown environment

Cyber-Physical Systems

- React to physical environment
- Safe behavior: safe actions at correct time (e.g., prevent crash)

SEI Focus

Effective assurance techniques for emerging technologies

- Machine Learning
- Autonomous systems

Engineering to enhance usability of AI

- To reduce uncertainty
- To simplify assurance

Aims for this Line of Work

Affordable
Be Affordable such that the cost of acquisition and operations, despite increased capability, is reduced and predictable

Capable
Bring Capabilities that make new missions possible or improve the likelihood of success of existing ones

Trustworthy
Be Trustworthy in construction, correct in implementation, and resilient in the face of operational uncertainties

Timely
Be Timely so that the cadence of fielding is responsive to and anticipatory of the operational tempo of the warfighter

SOTA SOTP

Line-funded Strategic Initiative Projects

What will the robot do next? (Human-Machine Teaming)

PI: Drew Gifford

- Understanding autonomous behavior key for
  - Simplification of behavior validation
  - Human-robot teaming

Certifiable Distributed Runtime Assurance

PI: Dionisio de Niz

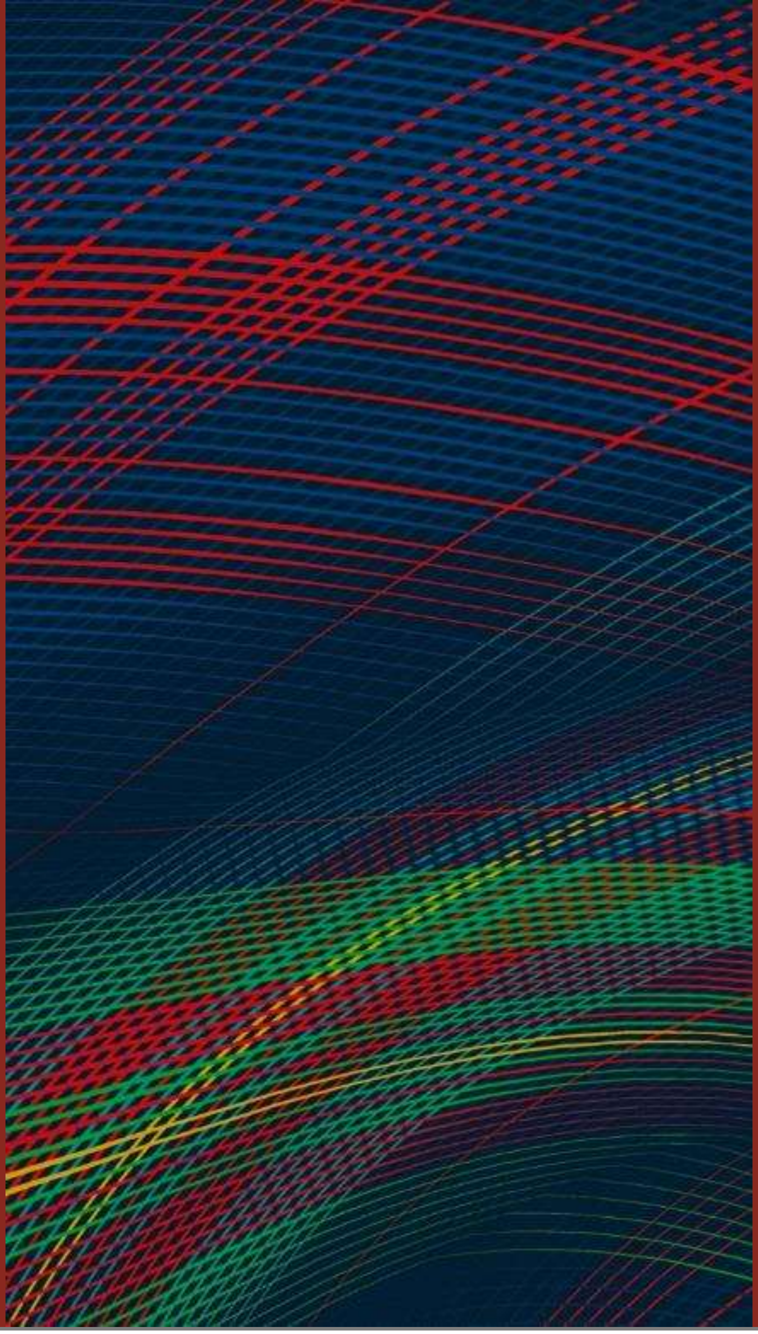
- Use enforcers to ensure safe behavior
- Verification limited to enforcers resulting in verified system-wide safety

Previous Projects

- Verifying Distributed Adaptive Real-Time Systems (DART)
  - Languages: DMPL, Schedulers: ZSRM, Tools: DEMETER
- Why did the robot do that?

What will the robot do next? (Human-Machine Teaming)

PI: Dr. Drew Gifford



What will the robot do next?

Human-Machine Teaming

SIAMESE CONNECTION FOR FIRE DEPARTMENT SPRINKLERS THROUGHOUT BUILDING
FIRE HOSE CONNECTION
ROBOTIQ
Carnegie Mellon NREC

Understanding robot behavior is important:

- Robots are increasingly being utilized in important tasks such as search and rescue operations.

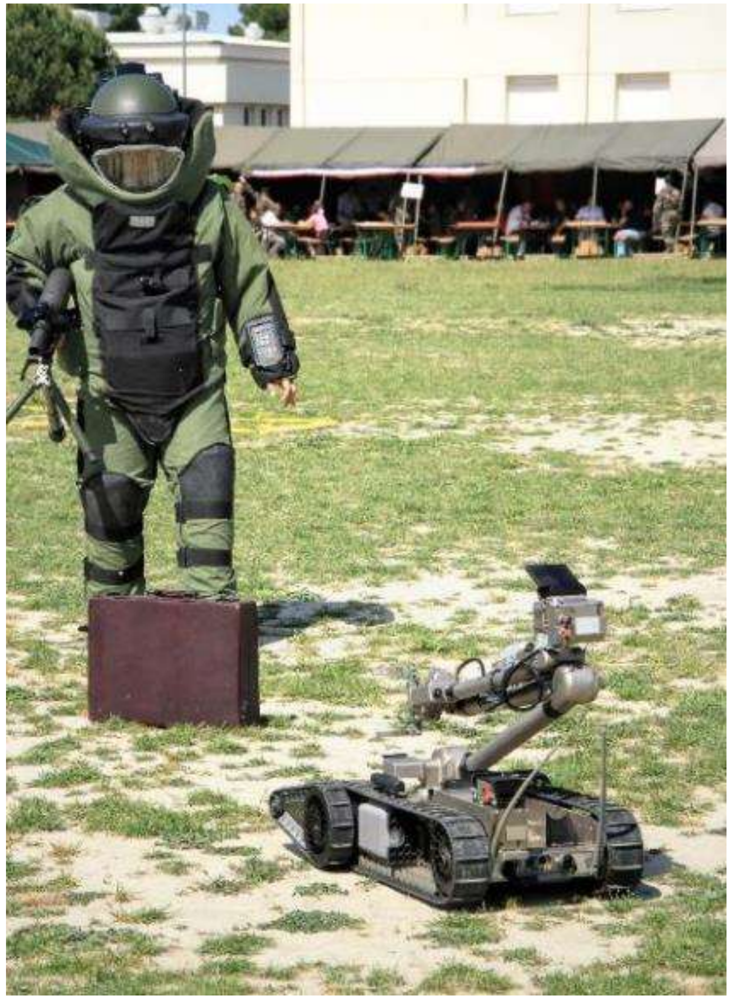
Understanding robot behavior is difficult:

- Their behaviors are often hard to understand, leading to users' mistrust and often abandonment of very useful tools.

People form beliefs about robot behavior through observation, but robots do not execute actions with the intent of conveying state preferences.

http://archive.defense.gov/DODCMSShare/NewsStoryPhoto/2013-12/hrs_tartan%20rescue.jpg

Modifying Robot Behavior Based on User Attention



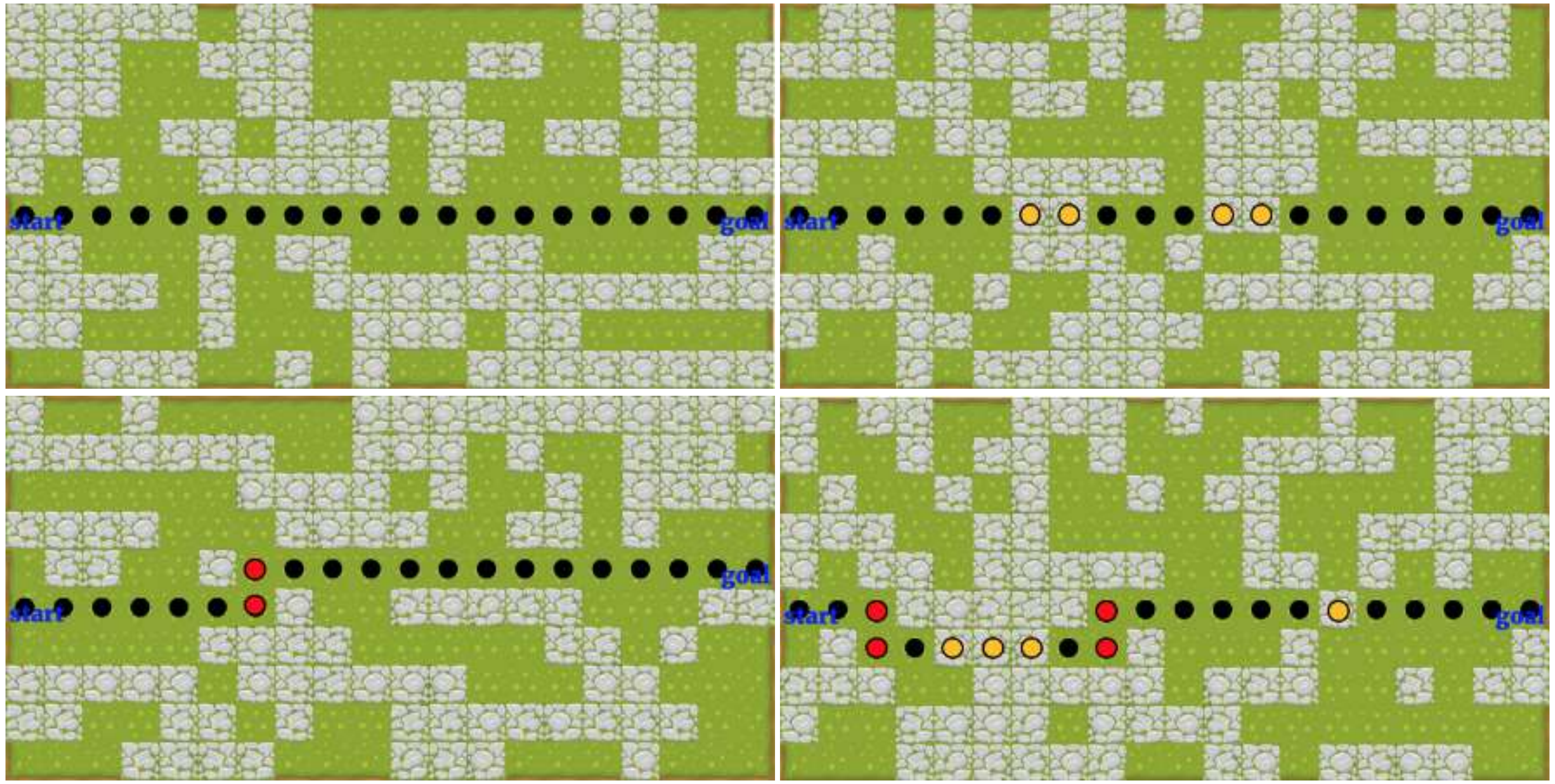
How can we increase neglect tolerance – the length of time that users are willing to look away from their robots before they proactively monitor them again?

Neglect tolerance is widely used as a measure of trust in robots.

We seek to measure and use operator gaze to adapt robot actions.

How might a robot behave (or “misbehave”) to confirm operator expectations of robotic motion?

Generating Trajectories to Convey System Intent



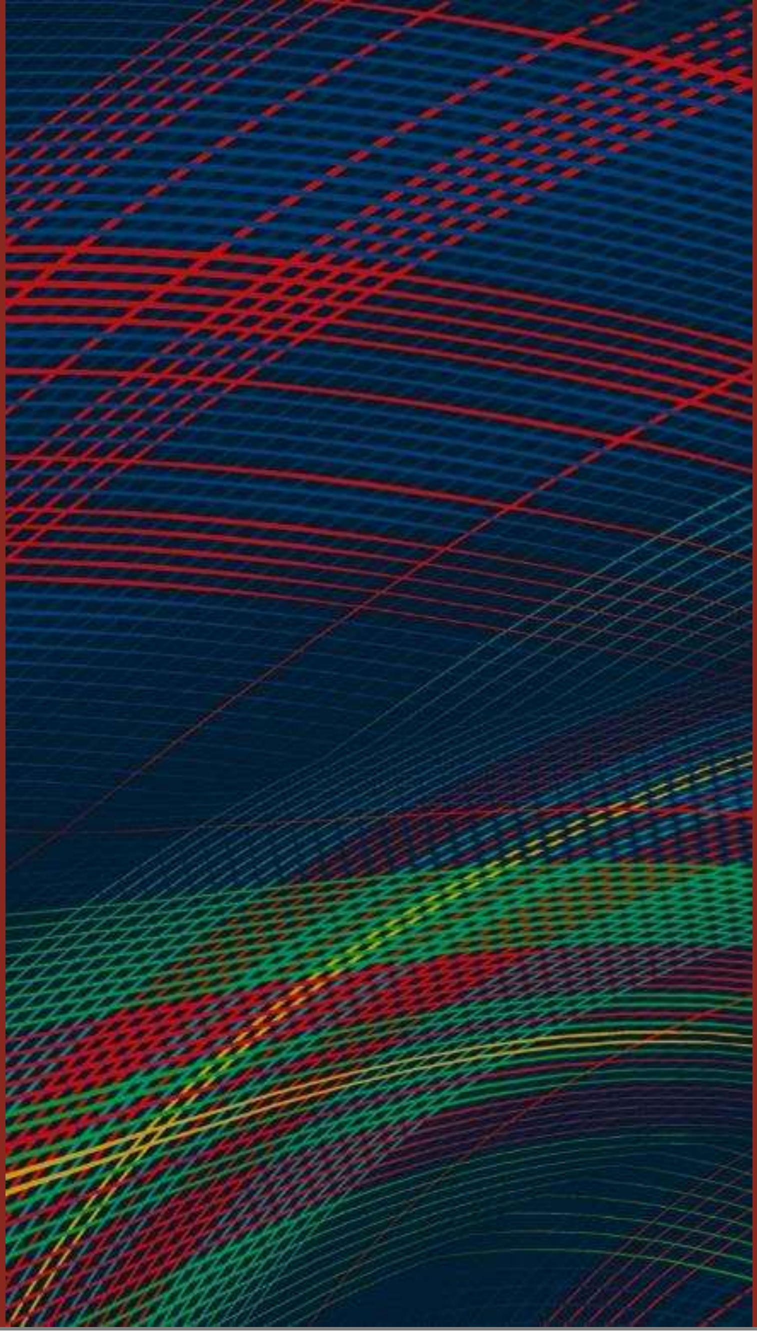
"Evaluating Critical Points in Trajectories" paper published and presented at 26th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN 2017)

How can the robot *generate* paths that indicate preference?

Can users infer preference based on these non-verbal explanations?

Certifiable Distributed Runtime Assurance (CDRA)

PI: Dr. Dionisio de Niz



Certifiable Distributed Runtime Assurance

Assurance Automation for Safe-Critical Cyber-Physical Systems

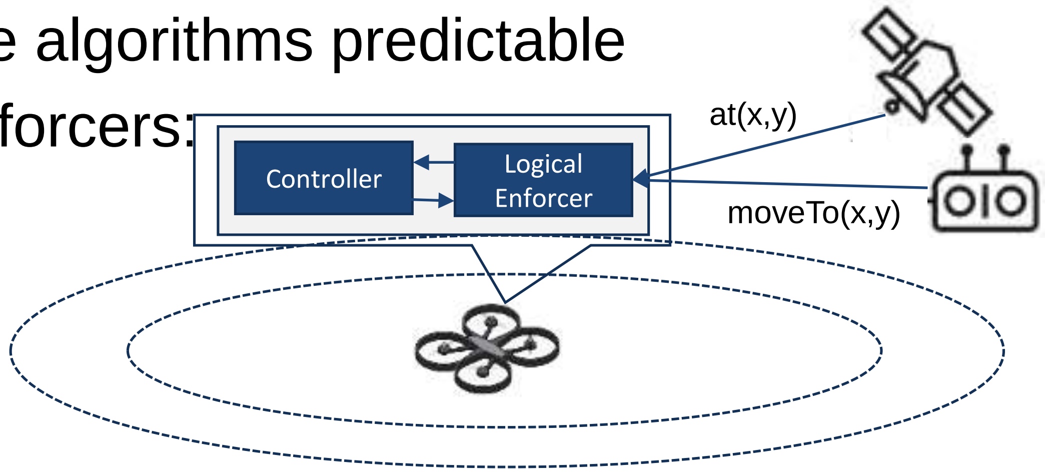
- Through formal verification

Challenge:

- Traditional Verification Does Not Scale
- Unpredictable Algorithms like machine learning (Autonomous CPS)
- Timely Interaction with Environment: correct **actions** at correct **time**

Our Solution:

- 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**
- Protect enforcers against failures/attacks



Related Work

Simplex Architecture (SEI)

- Lui Sha, Bruce Krogh, et al.
  - Normal untrusted controller guarded by simple safety controller
- Control-theoretic reachability verification (Bak, et al.)
- Unverified code

Control Theoretic / Hybrid Verification

- Claire Tomlin
- Unverified ML guarded by safety controller
  - Verified control-theoretic model via reachability
  - Unverified code, timing

Runtime Assurance

- Safety-Progress (Falcone): Logical Verification Only
- Edit Automata (Ligatti): Logical Verification Only
- CoPilot (Pike): sampling internal estate, logical verification, temporal sampling

Logical Model

Statespace

- $S = \{s\}$
- $\phi \subseteq S$

Periodic actions

- Transition: $R_P(\alpha) \subseteq S \times S$
- Destination state: $R_P(\alpha, s) = \{s' | (s, s') \in R(\alpha)\}$

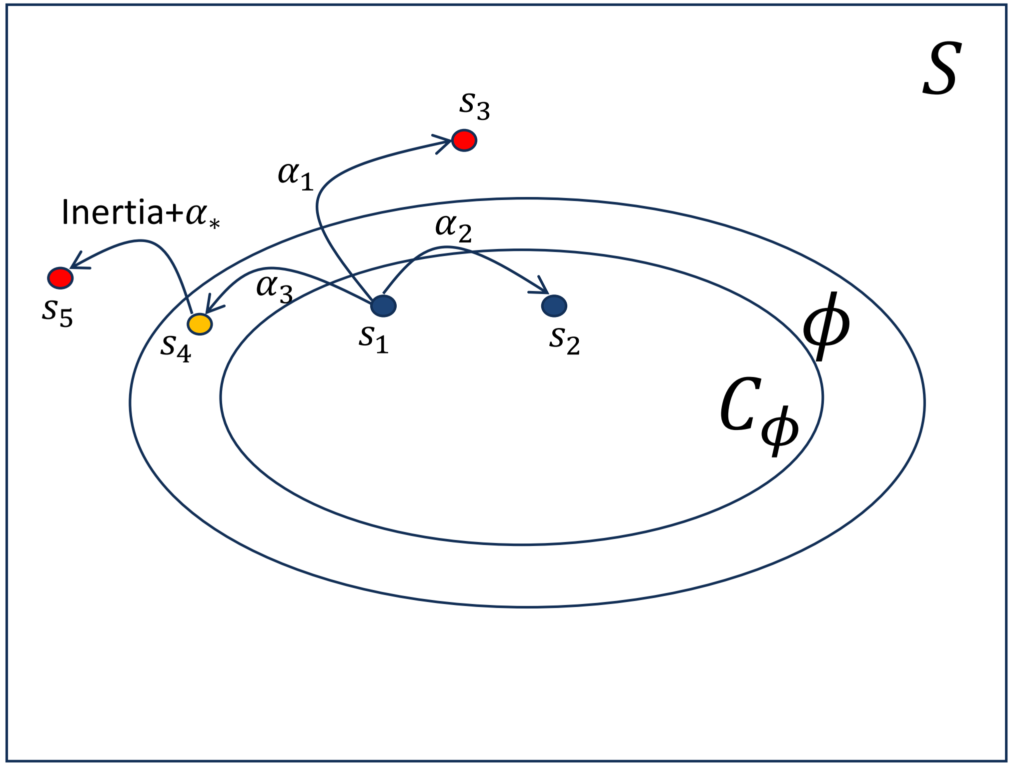
Identify states too close to safety border

- Inertia lead to unsafe state even if enforced
- Enforceable states:

$$C_\phi = \{s | \exists \alpha : R_P(\alpha, s) \in C_\phi\}$$

Safe actions:

- $SafeAct(s) = \{\alpha | R_P(\alpha, s) \in C_\phi\}$



Logical Enforcer

Statespace & actions

- $S = \{s\}, \phi \subseteq S$
- $R_P(\alpha) \subseteq S \times S;\ R_P(\alpha, s) = \{s' | (s, s') \in R(\alpha)\}$

Enforceable states

- $C_\phi = \{s | \exists \alpha : R_P(\alpha, s) \in C_\phi\}$

Safe actions:

- $SafeAct(s) = \{\alpha | R_P(\alpha, s) \in C_\phi\}$

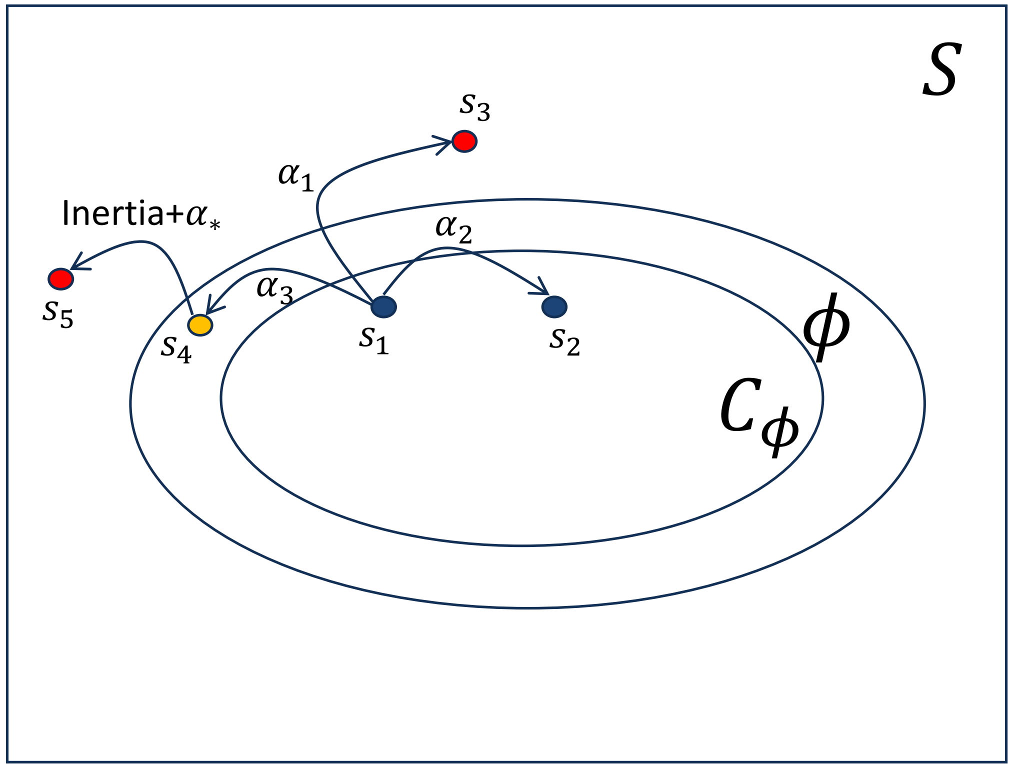
Logical Enforcer: $E = (P, C_\phi, \mu)$

- Set of safe actions:

$$\mu(s) \subseteq SafeAct(s)$$

- Monitor and enforce safe action:

$$\tilde{\alpha} = \begin{cases} \alpha, & \alpha \in \mu(s) \\ pick(\mu(s)), & otherwise \end{cases}$$



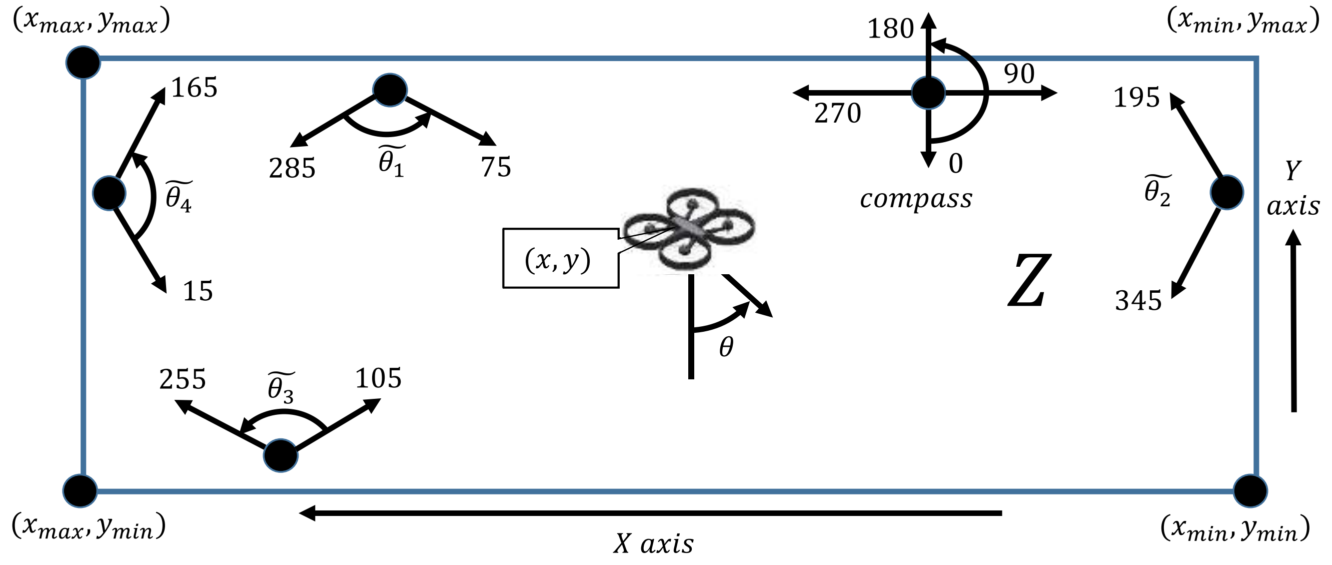
Drone Example

Statespace

- $S = \{s | s = (x, y, \theta)\}$
- $\phi = \{(x, y, \theta) | (x, y) \in Z\}$

Enforceable states

- δ_P: Max distance in one period P
- δ_B: Max distance in opposite direction of enforcement
- $C_\phi = \{(x, y, \theta) | (x + \delta_B, y + \delta_B) \in Z \wedge (x - \delta_B, y - \delta_B) \in Z\}$



Action: constant speed at angle θ

Enforcement: $$\tilde{\theta} = \begin{cases} \tilde{\theta} \in \tilde{\theta}_1,\ if\ Y_{max} - y \le \delta_B + \delta_P \\ \tilde{\theta} \in \tilde{\theta}_2,\ if\ x - X_{min} \le \delta_B + \delta_P \\ \tilde{\theta} \in \tilde{\theta}_3,\ if\ y - Y_{min} \le \delta_B + \delta_P \\ \tilde{\theta} \in \tilde{\theta}_4, if\ X_{max} - x \le \delta_B \theta + \delta_P \\ \theta, otherwise \end{cases}$$

Composing Enforcers

Enforcer Details: $\mathrm{E}{:}\left(P, C_{\phi}, \mu, U\right)$

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

Composition without conflict

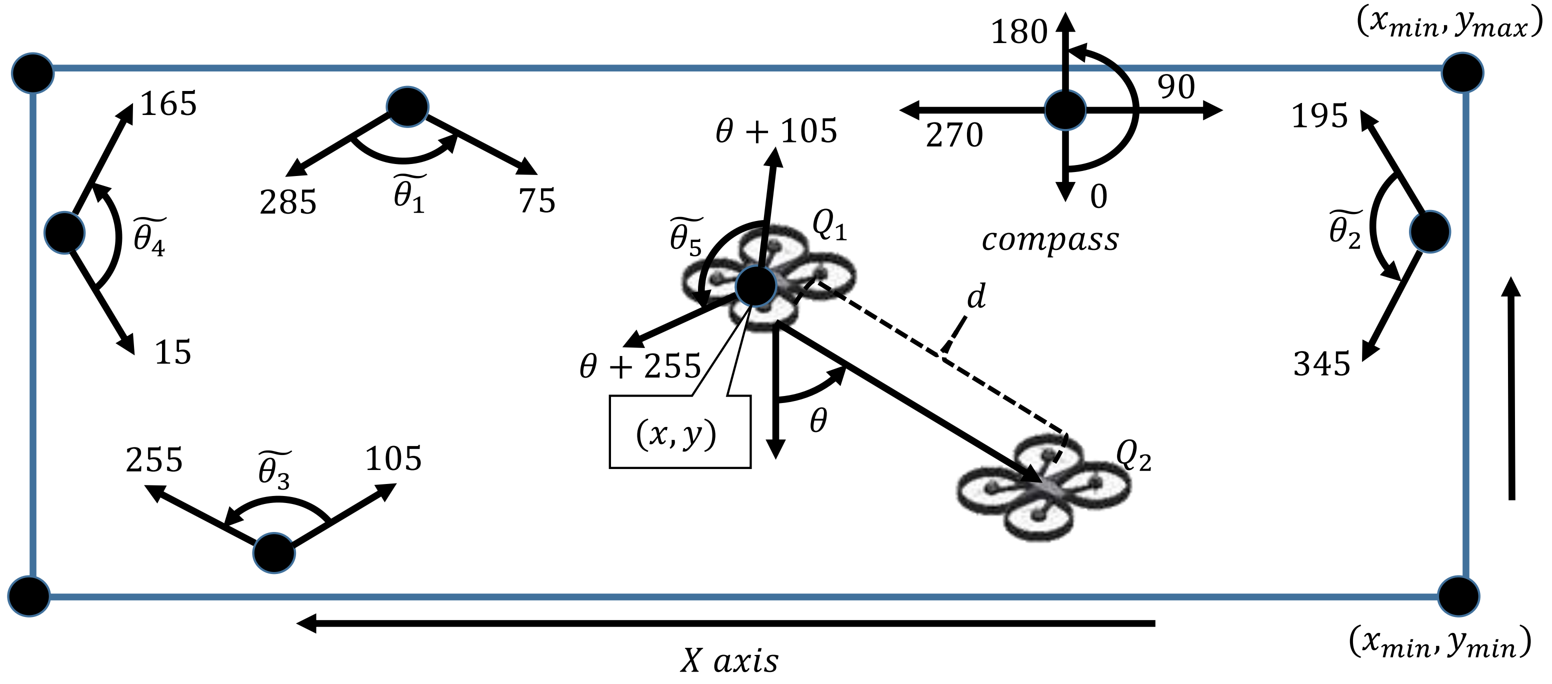
- $E_1{:}\left(P_1, C_{\phi_1}, \mu_1, U_1\right)$
- $E_2{:}\left(P_2, C_{\phi_2}, \mu_2, U_2\right)$
- $\mu_{1,2}{:}\ \mu_1 \cap \mu_2$

Conflicting: Priority:

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

Conflicting: Utility

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



Are We Done Yet?

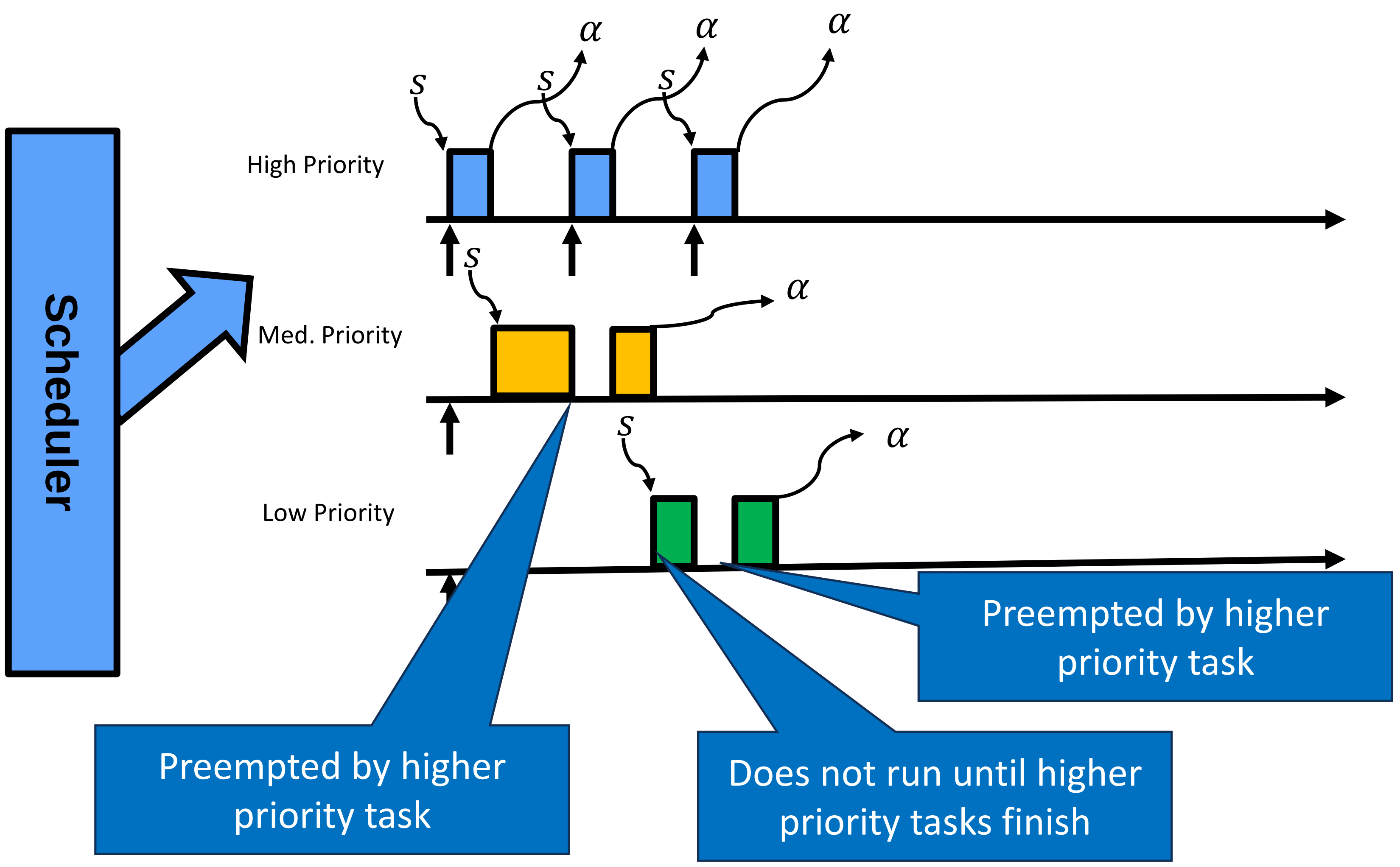
Timing Assumption:

- Unverified software + enforcer finish before end of every P period.
  - Unverified software executes for less than its Worst-Case Execution Time (WCET)
  - Other software running executes for less than its WCET
  - Schedulability analysis successful

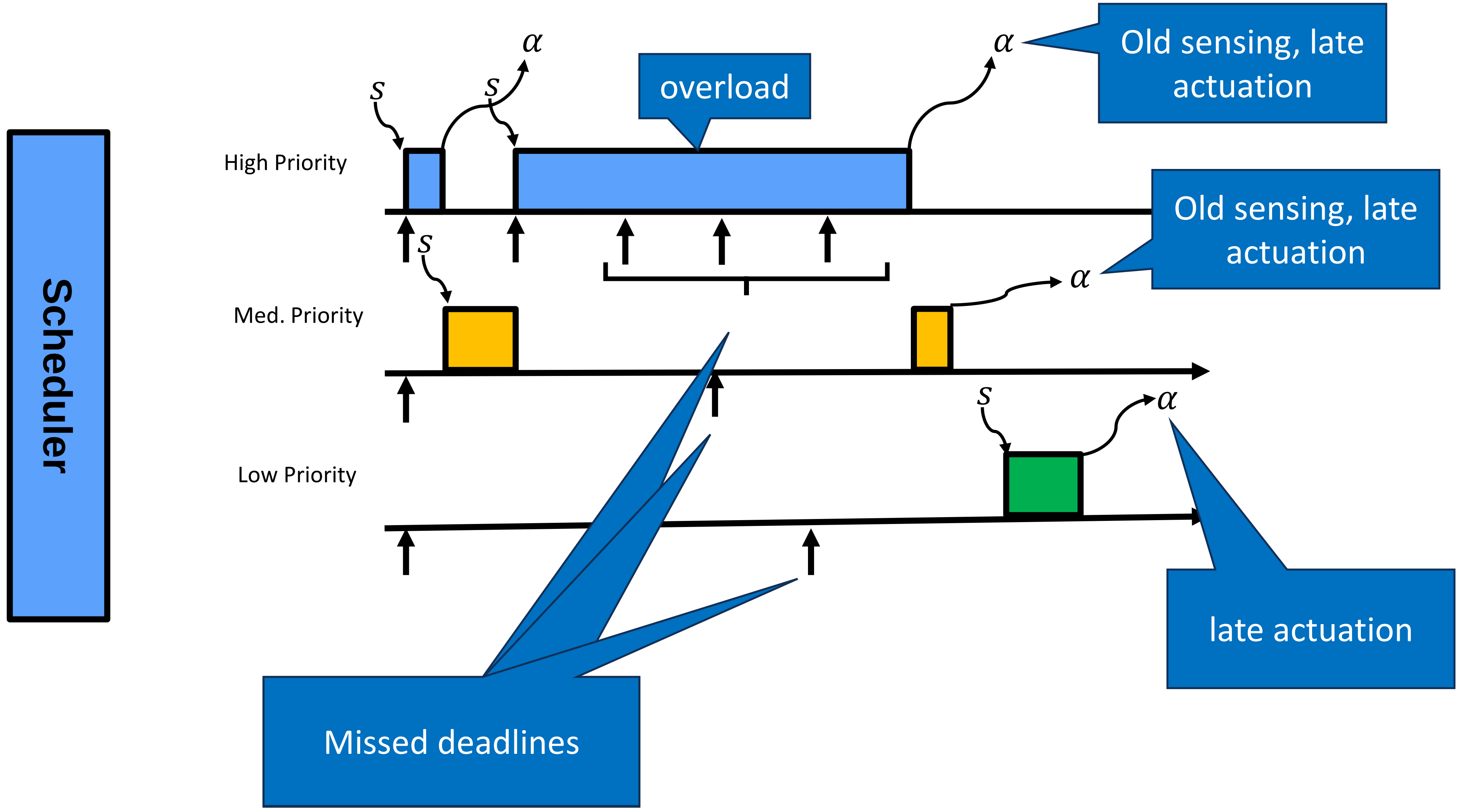
What can go wrong?

- Unbounded preemption
  - High priority software executes longer than WCET
  - Can make other software miss deadlines: late actions with old sensing
- Unbounded execution
  - Software executes longer than WCET
  - Misses its own deadline: Does **NOT** produce output on time: late action + old sensing
    - **Inertia** takes it to **unsafe state**

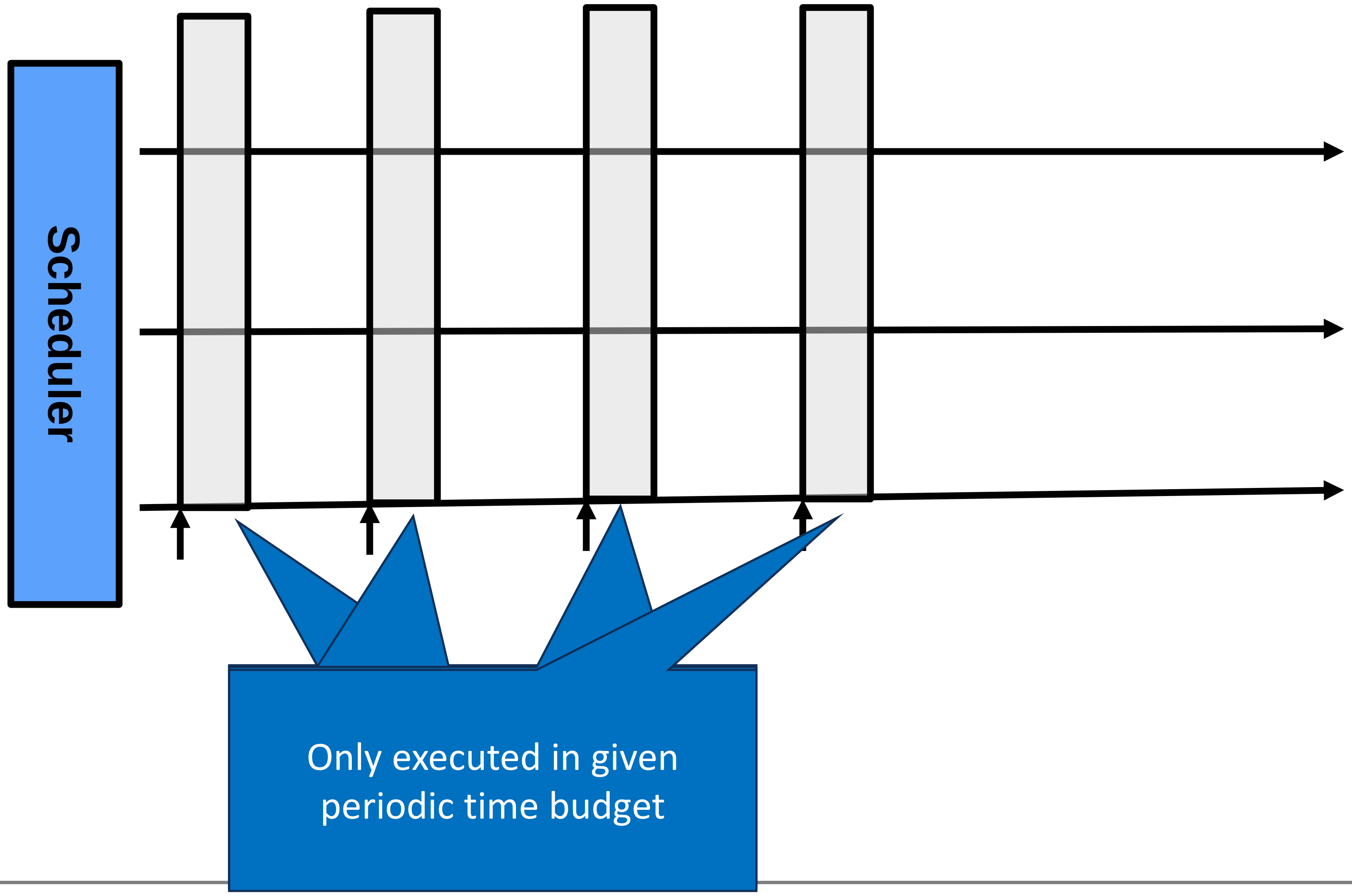
Fixed-Priority Scheduling + Rate Monotonic


Scheduler
High Priority
Med. Priority
Low Priority
s
α
Preempted by higher priority task
Does not run until higher priority tasks finish
Preempted by higher priority task

Overload -> old sensed data + late actuation

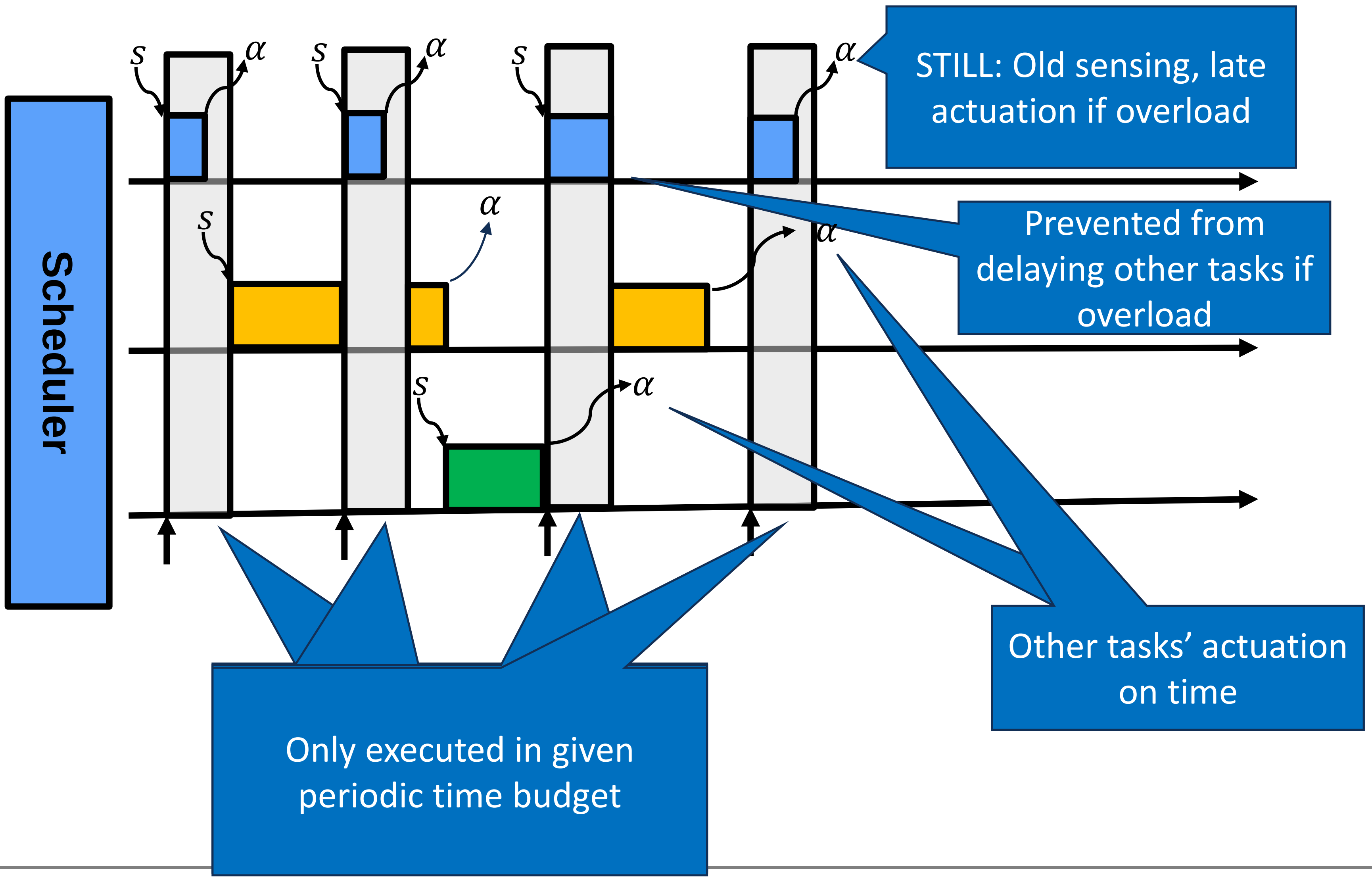


Unbounded preemption
Solution: Enforce timing budgets (timing enforcement)

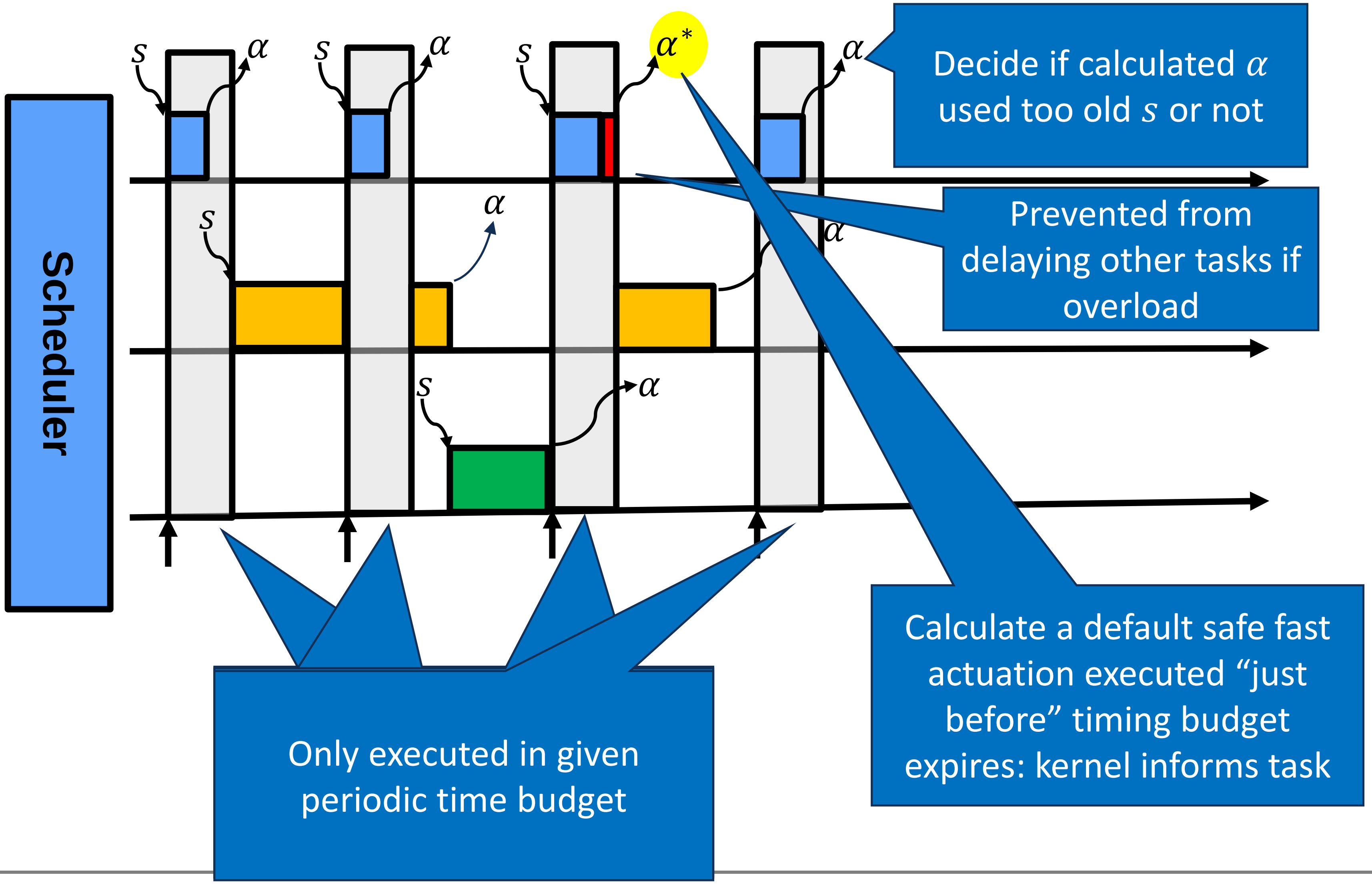


Only executed in given periodic time budget

Unbounded preemption
Solution: Enforce timing budgets (timing enforcement)


Scheduler
s
α
STILL: Old sensing, late actuation if overload
Prevented from delaying other tasks if overload
Other tasks' actuation on time
Only executed in given periodic time budget

Unbounded Execution: Solution: safe actuation on timing enforcement


Scheduler
s
α
α^*
Decide if calculated α used too old s or not
Prevented from delaying other tasks if overload
Only executed in given periodic time budget
Calculate a default safe fast actuation executed "just before" timing budget expires: kernel informs task

Are we done yet?

Unverified software may corrupt Logical Enforcer

- It can even be malicious

Unverified software uses

- Unverified OS/kernel
- Unverified libraries

Temporal Enforcer relies on

- Unverified kernel / scheduler

Mixed-Trust Computing

System composed of trusted (verified) and untrusted (unverified) components

- Trusted : Verified Enforcers
- Untrusted: Unverified software

Untrusted should not corrupt trusted

Trusted should not depend on untrusted

- Cannot depend on unverified kernel / scheduler

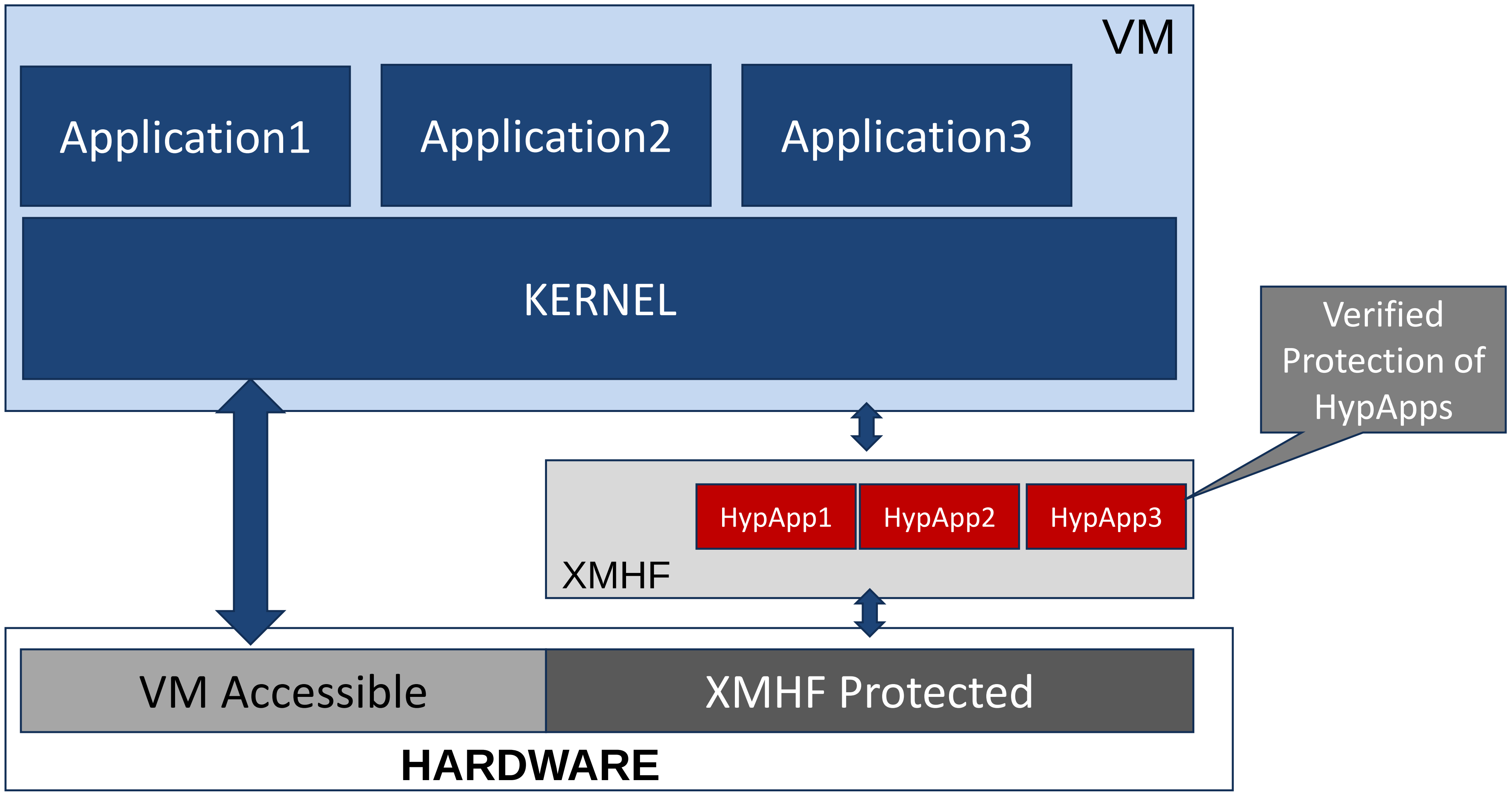
Trusted components

- Preserve safety

Untrusted components

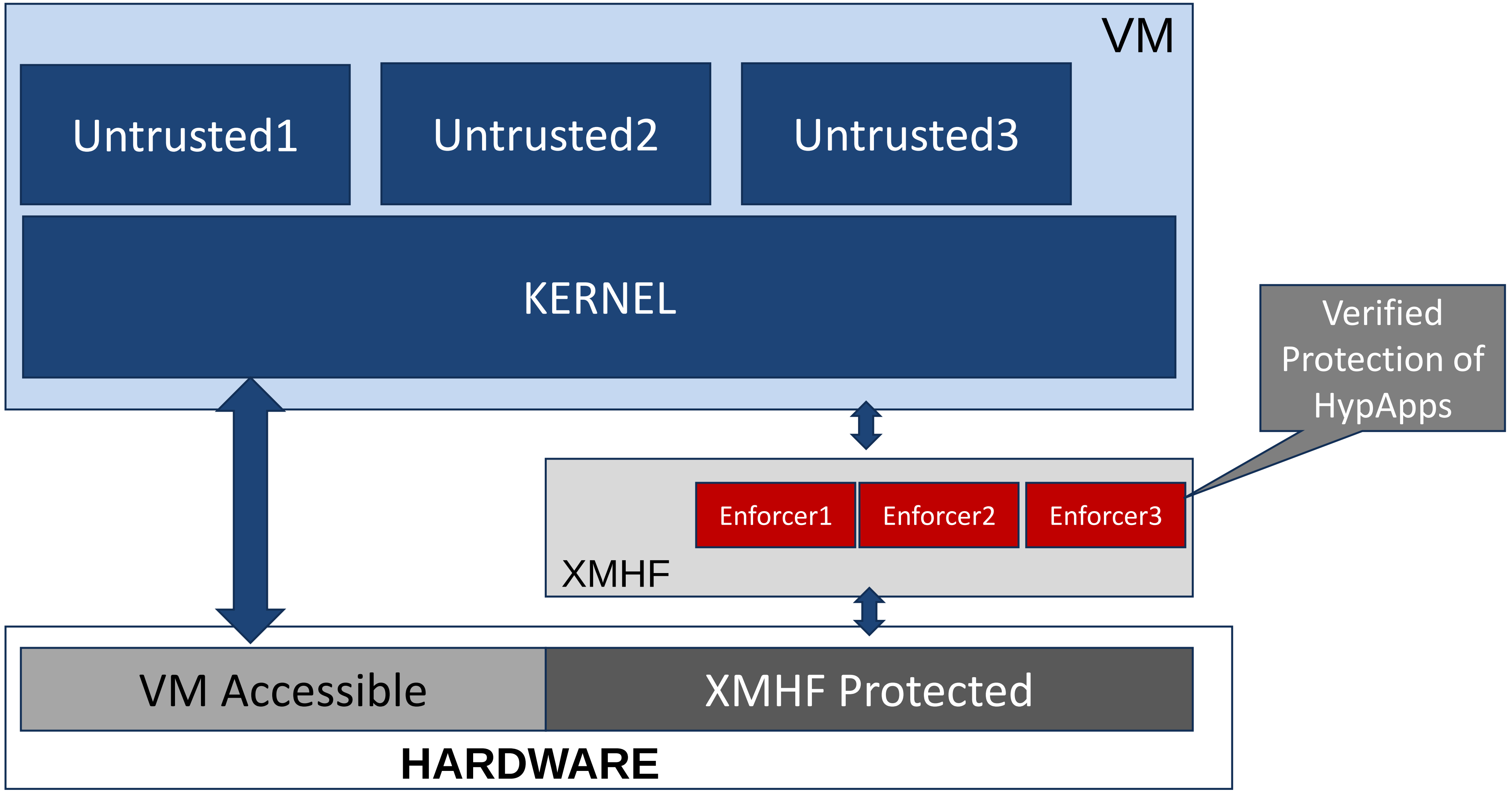
- Provide mission capability / performance
- Potential spurious failures

Uber XMHF: Verified Micro-Hypervisor Protection

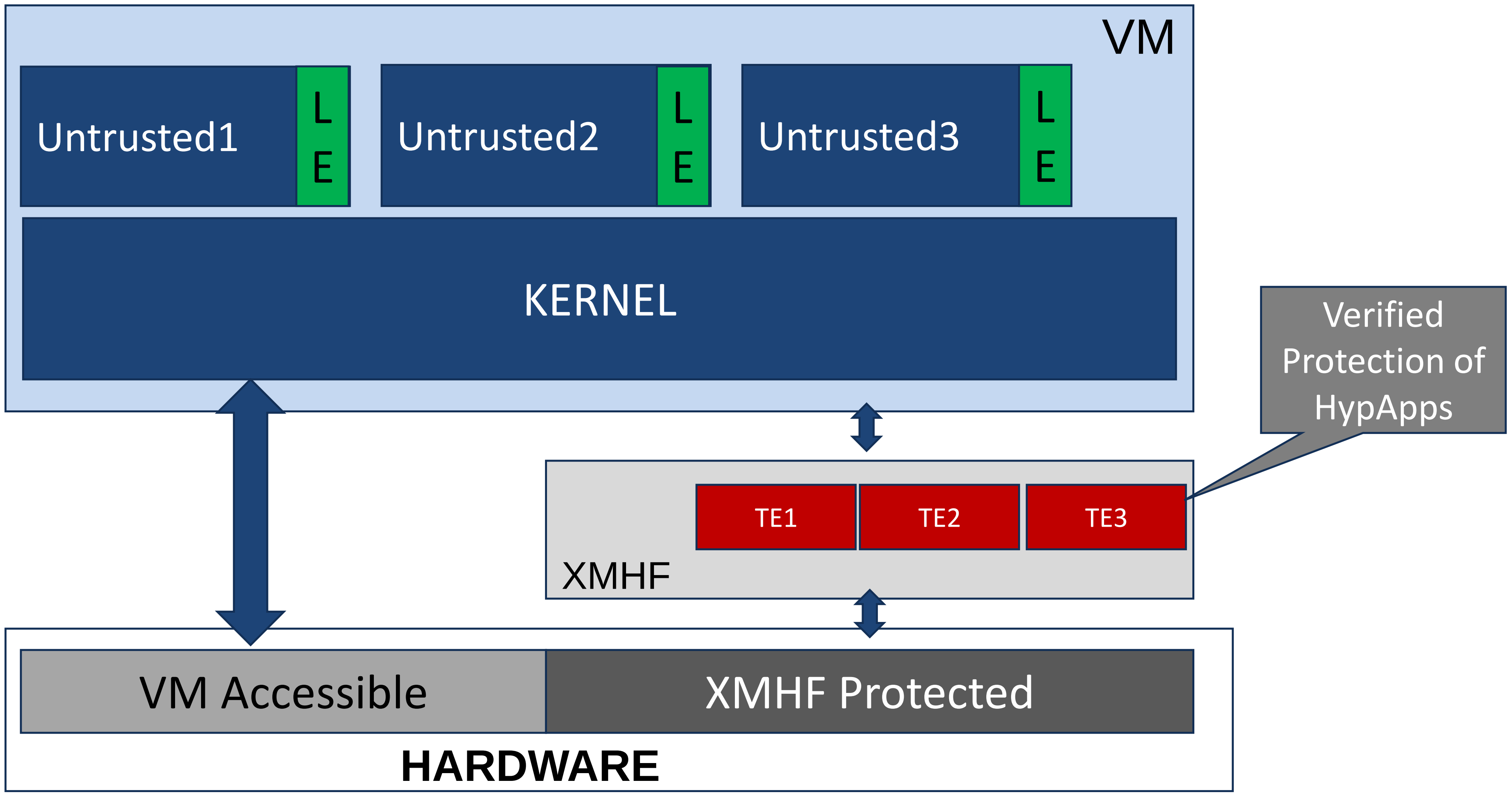


Uber XMHF: Verified Micro-Hypervisor Protection

Only temporal enforcer can be protected if untrusted does not finish

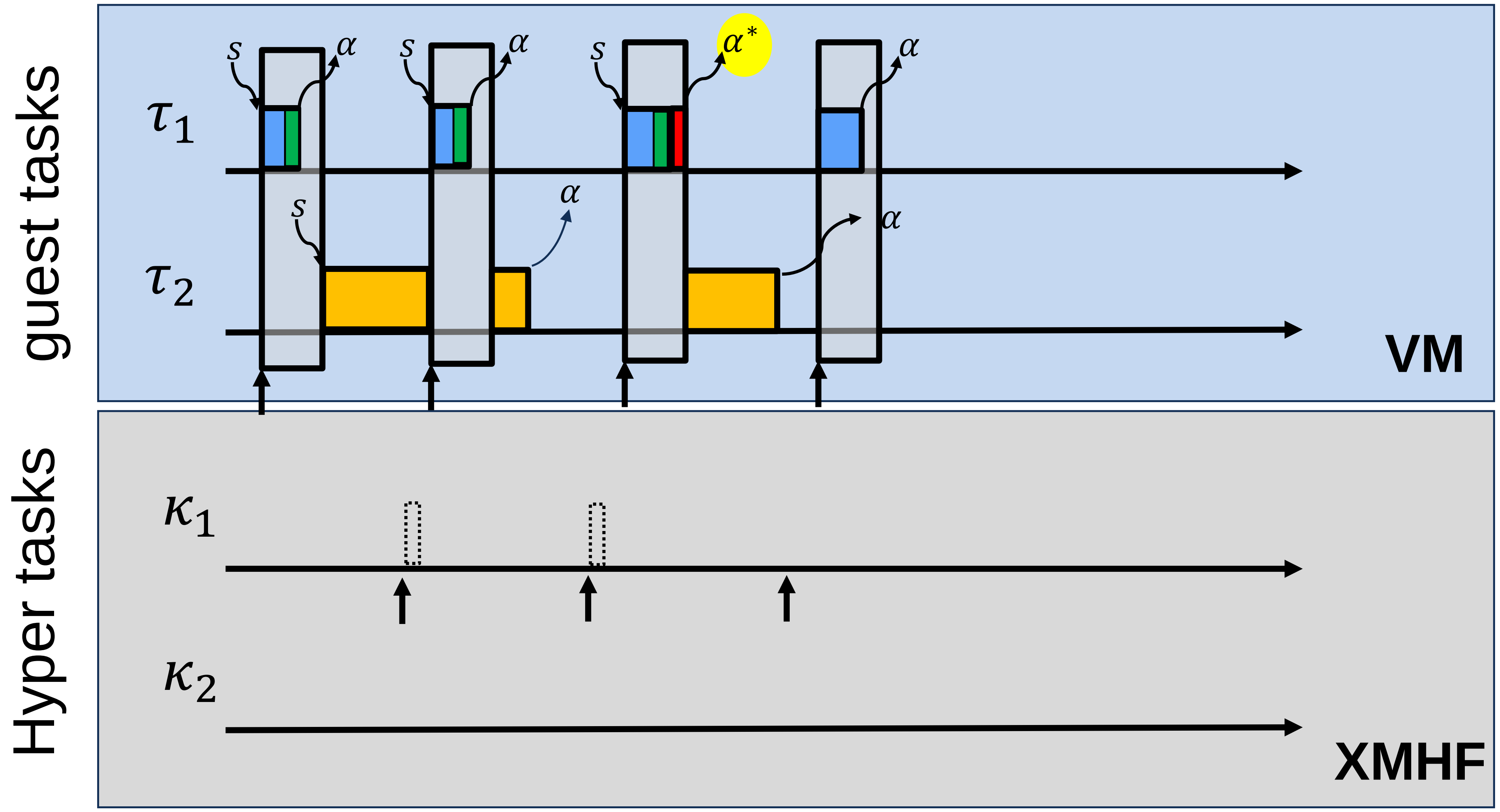


Uber XMHF: Verified Micro-Hypervisor Protection



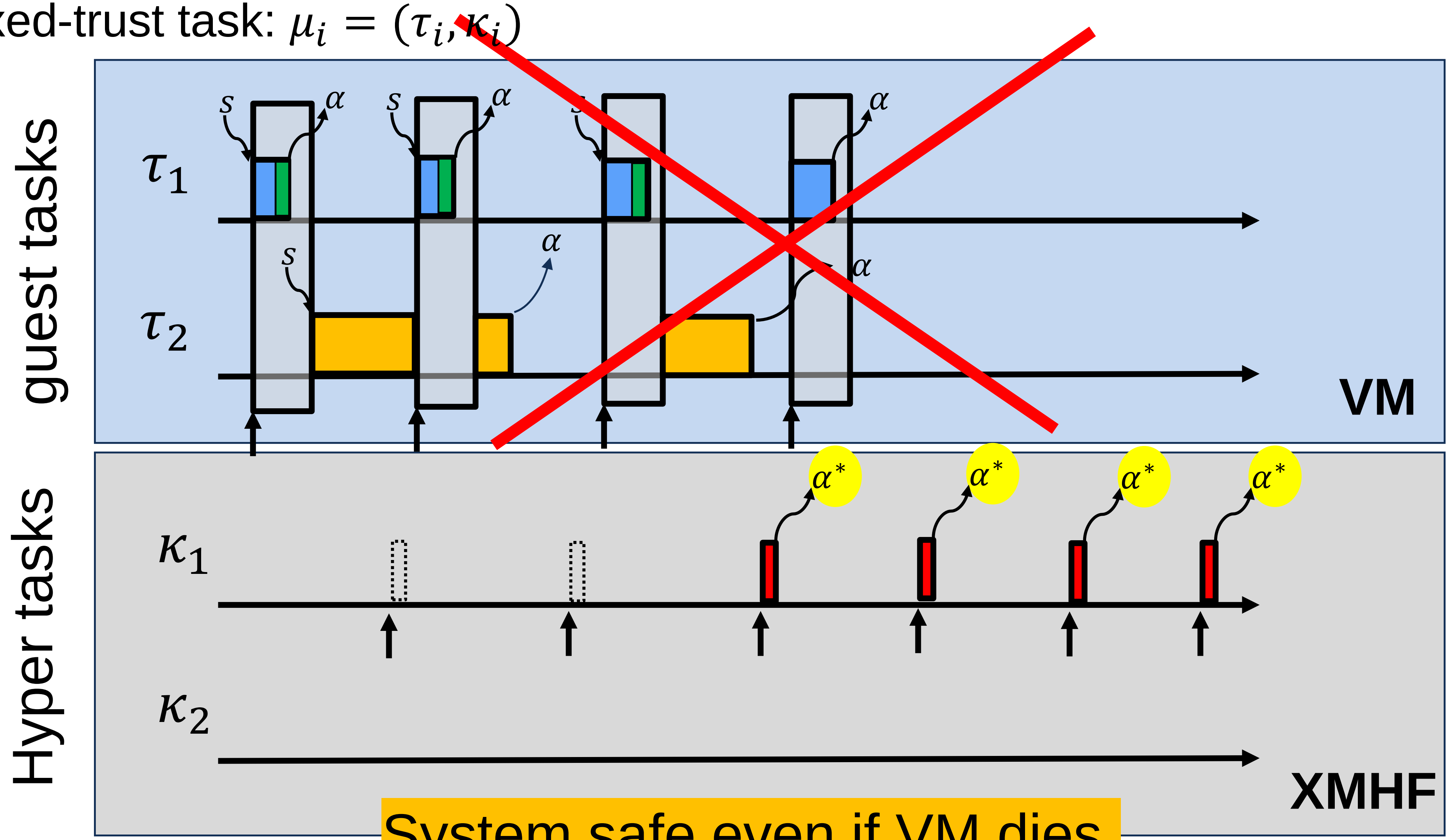
Two schedulers: VM scheduler + XHMF Scheduler

Mixed-trust task: $\mu_i = (\tau_i, \kappa_i)$



Two schedulers: VM scheduler + XHMF Scheduler

Mixed-trust task: $\mu_i = (\tau_i, \kappa_i)$



System safe even if VM dies

Mixed-Trust Scheduler

VM Scheduler

- Fixed Priority
- Preemptive
  - To maximize utilization

Hypervisor Scheduler

- Fixed priority
- Non-Preemptive
  - To simplify verification

New Timing Verification Equations

Mixed-Trust Computing Remarks

Verification applied only to small part of system: enforcers
- Unverified parts guarded by enforcers
- Increases speed of validation, decreases its cost
- Verified System-Wide Safe Behavior

Verified hypervisor protection allows any unverified part
- COTS, open source community
- Even malicious code is prevented from corrupting safe behavior

Enables Safe Use of
- COTS
- Open source code

Reduces Verification / Validation Time – Fielding Time

Reduces Verification Cost

Outcomes

Real-time schedulers

- Mixed-Trust Scheduler
- Uber XMHF hypervisor

Verification algorithms

- Mixed Trust Timing Verification
- UberSpark hyperapps verification framework

Experimental Platforms

- Drone Laboratory
- Demos: virtual fence, minimum separation

Publications

- Dionisio de Niz, Bjorn Andersson, and Gabriel Moreno,” Safety Enforcement for the Verification of Autonomous Systems,” SPIE Conference on Autonomous Systems. 2018
- Amit Vasudevan, Sagar Chaki, “Have Your PI and Eat it Too: Practical Security on a Low-Cost Ubiquitous Computing Platform,” EuroS&P 2018
- Bjorn Andersson, Sagar Chaki, and Dionisio de Niz, “Combining Symbolic Runtime Enforcers for Cyber-Physical Systems,” International Conference in Runtime Verification. 2017
- Sagar Chaki and Dioniso de Niz, “Certifiable Runtime Assurance of Distributed Real-Time Systems,” AIAA Information Systems-AIAA Infotech @ Aerospace. 2017.

Engagements

- AFRL PWP, ONR PWP

Are we done yet?

Oversimplification of physical interaction with environment

- Simple to stop a quadrotor
- Contrast: jet-fighter cannot be stopped mid-air or easily deviated from a trajectory

Simplified enforcer composition

- Tradeoffs in relaxing assumptions/guarantees

Tradeoffs between mission performance / safety

- How do I avoid a safe drone that does not move