SEI CPS Projects

Presenter: Dionisio de Niz

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.

DM17-0812

Implementations – Drone

YOLO reboots on Drone

- Snapshot at fully booted state
- Reboots and returns to saved snapshot.

Reboots at frequent intervals (1 sec).

- Tested on physical drone in controlled flight

Experimentation

Drone Flight Simulation Environment

- Simulates flight physical process
- Allows extreme experiments w/o physical loss

Initial simulation of reboots in extreme maneuvers

- Frequent reboots
- Stable in gentle manual flight
- Extreme maneuvers can lead to crash
- To be performed:
  - Evaluating fidelity of simulation to physical reality (drone)

Micro-Reboots

Reboot only part of the system

- Process
- Checkpoint & Rollback
  - Checkpoint: save clean state to protected space
  - Rollback: erase corrupted state and replace with saved state

Prototype

- Linux kernel module
- Saves checkpoint (memory backup) in kernel space
  - Including processor state (Instruction Pointers, CPU registers)
  - Cannot be modified from user code (if user code compromised)
- Rollback
  - Copies back saved kernel copy into process memory
  - Restores previous processor state (Instruction Pointer, CPU registers)

Recoverable / Non-Recoverable State (1)

Recoverable CPS physical process state

- Acquired through sensor readings (e.g. position, velocity)
- Recoverable through sensor readings

Non-Recoverable State

- Mission state (next waypoint)

Micro-reboots

- Rebooted part out of sync with non-rebooted
  - Drone: rebooted part must read time at checkpoint time

Recoverable / Non-Recoverable State (2)

Persistent state

- Preserved across rollbacks
  - Time at checkpoint
  - Mission state: next waypoint

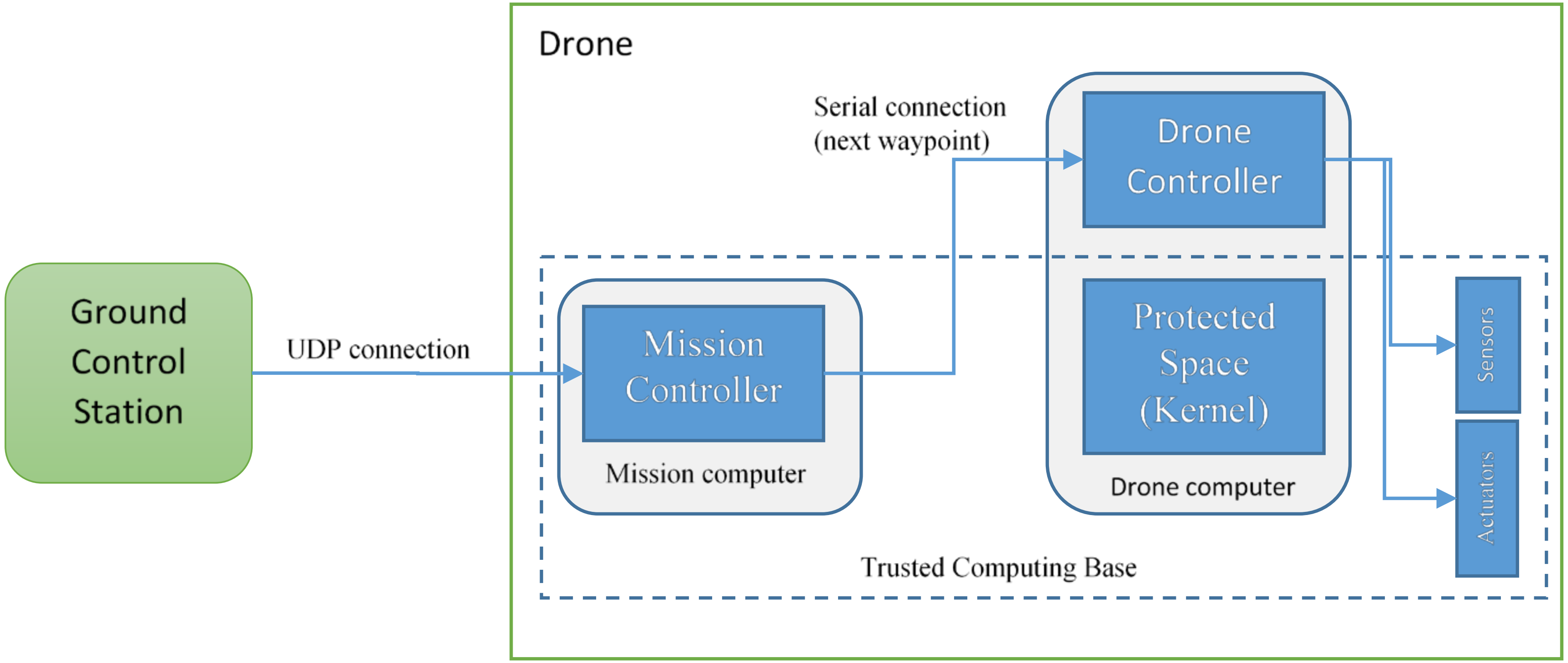
Persistent state protection

- Identify / implement Trusted Computer Base (TCB)
  - Tamper-proof hardware
  - Kernel space
  - Hypervisor

Place in TCB

- Saved state
- Rollback code

Drone Architecture



Threat Model

Attack Surface

- Code, Data, Stack of Drone Controller

Sample Attack Vector

- IP Connection: Can inject data and break into drone controller

Attack Lifetime

- Period of incubation (e.g. zero-day attack)
  - Reboot before incubation ends

Attacker Capabilities

- Observation : communication, variables, sensor, actuation
- Modification: communication, controller memory

Verifying Resilient System

Logic

- TCB correctness
  - XMHF / UberSpark
  - ZSRMV

Timing

- Can recover state fast enough (enough inertia)
- Reboot + rollback action finish on time
  - Modified deadline
  - New scheduling model

Control

- Reboot frequency does not destabilize system
- Recovered + Save state leads to stable behavior

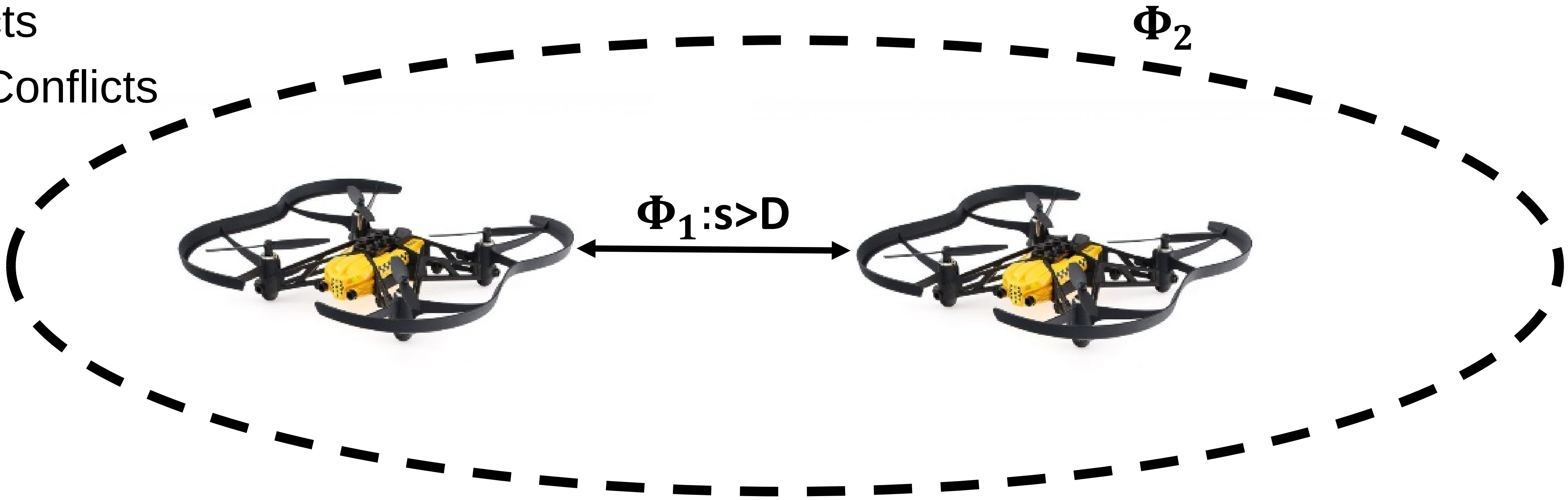
Certifiable Distributed Runtime Assurance

Constraining Behavior (satisfies Φ) with Enforcers

- Verifiable Constrained Behavior
- Verifiable Enforcer Implementation

Multiple Enforcers Φ_1, Φ_2:

- Identify Conflicts
- Resolution of Conflicts

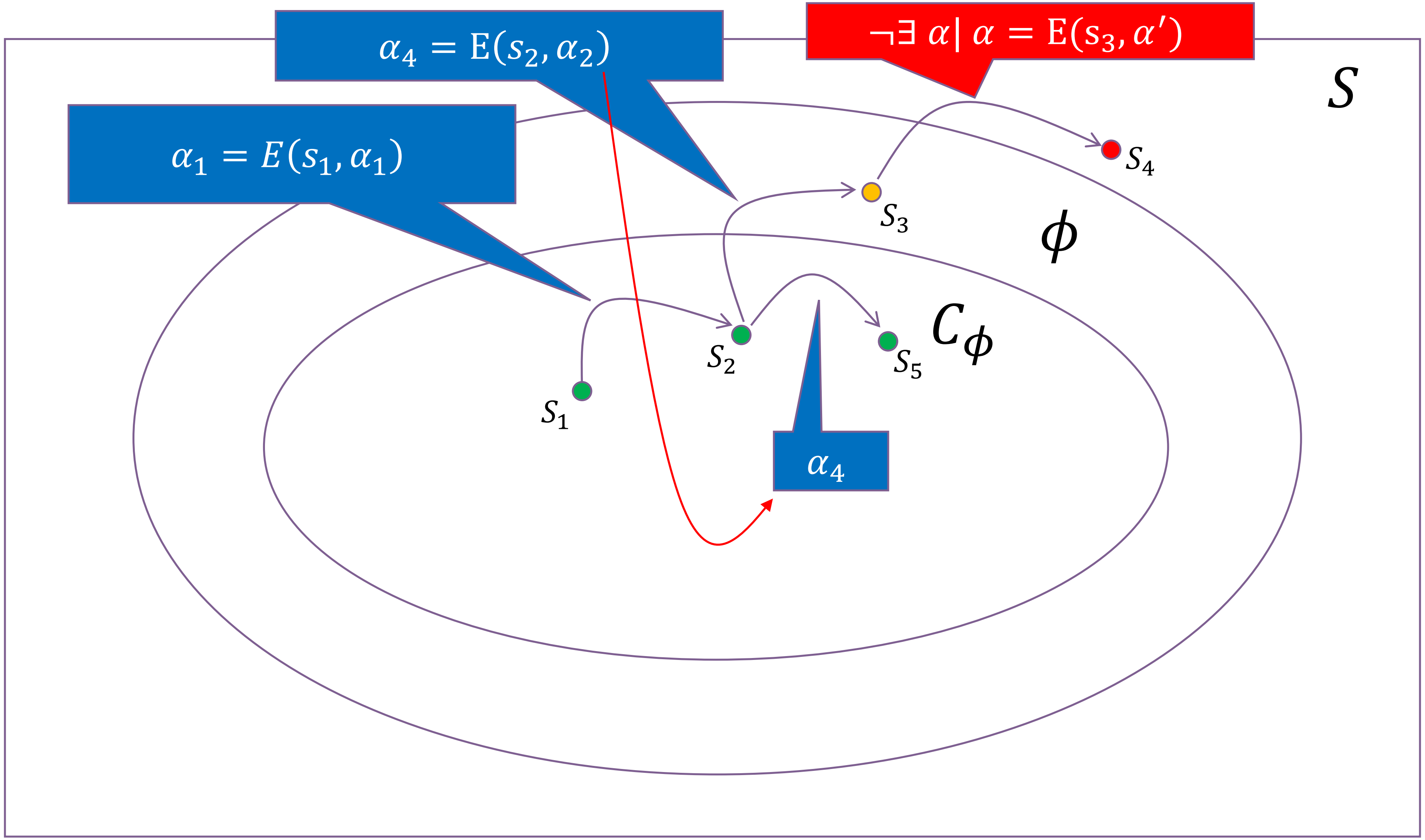


Formal Periodic Model: Representing Time-Aware Logic

State of the system: values of variables

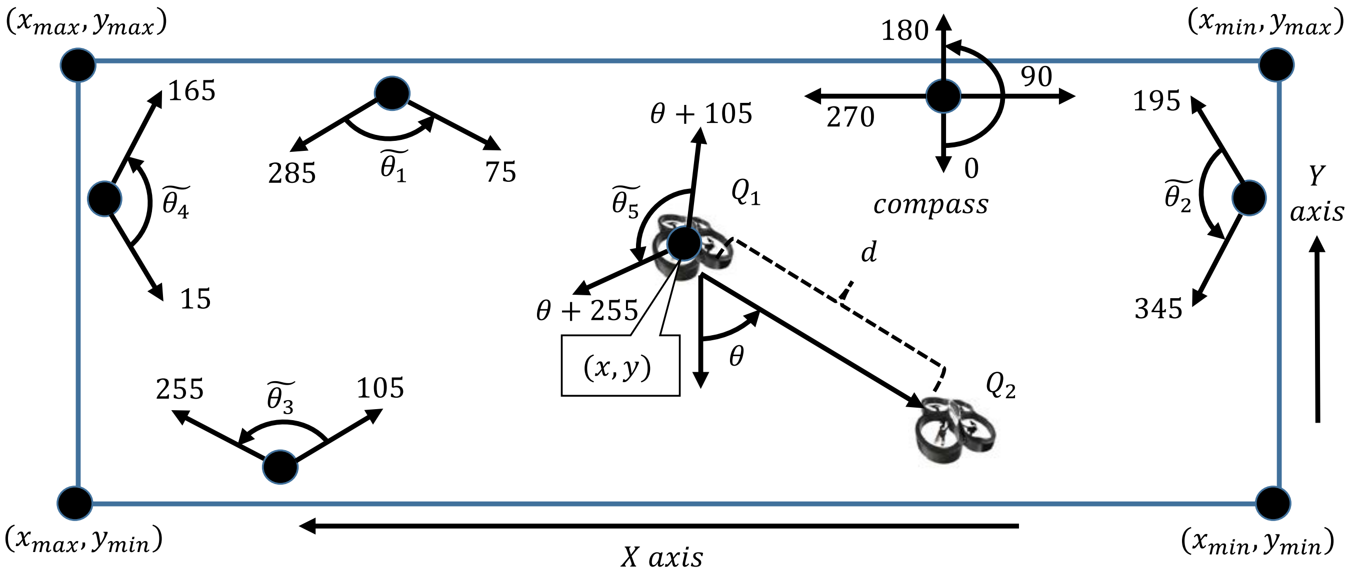
- State variables: V_S
- Action variables: V_Σ
- Variable values from domain: D
- System state: state variable: $s: V_S \mapsto D \in S$
- Actions: action variables valuations: $\alpha: V_\Sigma \mapsto D$
- Behavior: state transitions given actuation <u>every period</u> P: $R_P(\alpha) \subseteq S \times S$
  - Next state given action: $R_P(\alpha, s) = \{s' | (s, s') \in R_P(\alpha)\}$
- <u>Property to verify</u> subset of all possible states: $\phi \subseteq S$
- Enforceable state: $C_\phi \subseteq \phi \wedge C_\phi = \{s \mid \exists \alpha \in \Sigma: \mathrm{R_P}(\alpha, s) \in C_\phi\}$
- Safe actuation : $SafeAct(s) = \{\alpha | R_P(\alpha, s) \in C_\phi\}$

Enforcer



$\alpha_4 = \mathrm{E}(s_2, \alpha_2)$
$\neg\exists\ \alpha |\ \alpha = \mathrm{E}(s_3, \alpha')$
S
$\alpha_1 = E(s_1, \alpha_1)$
s_4
s_3
ϕ
C_ϕ
s_2
s_5
s_1
α_4

Example



Quadrotors Q_1, Q_2

State Variables: $V_S = \{x, y, \theta, d\}$

Action $V_\Sigma = \{\theta_\alpha\}$: move in direction θ_α

Z: Virtual Fence Zone

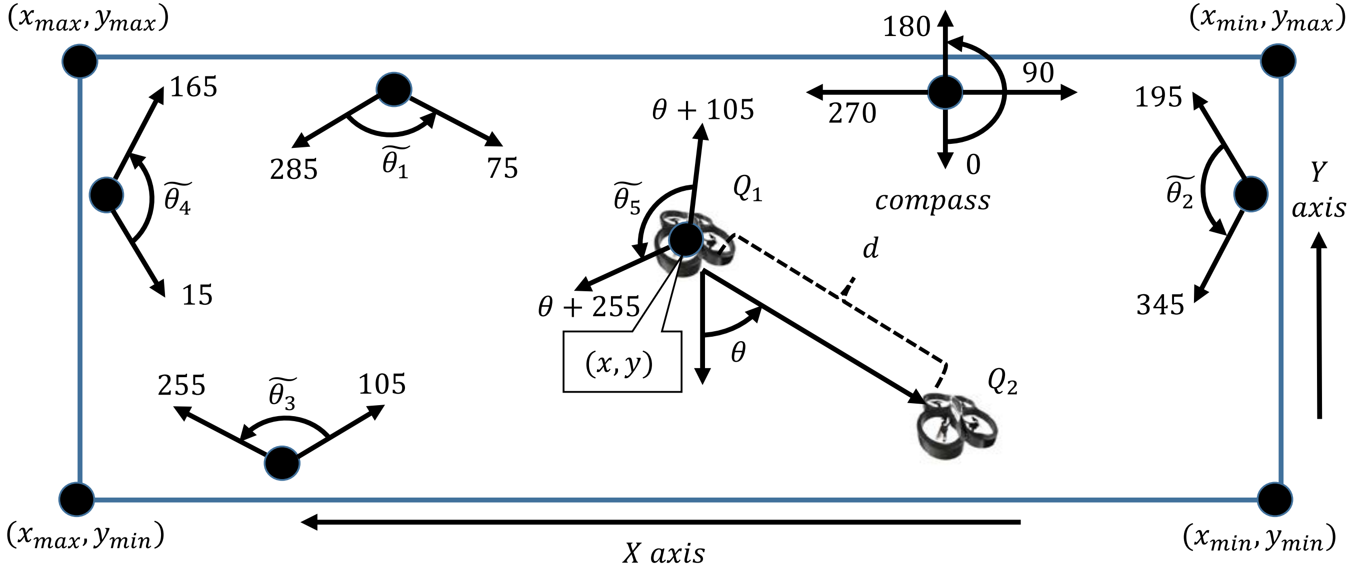
$C_{\phi_1} = \{(x, y, \theta, d) | (x + \delta_{B1}, y + \delta_{B1}) \in Z \wedge (x - \delta_{B1}, y - \delta_{B1}) \in Z\}$

- δ_{B1}: braking distance

$C_{\phi_2} = \{(x, y, \theta, d) | d + \delta_{B2} \geq D\}$

- δ_{B2}: largest reduction in d once separation enforcement applied

Example: Utility Enforcers



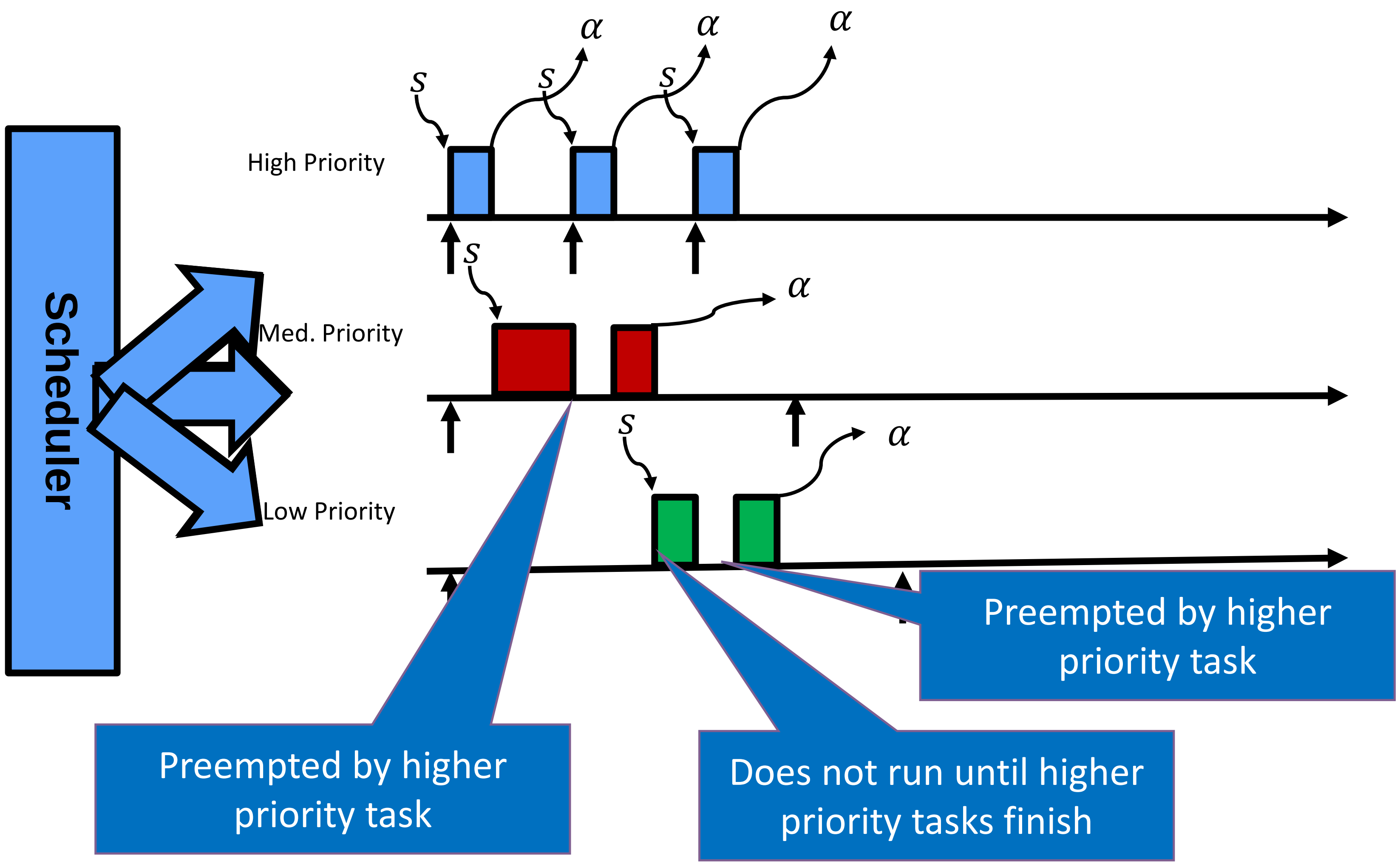
Angle Operations: $\theta \ominus \theta'$: min angular *distance* θ to θ',θ^{opp}: opposite angle to θ

Fence enforcer: $U_1(x, y, \theta, d, \theta_\alpha) = U^1 + U^2 + U^3 + U^4$ where:

- $U^i = 75 - \left(\theta^i_{mid} \ominus \theta_\alpha\right)$ if b_i 0 otherwise with:
  - $b_1 \equiv y_{max} - y \leq \delta_{PB1}, \mathrm{b}_2 \equiv x - x_{min} \leq \delta_{PB1}, b_3 \equiv y - y_{min} \leq \delta_{PB1}, b_4 \equiv x_{max} - x \leq \delta_{PB1}$
  - $\theta^1_{mid} = 0, \theta^2_{mid} = 270, \theta^3_{mid} = 180, \theta^4_{mid} = 90$

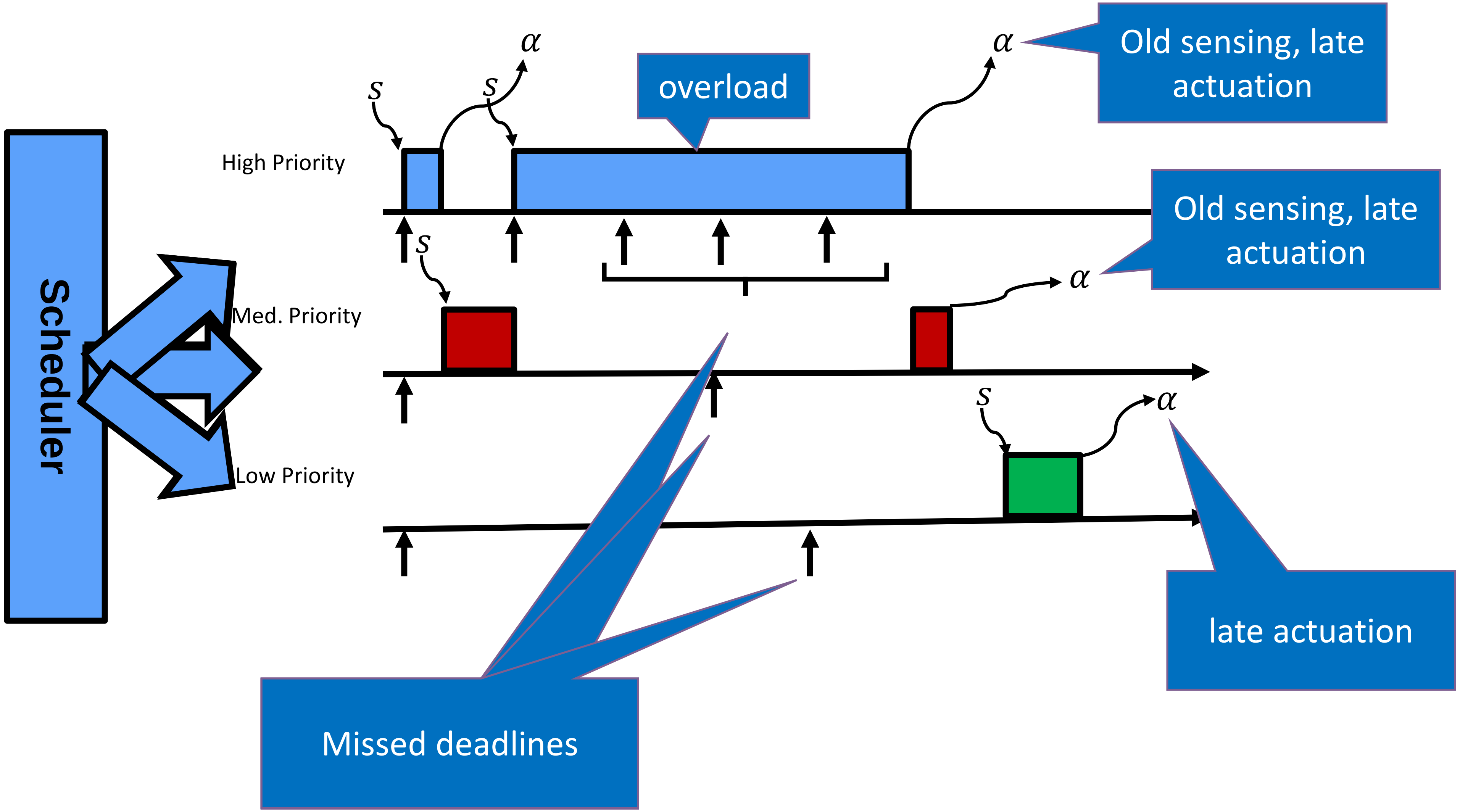
Separation enforcer: $U_2(x, y, \theta, d, \theta_\alpha) = 75\ - (\theta^{opp} \ominus \theta_\alpha)$ if b_{sep} 0 otherwise

Primer: Fixed-Priority Scheduling + Rate Monotonic



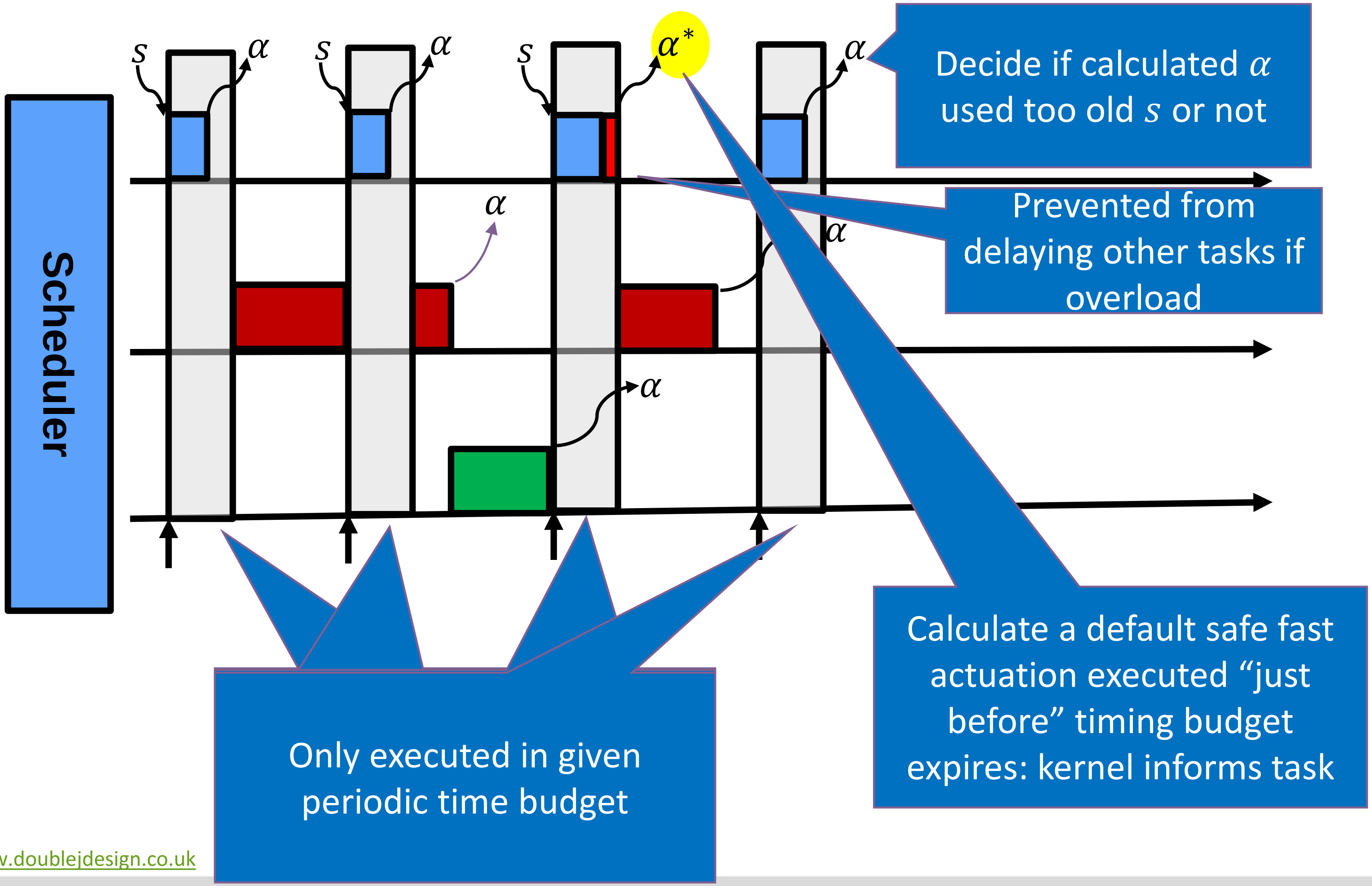
Icons credit: http://www.doublejdesign.co.uk

Overload -> old sensed data + late actuation



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

Solution step 2: safe actuation on timing enforcement



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