Mixed-Trust Computing for Real-Time Systems

Dionisio de Niz[*], Bjorn Andersson[*], Mark Klein[*], John Lehoczky[*], Amit Vasudevan[*], Hyoseung Kim[†], and Gabriel Moreno[*]

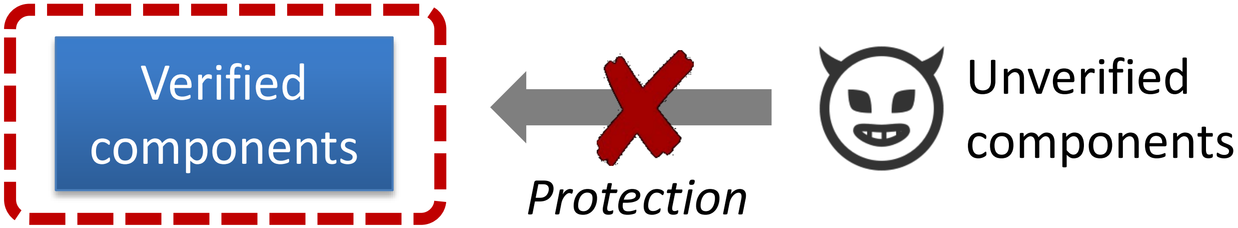
[*] Carnegie Mellon University

[†] University of California, Riverside

“Trust” in Safety-Critical Systems

- Verification via formal methods
  - Critical components, OSs, libraries, etc.

- Verified properties can be easily compromised if the verified components are *not* protected from unverified ones



Our notion of “Trust”

- **Both verification** and **protection** should be jointly considered

Challenges

- The complexity of today's OSs making them impractical to verify
- Alternative: minimize the trusted computing base (TCB) by developing small verified hypervisors (HVs) and microkernels
  - e.g., seL4, CertiKOS, and uberXMHF

Trusted parts in TCB

- Made small and simple due to verification difficulties
- Isolated from untrusted parts

Untrusted parts in VM

- Hosted in a virtual machine (VM)
- Implements rich functionalities on full-scale OSs, e.g., Linux

Disjoint-trust computing: trusted and untrusted parts co-exist but in a completely isolated and disjoint manner

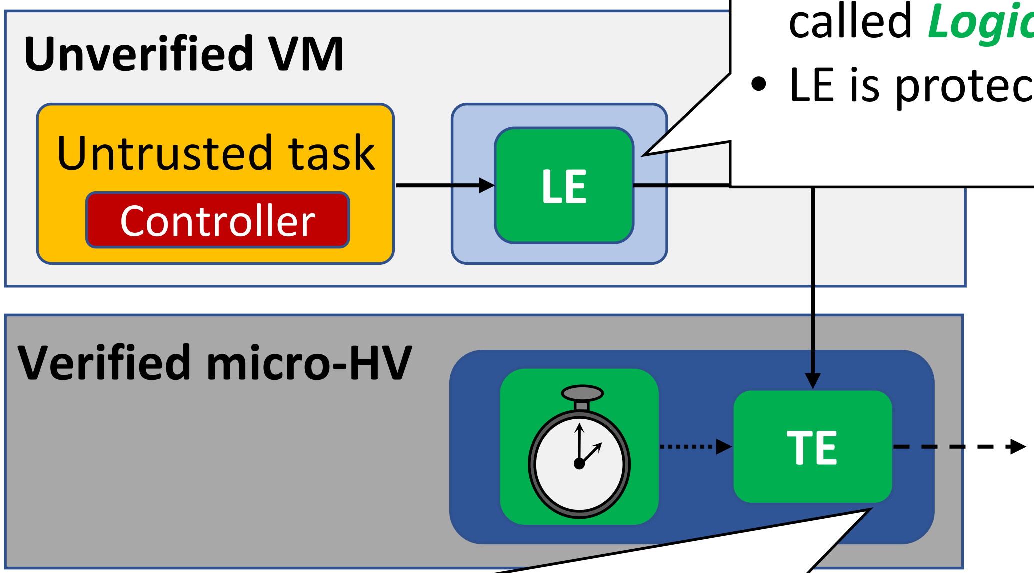
Limitations of Disjoint-Trust Computing

- Does *not* allow the use of untrusted components in critical functionality where safety must be assured through verification
  - ∵ The verified components must be isolated from the untrusted ones if they are to be trusted

- Example: self-driving car
  - Prevents untrusted machine learning algorithms to drive a car if such functionality needs to be verified
    - Very difficult or practically impossible to verify the entire software/hardware stack, e.g., GPUs, drivers, ML libraries, frameworks, etc.
  - Instead, a separate trusted component would need to be in charge of the driving, isolating it from any untrusted component

Real-time Mixed-Trust Computing

- **Goal:** to give the flexibility to use untrusted components even for critical functionalities



- Output checked by a verified component, called ***Logical Enforcer (LE)***
- LE is protected by HV (e.g., uberapps*, PAL†)

secure memory enclaves

- ***Temporal Enforcer (TE)***: performs a default safety action if the untrusted task has not produced a correct output by a specified time

* A. Vasudevan et al. uberspark: Enforcing verifiable object abstractions for automated compositional security analysis of a hypervisor. *USENIX Security*, 2016.
† J. M. Mccune et al., TrustVisor: Efficient TCB Reduction and Attestation. *IEEE S&P*, 2010.

Contributions

- Mixed-trust software architecture
  - Interplay of two schedulers
    1. Preemptive fixed-priority scheduler in the VM
    2. Non-preemptive fixed-priority scheduler in the HV
  - Mixed-trust task model & analysis

- Design of a mixed-trust coordination protocol
  - Preserves timing dependencies between trusted and untrusted parts
  - Prevents logical dependencies that can compromise the trusted part

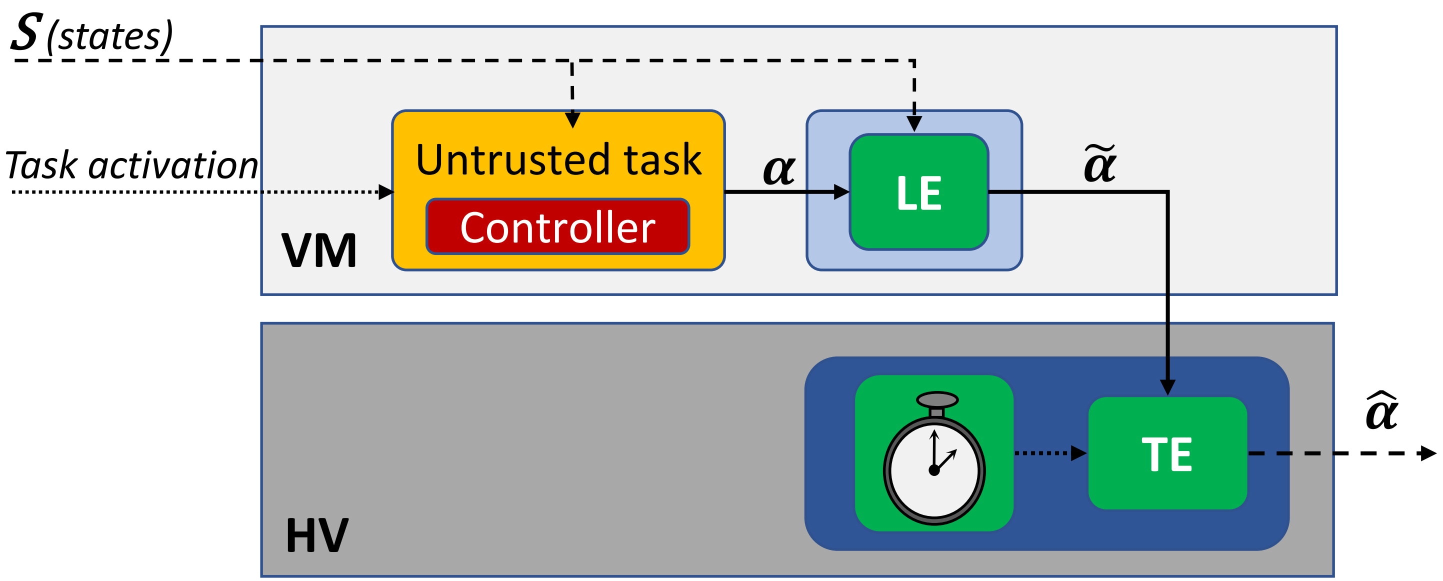
- Implementation in the uberXMHF hypervisor* on Raspberry Pi

* A. Vasudevan and S. Chaki. Have your PI and eat it too: Practical security on a low-cost ubiquitous computing platform. IEEE Euro S&P, 2018.

Outline

- Introduction
  - Motivation & Limitations
  - Overview
- Mixed-trust computing
  - Logical model and protection domains
  - Mixed-trust task scheduling and analysis
  - Fail-safe coordination protocol
- Case study results
- Conclusions

Logical Model of Mixed-Trust Computing



- LE-enforced action $\tilde{\alpha} = \begin{cases} \alpha & \mathit{if}\ \ \alpha \in \boxed{\mu(s)} \\ \mathrm{pick}(\mu(s)) & \mathit{otherwise} \end{cases}$ *Safe action for state s*

- TE-enforced action $\hat{\alpha} = \begin{cases} \boxed{\alpha_T \quad \mathit{if}\ \ \tilde{\alpha} = \perp} \\ \tilde{\alpha} \quad \mathit{otherwise} \end{cases}$ *Default safe action for any state if no output generated by a specific time **E***

Conditions Required by Logical Model

- To prevent an untrusted component from causing behaviors not present in the logical model

C1. Each task must produce an output every period

C2. There is only one output per period

C3. The output produced by a task in a period is either from LE or TE

C4. An output produced by the task and validated by the LE must be the product of a computation that executes within a single period

- i.e., sensing, computing, and output should be done within the same period

C5. The TE of a task must execute *E* time units after the arrival of the job it guards and finish before the end of the period

Behavior of Periodic Mixed-Trust Task

while *true* **do**
    try:
        $s \leftarrow \mathrm{currentState}()$
        $\alpha \leftarrow \mathrm{computeActuation}(s)$
        $\tilde{\alpha} \leftarrow LE(s, \alpha)$
        $\mathrm{actuate}(\tilde{\alpha})$
    catch *timeout(E)*:
        $\mathrm{actuate}(\alpha_T)$
    end
    $\mathrm{waitForNextPeriod}()$
end

Untrusted code

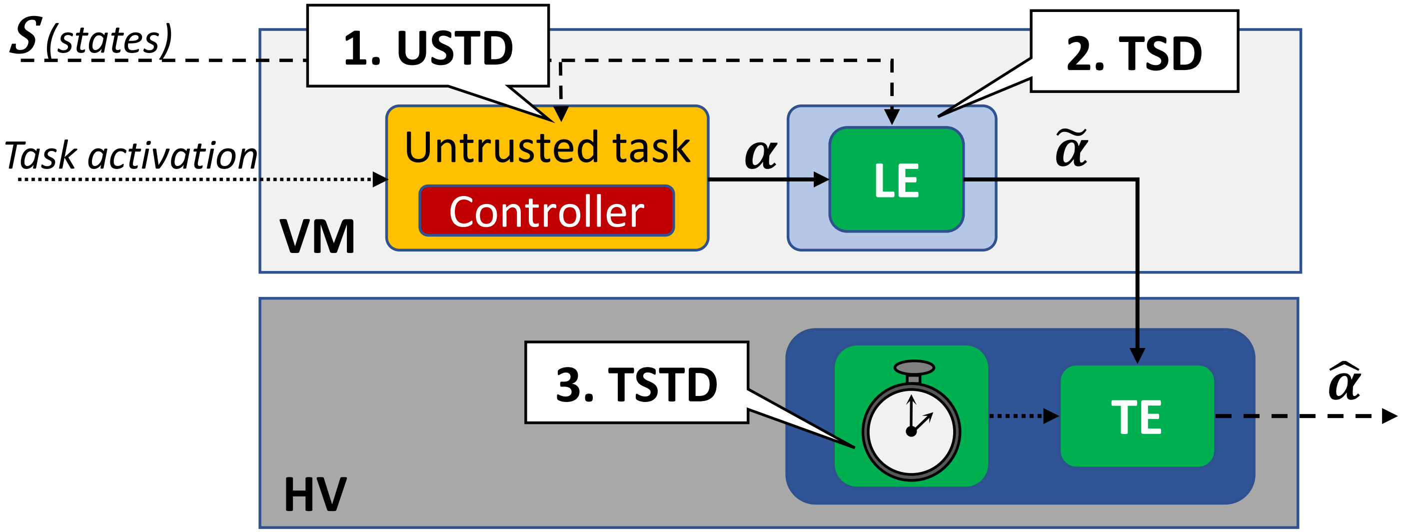
Logical enforcer

Temporal enforcer

Every period

Formally-verified LE and TE need to be protected against unintended modifications

Mixed-Trust Protection Domains



1. **Untrusted Spatio-Temporal protection Domain (USTD)**
   – Untrusted task code execution in the VM
2. **Trusted Spatial protection Domain (TSD)**
   – LE execution in secure enclaves (memory protection)
3. **Trusted Spatio-Temporal protection Domain (TSTD)**
   – TE execution in the verified HV (memory and spatial protection)

System Modeling

- Mixed-trust task $\mu_i = (T_i, D_i, \tau_i, \kappa_i)$
  - *T_i: period, D_i: deadline*
  - Two execution segments: *Guest Task τ_i* and *Hyper Task κ_i*
  1. Guest Task $\tau_i = (T_i, E_i, C_i)$
     - *C_i: worst-case execution time of τ_i*
     - *E_i: intermediate deadline for τ_i ← set by analysis such that κ_i can finish by D_i*
  2. Hyper Task $\kappa_i = (T_i, D_i, \kappa C_i)$
     - *κC_i: worst-case execution time of κ_i*

If there is no hyper-task part, $\kappa C_i = 0$

- Uniprocessor system

Preemptive scheduler in VM

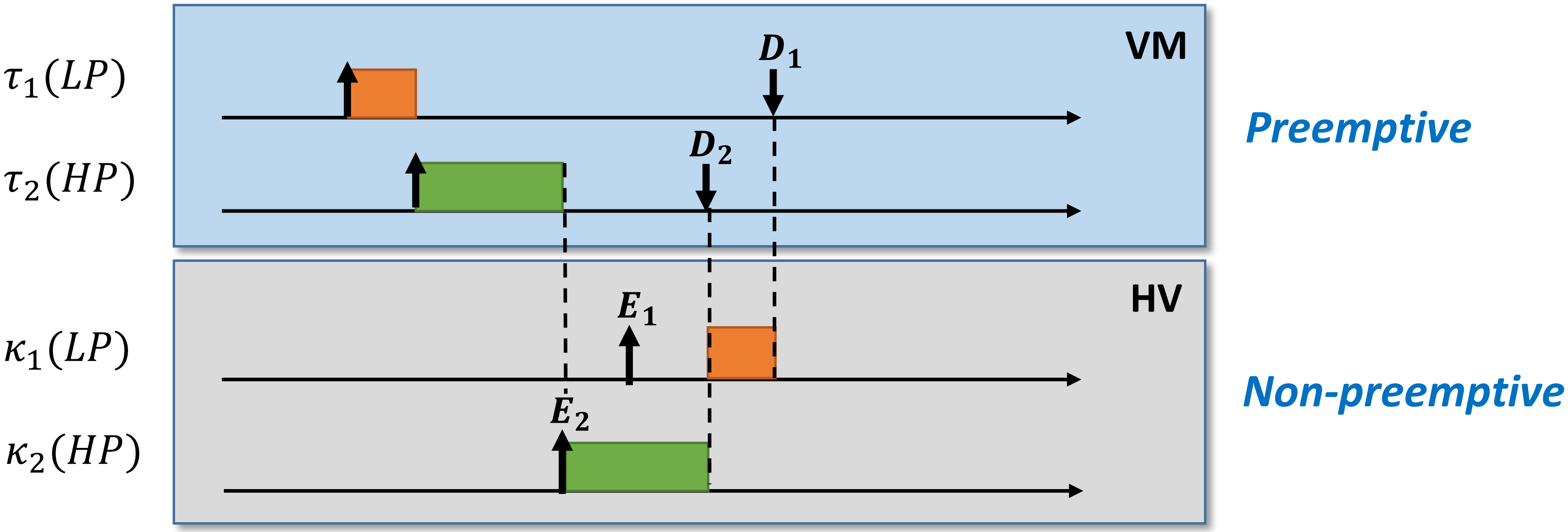
- Full-scale guest OSs, e.g., Linux

Non-preemptive scheduler in HV

- Simplifies HV logical verification by removing task interleavings*

* A. Vasudevan et al. Design, implementation and verification of an eXtensible and Modular Hypervisor Framework. *IEEE S&P*, 2013.

Mixed-Trust Task Scheduling



- How to determine E_i?
  - For hyper tasks to be schedulable, $\forall \mu_i \in \Gamma \quad E_i \leqslant D_i - \boxed{R_i^\kappa}$
  - This work uses $E_i = D_i - R_i^\kappa$ *Hyper task response time*
    - Optimal *E* assignment presented in online appendix

Hyper Task Schedulability

- Based on non-preemptive fixed priority scheduling*
  - Maximum duration of a level-i active period

$$t_i^\kappa = \max_{j \in \kappa L_i} \kappa C_j + \left\lceil \frac{t_i^\kappa}{T_i} \right\rceil \kappa C_i + \sum_{j \in \kappa H_i} \left\lceil \frac{t_i^\kappa}{T_j} \right\rceil \kappa C_j$$

  - Latest starting time of $\kappa_{i,q}$ in the level-i active period

$$w_{i,q}^\kappa = \max_{j \in \kappa L_i} \kappa C_j + (q-1)\kappa C_i + \sum_{j \in \kappa H_i} (\left\lfloor \frac{w_{i,q}^\kappa}{T_j} \right\rfloor + 1)\kappa C_j$$

  - Worst-case response time of κ_i

$$R_i^\kappa = \max_{q \in \{1 \ldots \lceil \frac{t_i^\kappa}{T_i} \rceil\}} (w_{i,q}^\kappa + \kappa C_i - (q-1)T_i)$$

- The corresponding guest task τ_i's deadline $E_i = D_i - R_i^\kappa$

* R.I. Davis, A. Burns, R.J. Bril, and J.J. Lukkien. Controller area network (CAN) schedulability analysis: Refuted, revisited and revised. Real-Time Systems, 2007.

Guest Task Schedulability

- Level-i busy period (BP) for a guest task τ_i
  - Processor is busy with
    - Hyper tasks (HTs)
    - Guest tasks (GTs) with higher priority than τ_i

Theorem IV.1. *The longest response time for all GT jobs of task τ_i of a mixed-trust task μ_i occurs in a level-i busy period initiated by the arrival of either τ_i or κ_i and the arrival of higher-priority GTs or HTs of other mixed trust tasks, μ_j.*

- Request-bound function

$$\mathrm{rbf}_i^y(t,b) = \begin{cases} \left\lceil \frac{t-(T_i-E_i)}{T_i} \right\rceil^+ C_i b + \left\lceil \frac{t}{T_i} \right\rceil \kappa C_i & \text{if } y = E, \\ \left\lceil \frac{t}{T_i} \right\rceil C_i b + \left\lceil \frac{t-E_i}{T_i} \right\rceil^+ \kappa C_i & \text{if } y = A, \end{cases}$$

BP starts with HT (case $y = E$)

BP starts with GT (case $y = A$)

Guest Task Schedulability (cont'd)

- Maximum level-i busy period

$$t_i^{g,x} = \left(\sum_{j \in L_i} \mathrm{rbf}_j^E(t_i^{g,x}, 0) \right) + \mathrm{rbf}_i^x(t_i^{g,x}, 1)$$
$$+ \sum_{j \in H_i} \max_{y \in \{E,A\}} \mathrm{rbf}_j^y(t_i^{g,x}, 1).$$

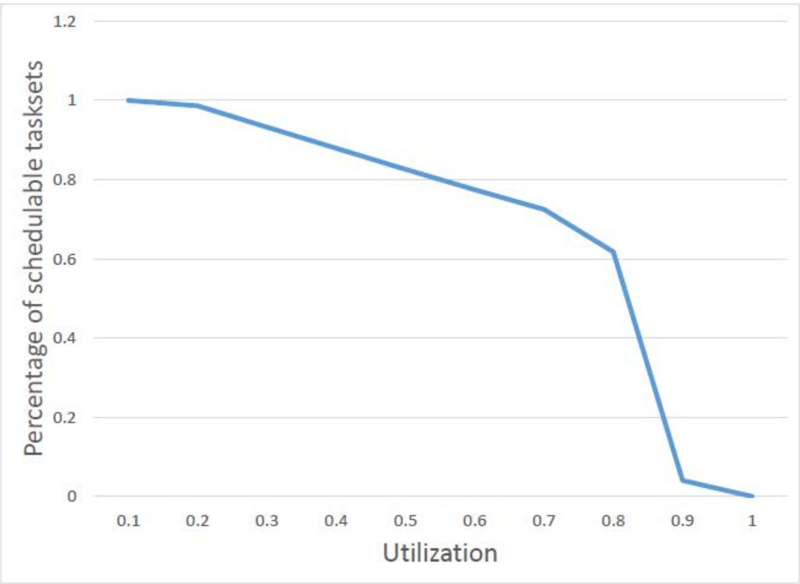
- Maximum finishing time of $\tau_{i,q}$ in the level-i busy period

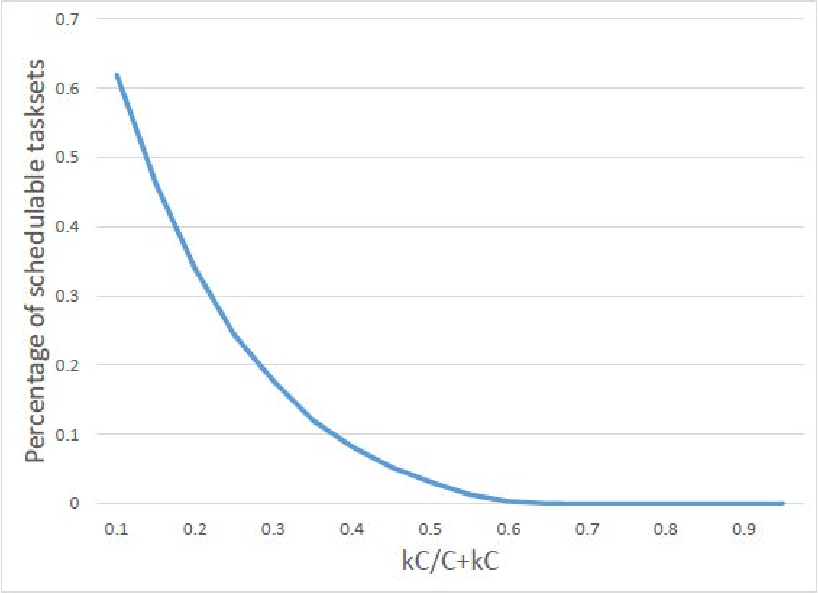
$$w_{i,q}^{g,x} = \left(\sum_{j \in L_i} \mathrm{rbf}_j^E(w_{i,q}^{g,x}, 0) \right) + qC_i + (q - 1 + I_{(x=E)})\kappa C_i$$
$$+ \sum_{j \in H_i} \max_{y \in \{E,A\}} \mathrm{rbf}_j^y(w_{i,q}^{g,x}, 1).$$

- Worst-case response time of τ_i: $R_{i,q}^{g,x} = w_{i,q}^{g,x} - ((q-1)T_i + I_{(x=E)}(T_i - E_i))$

$$R_i^{g,x} = \max_{q \in \left\{1 \ldots \left\lceil \frac{t_i^{g,x} - I_{x=E}(T_i - E_i)}{T_i} \right\rceil \right\}} R_{i,q}^{g,x}$$

Experiments





- Impact of HTs
  - A GT can experience delay from the HTs of other tasks
  - A HT can experience delay from the HTs of other tasks

double-accounting effect

- But HTs are made small for verifiability

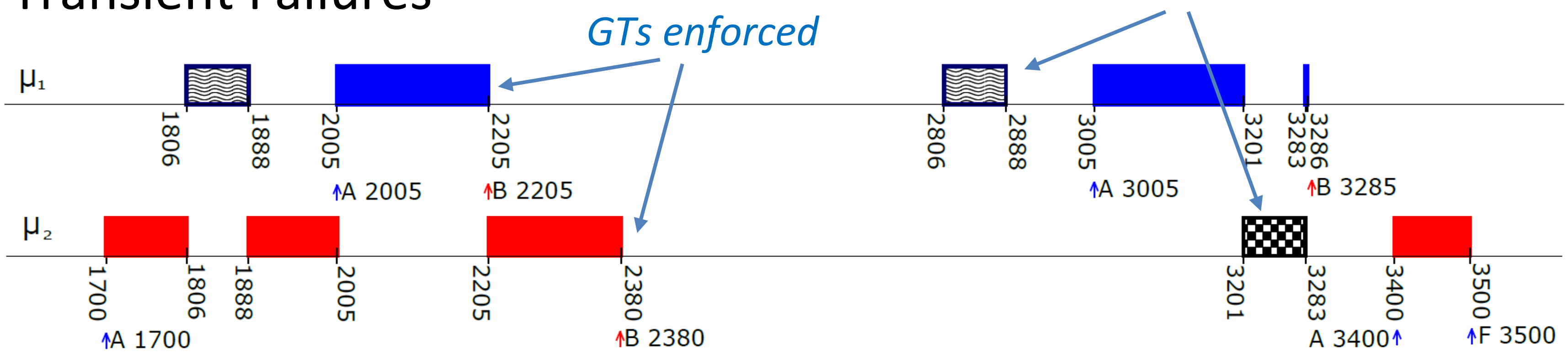
Fail-safe Coordination Protocols

- To prevent any dependency of trusted code from untrusted code

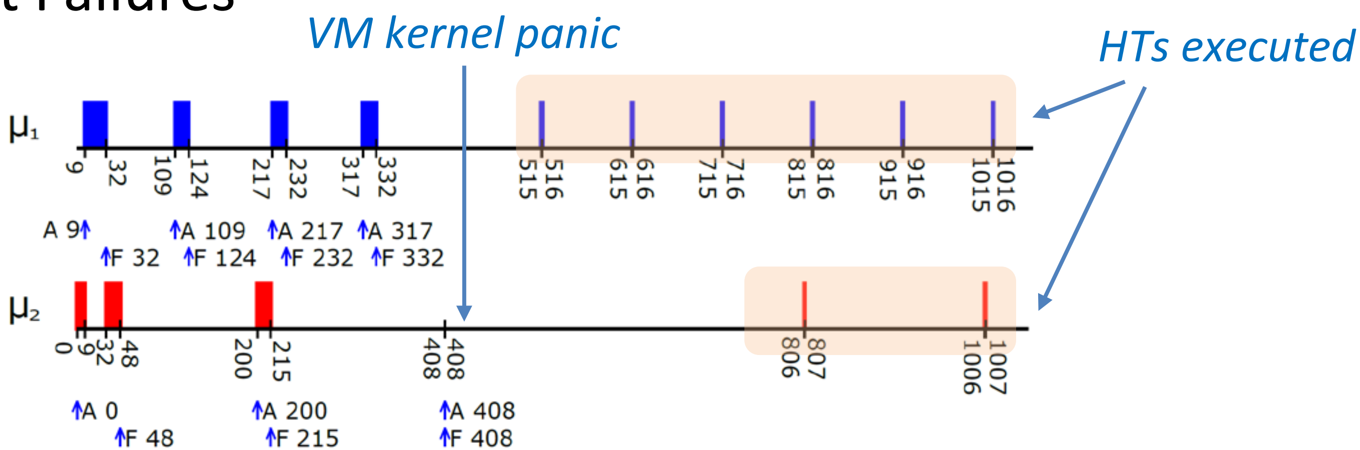
1. Secure HT Bootstrapping
   - Required for **Trusted Spatio-Temporal protection Domain (TSTD)**
   - Ensures that HTs can start and execute periodically even if the VM is unable to run GTs
2. Fail-Safe HT Triggering
   - Prevents a failure in the VM from disabling or corrupting the periodic arrival of HTs
3. Late-Output Prevention
   - Prevents the output of a GT τ_i after its deadline E_i

Case Study: Temporal Failure Scenarios

- Implemented in uberXMHF on Raspberry Pi 3
- Transient Failures

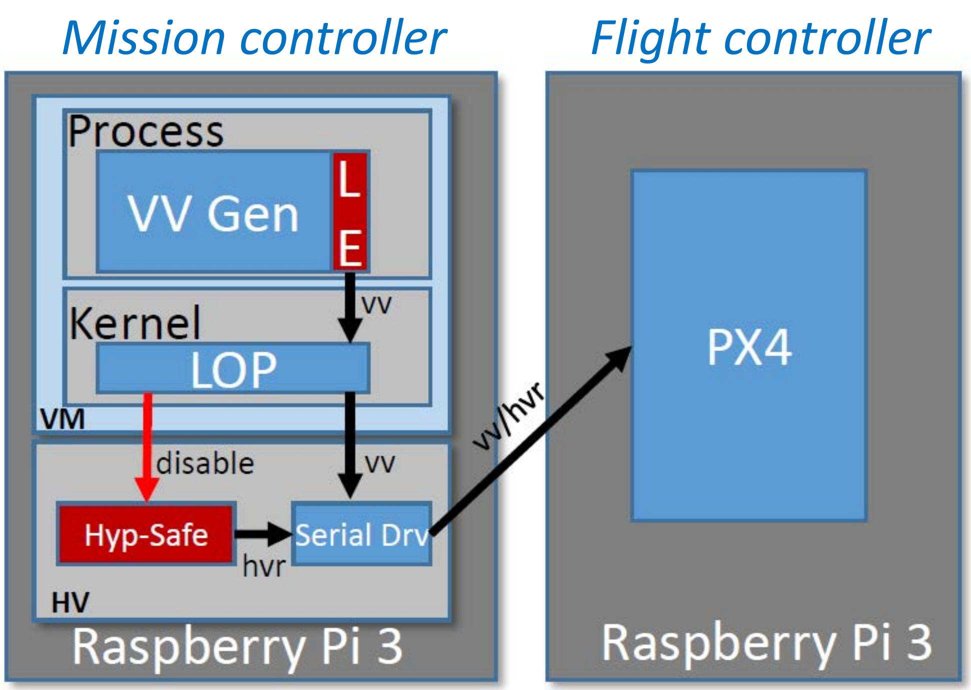


- Permanent Failures



Case Study: Drone Application

- Mission controller: sends velocity vectors (VV) to Flight controller
  - Guest task (VV Gen) generates velocity vectors
  - Hyper task (Hyp-Safe) generates the safe drone action
- Tested with hardware-in-the-loop simulation



Conclusions

- Both *protection* and *verification* are required for the safe use of untrusted components in critical functions
- Real-time mixed-trust computing
  - First framework that satisfies these two requirements
    1. Using trusted components to monitor and replace unsafe untrusted component outputs with safe ones
    2. Protecting the logical and temporal behavior of trusted components
  - Mixed-trust task
    - *guest task* (untrusted component & logical enforcer)
    - *hyper task* (temporal enforcer)
- Prototype implementation in the uberXHMF hypervisor
- Tested with transient and permanent failures

Thank You

Mixed-Trust Computing for Real-Time Systems

Dionisio de Niz[*], Bjorn Andersson[*], Mark Klein[*], John Lehoczky[*], Amit Vasudevan[*], Hyoseung Kim[†], and Gabriel Moreno[*]

[*] Carnegie Mellon University

[†] University of California, Riverside