Vigilare : 시스템 무결성을 보장하는 하드웨어기반 상시 보안감시자를 목표로

Yunheung Paek
S(W/oC/ecurity) Optimizations and Restructuring
Seoul National University

2012년 하계 ERC Workshop

Contents

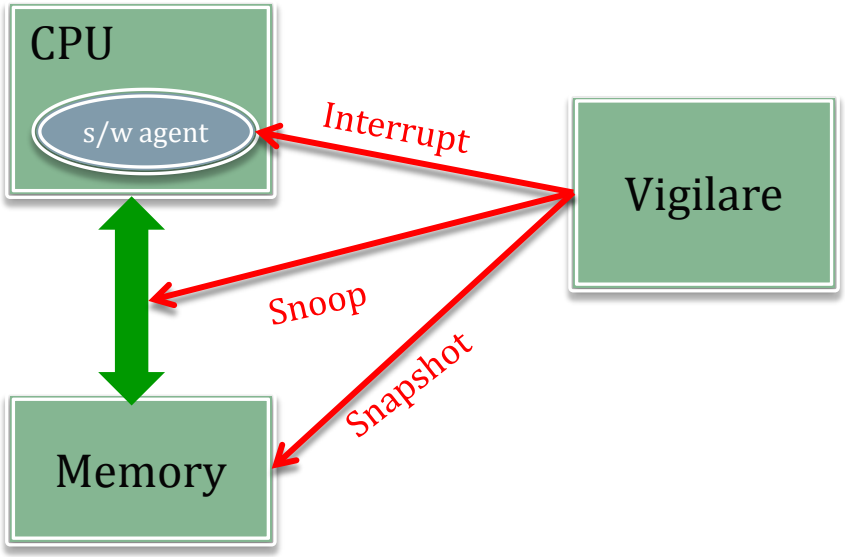
- Introduction
- Implementation of Vigilare version 1
  - Attack Model
  - Design and Implementation
  - Evaluation
- Our roadmap
  - Short-term goals
  - Long-term goals
- Conclusion

Our research goal

- 전통적인 3가지 Security types
  - Confidentiality
    - 허락 되지 않은 사용자가 정보의 내용을 알 수 없도록 하는 것
  - Integrity
    - 허락 되지 않은 사용자가 정보를 함부로 수정할 수 없도록 하는 것
  - Availability
    - 허락된 사용자가 정보에 접근하려 하고자 할 때 방해 받지 않도록 하는 것
- 우리 연구 주제: 어떻게 보안성을 유지 할 것인가?
  - *Purely software based monitoring system*
  - *Hardware based or hardware supported monitoring system*

Vigilare

- *Hardware based integrity monitoring system*
- Missions of Vigilare
  - Host system(CPU, memory)에서 일어나는 행동을 항상 감시
  - 감시 기능의 customized & distributed 연산으로 저전력 고성능 실현
  - Host의 보안 감시 기능을 isolated 환경에서 실행해서 safer 감시 실현

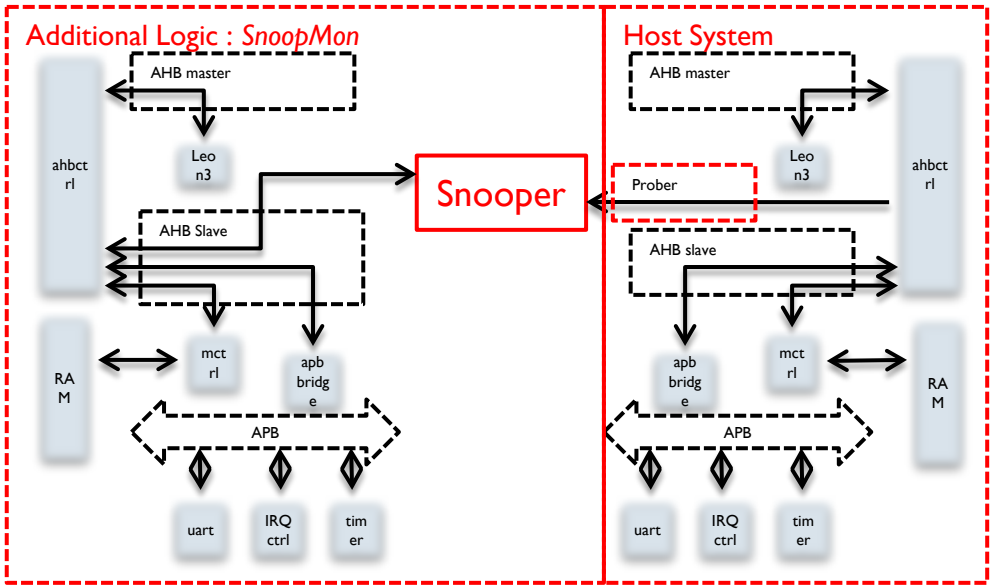


Code errors

- Computer system에서 발생할 수 있는 오류의 원인
  - ➔ Anything that causes abnormal behavior of SW or HW
  - Accidental & inadvertent Error
    - Software Bug
    - Hardware 오류 – soft/hard error
  - Intentional, malicious Error
    - 악성코드(Malware)의 공격
- Vigilare 미션의 재해석
  - ➔ 어떻게 악성코드에 의한 오류 감시 및 방지?

초기 연구 결과 (2011 가을 – 2012 봄)

- Vigilare version 1 구현
  - Hardware design and implementation on FPGA board
  - Malware attack model Implementation and test on Vigilare v.1



- 논문 게제

  Vigilare : Toward Snoop-based Kernel Integrity Monitor
  - H. Moon, H. Lee, J. Lee, K. Kim, Y. Paek and B.B. Kang
  - Conference on Computer and Communications Security (**CCS**), 2012

    ➔ Top conference in ACM SIGSAC group with acceptance rate < 20%

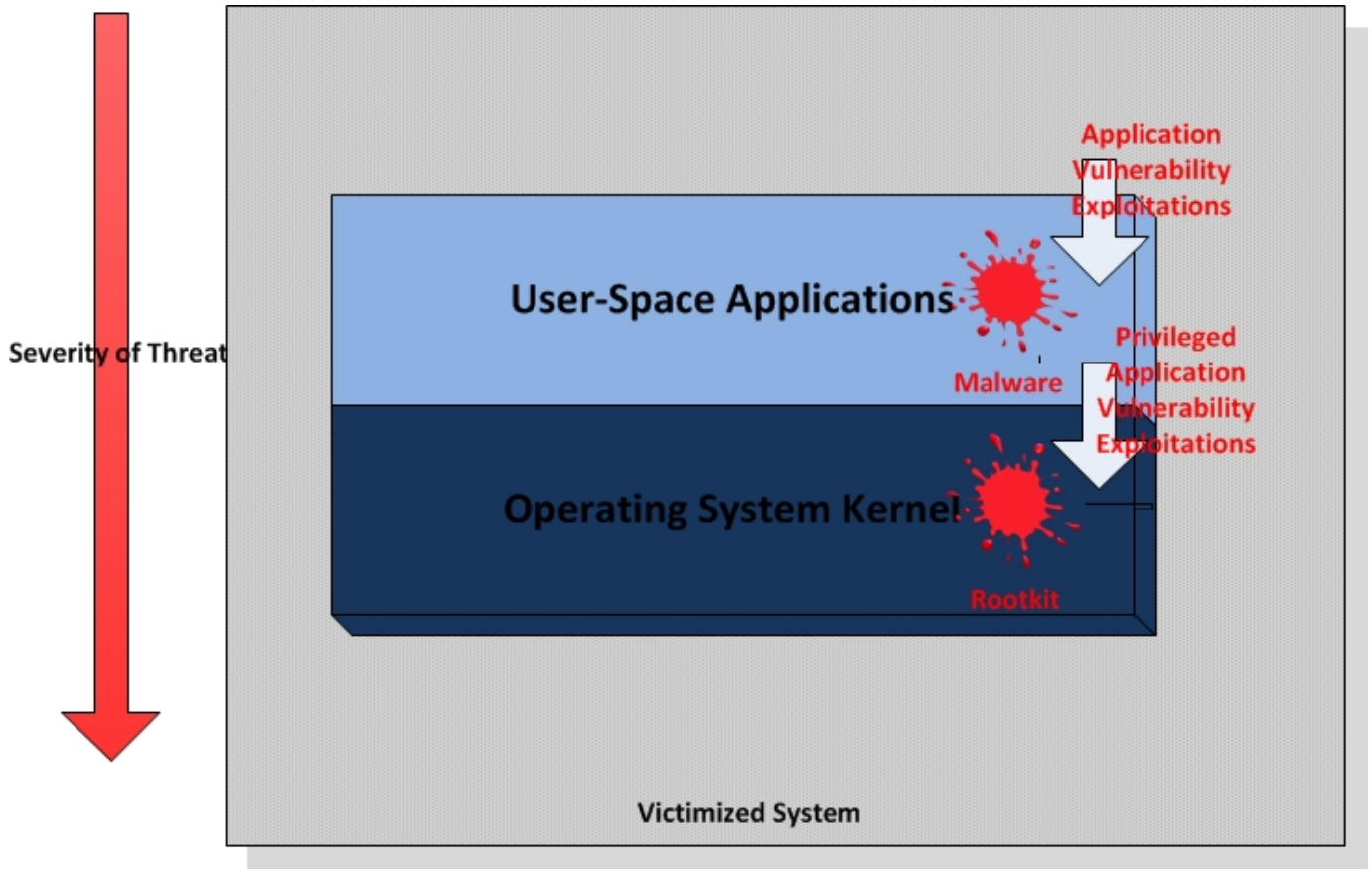
Malware attack 유형들

- Personal Identification Stealing
  - 은행 계좌 및 인증 정보
  - 게임 계정
  - 프라이버시
- Advanced Persistent Threat (APT) for Corporate Espionage
  - 기업 핵심 기술 유출
  - 고급 내부 정보 습득
- Used as Tool for Other Cybercrimes
  - DDoS ← embedding time-bomb in numerous machines to shutdown normal services with heavy workloads

공격 위치에 따른 분류

- User space
  - 수행할 수 있는 악성 행위가 제한적
  - 응용의 특성 및 동적 행위 파악이 되면 탐지가 비교적 용이
- Kernel space (Rootkit)
  - 이 region에 대한 공격 수단으로 일반적으로 **rootkit**이라는 kernel 수준의 권한을 불법적으로 획득한 소프트웨어 tool을 사용
  - 시스템이 접근 가능한 모든 장비와 커널 수준의 데이터를 임의로 접근 및 변조 가능
  - 모든 시스템 상태에 의존하는 다른 응용 등에 거짓 상태 보고 가능
  - 백신등의 탐지도구와 동등한 (또는 Android 같은 경우에는 심지어 더 우월한) 권한으로 동작
    - 백신을 무력화 가능
    - 백신의 탐지 메커니즘을 파악/회피

States of system compromise

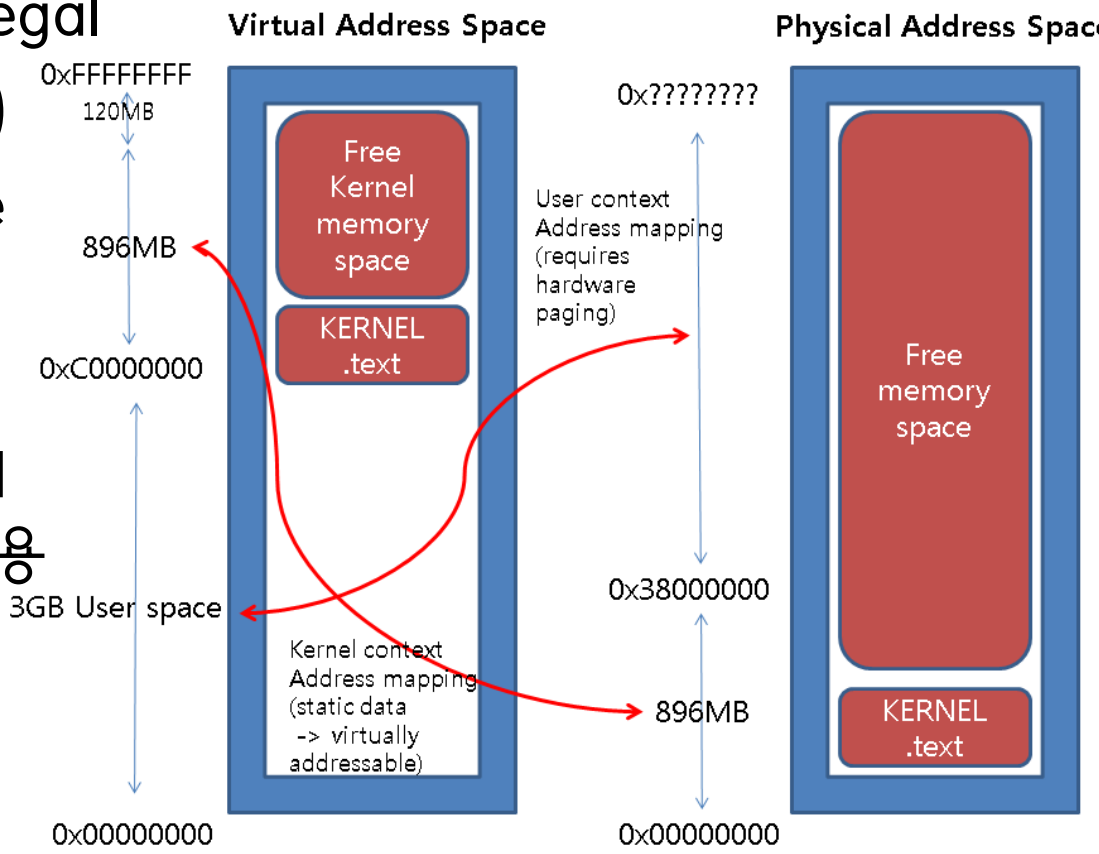


공격 위치1: Kernel Static Region

- This is immutable region, so any access/modification is illegal
- Text segment (Kernel code)
- Interrupt Descriptor Table
- Sys_call_table
  - Dispatch system calls
- Linux에서 linear-mapped kernel virtual address 사용
  - 외부 HW에서 감시 가능

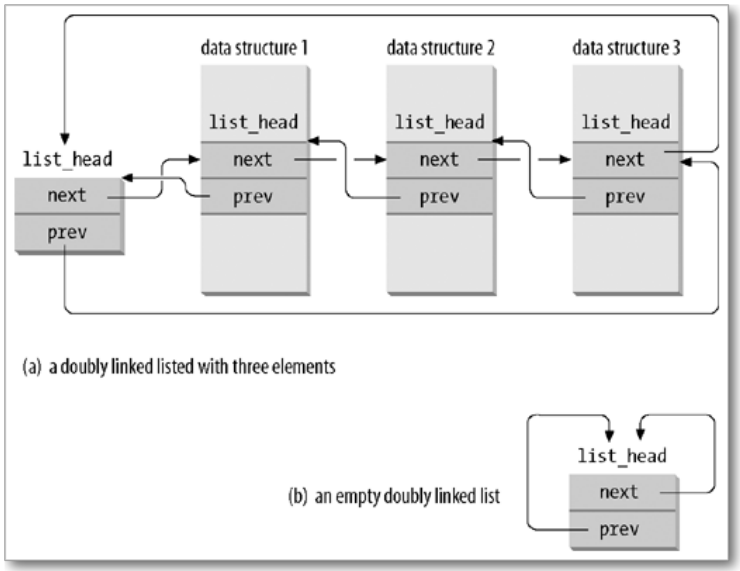
Mapped statically → simple to trace because violation is verdicted based on structural rules (analogy: lexical/syntax analysis in compiler)

32bit Linux on x86



공격 위치2: Kernel Dynamic Region

- Accesses/modifications are possible in principle but some that cause semantic mismatch are illegal (ex: removing a process entry from process table)
- Loadable Kernel Modules (LKMs)
  - Makes monolithic kernel more flexible. (ex: device drivers)
  - Dynamically loaded into kernel memory space at runtime
- Task linked list (process table)
  - Dynamically created/deleted at runtime
- Virtual File System structure
  - Varies with time



Protection region has to be calculated for every change → more complex and time consuming because violation is determined by behavioral rules (analogy: semantic analysis)

공격 위치3: User Memory Region

- Allocated in ZONE_HIGHMEM area
  - No linear address mapping, so difficult to pinpoint the exact location of wanted objects or data structures
- Virtual memory is managed by kernel
  - CR3 -> Page directory -> Page Table -> Page Frame Number
- Shared Library, Page fault
  - Makes attack analysis more complex.

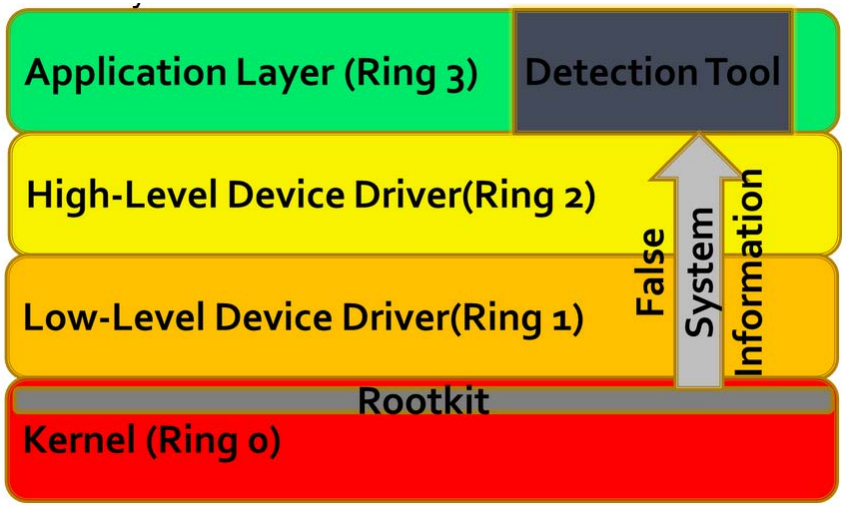
Non linear address mapping, shared library, page faults, app-specific security requirements ➔ Identifying data structure, protection domain becomes more difficult and app-specific because definition of violation differs from app to app or case by case at run time (analogy: runtime error checks)

1st goal of Vigilare

- Kernel static region에 대한 감시 기능 수행
- Rootkit 공격 예제
  - Static hooks/patching
    - 시스템 콜 Hooking
    - 커널코드의 static patching (jmp insertion)
  - DKOM (Dynamic Kernel Object Manipulation)
    - PCB (Process Control Block) 조작 for Process Hiding
    - Filesystem Manipulations (ex: File Hiding, Content Filtering)

How Rootkit works

- a class of malware that manipulates OS kernel itself
- Perpetuates privileged access to the victim
- Hides its traces by modifying kernel' s system information reporting functions
- Difficult to detect since it operates in the very core of the system



Kernel integrity monitoring

- Fig 1: Monitor in application layer
  - Compromised OS kernel may disturb(?) the kernel monitor
- Fig 2: Monitor independent from OS kernel with VMM

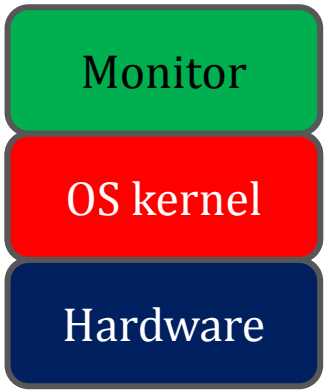


Fig 1

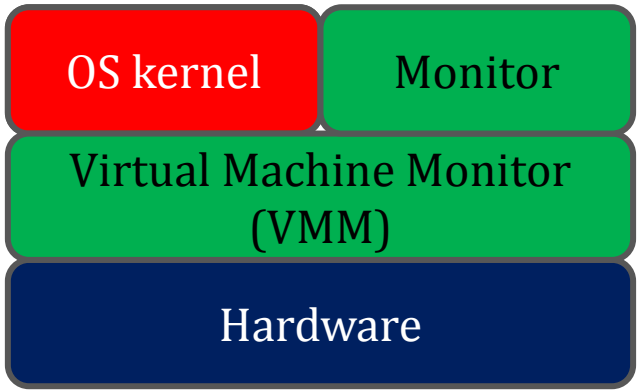


Fig 2

Kernel integrity monitoring

- Fig 3: Monitor independent from OS kernel with VMM
  - VMM can also be compromised!!
- Fig 4: Monitor independent from OS kernel with Hardware

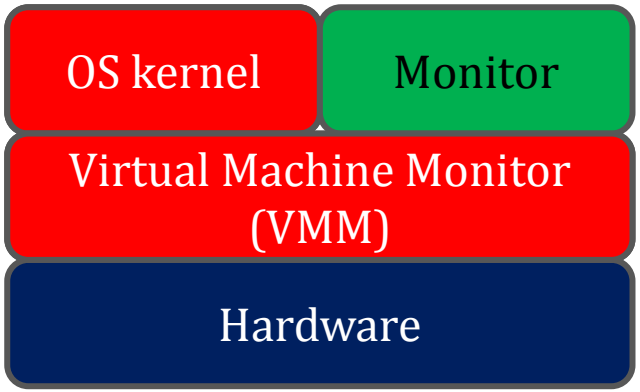


Fig 3

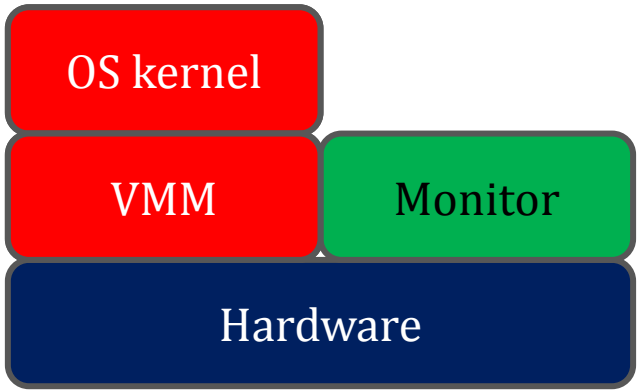


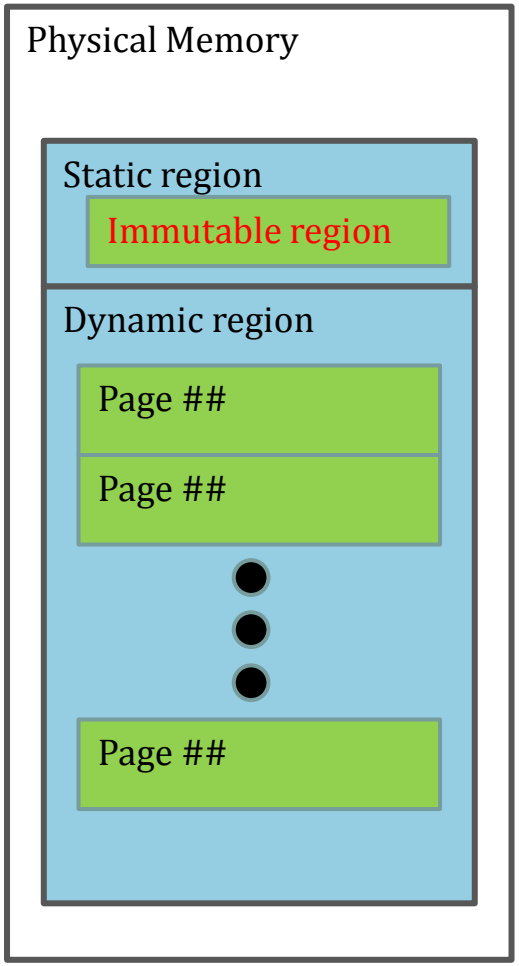
Fig 4

Previous work vs. Vigilare

	Copilot (Security 2004)	HyperSentry (CCS 2010)	Vigilare (CCS 2012)
Target	Static, immutable regions of Linux kernel	Static, immutable regions of Xen + guest memory isolation of Xen	Static, immutable regions of Linux kernel
How	Analyzing periodic snapshots	Calling in-context monitor periodically	Snooping traffic between processor and memory
Performance degradation	6-9%	0-5%	0%
Transient attacks	May miss	May miss	Detects all

Attack model (1/4)

- Target: Kernel static/immutable region
  - Linux kernel is a program: needs codes and data
    - Data generated on runtime (task_struct, mm_struct, etc)
      - Stored in "pages"
      - Dynamic region
    - Most of the codes and some of the data should not be in "pages"
      - At least codes/data that manage "pages"
      - Static region
  - All codes and some data (e.g. sys call table) of the kernel should not be modified at runtime
    - Immutable region

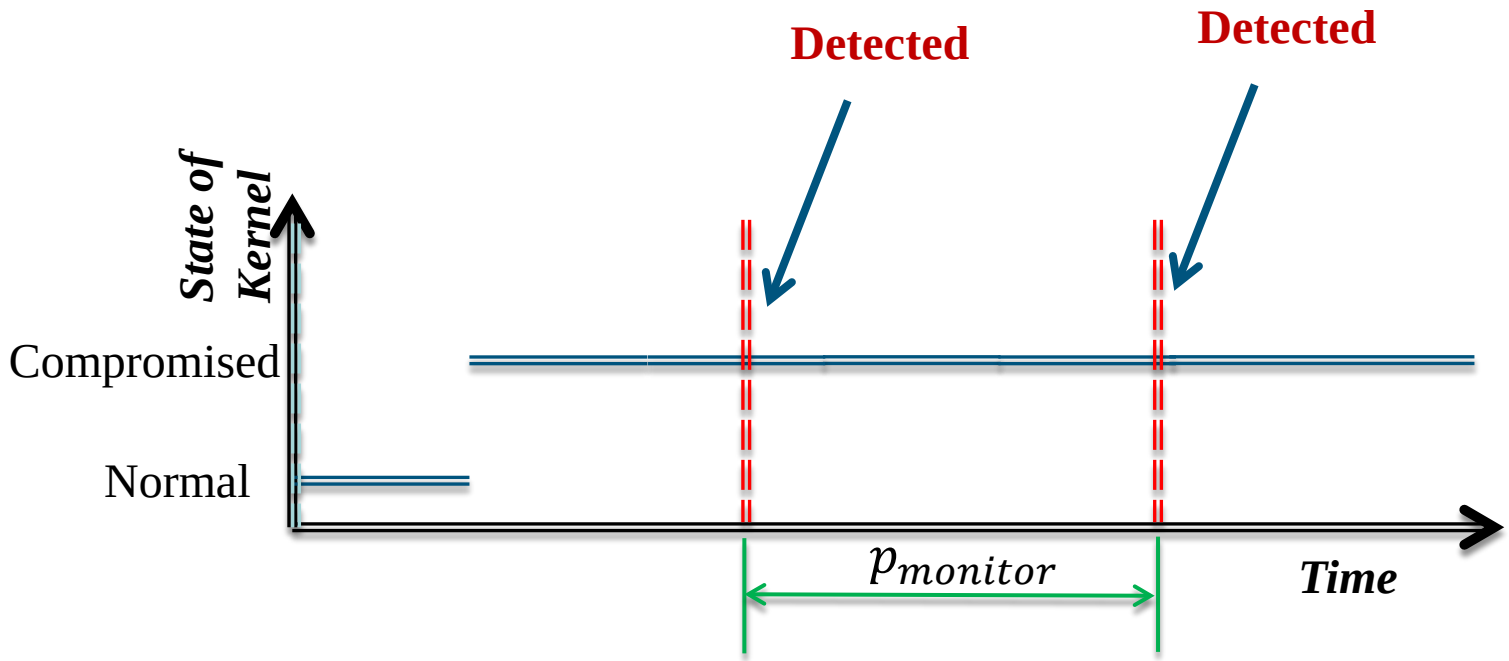


Attack model (2/4)

- Persistent attack model
- Transient attack model
  - Opposed to persistent attacks , traces of attacks in memory are not permanent
  - Possibly evade periodic scans
  - Our Transient Rootkit
    - Modifies and Restores kernel code at a fixed time intervals

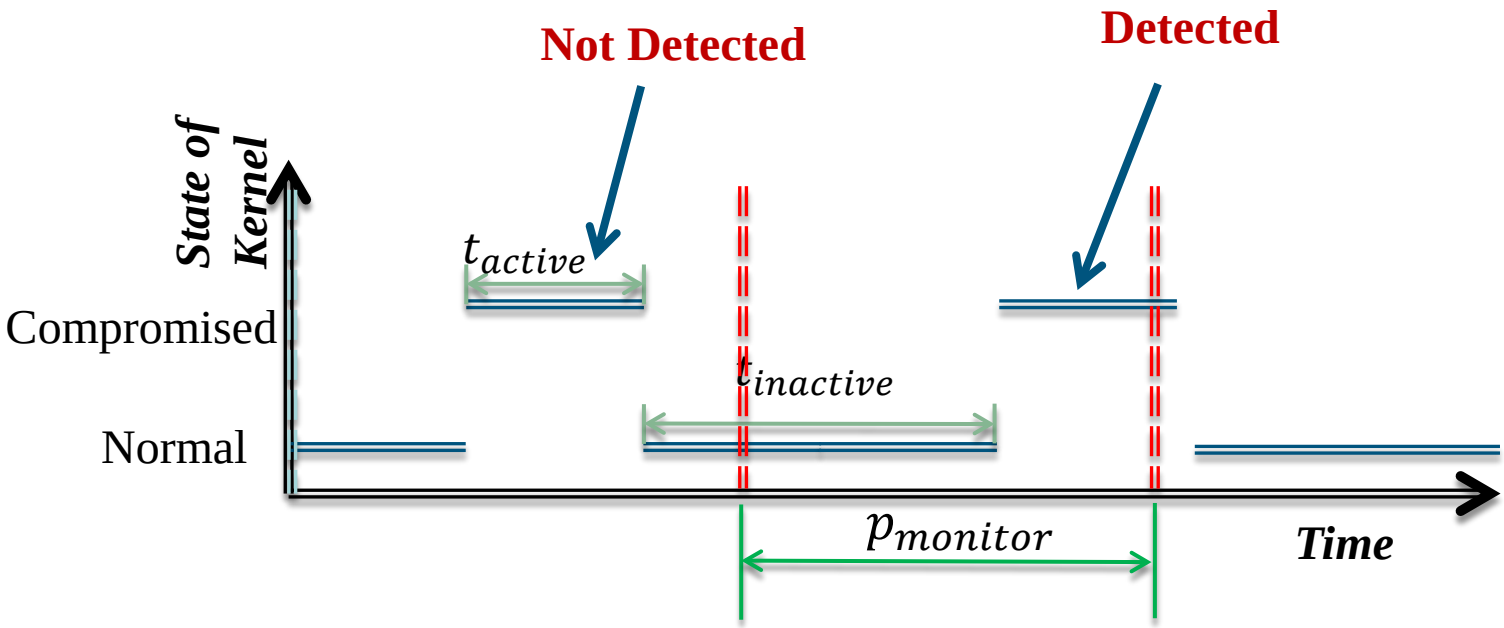
Attack model (3/4)

- Persistent attacks
  - Previous approach works



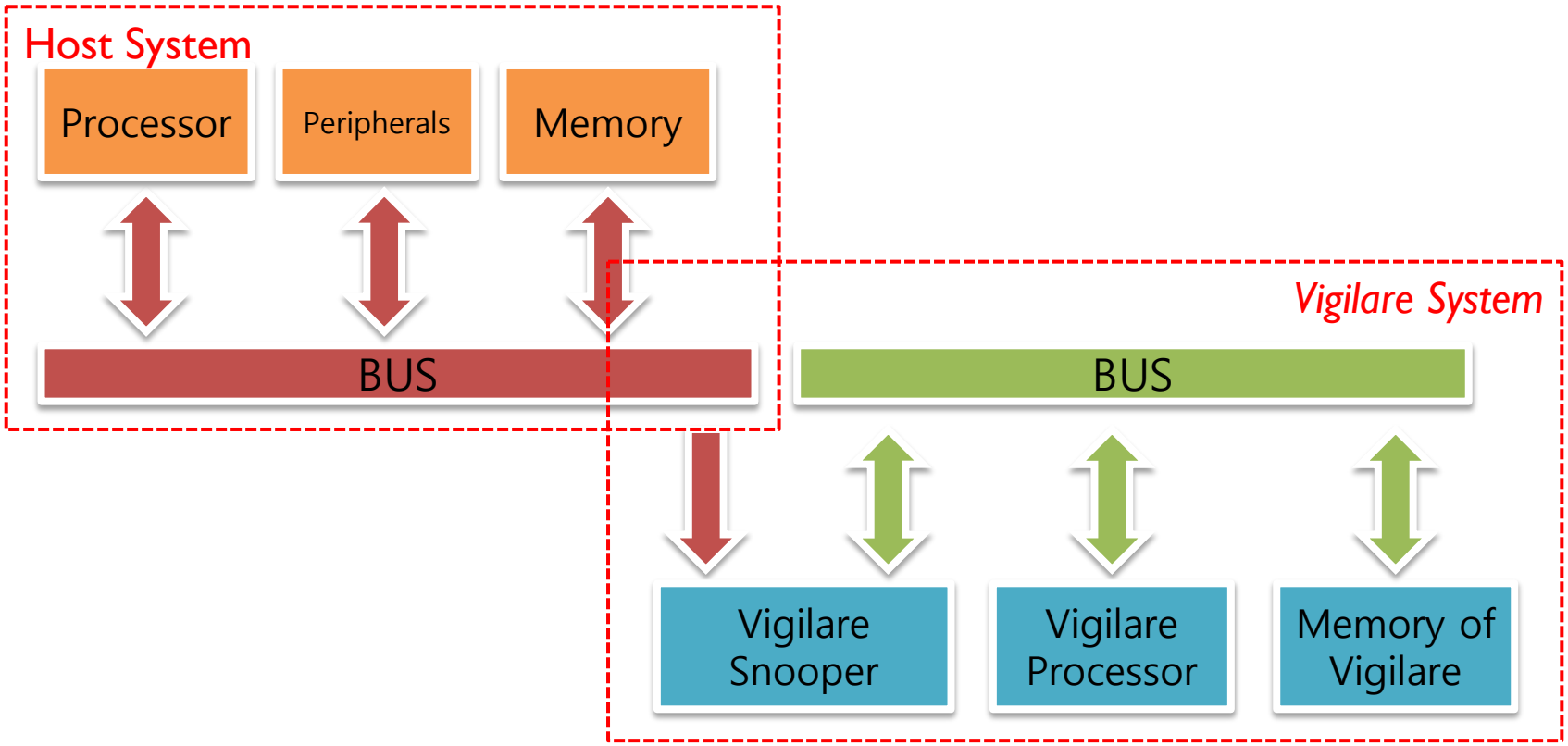
Attack model (4/4)

- Transient attacks
  - Previous periodic monitoring may fail to detect an attack
  - Our attack model: transient attacks on kernel static/immutable region

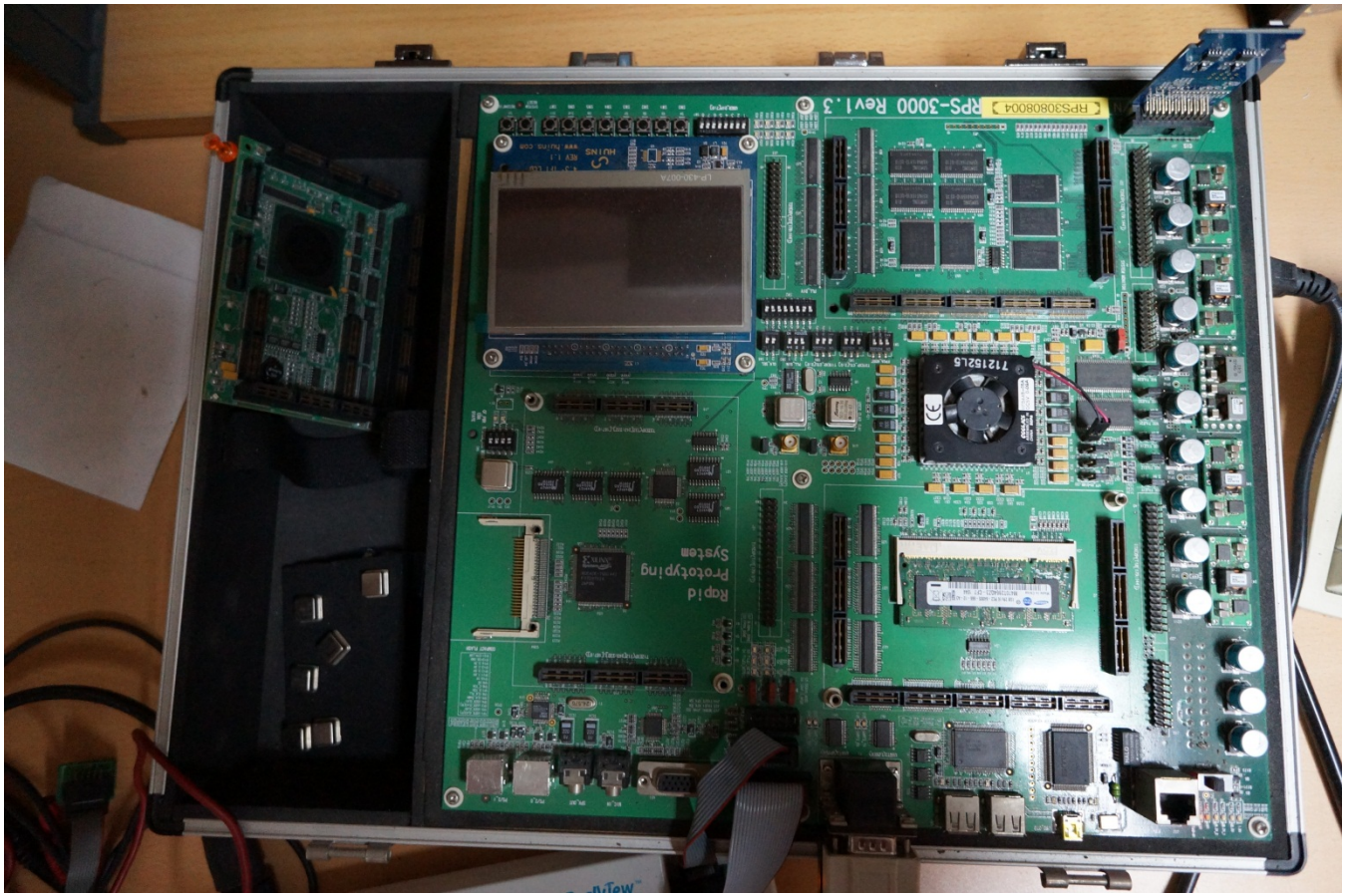


Design and Implementation (1/7)

- Conceptual design of a system with Vigilare System



Design and Implementation(2/7)

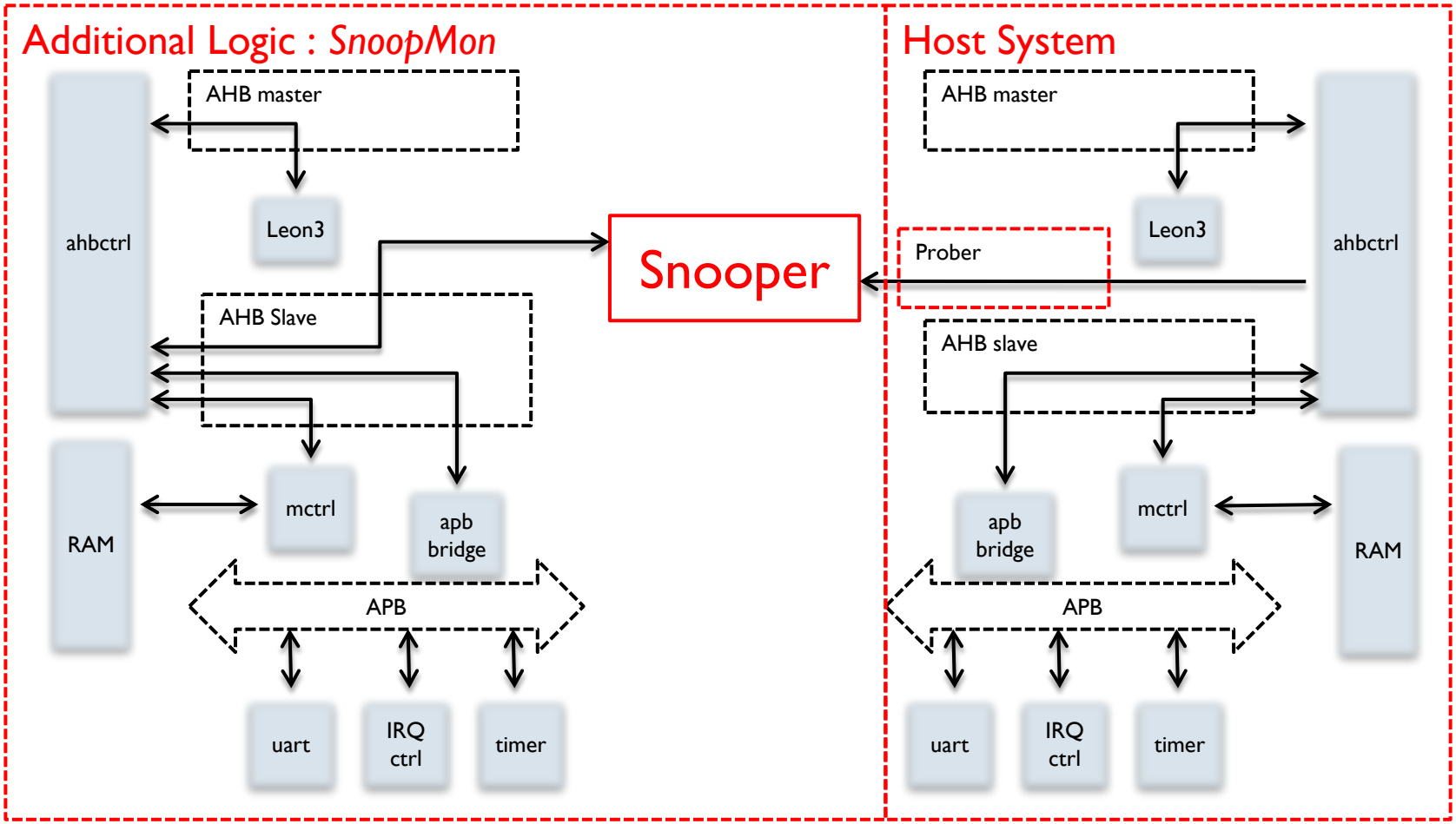


Design and Implementation(3/7)

- Design spec.
  - Host system
    - 50MHz Leon 3 processor (SPARC V8)
    - 64MB SDRAM
  - SnoopMon
    - 50MHz Leon 3 processor (SPRAC V8)
    - 2MB SRAM
    - Snooper
  - SnapMon
    - 50MHz Leon 3 processor (SPRAC V8)
    - 2MB SRAM
    - Direct Memory Access (DMA)
    - Hash accelerator

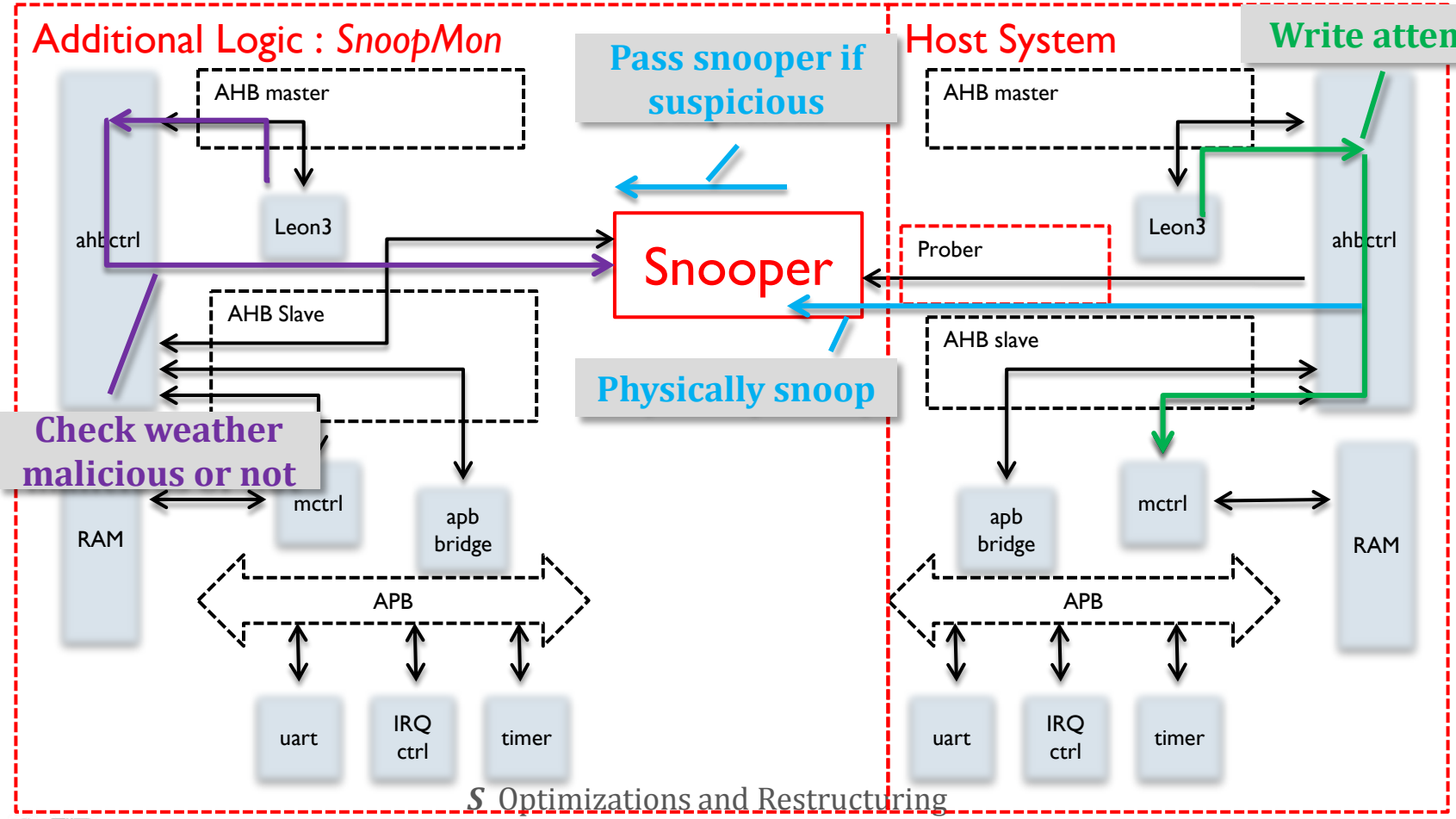
Design and Implementation(4/7)

- The prototype we used for experiment (SnoopMon)



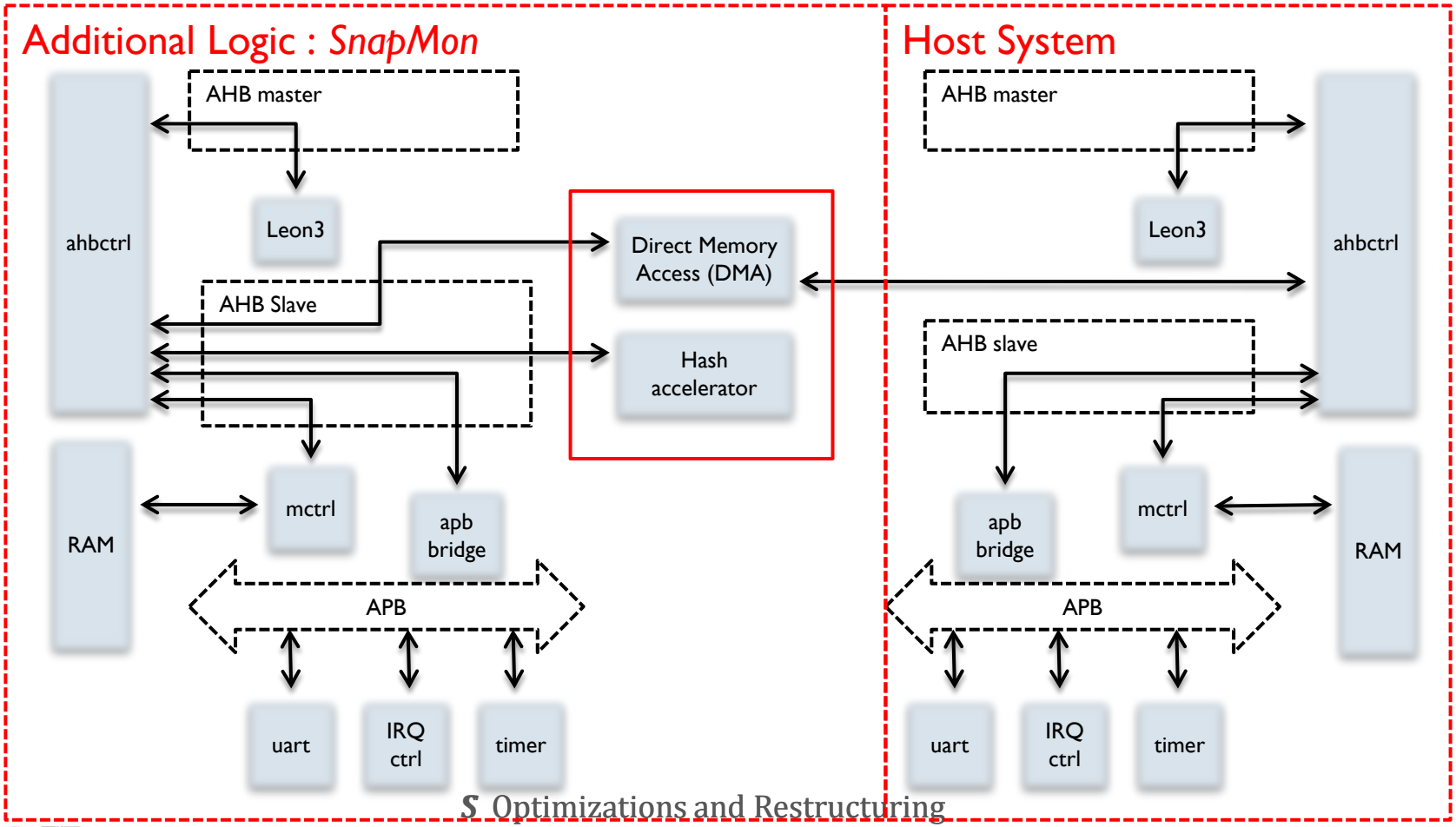
Design and Implementation(5/7)

- How SnoopMon works



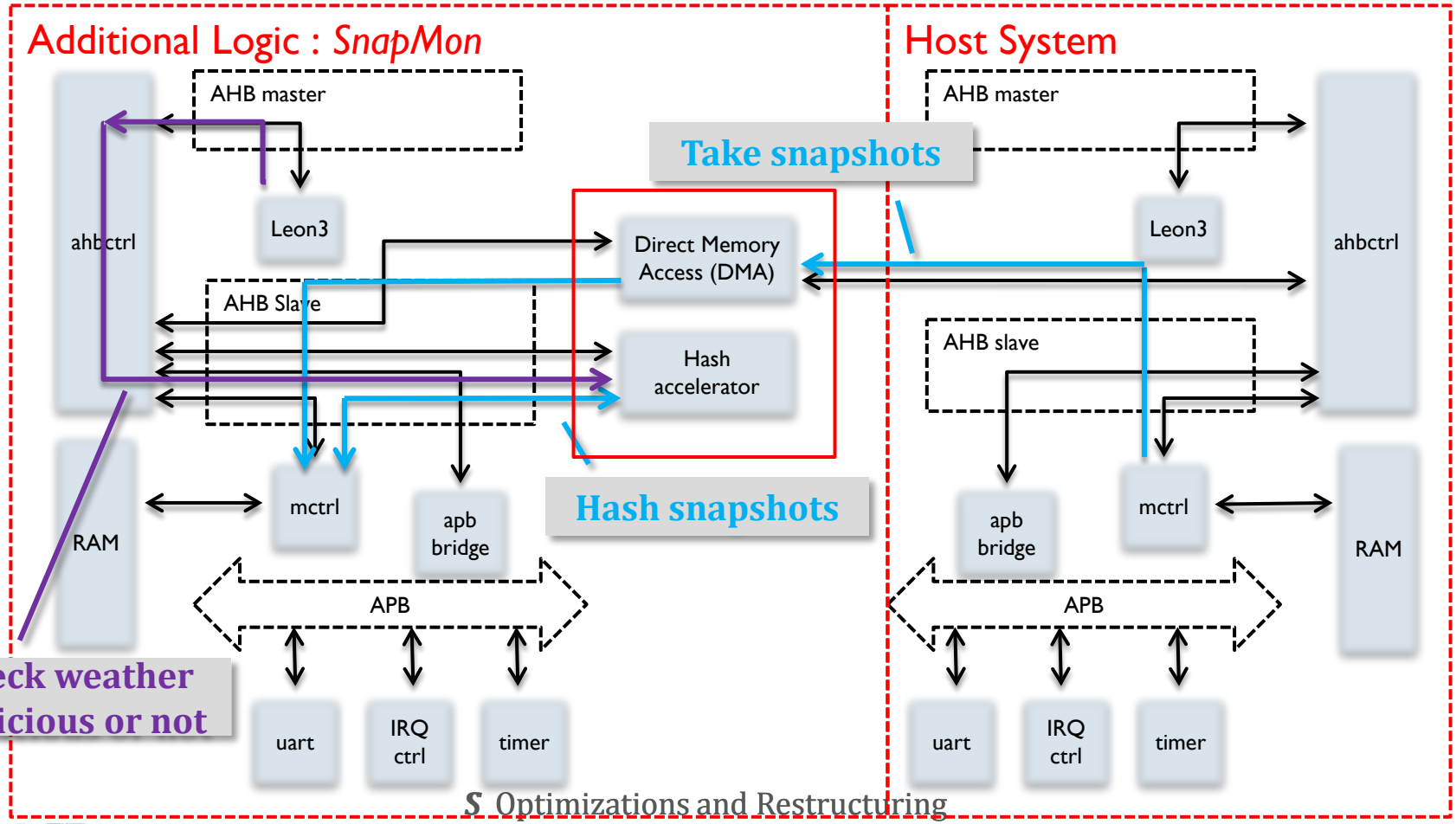
Design and Implementation(6/7)

- Copilot-like snapshot monitor (SnapMon) example



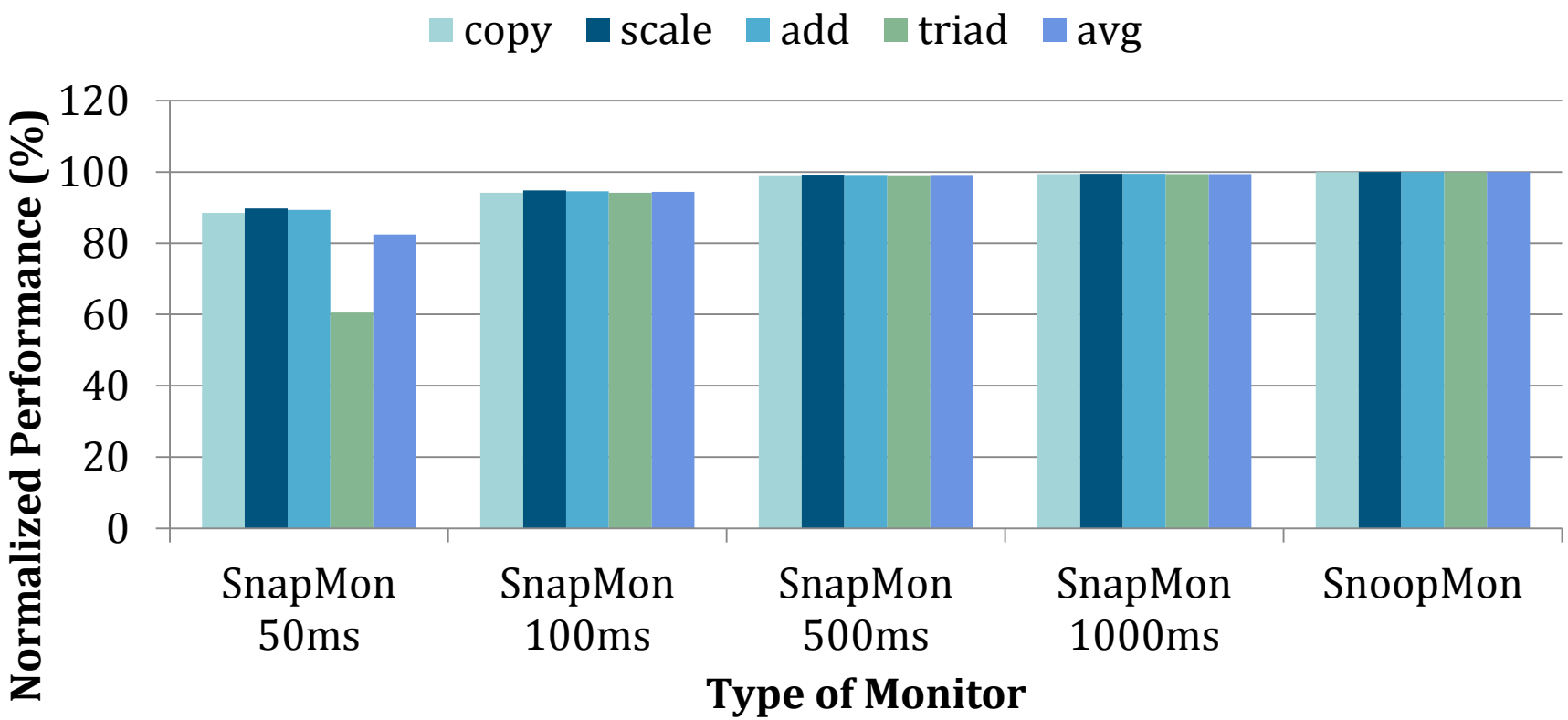
Design and Implementation(7/7)

- How SnapMon works



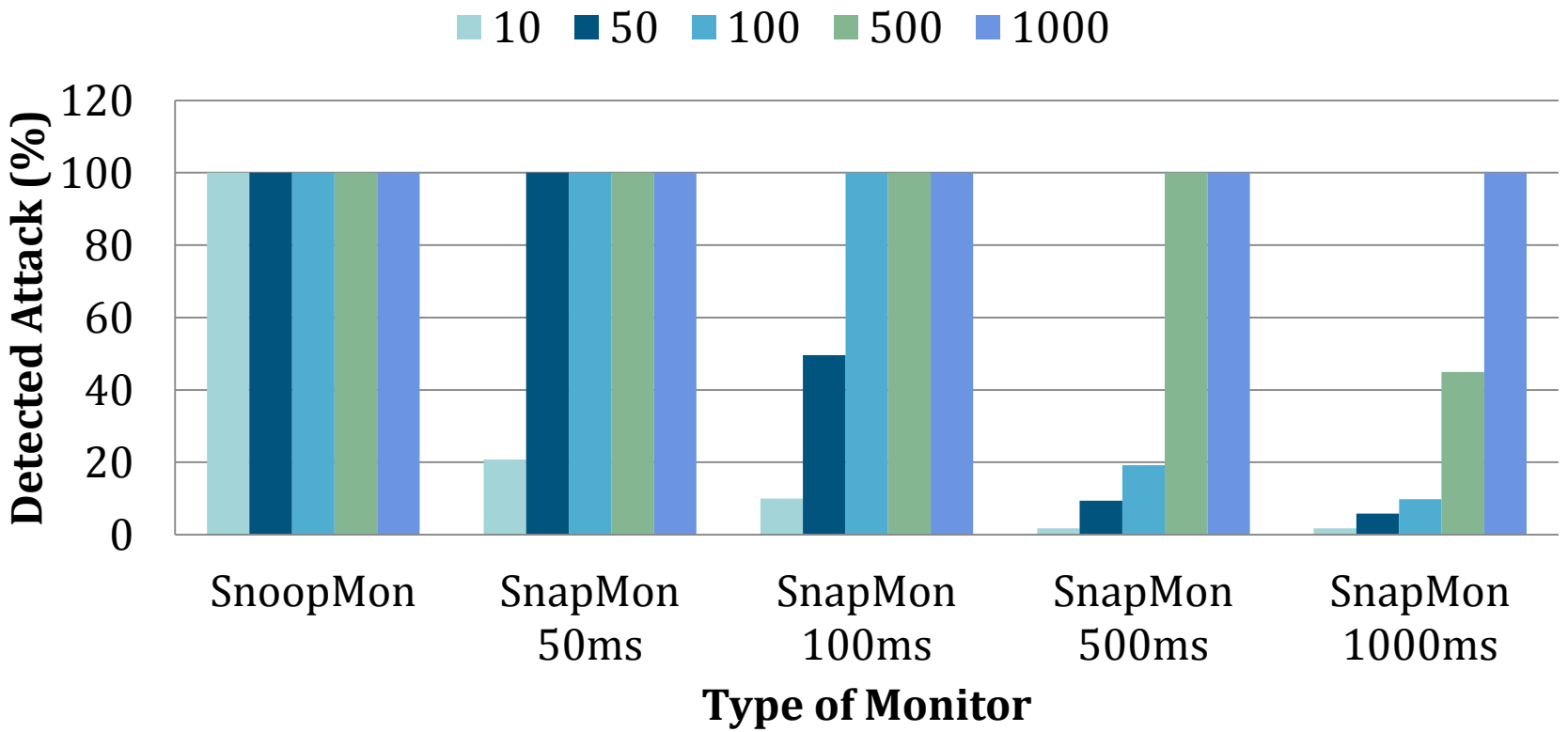
Evaluation (1/3)

- Performance degradation



Evaluation (2/3)

- Transient attack detection

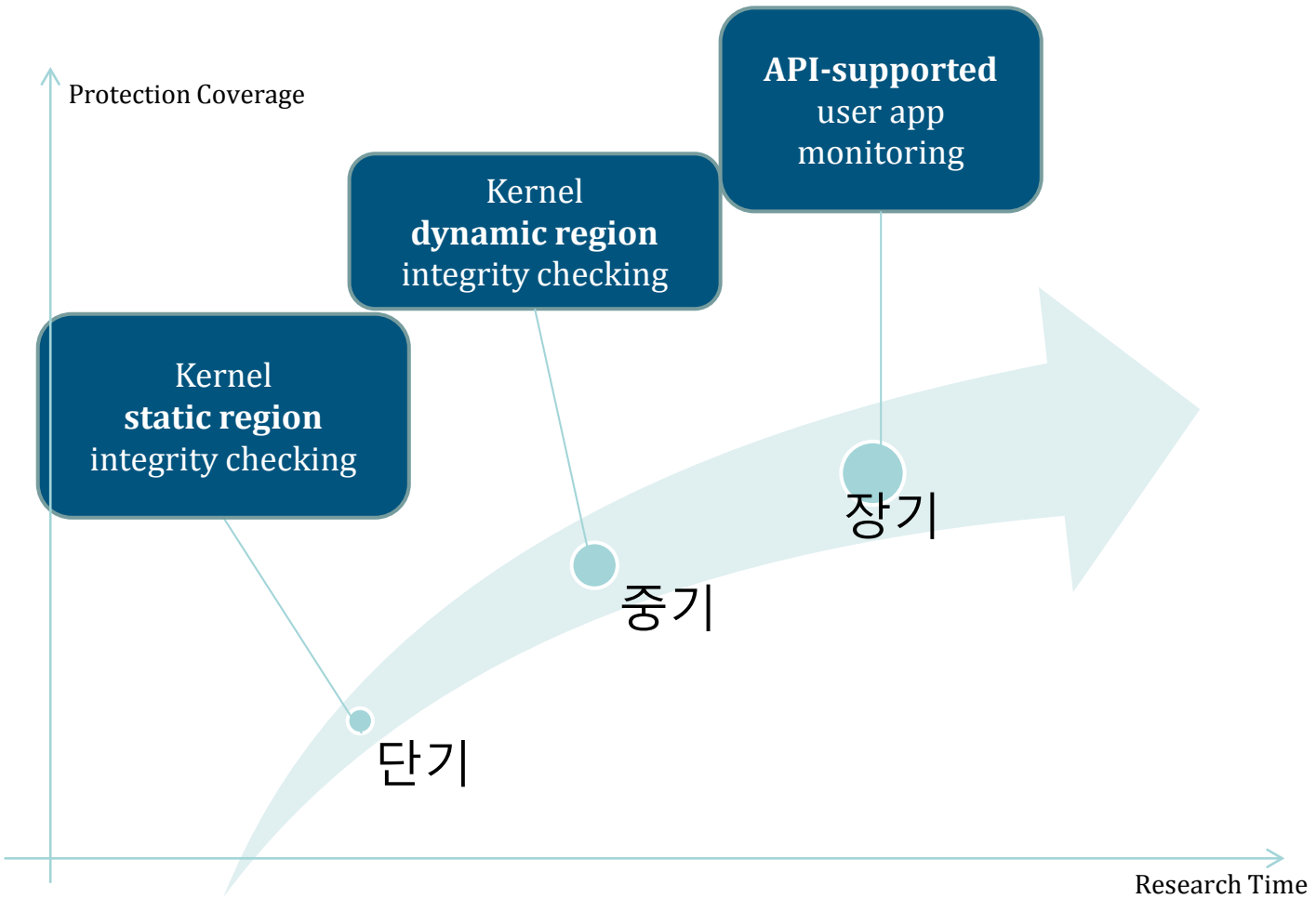


Evaluation (3/3)

- Fundamental trade-off of SnapMon
  - Longer period, less performance degradation
  - Longer period, less ability to detect transient attacks

- No such trade-off does SnoopMon have
  - Detect all transient attacks
  - No performance degradation

Overall roadmap in big picture

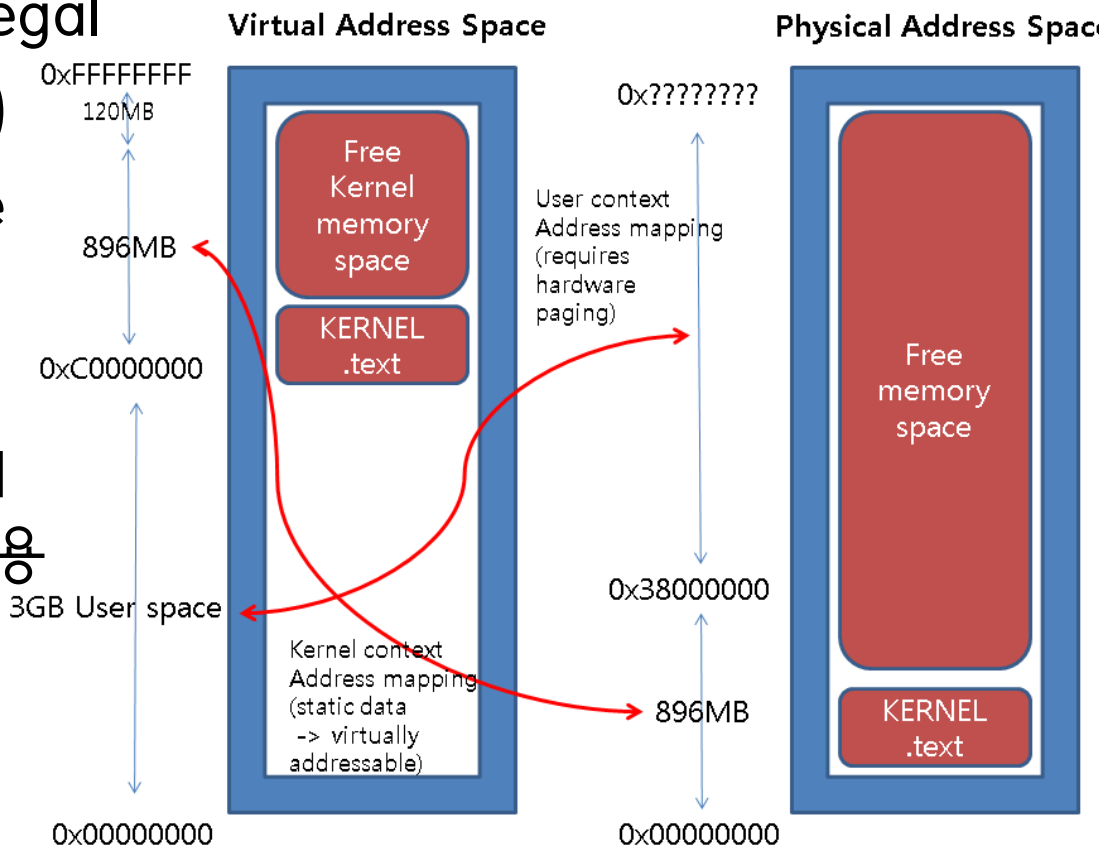


Kernel Static Region

- This is immutable region, so any access/modification is illegal
- Text segment (Kernel code)
- Interrupt Descriptor Table
- Sys_call_table
  - Dispatch system calls
- Linux에서 linear-mapped kernel virtual address 사용
  - 외부 HW에서 감시 가능

We have implemented basic Vigilare hardware platform for the attacks on this region.

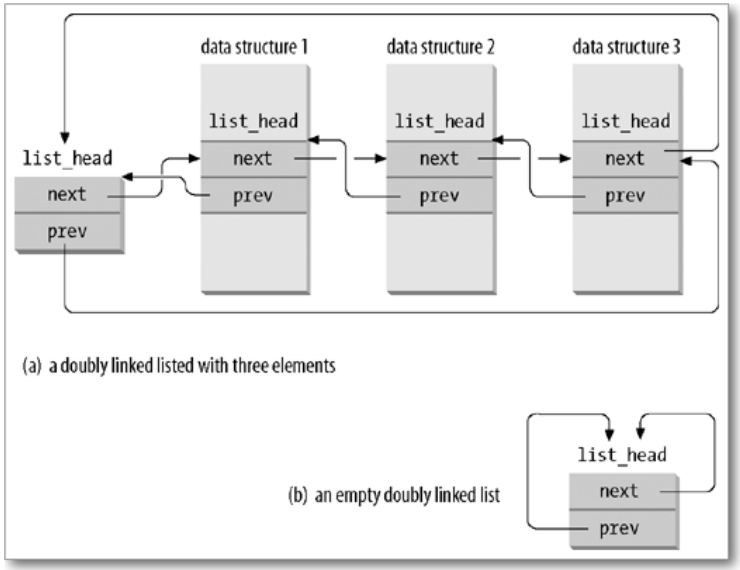
32bit Linux on x86



Kernel Dynamic Region

- Accesses/modifications are possible in principle but some are illegal (ex: removing a process entry from process table)
- Loadable Kernel Modules (LKMs)
  - Makes monolithic kernel more flexible.
  - Dynamically loaded into kernel memory space at runtime
- Task linked list (process table)
  - Dynamically created/deleted at runtime
- Virtual File System structure
  - Varies with time

The next research topic!
We are enhancing our Vigilare architecture to handle the attack on these regions

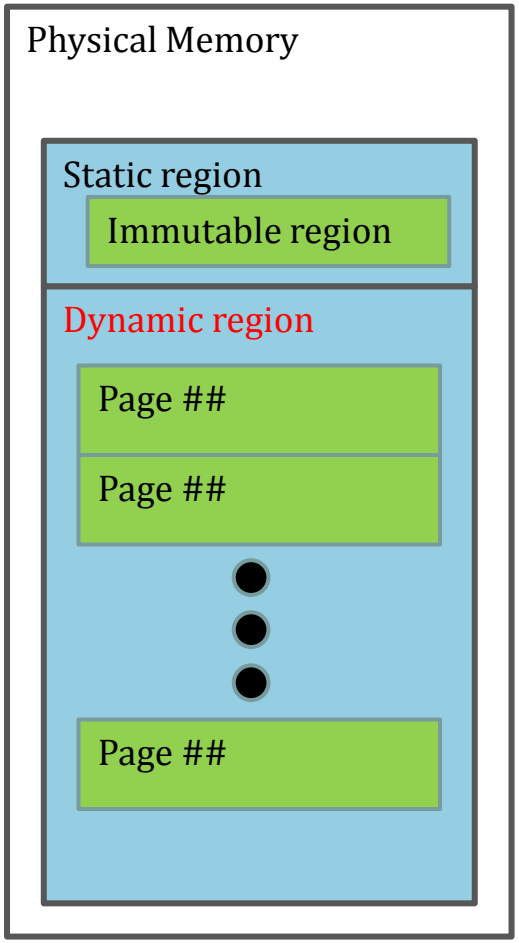


Roadmap - 단기

1. Kernel dynamic region에 대한 공격에 대한 방어 수단 구축
2. 기존에 구현된 Vigilare System을 보다 현실적 환경에 특화된 구조로 변경해 구현
   - 국내 스마트폰 제조사와 차세대 스마트폰 보안 장비 개발
     - 스마트폰의 AP SOC에 맞게 One chip 설계 중 → Vigilare version 2
     - 2개의 독립적 memory 모듈을 사용하는 Vigilare version 1는 일반적인 스마트폰에 적용하기에 한계가 있음
     - 한 개의 memory를 공유하면서 생기는 보안취약성 보안
   - ADD와 차세대 군 전자장비 보안 장치 개발 계획 중
     - Linux대신 군장비에 보편적인 VxWorks 같은 RTOS에 특화된 알고리즘 개발
     - 다양한 군사 전자 장비에 특화된 보안 attack 방법에 대한 대응

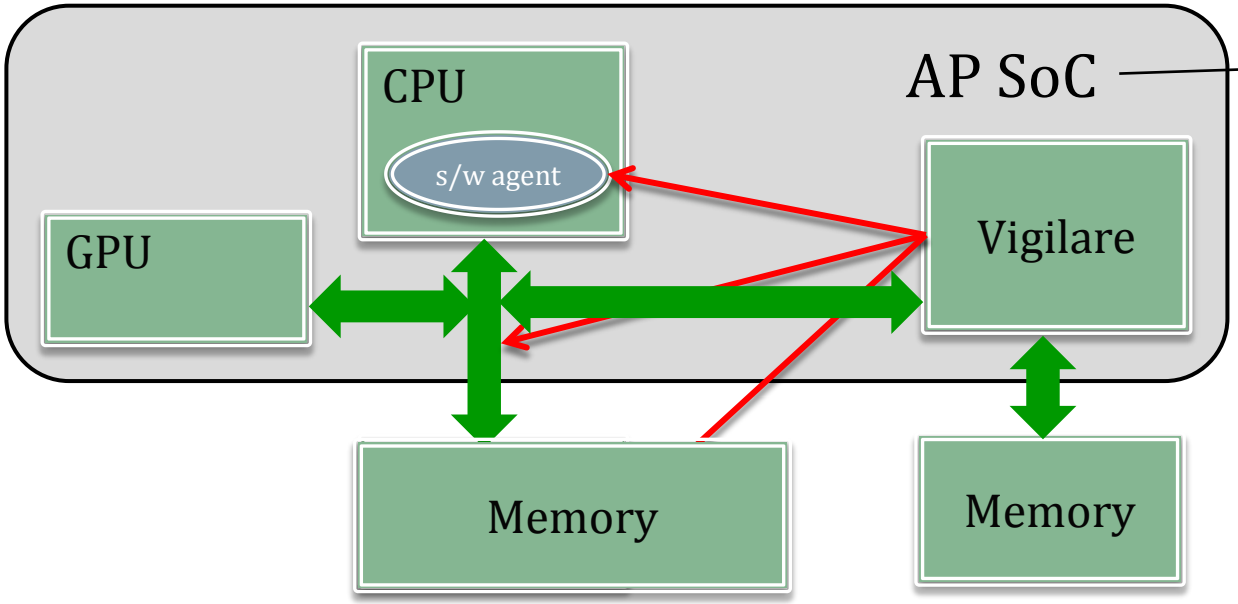
Roadmap - 단기

- Monitoring dynamic region
  - Challenges:
    - Locating a certain page
    - Physical addr to logical addr translation
- Monitoring mutable region
  - Challenges:
    - Designing integrity policies:
      - "What is legal modification?"

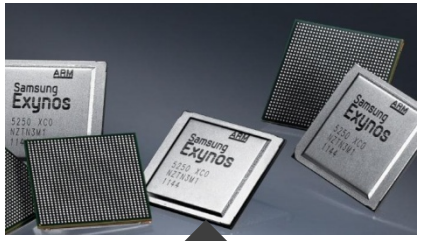


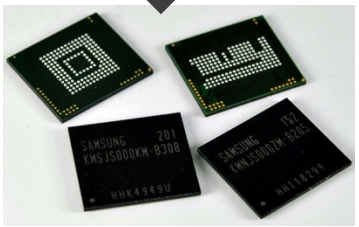
Roadmap - 단기

- 스마트폰에서는 모든 processor core들이 Application processor (AP) SoC안에 모두 integrated circuit으로 packaging 되고, 메모리는 모두 외부에 위치
- Vigilare v1은 그런 현실을 반영하지 못함



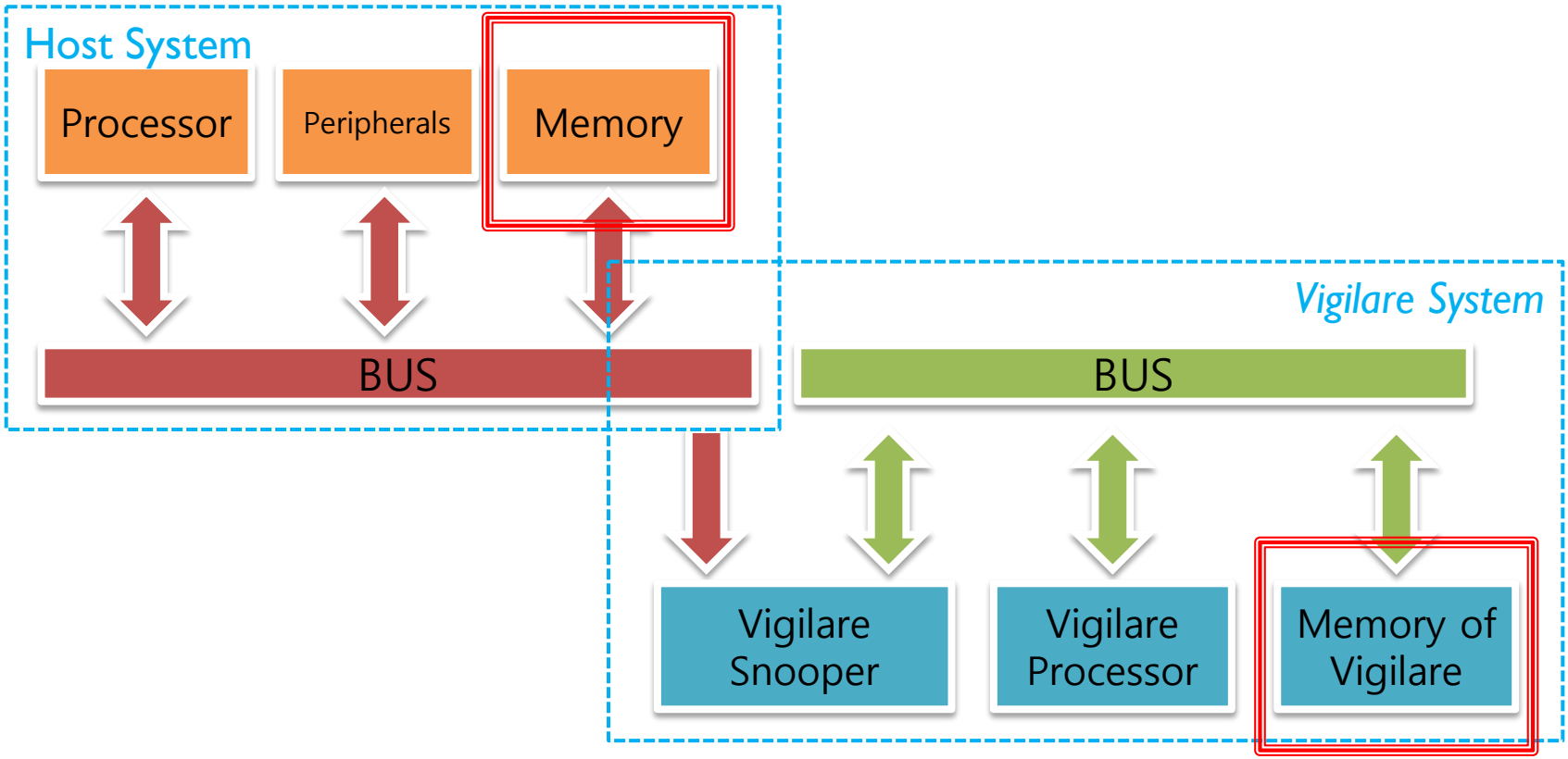
Apple Ax (A4,A5)
Samsung Exynos





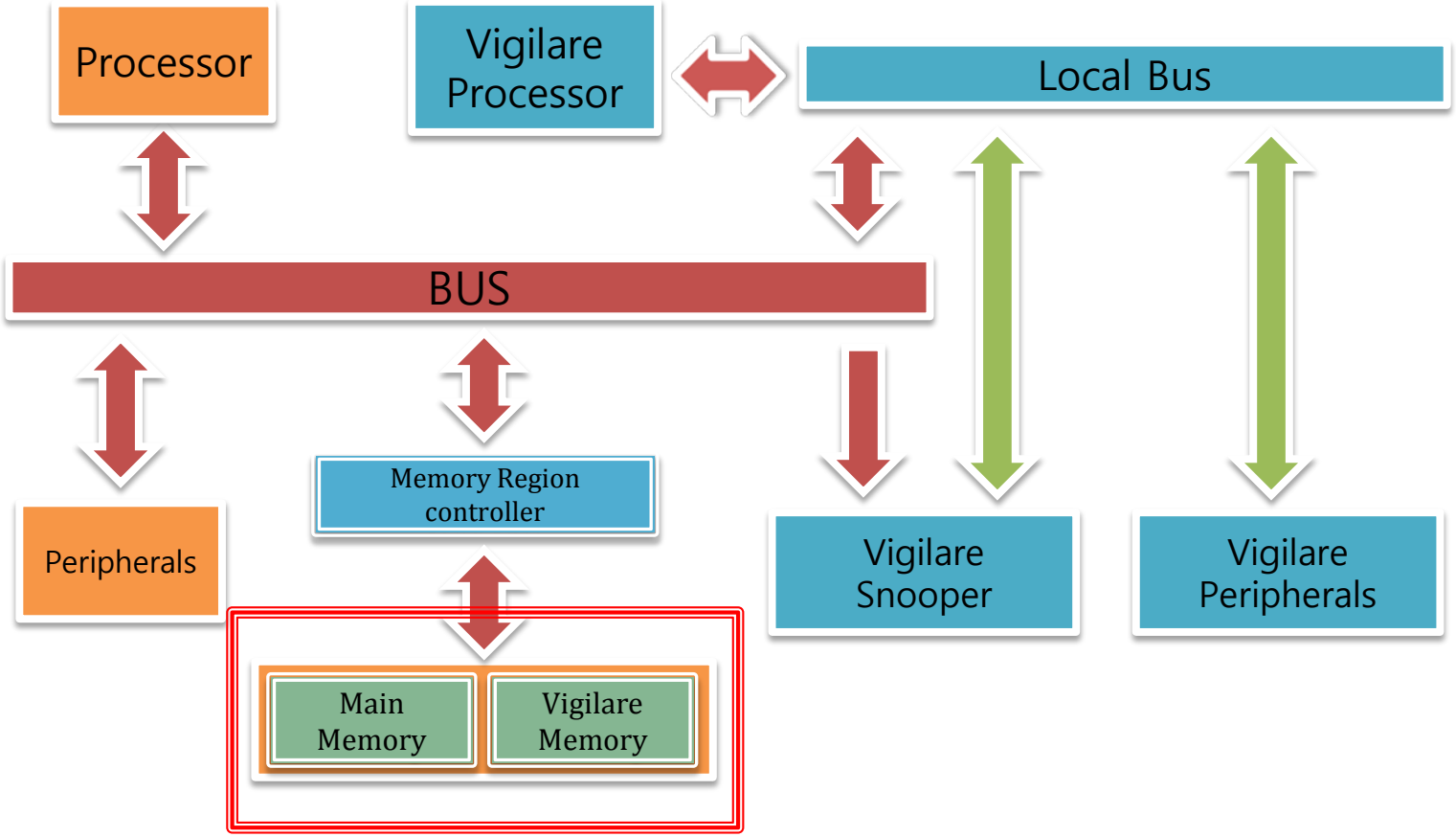
Roadmap - 단기

- Old architecture revisited



Roadmap - 단기

- New Architecture for the work with AP SoC



User Memory Region – 중장기

- Allocated in ZONE_HIGHMEM area
  - No linear address mapping, so difficult to pinpoint the exact location of wanted objects or data structures
- Virtual memory is managed by kernel
  - CR3 -> Page directory -> Page Table -> Page Frame Number
- Shared Library, Page fault
  - Makes system more complex.

In a few years, we will deal with the attacks on user memory regions.
We already have some ideas about it.

Roadmap - 중장기 목표

- API support for monitoring user programs
  - Design APIs and supporting architectures for user programs
  - Monitor the integrity of user programs that used the APIs
  - Early stage: rely on information provided by developer
  - Advanced: generate more information by analyzing user programs
  - Possibly can be a new programming model
    - "Programming model for attack-tolerant application"

API support 예제

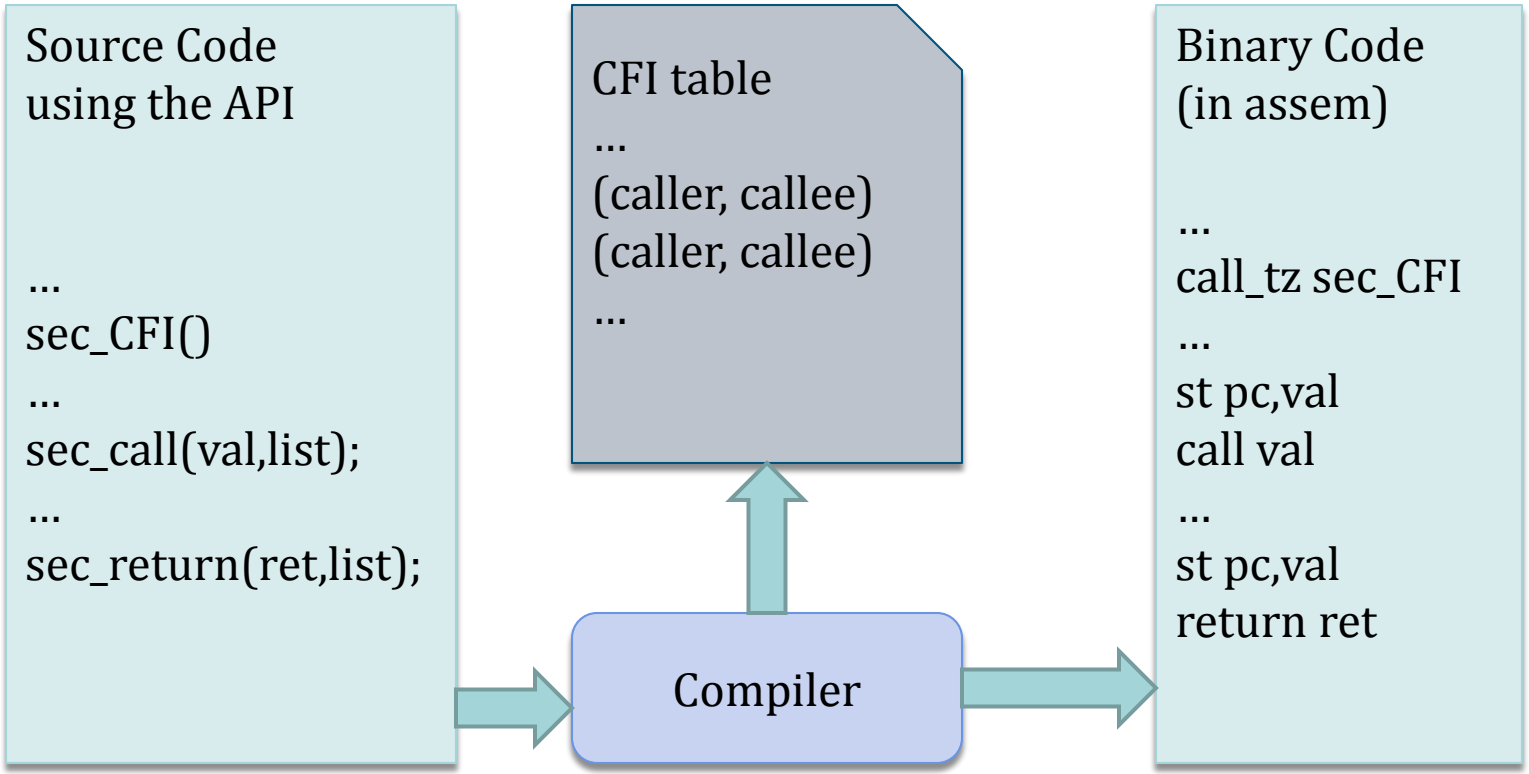
- Monitoring Control Flow Integrity (CFI) of user apps
  - Why CFI?
    - Attacks on user app break CFI in many cases
  - One key challenge: defining "legitimate" control flow - CFI table
    - Relying on developer
      - Manually generate CFI table
    - Analyzing source/binary code
      - Automatically generate CFI table
    - Tools to assist developer
      - Interactively generate CFI table

API support 예제

- Relying on developer
  - API 구성: return과 function pointer를 통한 function call을 대체하는 functions
    - sec_return(ret,legal_functions[])
    - sec_call(function_pointer, legal_values[])
    - CFI보호를 위한 table 작성에 사용
    - "code 분석" 에 대한 burden을 개발자에 전가
  - 이 app에 대한 CFI보호를 시작하도록 하는 function
    - sec_CFI();

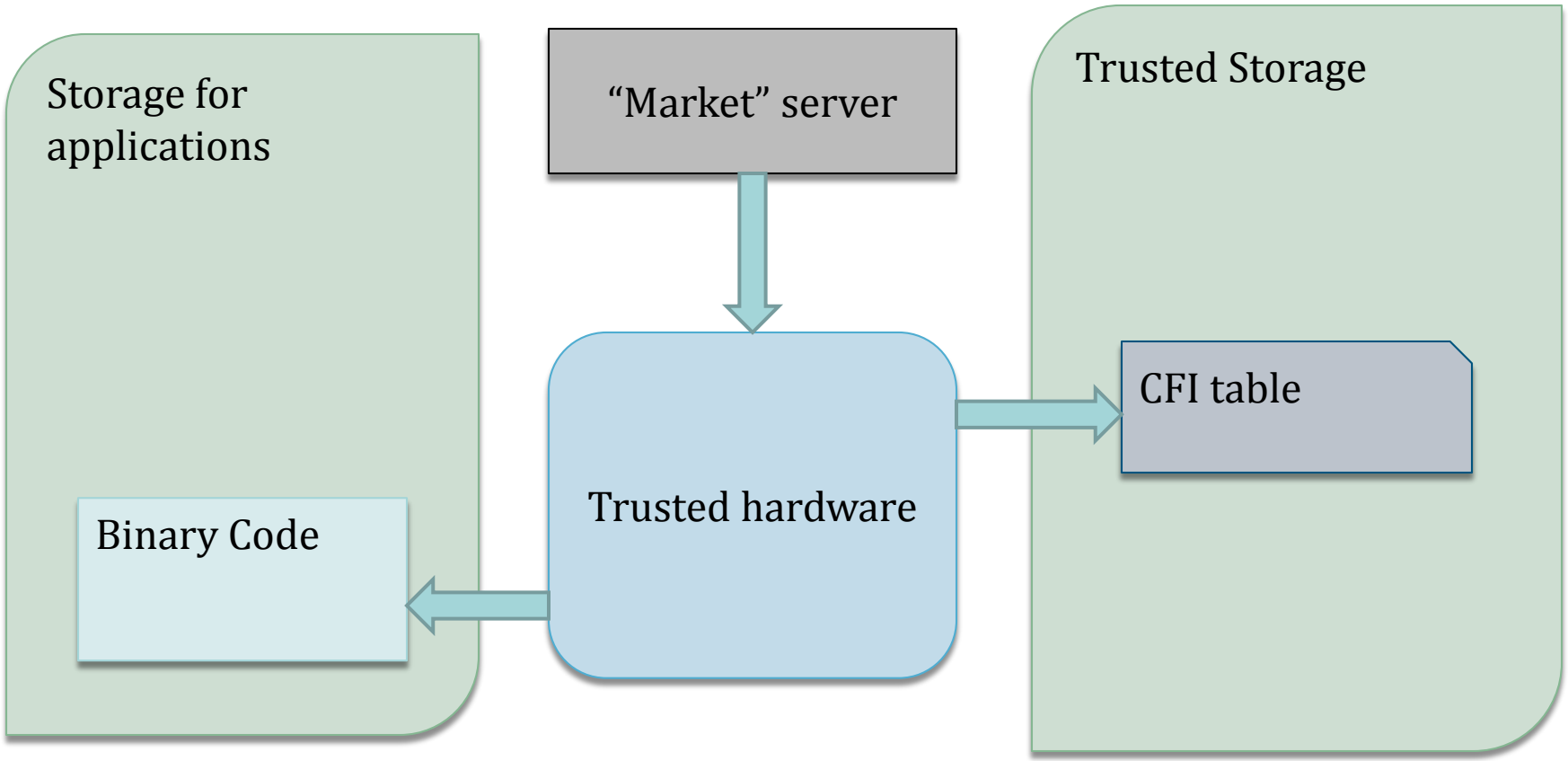
API support 예제

- Compile time



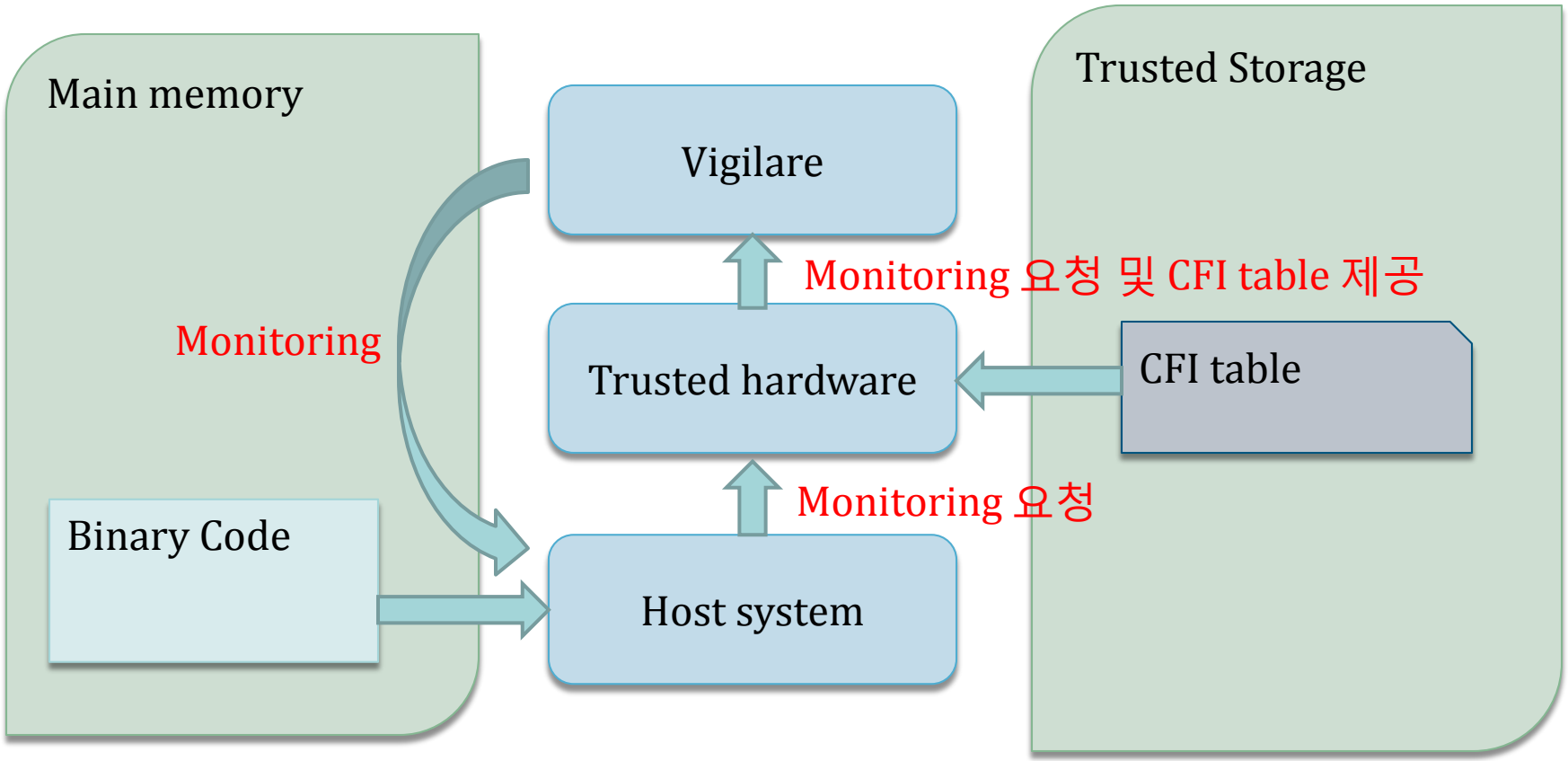
API support 예제

- Install time



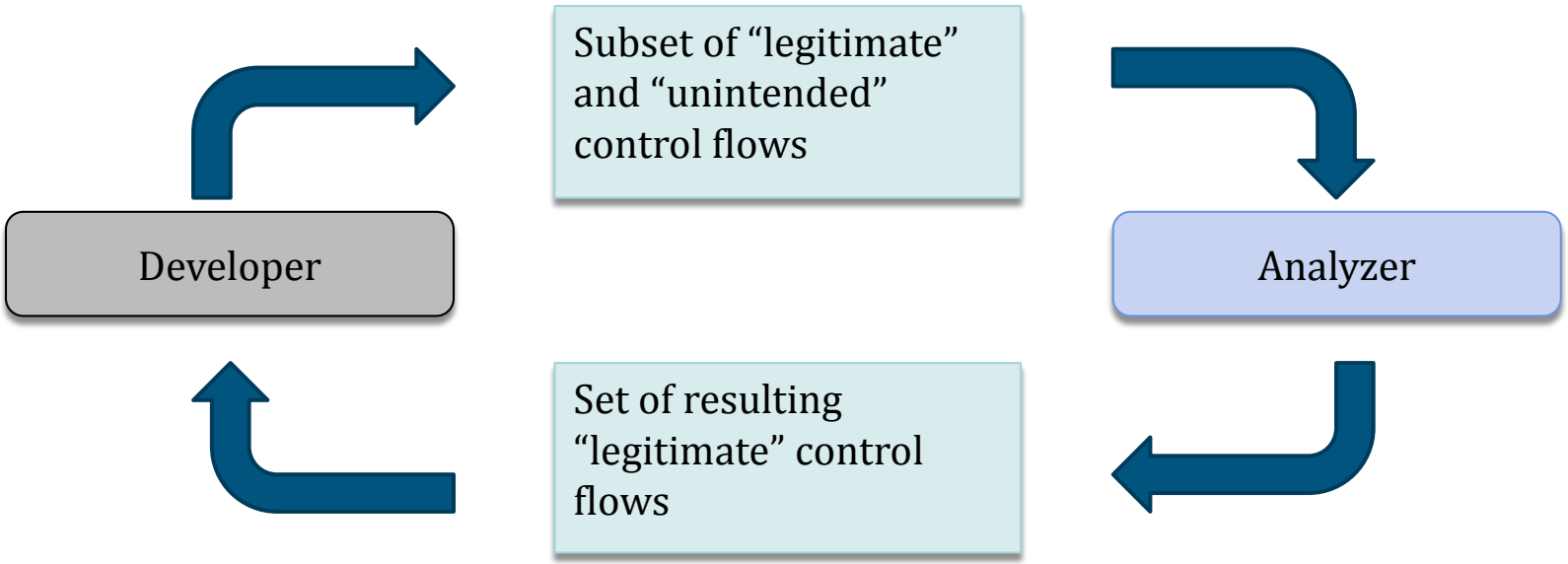
API support 예제

- Runtime



API support 예제

- Analyzing source/binary code
  - Difficult to generate control flow graph in general for binary codes
  - Resulting "legitimate" control flow may include "unintended" one
- Tools to assist developer



Vigilare API 활용가능성

- CFI table: an example of app-specific information

- More examples:
  - Critical data protection
    - Critical data table
    - Secure data allocation in user application
  - Peripheral access control
    - Access permission table

Conclusion

- Vigilare는 기존에 SW 기반의 Integrity monitoring 기법들을 하드웨어의 도움을 통해 more secure & energy-efficient, faster 하게 수행되도록 해주는 HW기반 보안감시장치
- Vigilare v1을 통해 본 연구의 feasibility를 입증
- 실제 스마트폰등에 적용을 위한 하드웨어 구조 개선이 현재 우리 연구실의 당면 목표 → 차세대 상용화 스마트폰에 탑재를 목표로 v2, v3로의 새 버전들 개발
- Vigilare API지원을 통해 Vigilare processor들을 Host에서 프로그래밍 가능하도록 하여, 다양한 user-level 보안 기능 지원 가능

  → 기존의 정적 분석 기술과의 결합도 가능