

# Malware Resistance by Memory Randomization and Remote Attestation

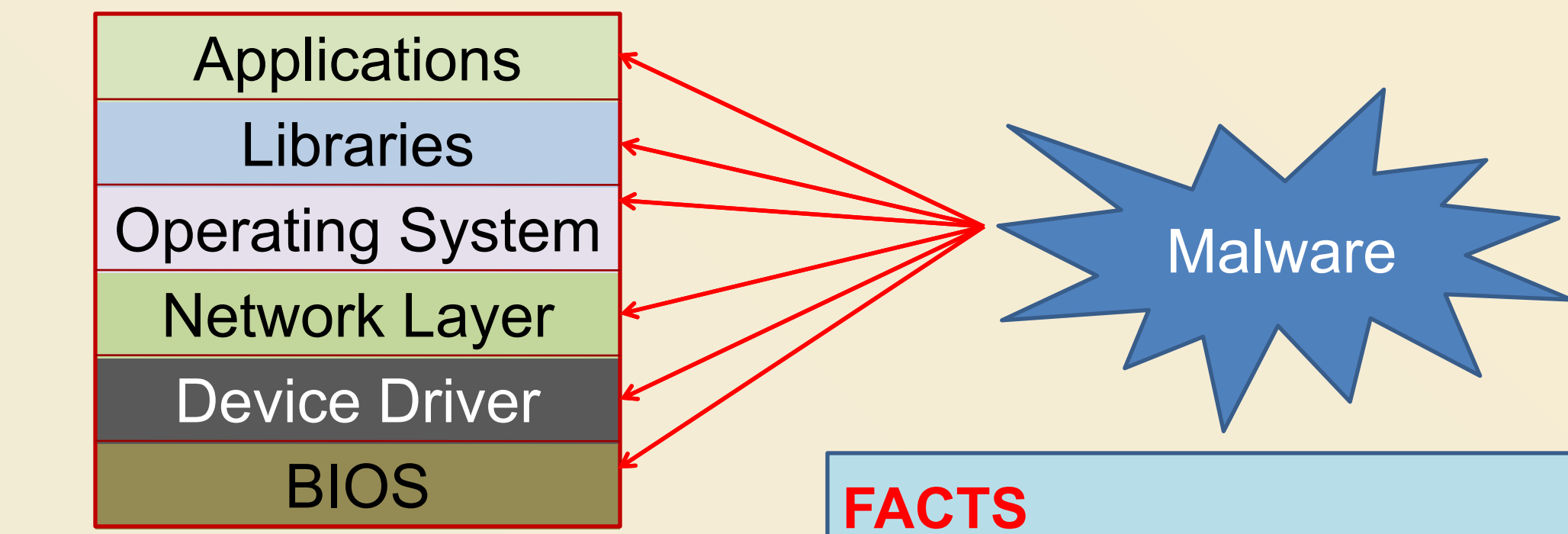
Raghunathan Srinivasan, Partha Dasgupta, Amit Kanitkar, Vivek Iyer  
School of Computing and Informatics, Arizona State University



## Introduction

**SYSTEMS SECURITY**

- Vulnerabilities common in software
- Spike of over 100 percent seen in attacks between 2003 and 2004
- Consumer computing security is broken
- Increase in the number of overflow based attacks
- Compromised systems leak data or used as part of a botnet
- Attacker's motivation may be financial gain or bragging rights
- Prevention and detection of malware required

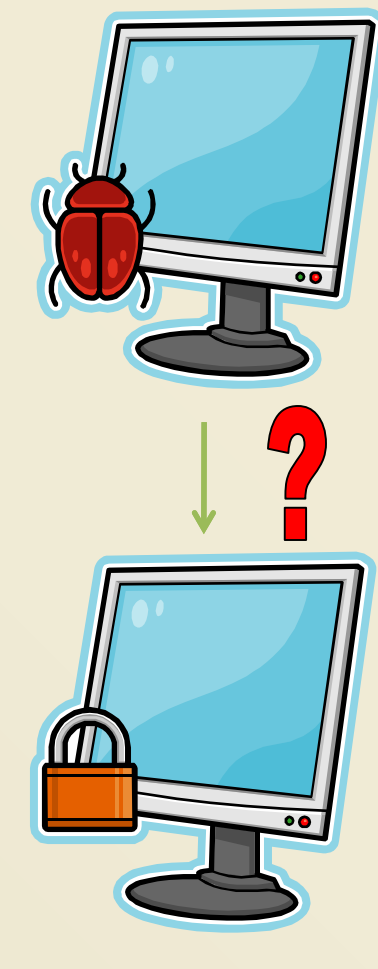


**FACTS**

- End systems are insecure
- Installed software is vulnerable and may get hacked
- Malware is not detectable from within
- Security experts do not build systems
- Designers do not understand security
- Programmers do not know what secure coding is
- Private information is easy to steal
- Viruses have unlimited power

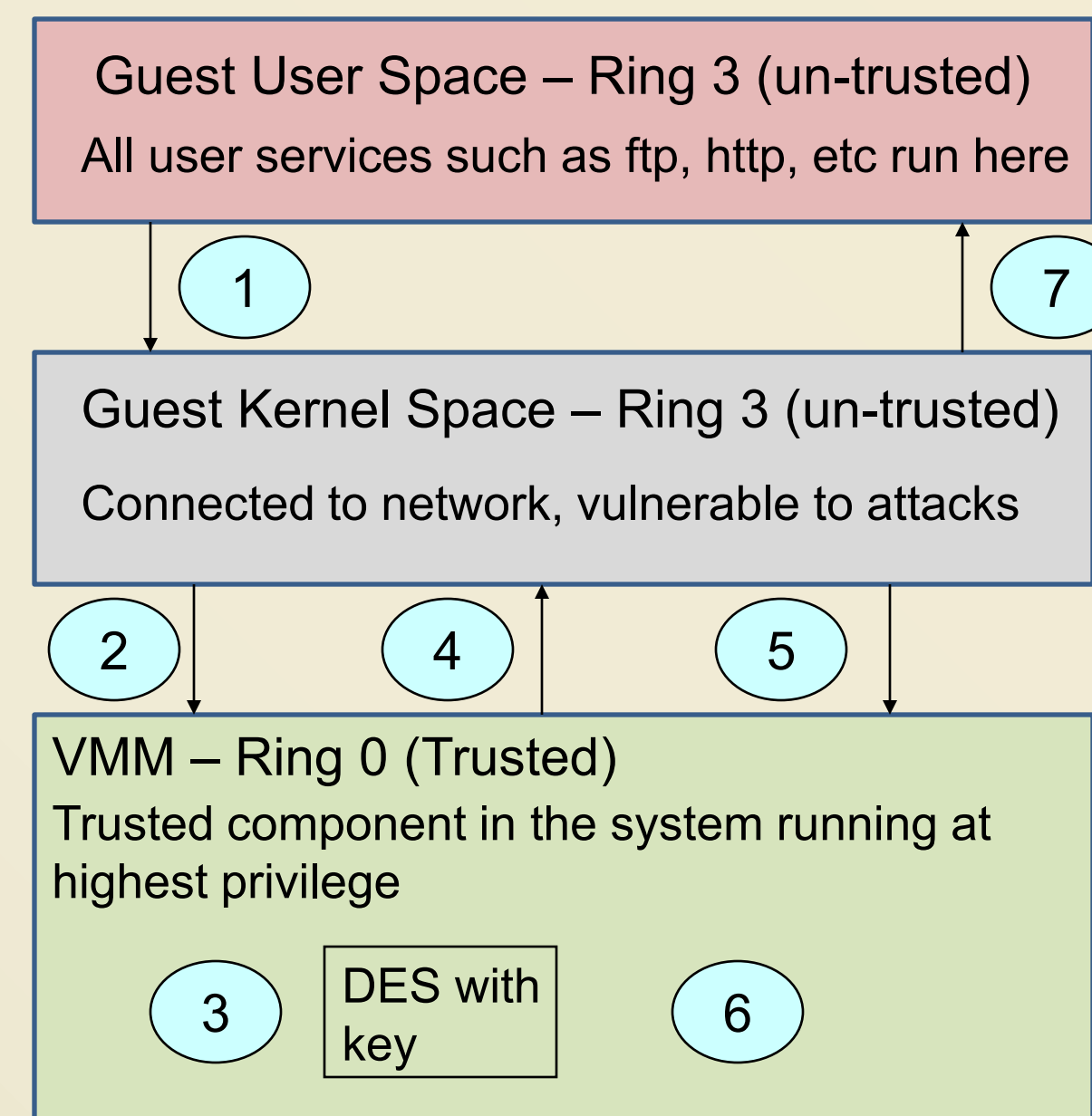
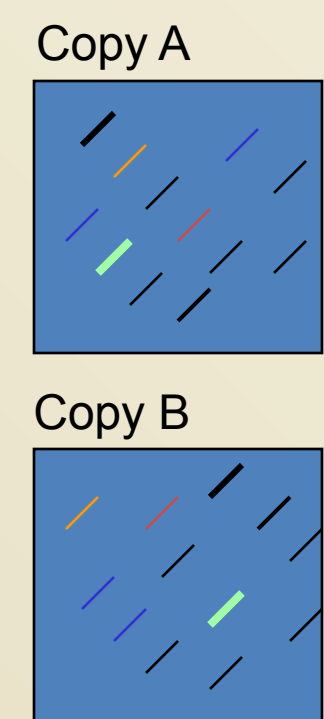
**FIXING HOLES**

- Software Diversity
- Remote Attestation
- Virtual Machines
- Data Hiding
- Code Obfuscation
- Verified Execution
- Integrity Checking



**SOFTWARE DIVERSITY**

- Same code/data but different layout
- Creates genetic diversity among binaries
- Exploitation by static analysis (pre computation) difficult



**VMM**

- Store secret data in hypervisor
- Hypervisor provides encryption interface
- Ensure Hypervisor attests calling function

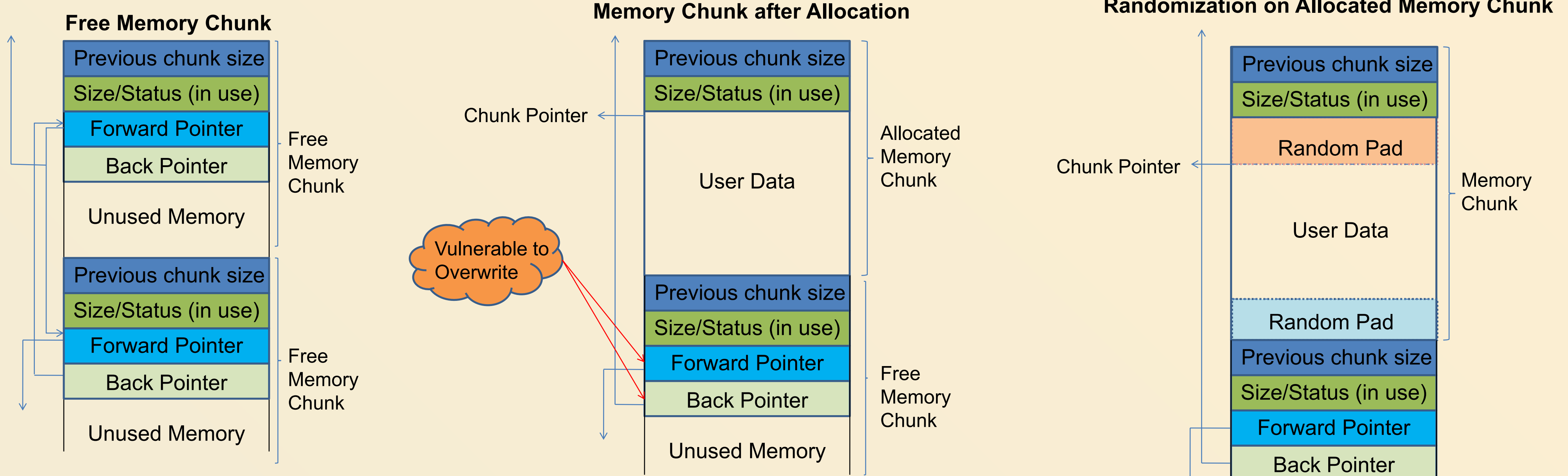
## Heap Randomization

**NEED**

- Heap based overflow overwrites control structures of heap memory chunks
- Can alter flow of execution
- Randomization can mitigate effects of overflows
- Diminished predictability of memory locations

**RANDOMIZATION CONCEPTS**

- Identify functions that allocate and free memory on the heap (*malloc, calloc, free, etc*)
- Modify system libraries so that the functions add random pads between chunk allocations



**HEAP MANAGEMENT**

- Free memory maintained in form of doubly-linked lists of memory chunks
- Chunks broken off from the lists and provided to program when requested
- Control pointers (fwd/back) changed to accommodate allocation and de-allocation

**ATTACKS ON HEAP**

- Overflow causes control pointers to be overwritten
- Can in turn allow arbitrary memory overwrite (2 phase overwrite)
- Ex: *Unlink, Double-free*

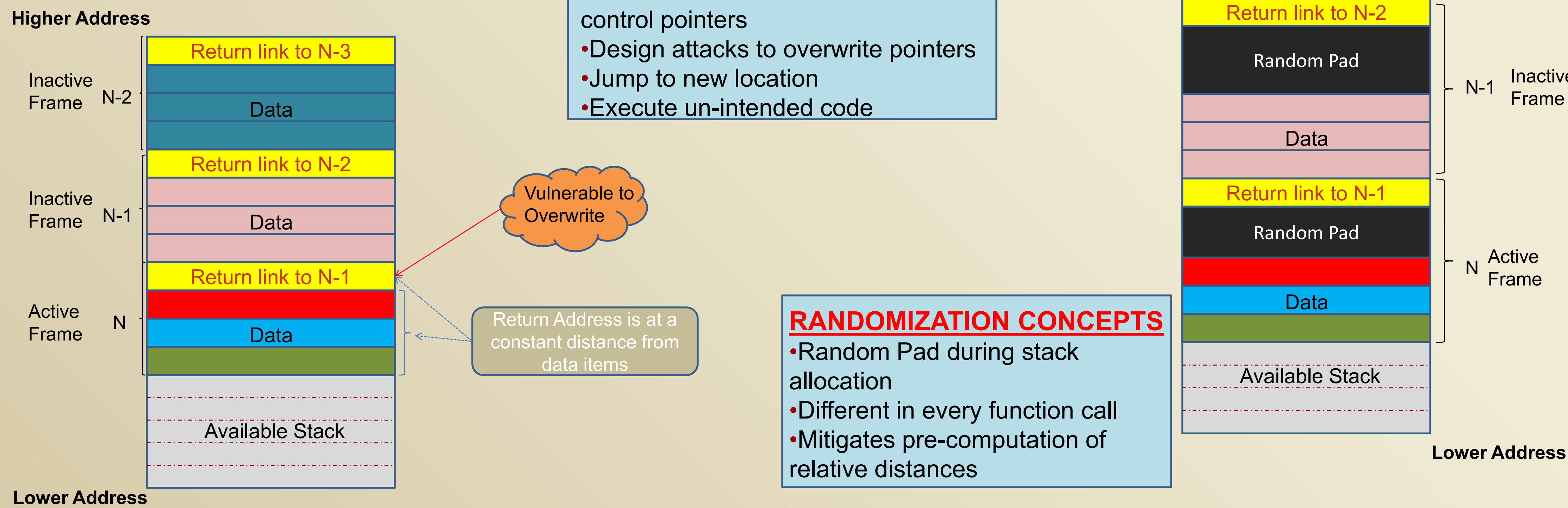
## Stack Randomization

**NEED**

- Stack smashing most common form of overflow based attack
- Oss and system libraries do not check bounds on buffers
- Function return pointers and data share stack

**ATTACK METHODS**

- Analyze binary
- Obtain relative locations of data and control pointers
- Design attacks to overwrite pointers
- Jump to new location
- Execute un-intended code



**RANDOMIZATION CONCEPTS**

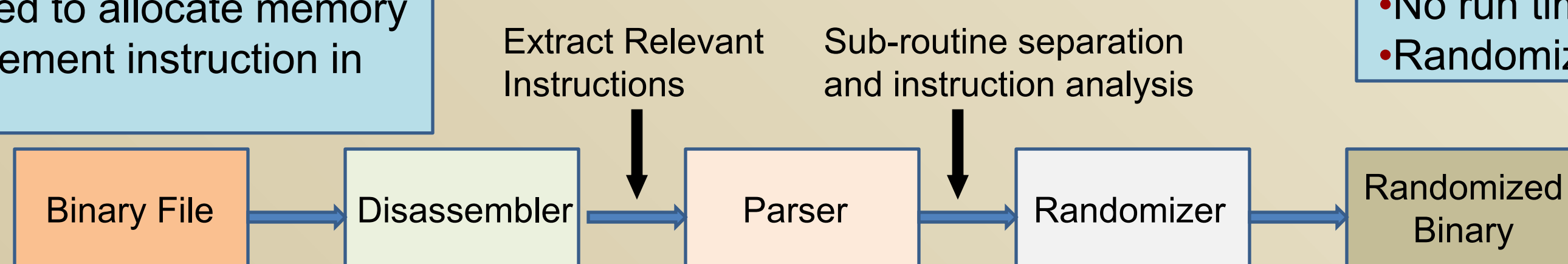
- Random Pad during stack allocation
- Different in every function call
- Mitigates pre-computation of relative distances

**IMPLEMENTING RANDOMIZATION**

- Stack allocation determined during compilation
- Stack pointer decremented to allocate memory
- Replace operand of decrement instruction in binary with random value

**REQUIREMENTS**

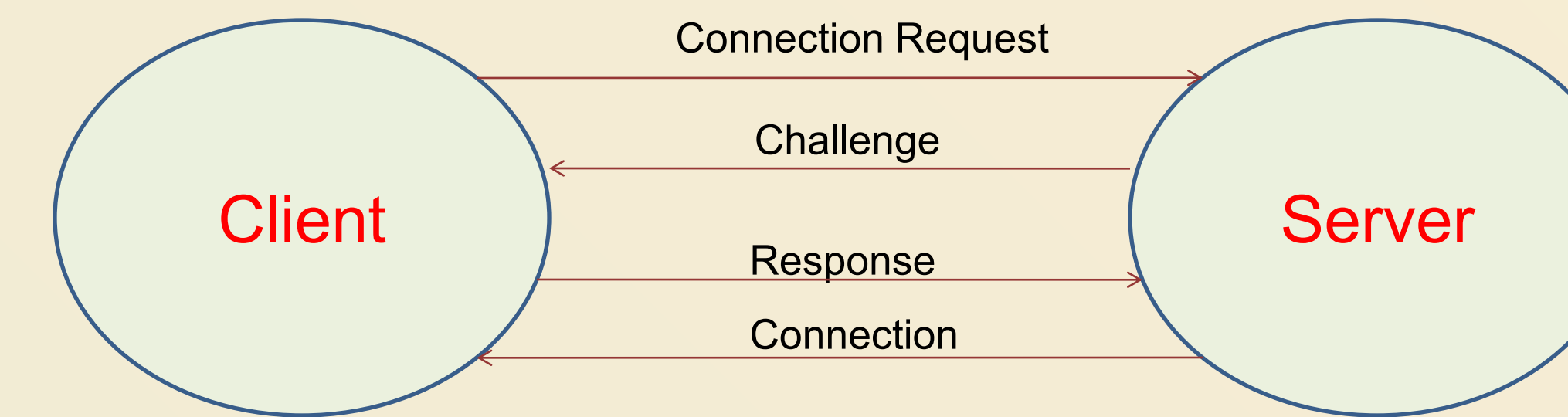
- No compiler or system support
- No run time overheads
- Randomization of legacy binaries possible



## Remote Attestation

**NEED**

- Smart viruses patch on various components
- Difficult to detect malwares from within the OS
- Integrity checks should be performed by external entities



**REMOTE ATTESTATION**

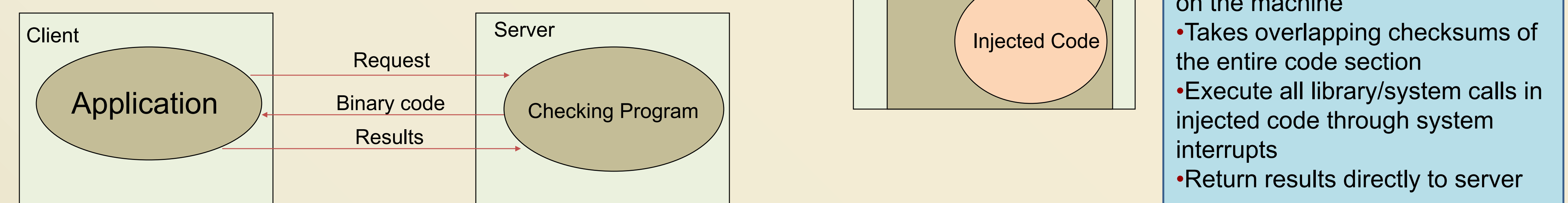
- Trusted host provides attestation for client
- Set of methods to determine if an application has been tampered
- Primarily used by TCG for DRM enforcement
- Can be extended to check the integrity of the OS

**CODE GENERATION**

- Code generated for every request
- Code passed through pre-processor to ensure randomization for every instance of verification

**METHOD**

- Attestation based entirely in software
- Client receives executable code from server
- Client program injects code on itself
- Code computes integrity and returns results
- Injected code verifies that it was not bounced to another location



**EXECUTION**

- Execute code not initially present on the machine
- Takes overlapping checksums of the entire code section
- Execute all library/system calls in injected code through system interrupts
- Return results directly to server