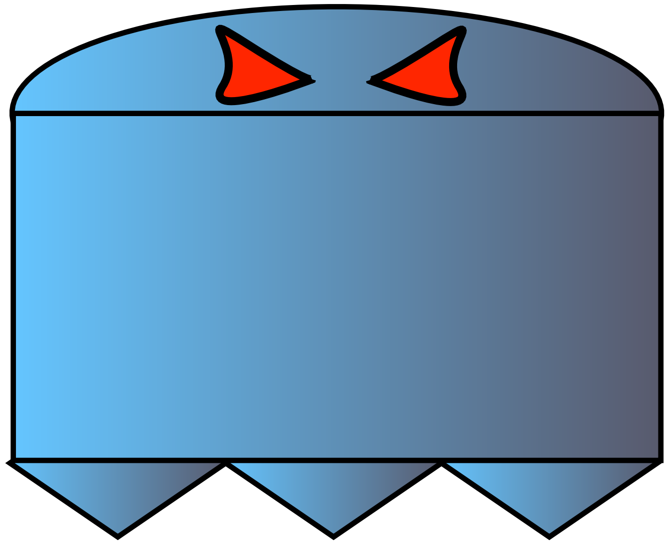
Virtual Ghost: Protecting Applications from Hostile Operating Systems

John Criswell, Nathan Dautenhahn, and Vikram Adve



New Job

New Job

Do You Trust Your Operating System?

Online Shopping!

Do You Trust Your Operating System?

Online Shopping!

Filing Taxes!

Do You Trust Your Operating System?

Online Shopping!

Filing Taxes!

Do You Trust Your Operating System?

Medical Data!

Filing Taxes!

Online Shopping!

Do You Trust Your Operating System?

Voting Machines!

Medical Data!

Filing Taxes!

Online Shopping!

Do You Trust Your Operating System?

Voting Machines!

National Security!

Medical Data!

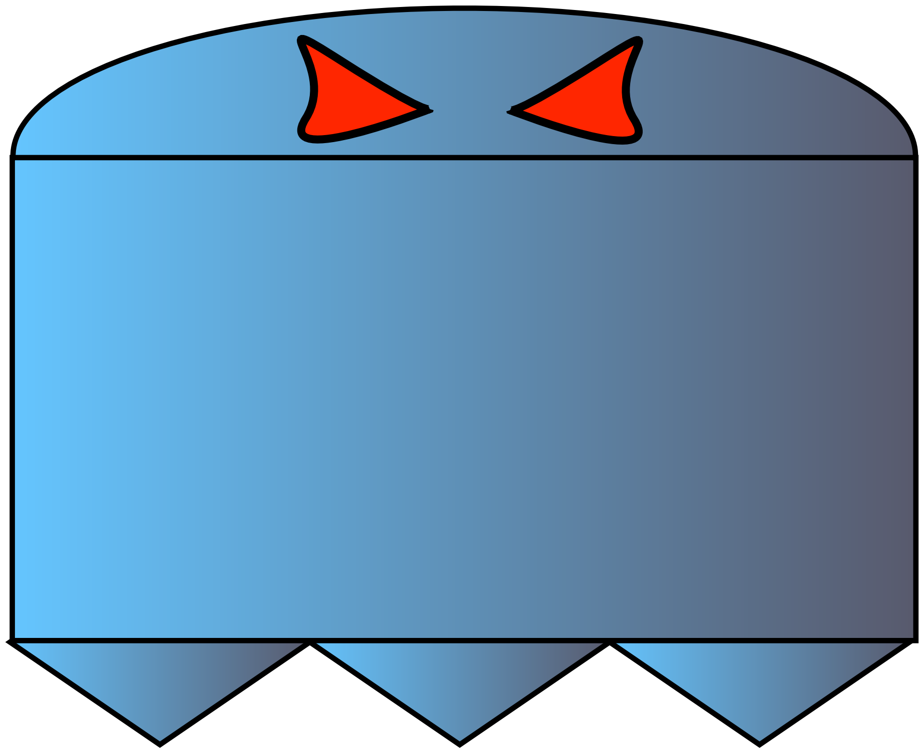
Commodity Operating Systems Are Vulnerable!

Vulnerability	Examples
Buffer Overflows	BugTraq ID 12911, 13589, 13207, 13225, 12295
Integer Overflows	BugTraq ID 10179, 63707
Information Leaks	BugTraq ID 8831, 64677, 64746, 64742, 62405
Kernel-level Malware	Adore rootkit

If the operating system kernel is exploited, all security guarantees are *null* and *void*.

Virtual Ghost Contributions

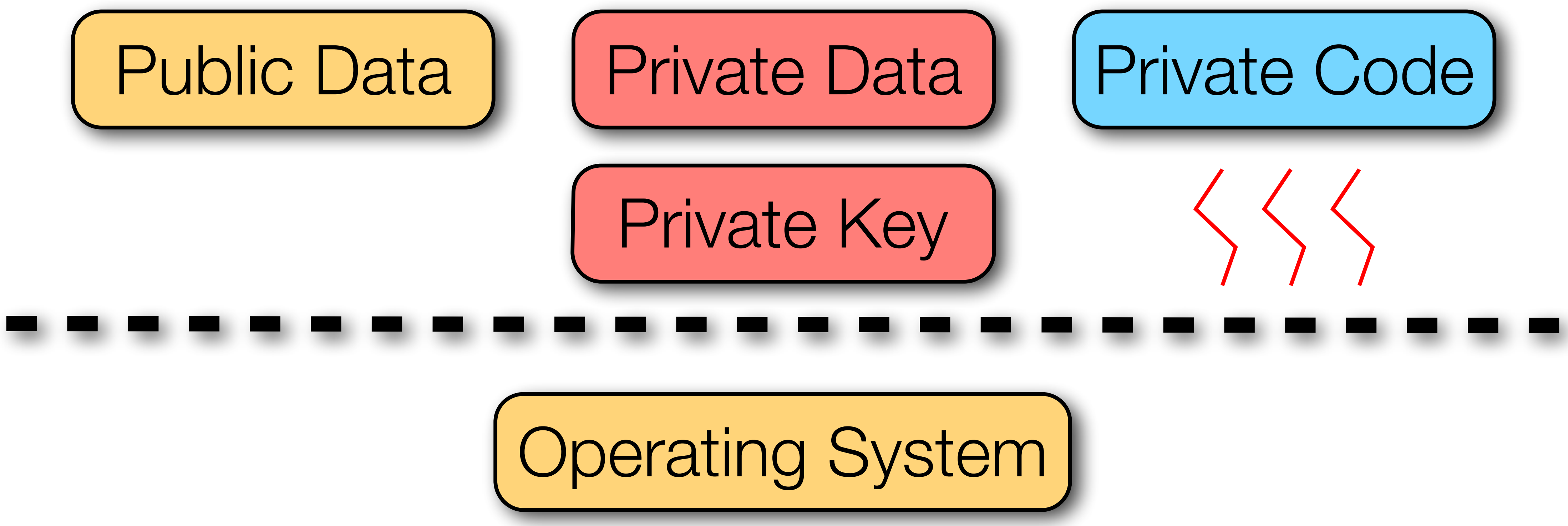
- *Protects* application data *confidentiality* and *integrity*
- Uses *compiler techniques* thanks to *LLVM*
- *Same privilege level* as kernel
- *Faster* than hypervisor-based approaches



Outline

- Motivation
- ***Design***
- Results
- Future Work

Goal: Application That *Protects* Itself from OS



Required Features

1. Private data and code
2. Incorruptible control flow
3. Reliable encryption key delivery

Challenges

Challenges

1. Processor lets privileged software access all memory

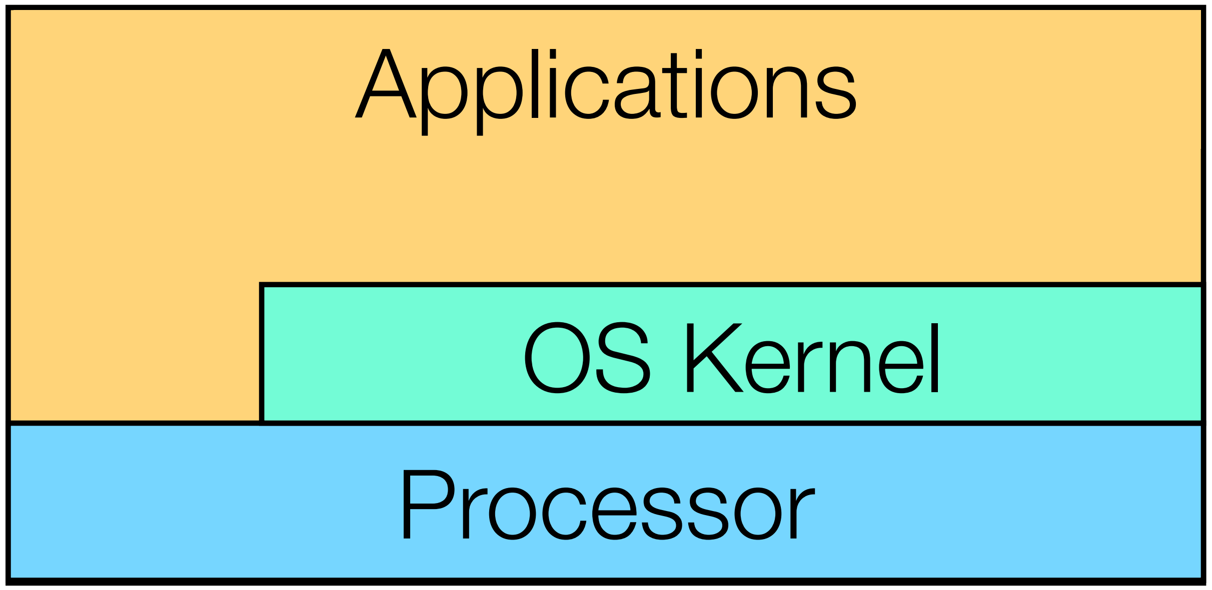
Challenges

1. Processor lets privileged software access all memory

2. Operating System *must* manipulate application state

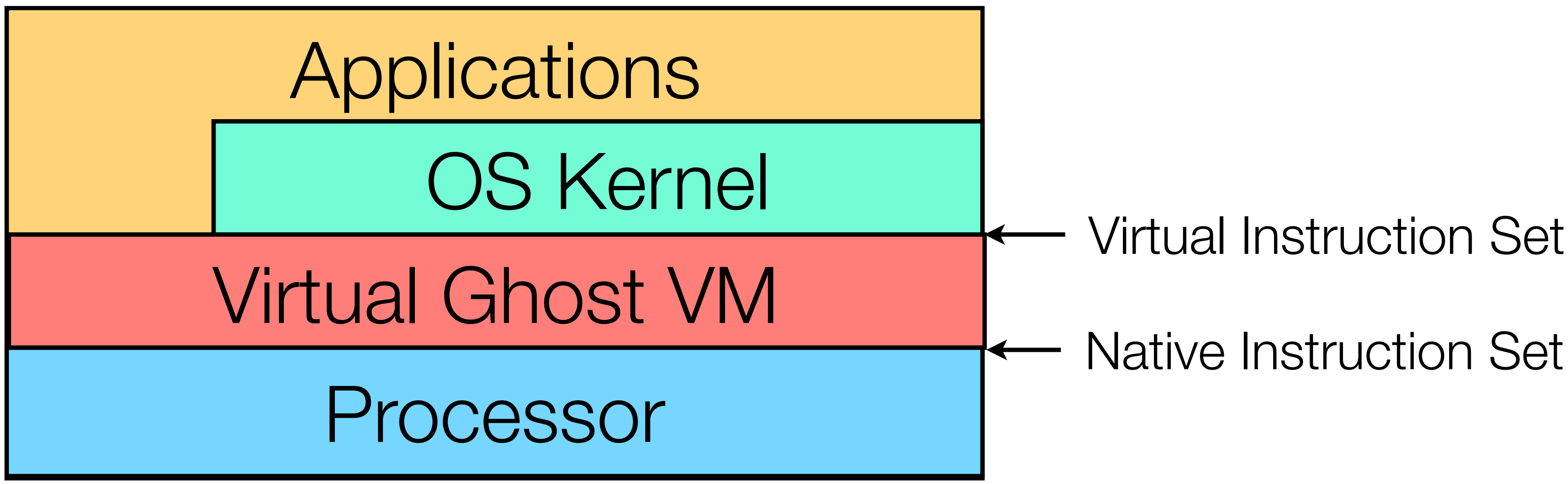
- Process and thread creation
- Executing new programs (exec() family of system calls)
- Signal handler dispatch

Virtual Ghost



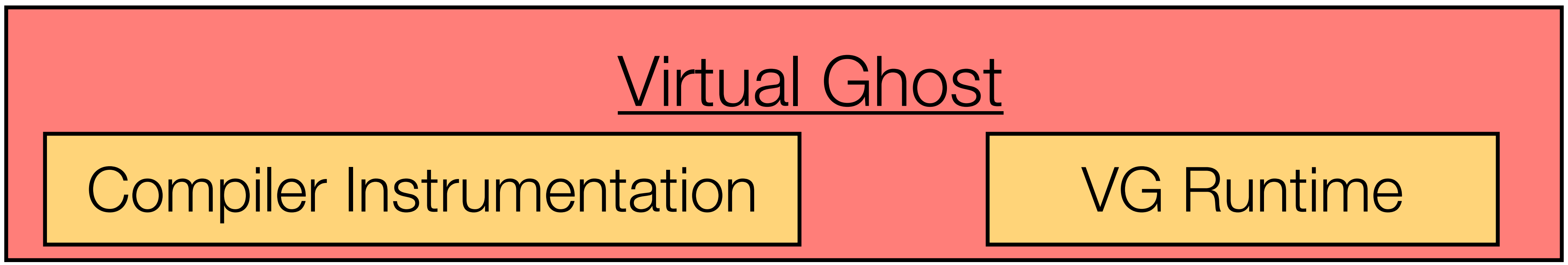
- OS compiled to virtual instruction set
  - Designed to be easy to analyze and instrument
  - Low-level instructions (SVA-OS) replace assembly code
- Translate ahead-of-time, boot-time, or run-time

Virtual Ghost

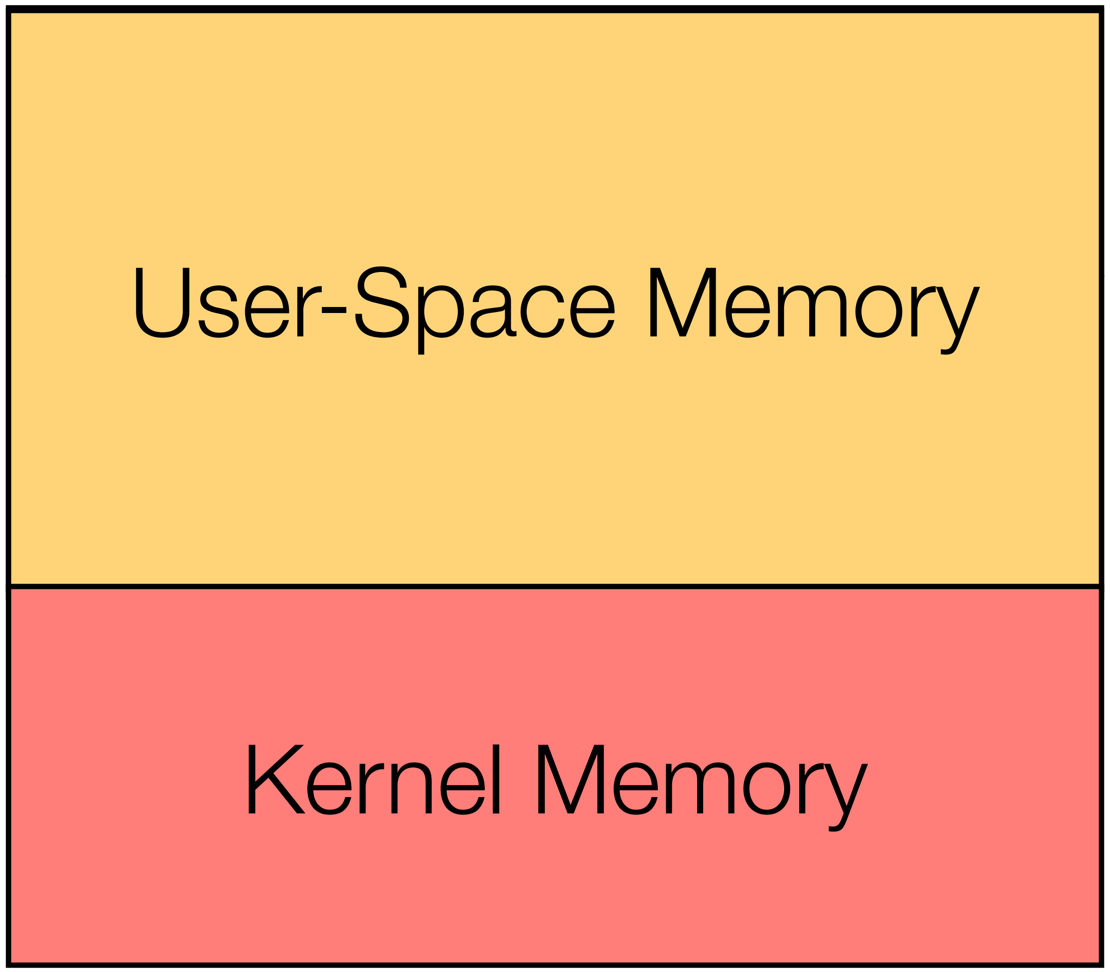


- OS compiled to virtual instruction set
  - Designed to be easy to analyze and instrument
  - Low-level instructions (SVA-OS) replace assembly code
- Translate ahead-of-time, boot-time, or run-time

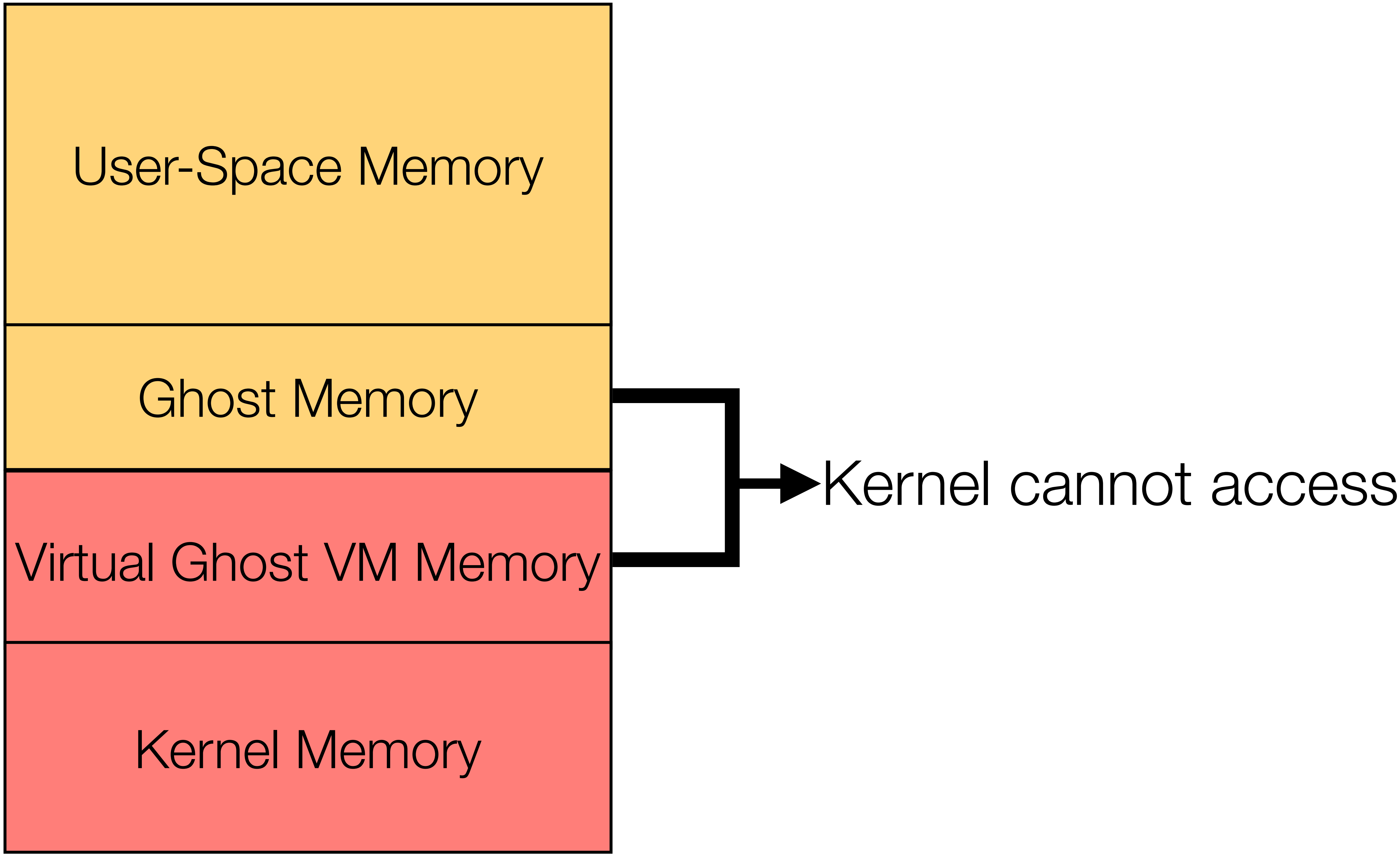
Virtual Instruction Set



- SVA-Core: Compiler Instrumentation
  - Based on LLVM IR: Typed, Explicit SSA form
  - Sophisticated compiler analysis and instrumentation
- SVA-OS: Virtual Ghost Runtime
  - OS-neutral instructions to support a commodity OS
  - Encapsulates & controls hardware and state manipulation
  - Implemented as a run-time library linked into kernel

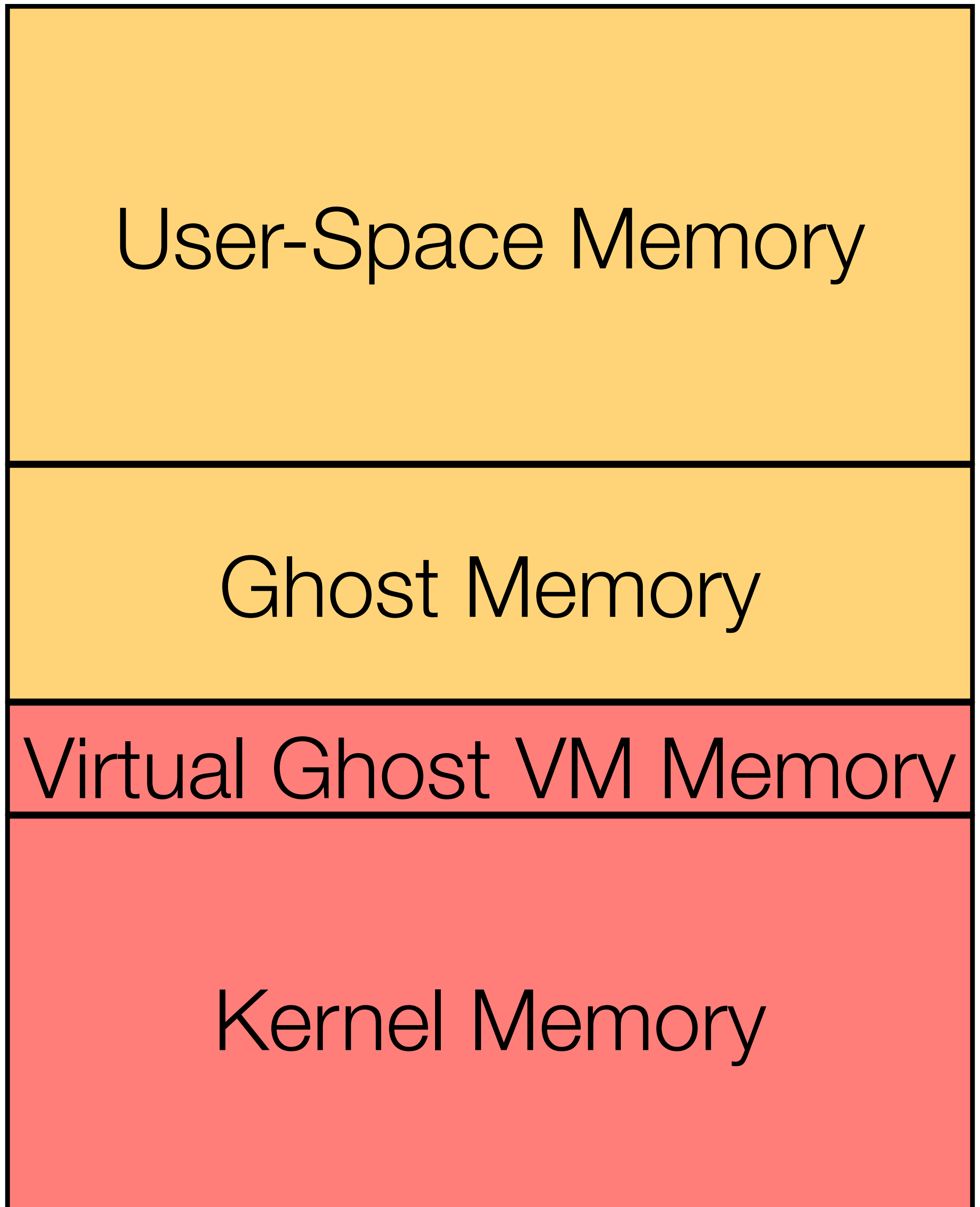


Private Data and Code | Ghost Memory



Private Data and Code

Ghost Memory Instrumentation



- Software Fault Isolation
  - Protects Ghost and VM Memory
  - Avoids TLB flush
- Control-Flow Integrity
  - Prevents instrumentation bypass
  - Provides kernel protection

Software Fault Isolation Instrumentation

Ghost Memory

0xffffff0000000000 – 0xffffff8000000000

```
mask = (((p >> 32) == 0xffffff00 ? 0x8000000000 : 0);
p |= mask;
store v, *p;
```

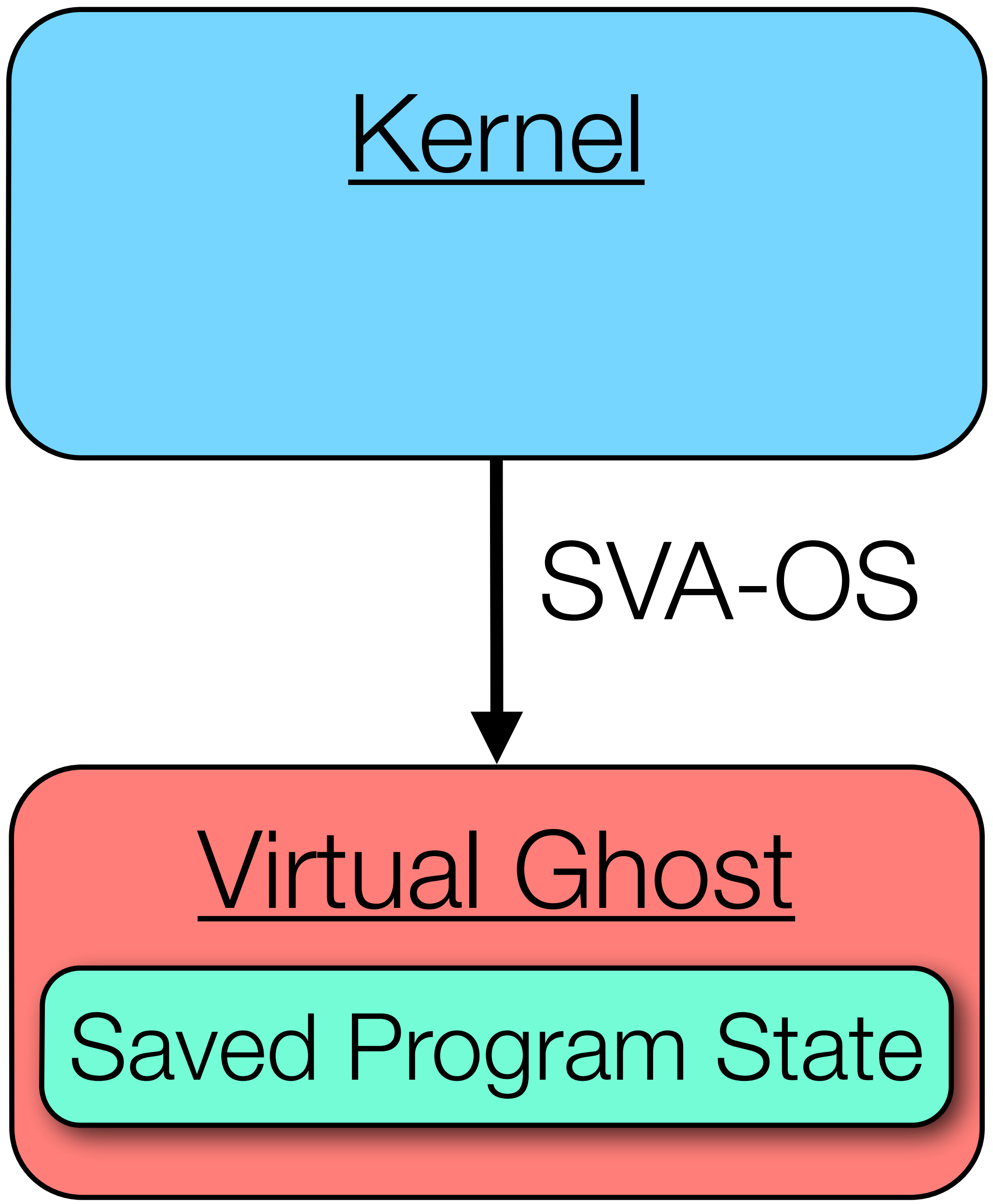
Control-Flow Integrity Instrumentation[1]

- Insert NOP labels at target addresses
  - Function entry
  - Call sites
- Instrument all computed jumps
  - Bitmask to force pointer into kernel code
  - Check label at target of computed jump

```
foo:
  NOP Label
  pushq %rax
  add %rax, %rax
  sub $1, %rbx
  call foo
  NOP Label
  popq
```

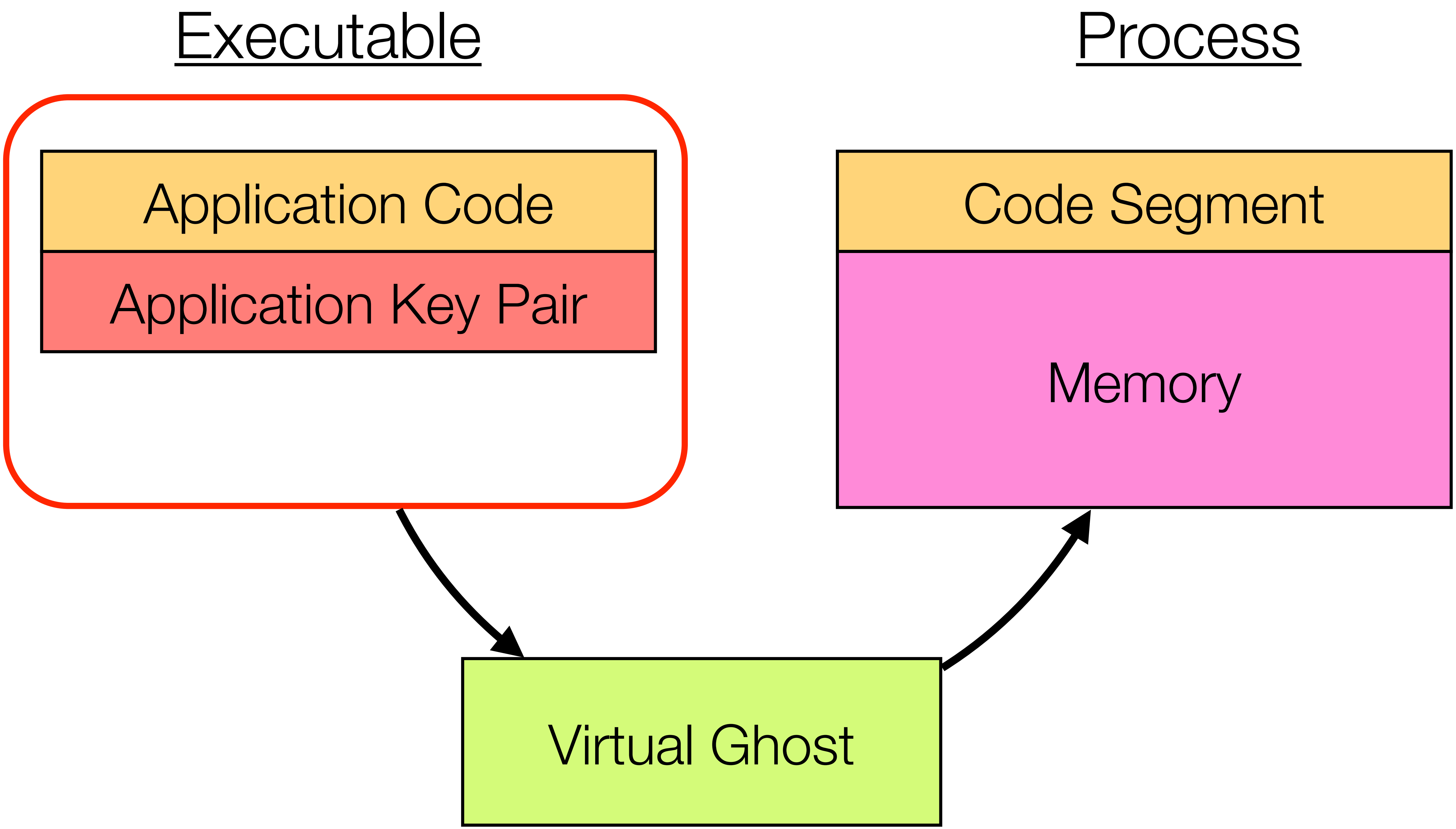
1. Zeng, Tan, and Morrisett, *Combining Control-flow Integrity and Static Analysis for Efficient and Validated Data Sandboxing*, CCS 2011

Secure Application Control Flow

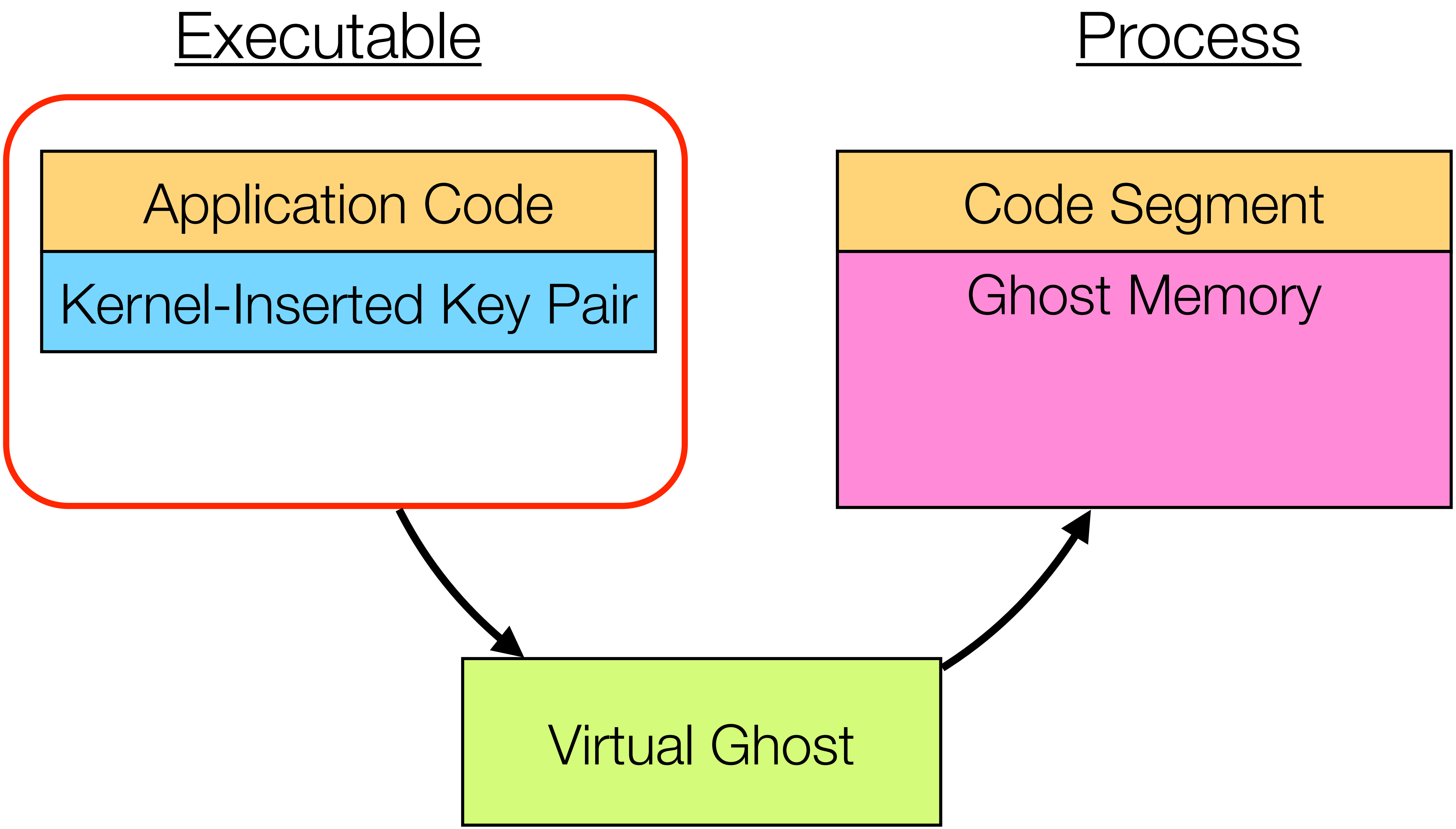


- Program state in VM Memory
  - OS cannot modify directly
- SVA-OS vets/performs changes
  - Signal handler dispatch
  - Thread creation
  - Exec() system calls

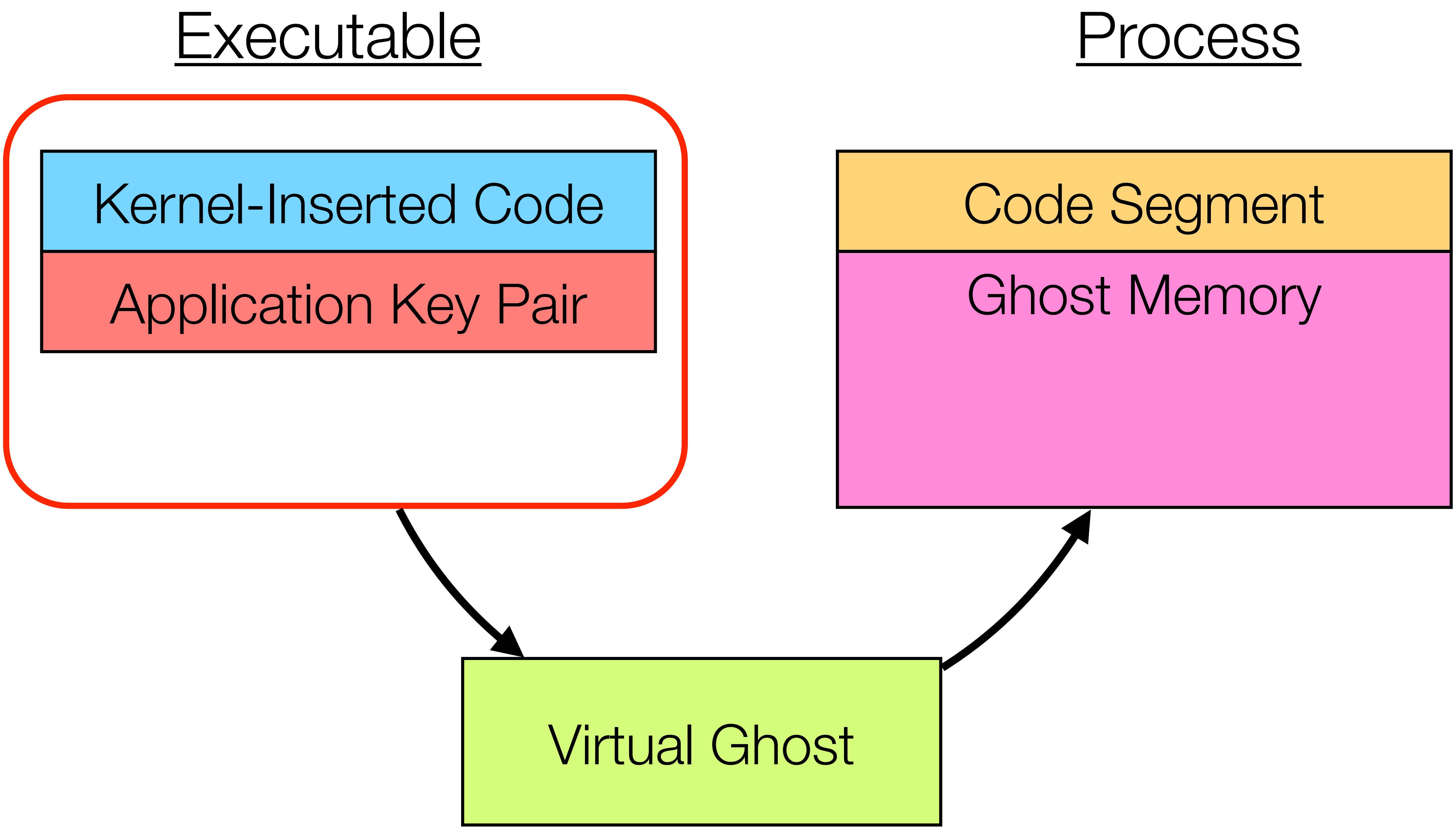
Secure Application Encryption Keys



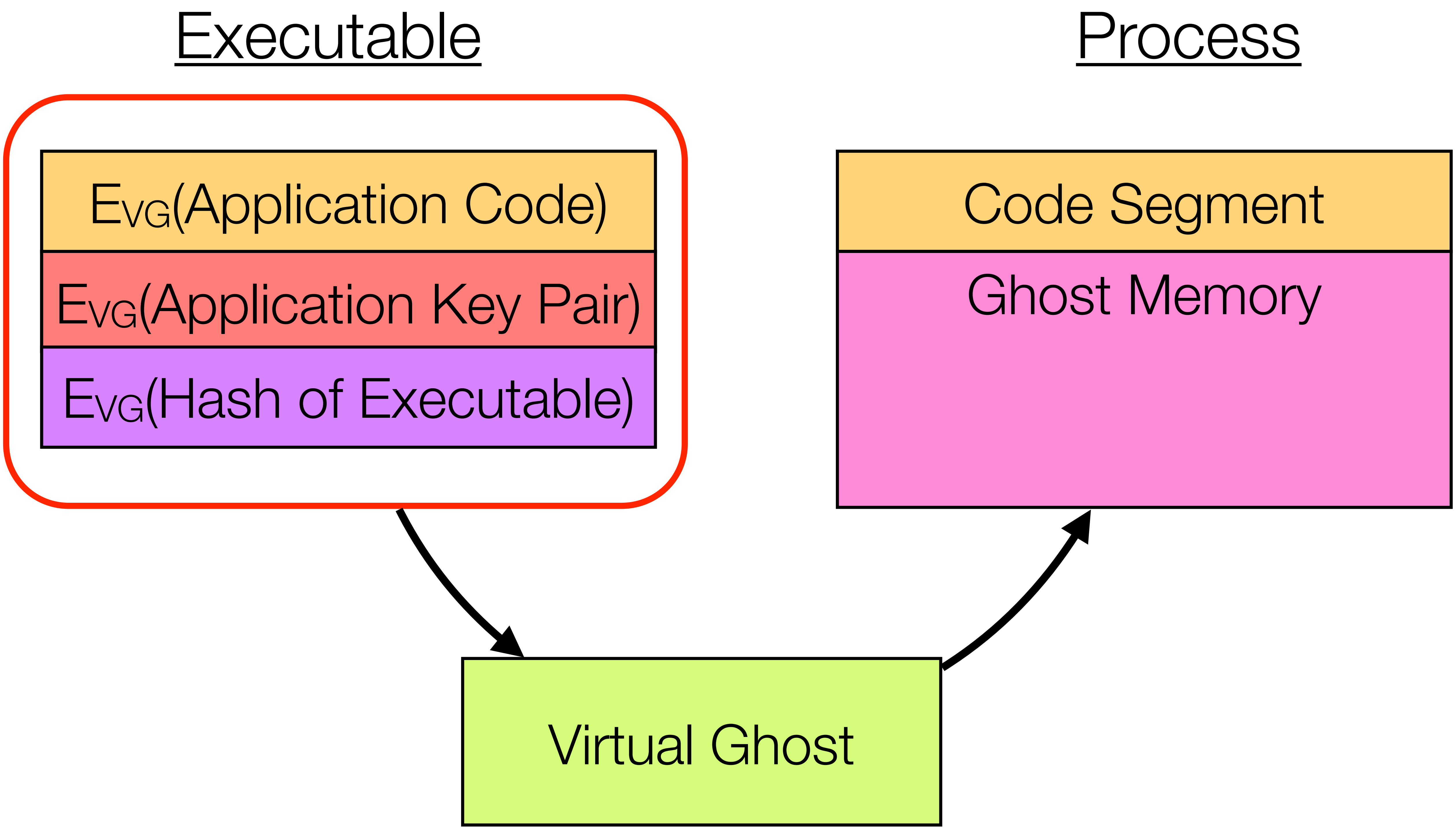
Kernel Injects Wrong Key



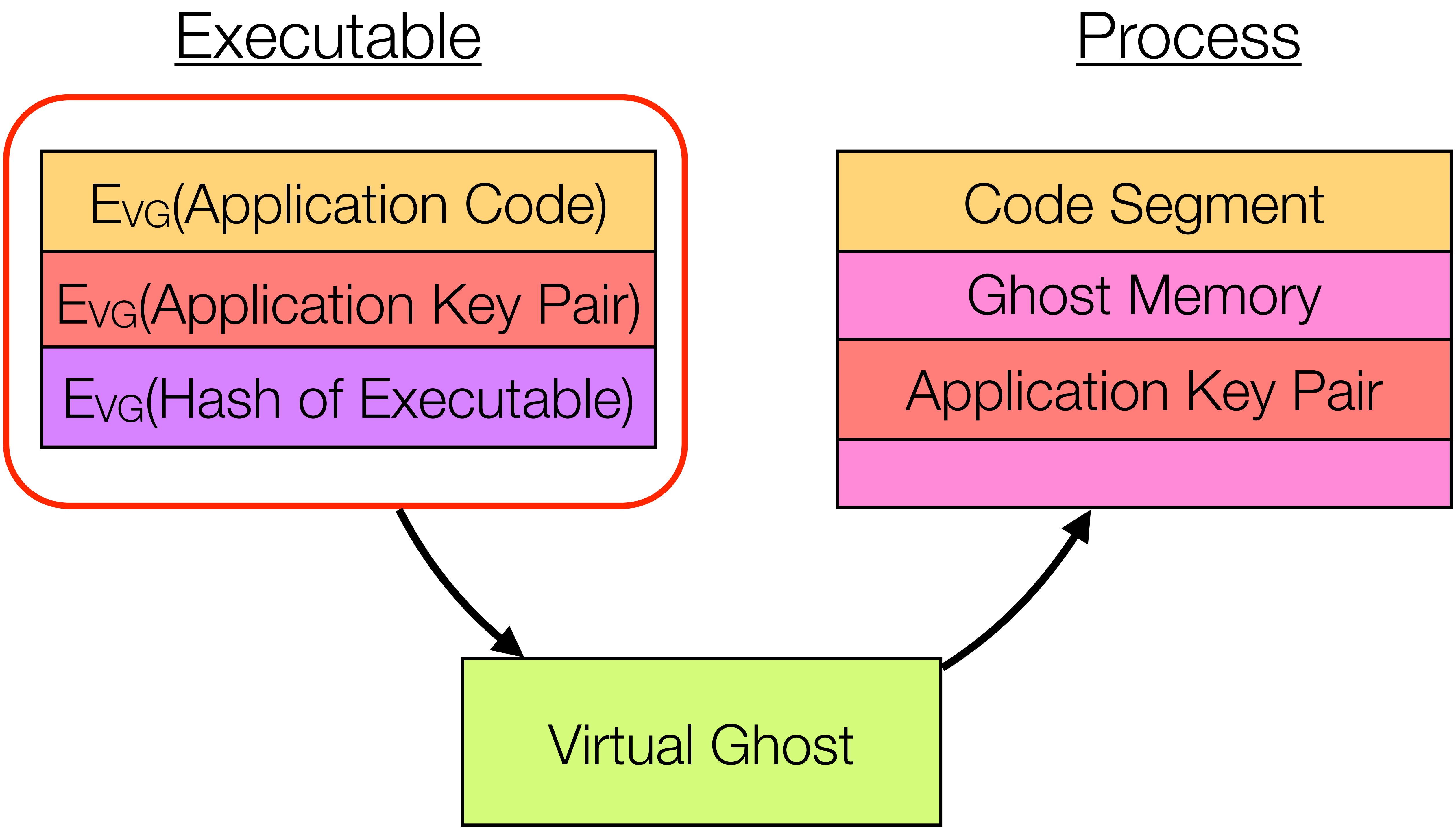
Kernel Replaces Code



Secure Application Encryption Keys



Secure Application Encryption Keys



Outline

- Introduction
- Design
- ***Results***
- Future Work

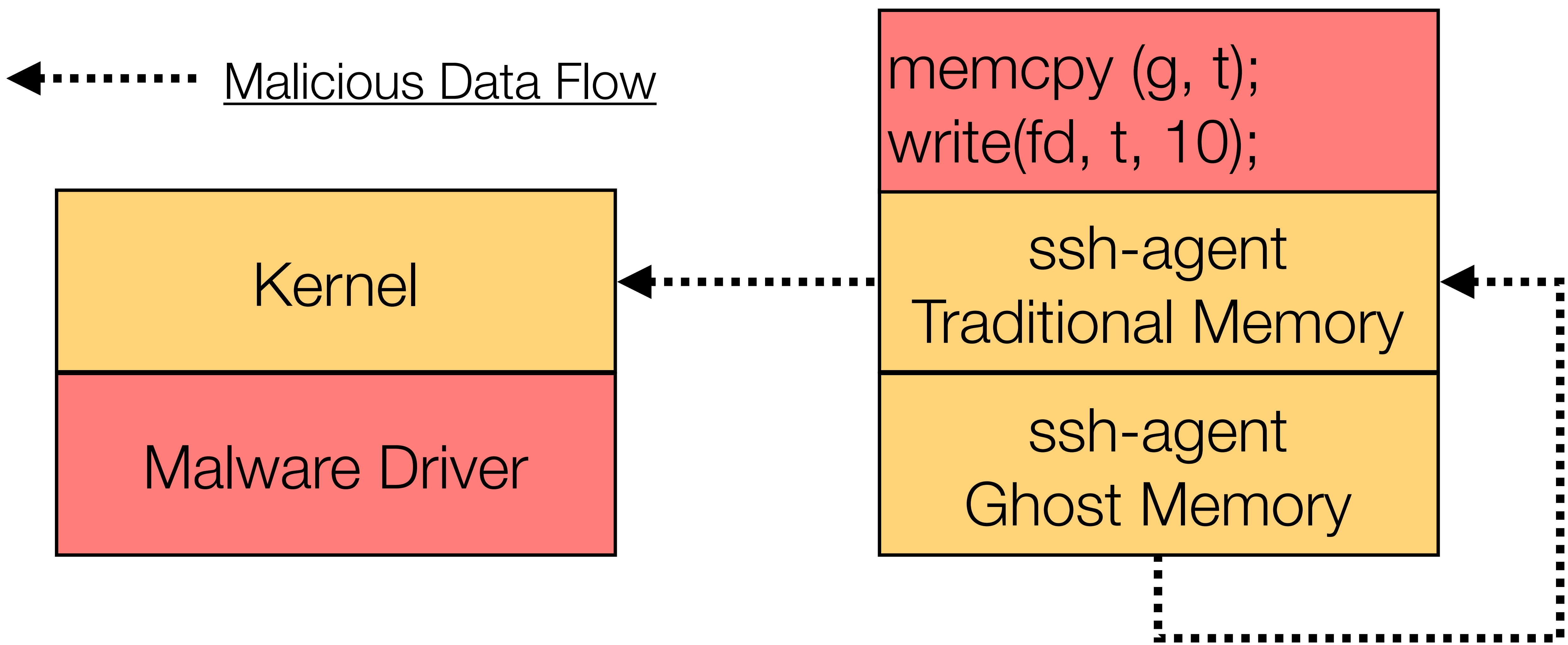
Implementation

- Developed a x86_64 64-bit implementation of Virtual Ghost
- Ported FreeBSD 9.0 to Virtual Ghost
  - FreeBSD compiles with LLVM out of the box
- Modified OpenSSH applications to use ghosting
  - `ssh` client
  - `ssh-agent` key-chain server
  - `ssh-add` utility

Kernel Malware Attack

Trick Application into Putting Data into the Clear

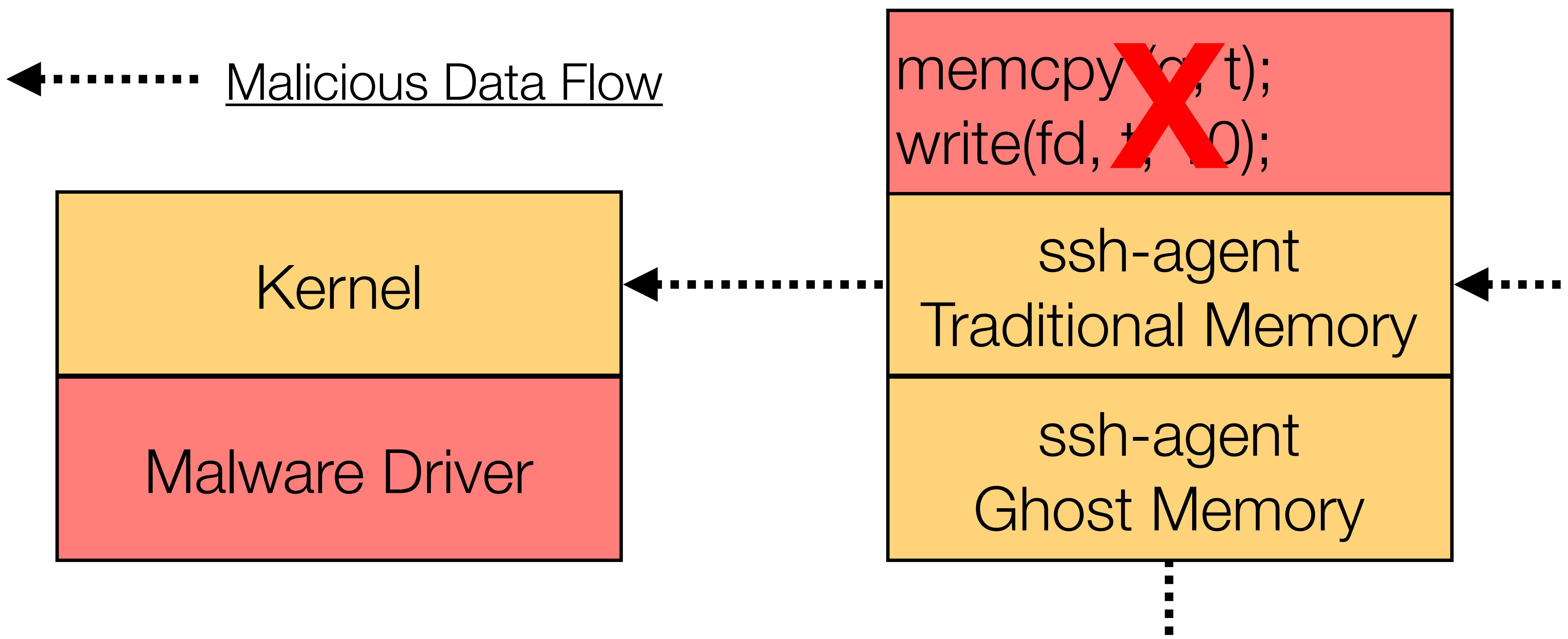
- Install signal handler to malicious code in application
- Malicious code copies data to traditional memory



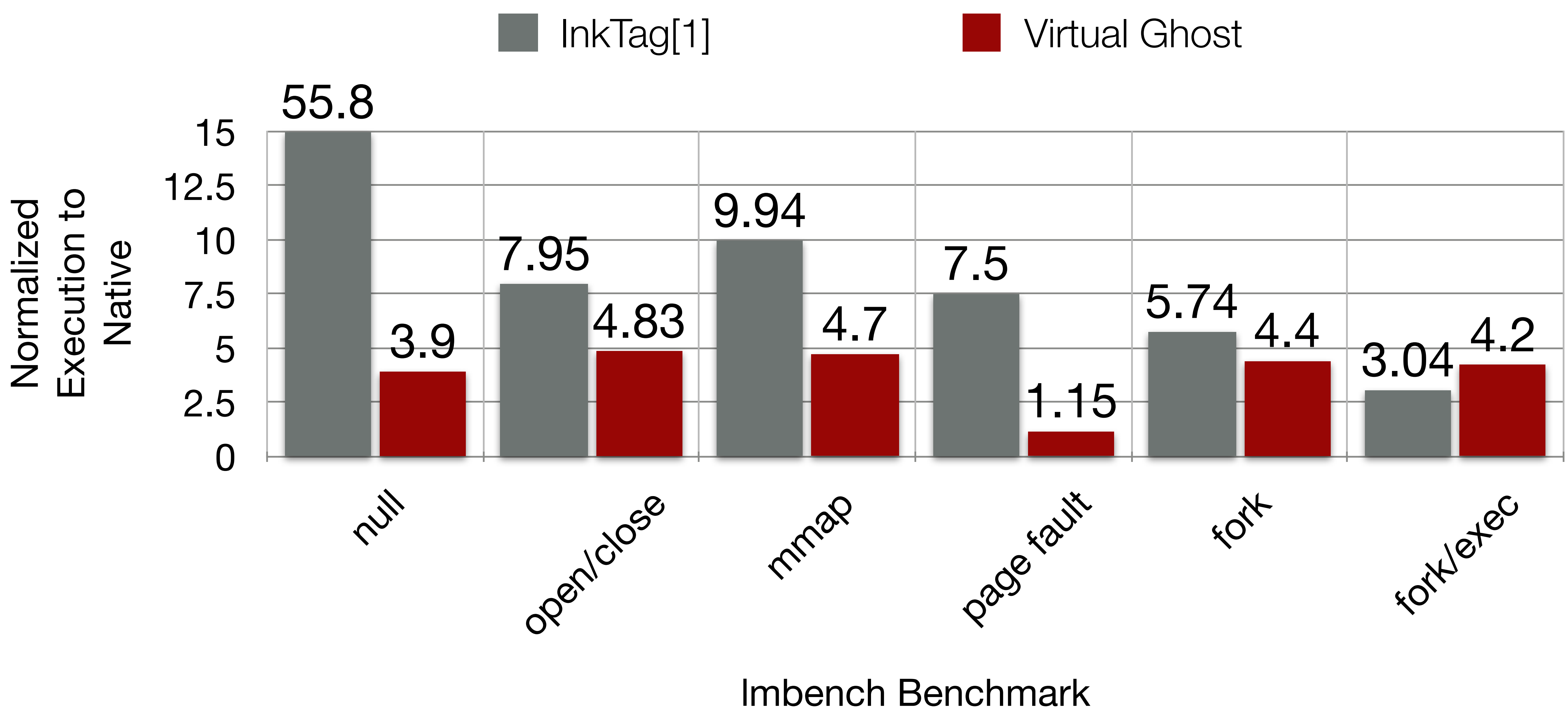
Kernel Malware Attack

Trick Application into Putting Data into the Clear

- Install signal handler to malicious code in application
- Malicious code copies data to traditional memory

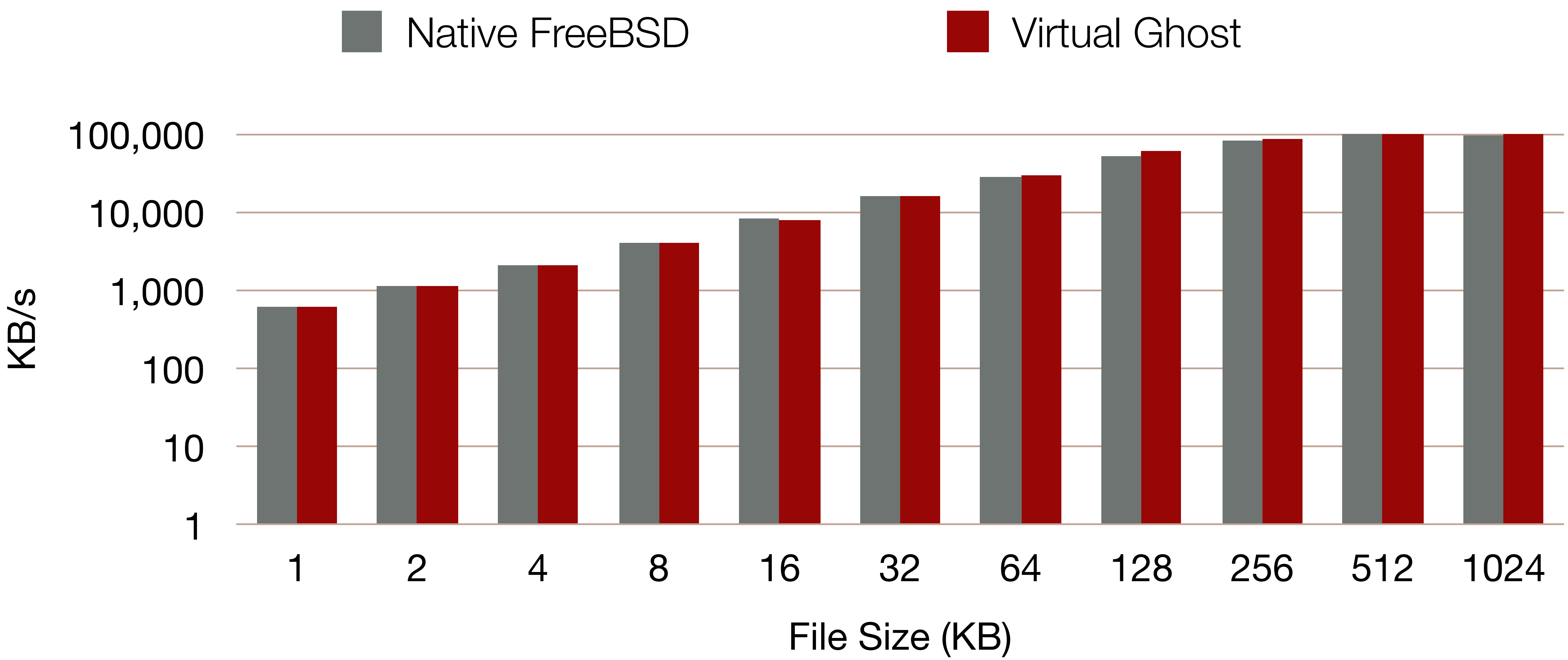


LMBench Execution Time Normalized to Native



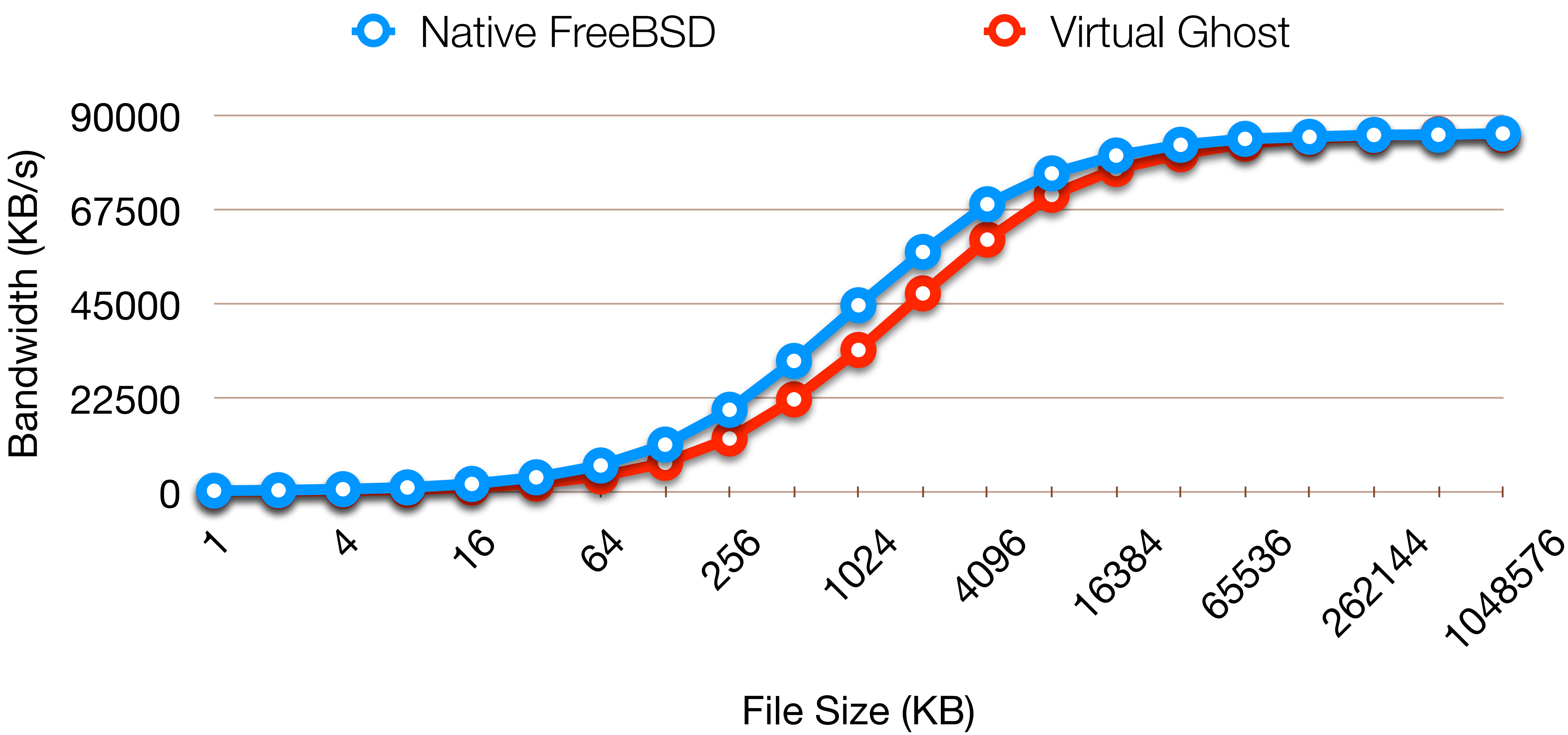
[1] InkTag: Secure Applications on an Untrusted Operating System, ASPLOS 2013

Web Server Performance for thttpd



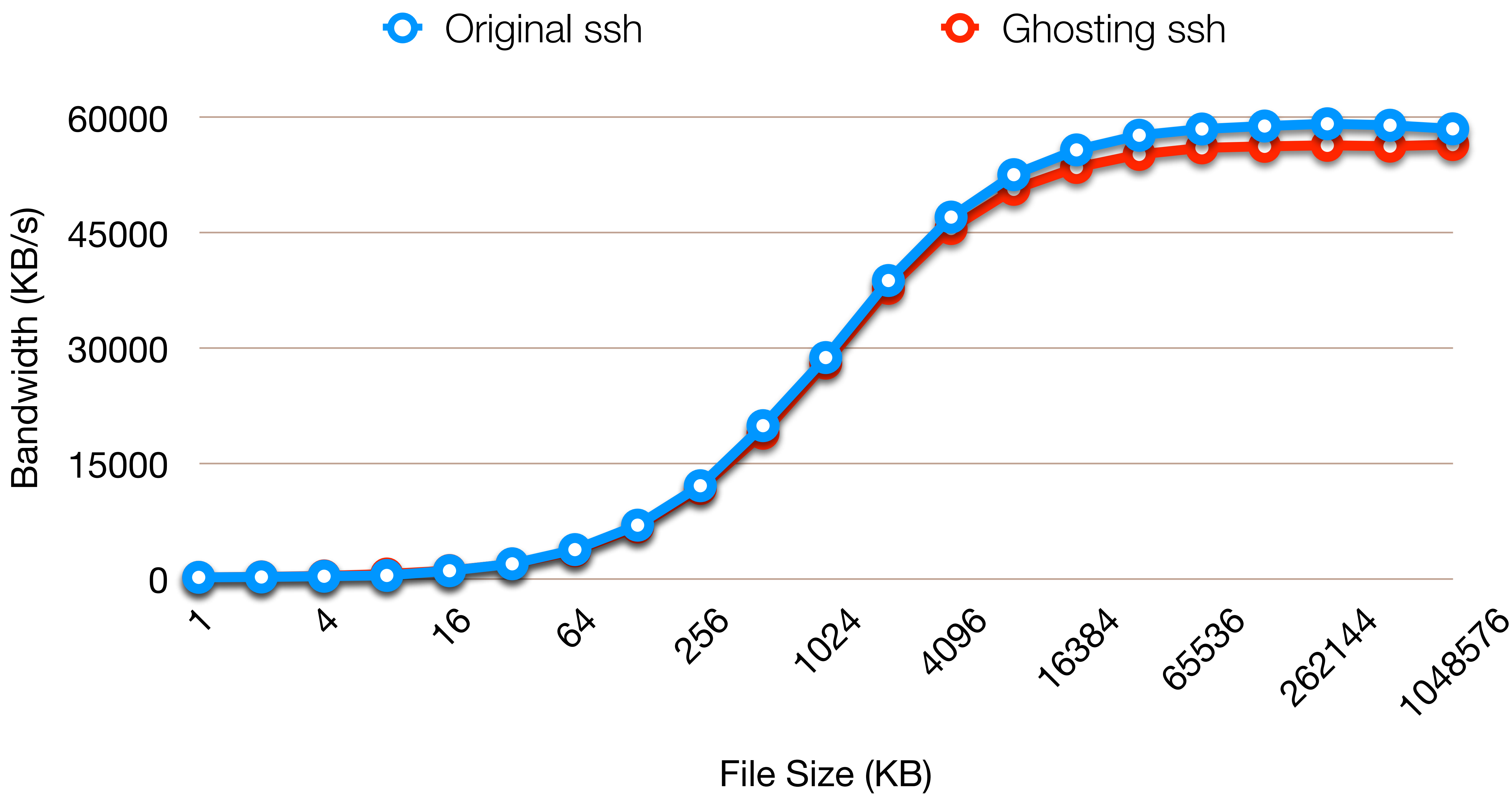
- ApacheBench: 100 clients, 100,00 requests
- Performance overhead negligible

Unmodified SSH Server Performance



- 23% reduction of bandwidth on average
- 45% reduction in worst case

Ghosting SSH Client Performance



- 5% reduction in worst case

Outline

- Introduction
- Design
- Results
- ***Future Work***

Future Work

- Improved performance
  - Advanced optimization (e.g., type safe check optimization)
- Cryptographic protocols for preventing OS attacks
  - Prevent replay attacks
- Compiler transforms to use Virtual Ghost features

Started Open-Source Release

- LLVM Compiler Extensions
- Virtual Ghost Run-time Library

Summary

- Virtual Ghost allows applications to protect themselves from an OS
- Uses compiler instrumentation
  - Keeps higher processor privilege levels free
- Faster than hypervisor-based approaches

See what we do at http://sva.cs.illinois.edu!