Oakland 2005

Hardware-assisted circumvention of self-hashing software tamper resistance

Glenn Wurster, Paul Van Oorschot, Anil Somayaji

School of Computer Science

Carleton University, Canada

Outline

- Self-hashing tamper resistance overview
- High level overview of our attack
- Hardware memory management design and attack details
- Results and implications

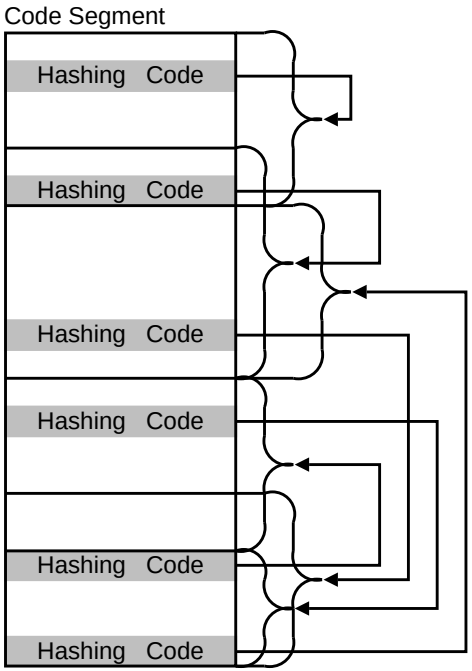
Self-Hashing Tamper Resistance Problem

- Protect an application binary against undetected modifications
  - Verifying that an application for DRM has not been modified
  - Protecting copy protection algorithms
  - Guard against unfair advantages in networked environments
- Do so without dependence on external hardware or software
- Use some form of self-hashing to detect changes
- Assumes a *Hostile Host* model

Self-Hashing Software Tamper Resistance

- Read into the code segment to compute a hash
- Rely upon a known good value to detect modifications
- Obscure reads into the code by hiding address calculations
- Protect the hashing code against alterations

A Network of Hash functions[ab]



[a]Chang et al. *Protecting Software Code by Guards*, DRM-2001

[b]Horne et al. *Dynamic Self-Checking Techniques...*, DRM-2001

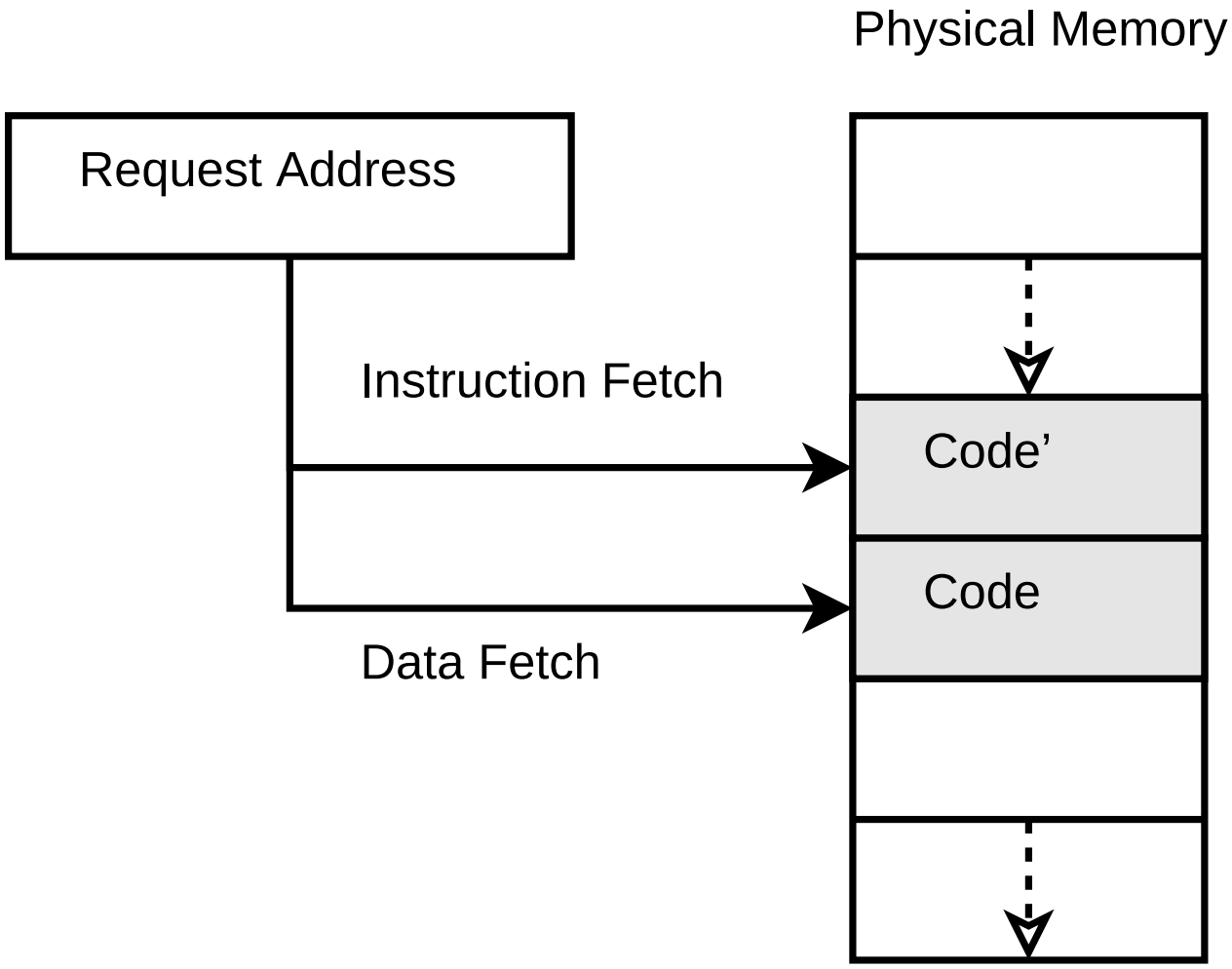
Our Results

- Self-hashing is not secure against attack on modern hardware
  - Can modify an application without being detected and without altering hashing algorithms
- Attack applies to proposals including:
  - Chang et al. DRM-2001
  - Horne et al. DRM-2001
  - Aucsmith, IHW-1996 (despite digital signatures)

Processor Design Elements Enabling our Attack

- There does not exist a 1:1 correspondence between virtual and physical addresses
- CPU caches are managed differently depending upon whether they contain information on program instructions or data

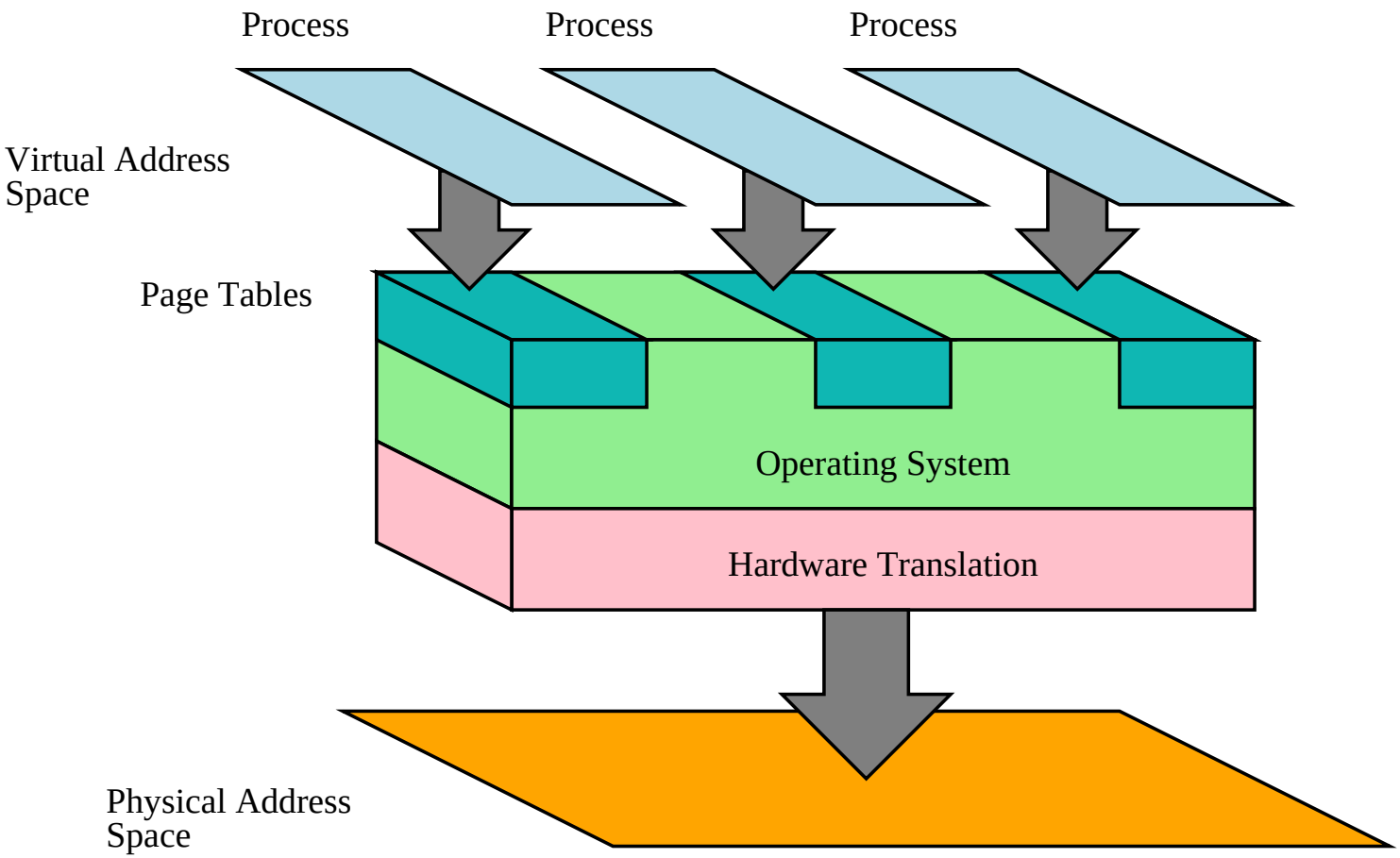
Graphical Representation of Our Attack



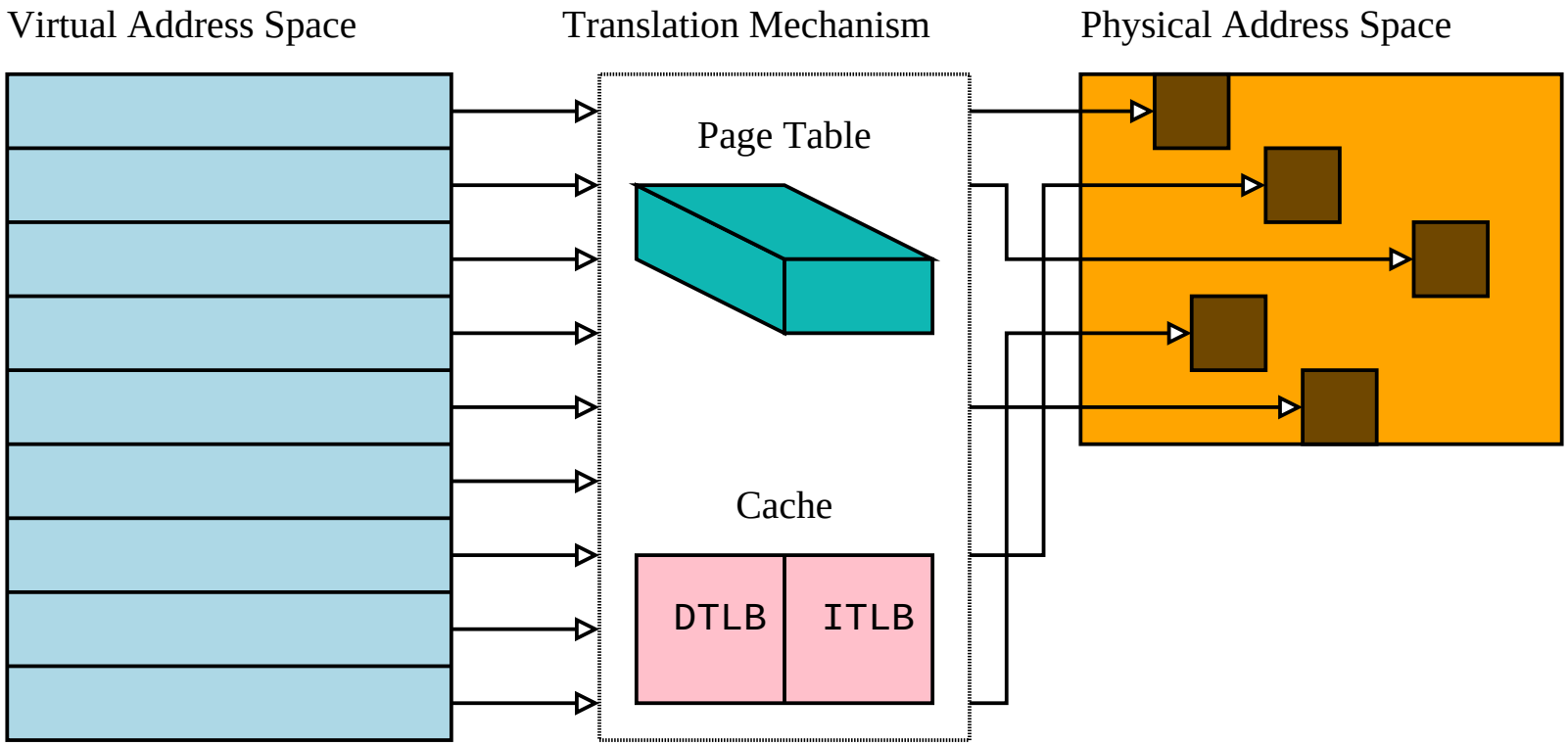
High-Level Overview of Attack

- Create a copy of the application which will remain unmodified
- Modify the application as desired
- Modify the kernel to contain the run-time attack code
- Load the modified application, installing and mapping both original and modified code pages in physical memory
- Run application - attack kernel vectors reads appropriately
  - This is the core of our attack

Virtual Memory Translation



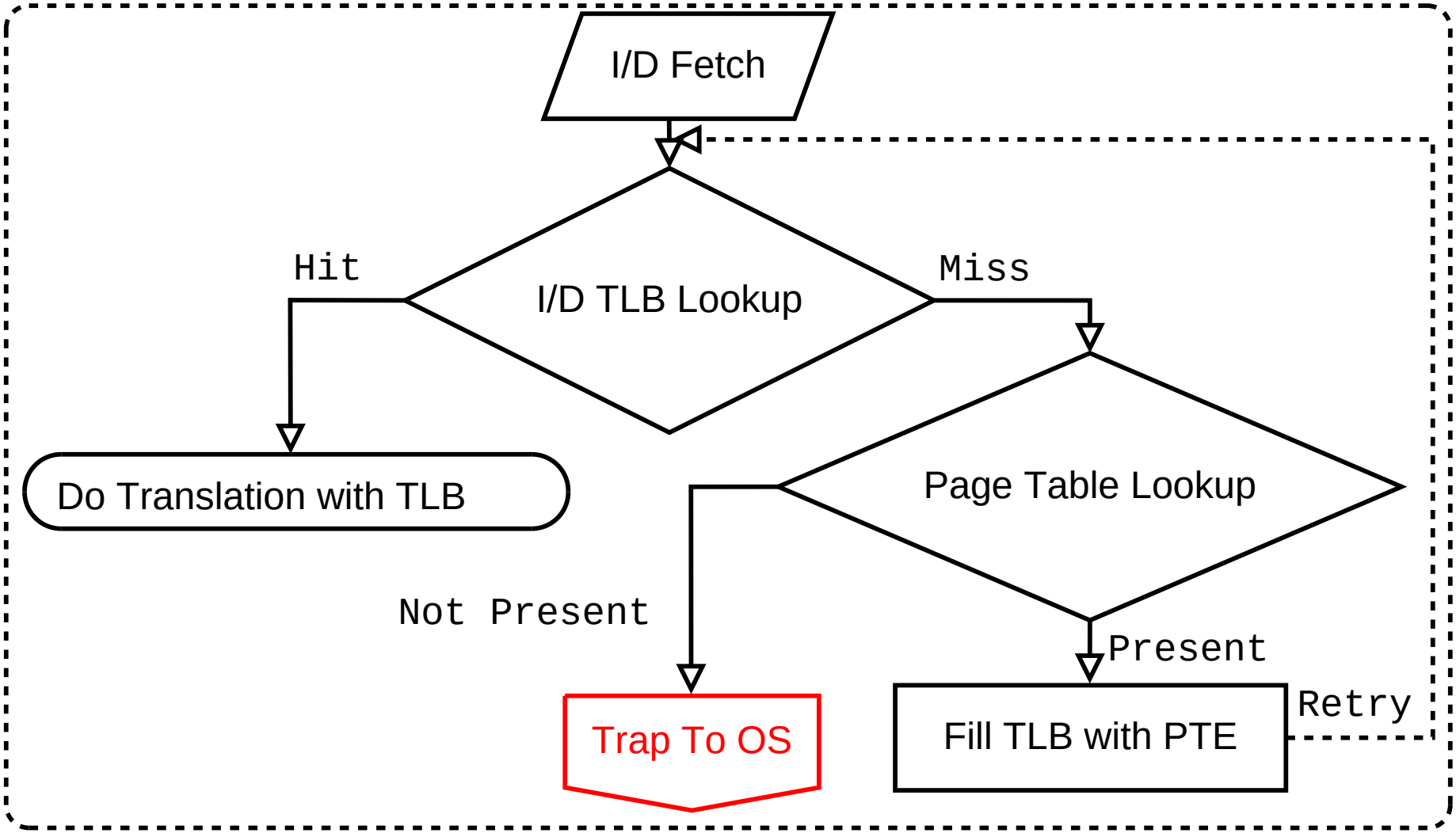
Hardware Architecture: Virtual Memory Translation



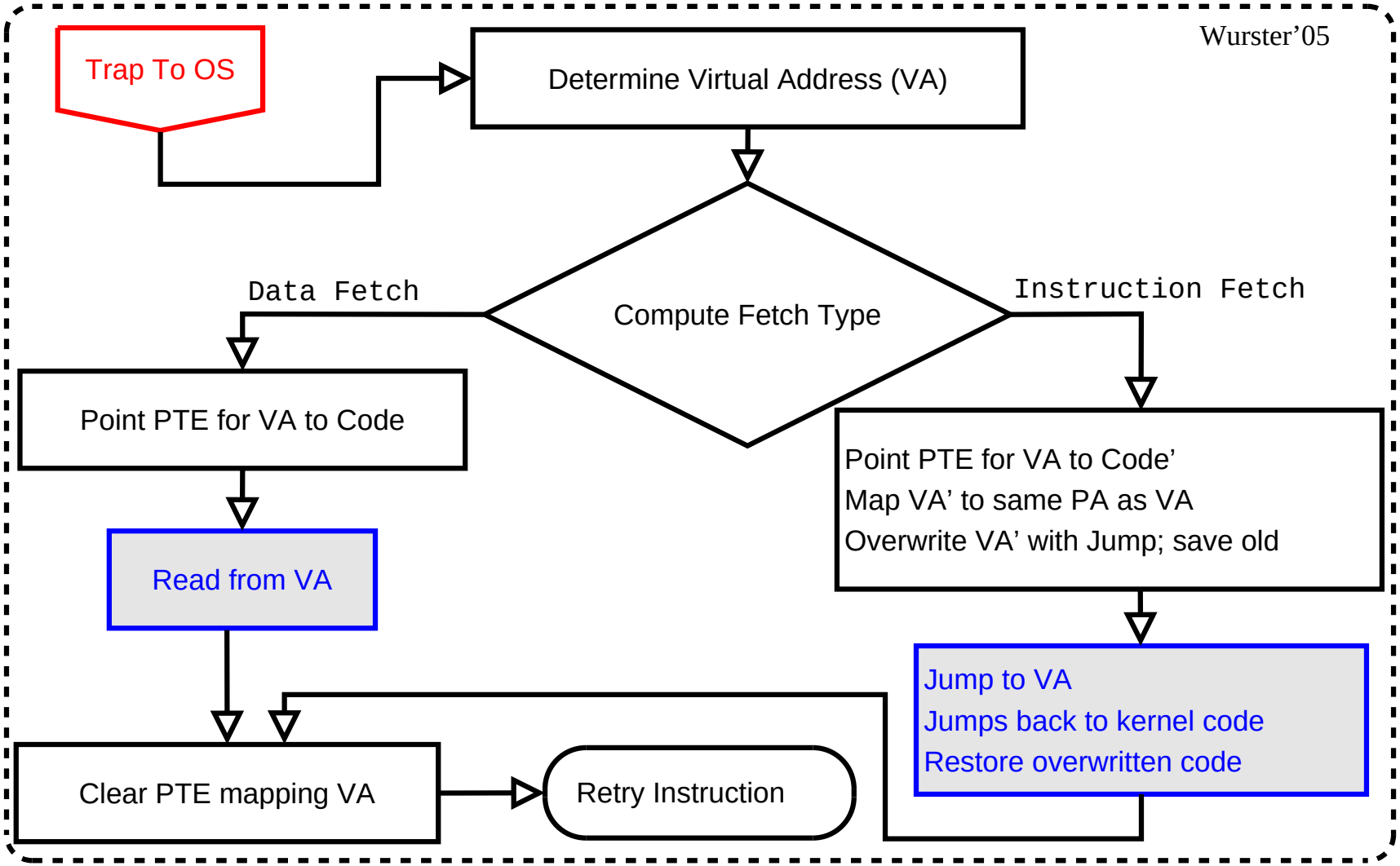
Alternative Attack Implementations

- We use any of several methods to separate code and data reads
  - Software TLB miss handlers
  - Hardware page table miss handlers
  - Hardware segmentation translation

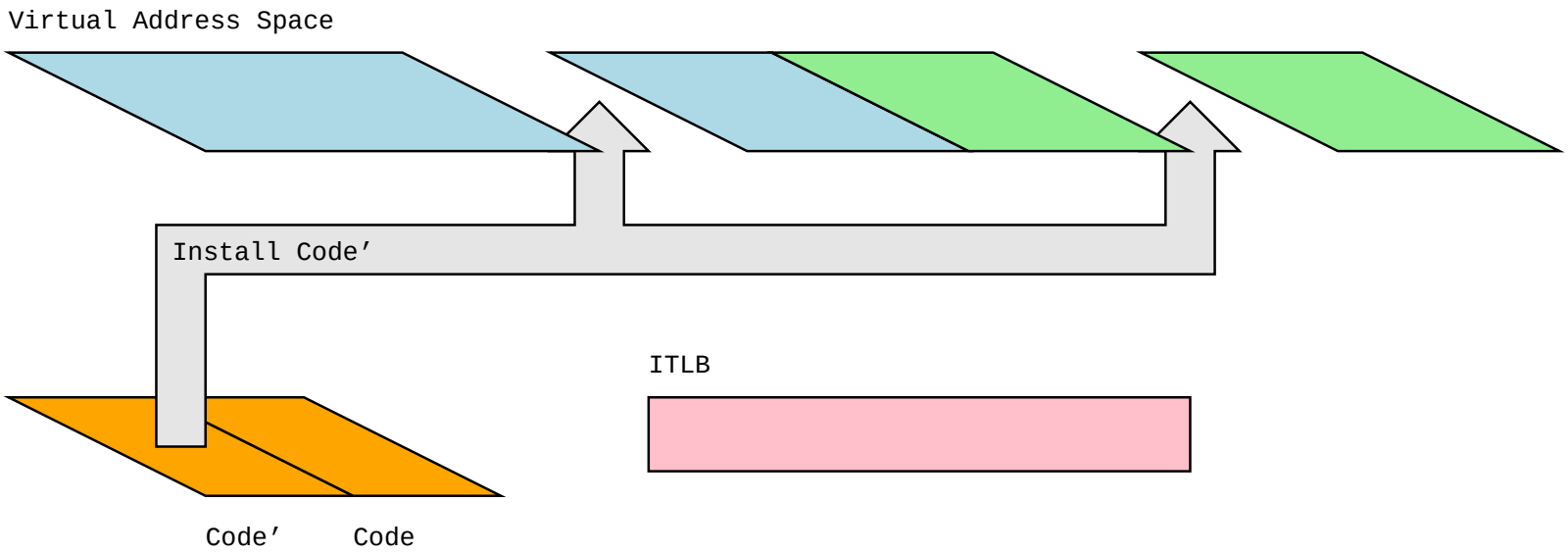
Hardware Translation Mechanism



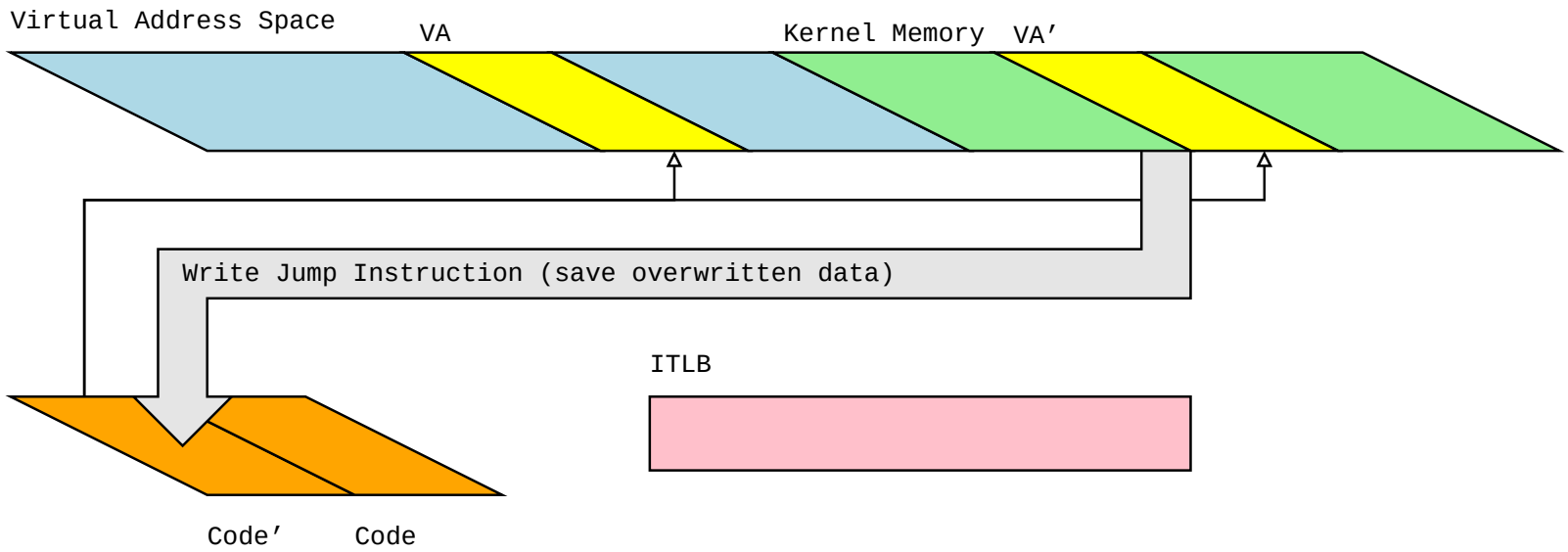
Attack Implementation



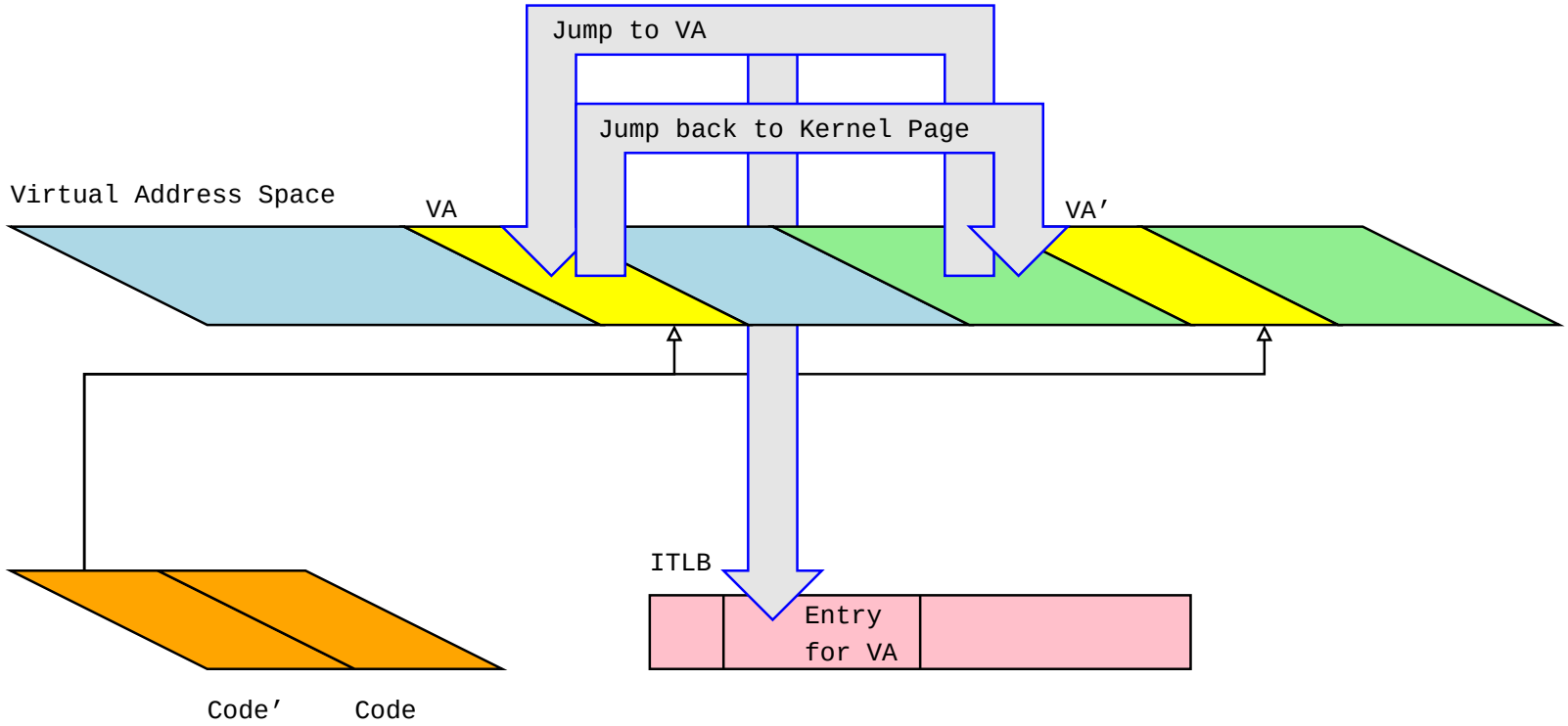
Filling the ITLB in Generic implementation



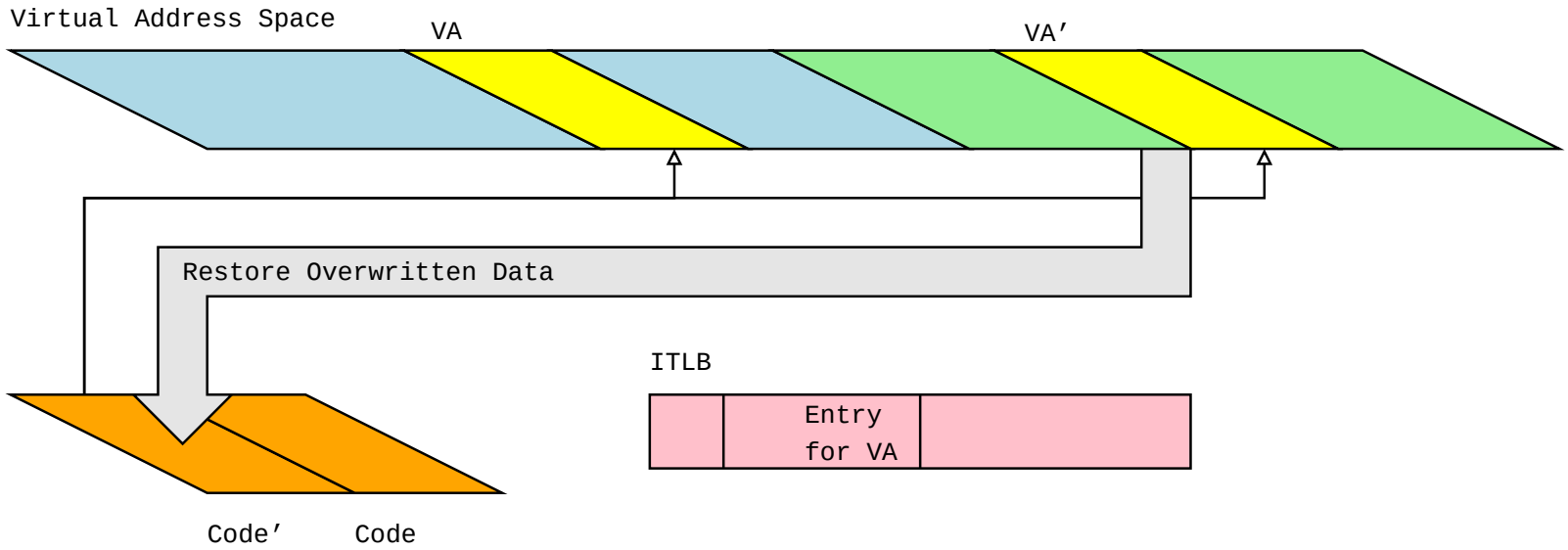
Filling the ITLB in Generic implementation



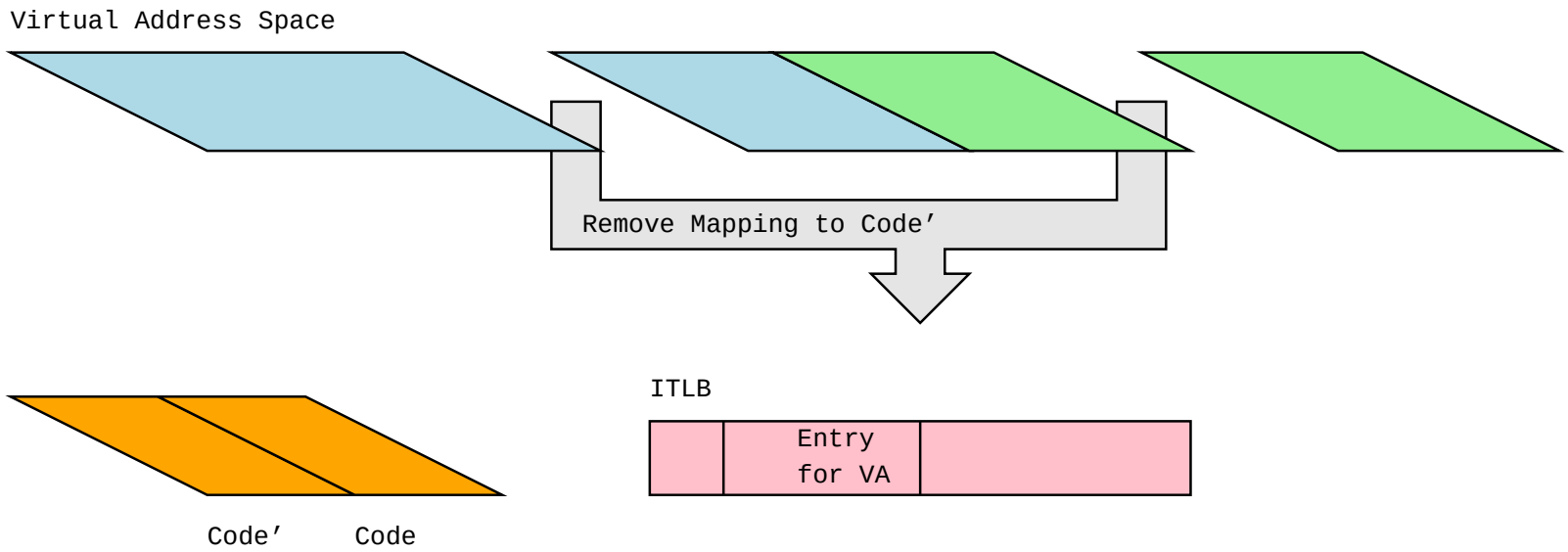
Filling the ITLB in Generic implementation



Filling the ITLB in Generic implementation



Filling the ITLB in Generic implementation



Result of attack

- Code read as data can differ from code executed
- By ensuring code read as data is unmodified code, self-hashing always uses unmodified code – yielding the “correct” hash
- Attack applies to most modern general-purpose processors e.g. UltraSparc, x86, PowerPC, AMD64, Alpha, ARM

Note: The attack is not prevented by stealthy address computations[a]

[a] Linn et al. *Enhancing Software Tamper-Resistance...* ACSAC-2003

Overhead

- Implementation Work
  - Must install a modified kernel
  - Per-application overhead is negligible (`copy` command)
- Run-Time Overhead
  - Only on a TLB cache miss (0.1% of time on UltraSparc)
  - Each DTLB miss adds 6 assembly instructions on UltraSparc
  - Overhead is less than existing time spent on cache misses

Variations of the Attack

Variation	Oakland'05	TDSC'05
TLB Load (Ultrasparc)	✓	✓
Generic Attack		✓
Segment (x86)	✓	✓
Microcode		✓
Performance Counters		✓

Realities of the Attack

- Implemented on the UltraSparc, experimented on x86
- Hash functions need not be found or modified
- Exploits translation and caching capabilities of processor
  - Negligible performance hit
- Attack is possible on wide range of processors

Conclusions

- Typical self-hashing can be subverted on modern processors
- Need new protection to secure self-hashing tamper resistance
  - must withstand real-time detection and separation of code/data

Questions?

http://www.scs.carleton.ca/~gwurster