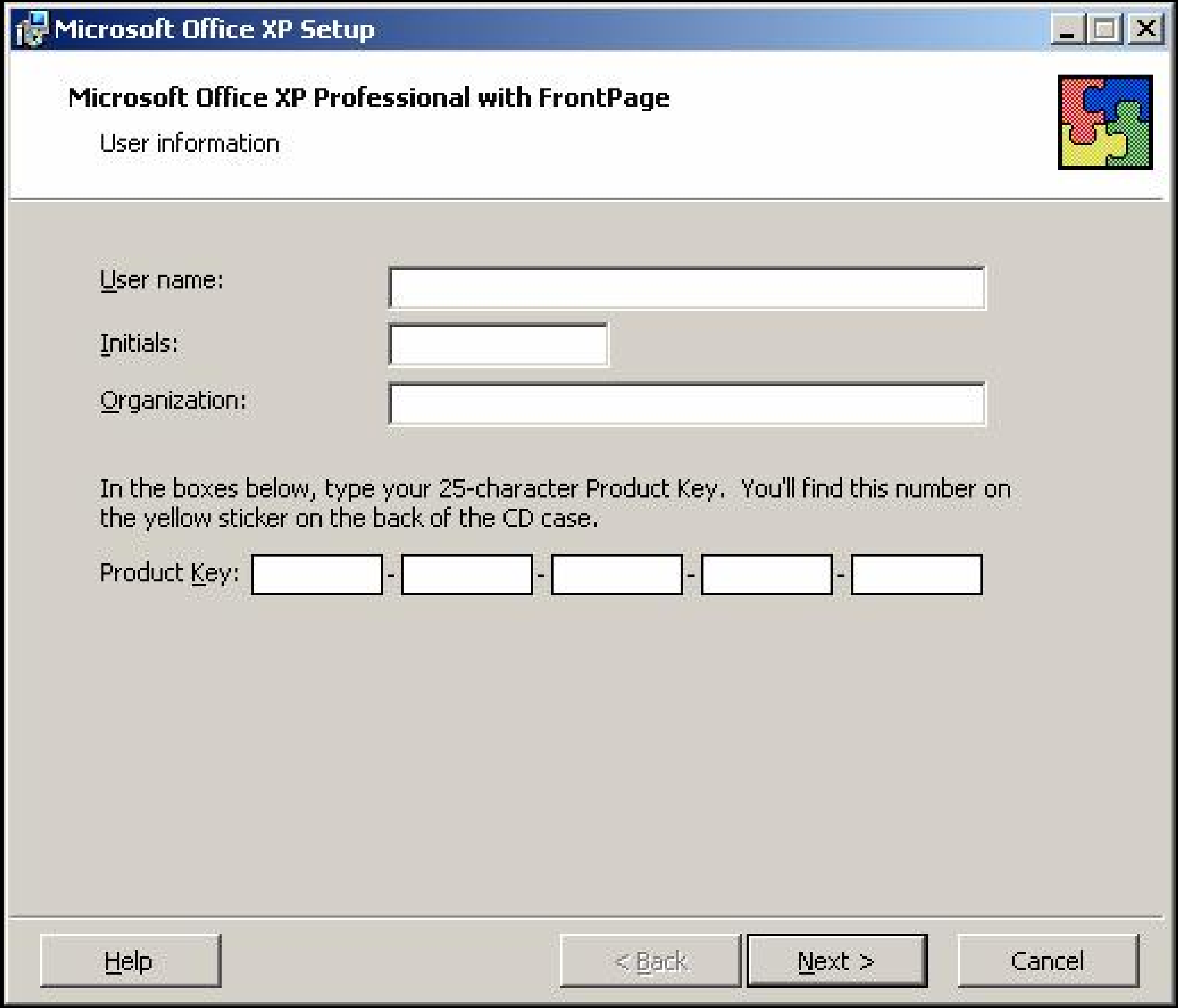
# Strengthening Self-Checksumming via Self-Modifying Code

*Jonathon T. Giffin, Mihai Christodorescu, Louis Kruger*

Computer Sciences Department
University of Wisconsin

**`{giffin,mihai,lpkruger}@cs.wisc.edu`**



Microsoft Office XP Professional with FrontPage

User information

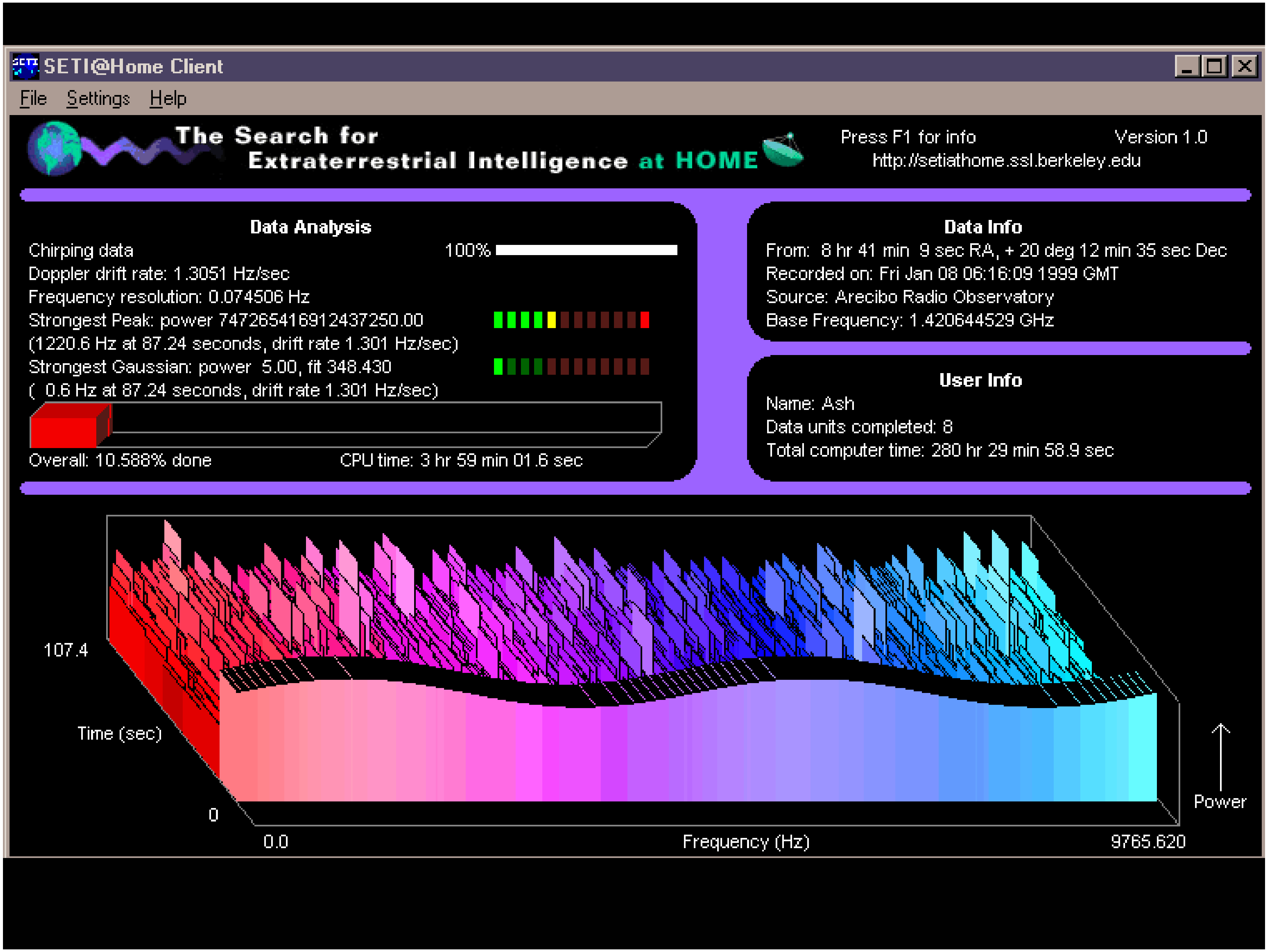
User name:

Initials:

Organization:

In the boxes below, type your 25-character Product Key. You'll find this number on the yellow sticker on the back of the CD case.

Product Key: _____ - _____ - _____ - _____ - _____



Press F1 for info Version 1.0
http://setiathome.ssl.berkeley.edu

**Data Analysis**

Chirping data 100%
Doppler drift rate: 1.3051 Hz/sec
Frequency resolution: 0.074506 Hz
Strongest Peak: power 747265416912437250.00
(1220.6 Hz at 87.24 seconds, drift rate 1.301 Hz/sec)
Strongest Gaussian: power 5.00, fit 348.430
( 0.6 Hz at 87.24 seconds, drift rate 1.301 Hz/sec)
Overall: 10.588% done CPU time: 3 hr 59 min 01.6 sec

**Data Info**

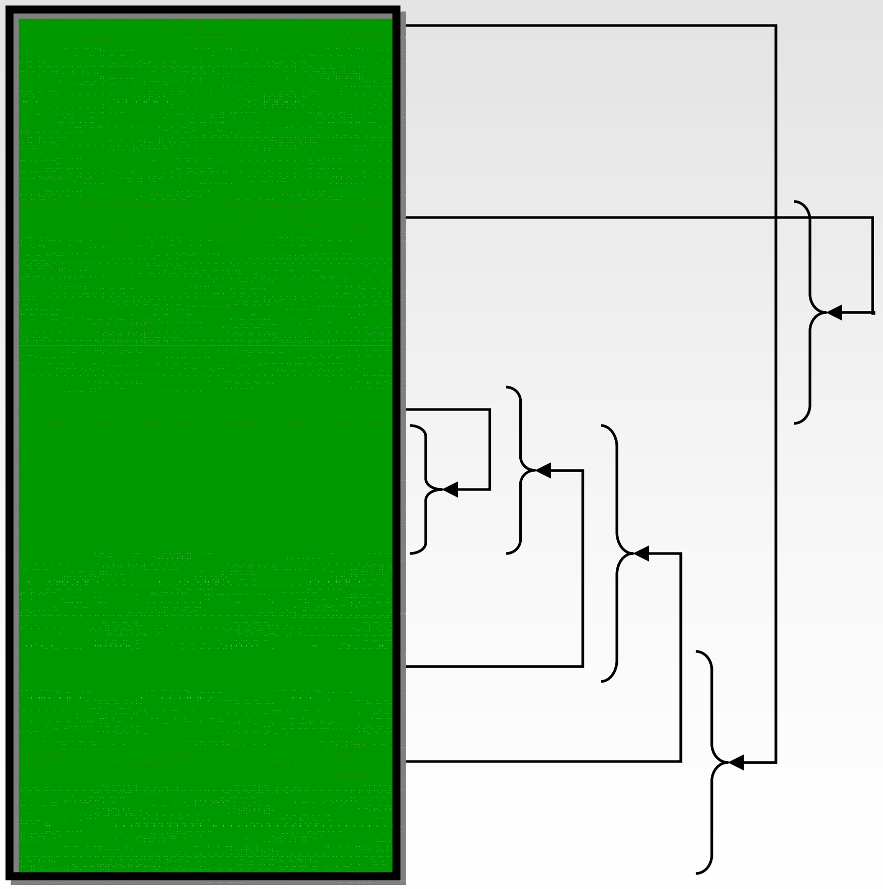
From: 8 hr 41 min 9 sec RA, + 20 deg 12 min 35 sec Dec
Recorded on: Fri Jan 08 06:16:09 1999 GMT
Source: Arecibo Radio Observatory
Base Frequency: 1.420644529 GHz

**User Info**

Name: Ash
Data units completed: 8
Total computer time: 280 hr 29 min 58.9 sec

# Self-Checksumming

- Program contains code to checksum parts of its own code.



## Self-Checksumming

- Integrity Verification Kernels [Aucsmith 1996]
  - Multithreaded, self-checking, checksumming components
- Testers and correctors [Horne *et al.* 2001]
- Network of guards [Chang & Atallah 2001]
  - Many overlapping checksumming components

## Assumptions

The attacker cannot identify all relevant checksum code within the protected program.

The attacker runs the protected program at full speed or with only a reasonable slowdown.

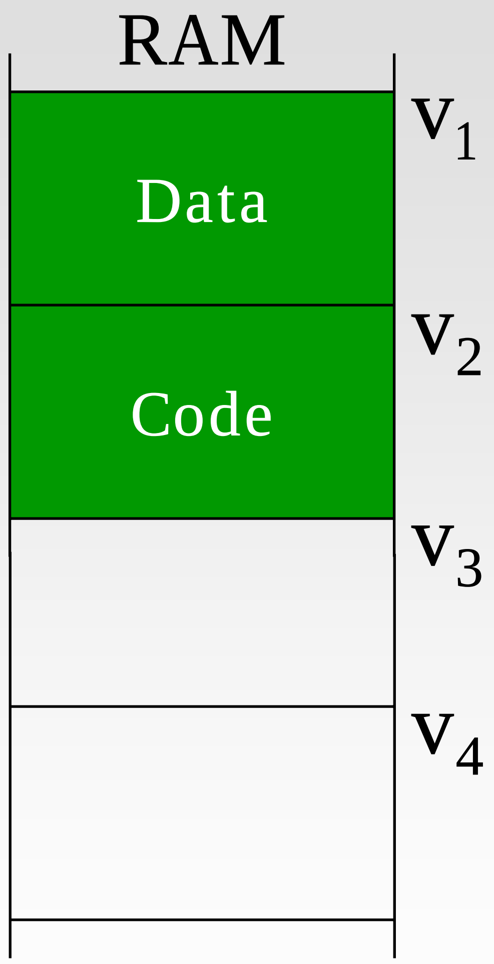
Self-checksumming programs execute on a commodity von Neumann machine.

## Memory Architectures

von Neumann architecture

- Unified code & data memory
- Memory addresses unique

[von Neumann 1945]

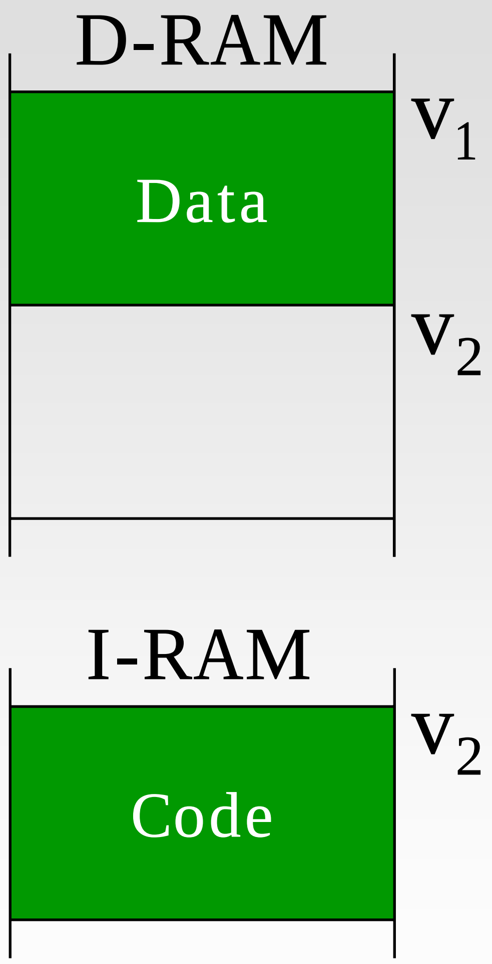


# Memory Architectures

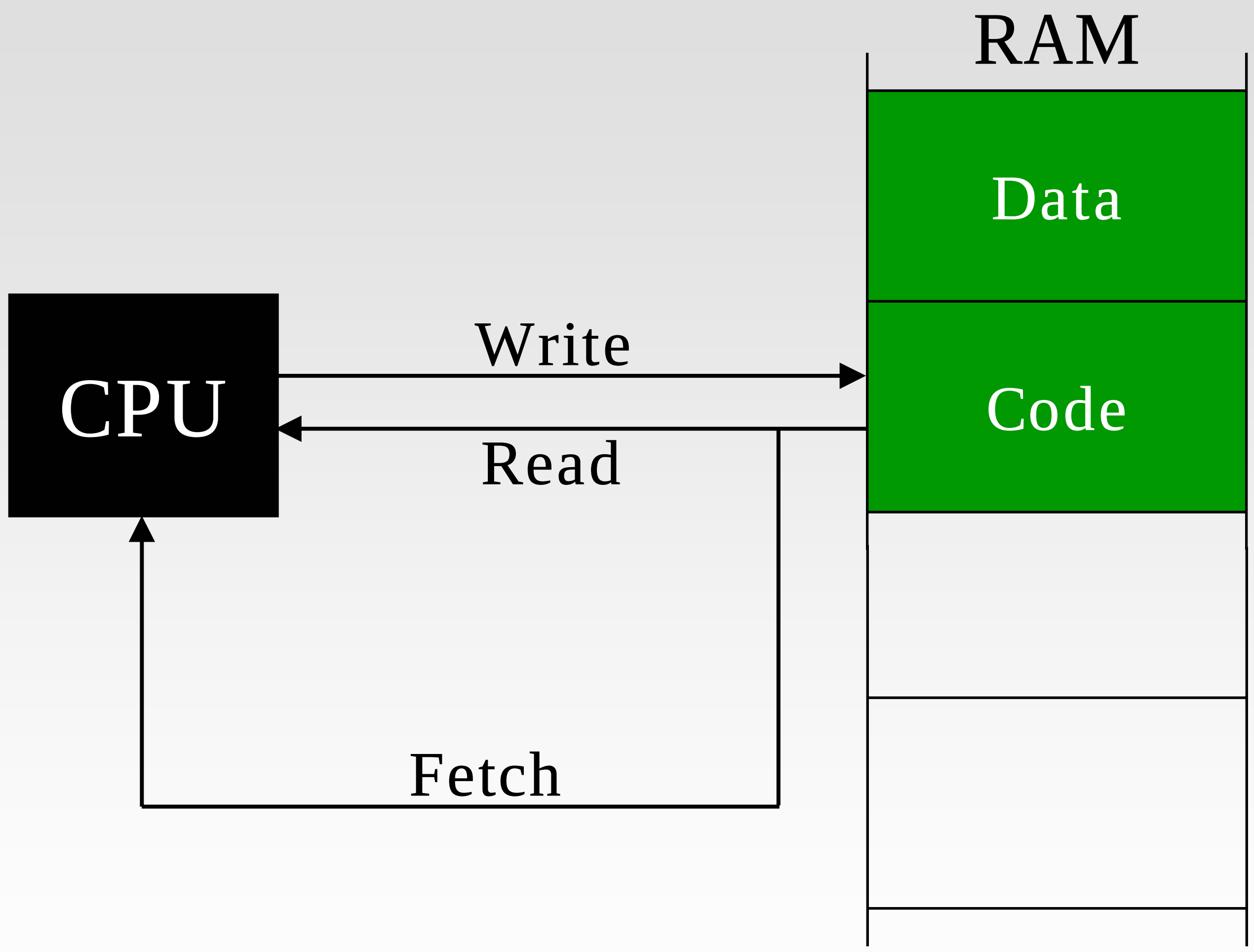
Harvard architecture

- Separate code & data memory
- Memory addresses duplicated

[Aiken & Hopper 1946]

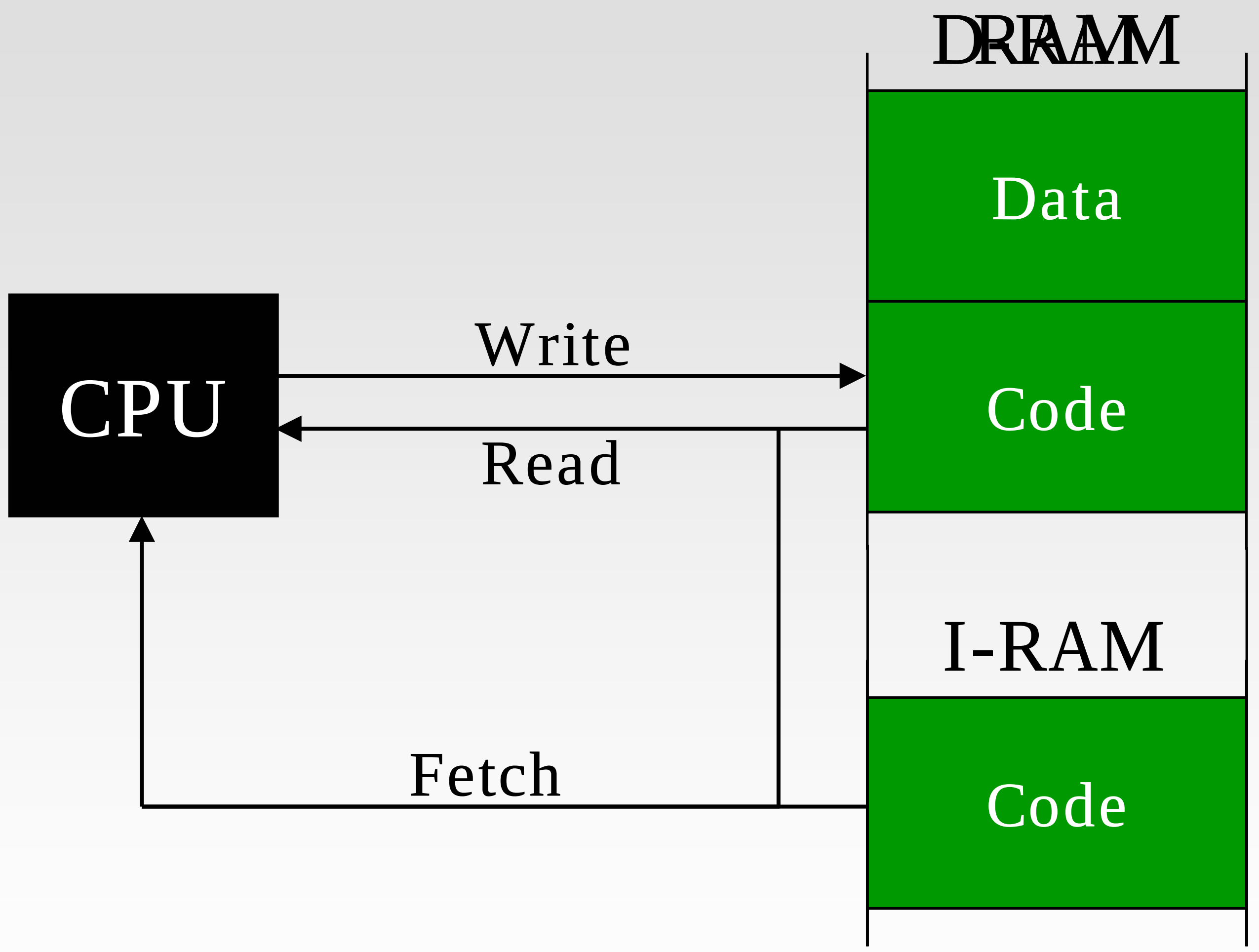


## Page-Replication Attack



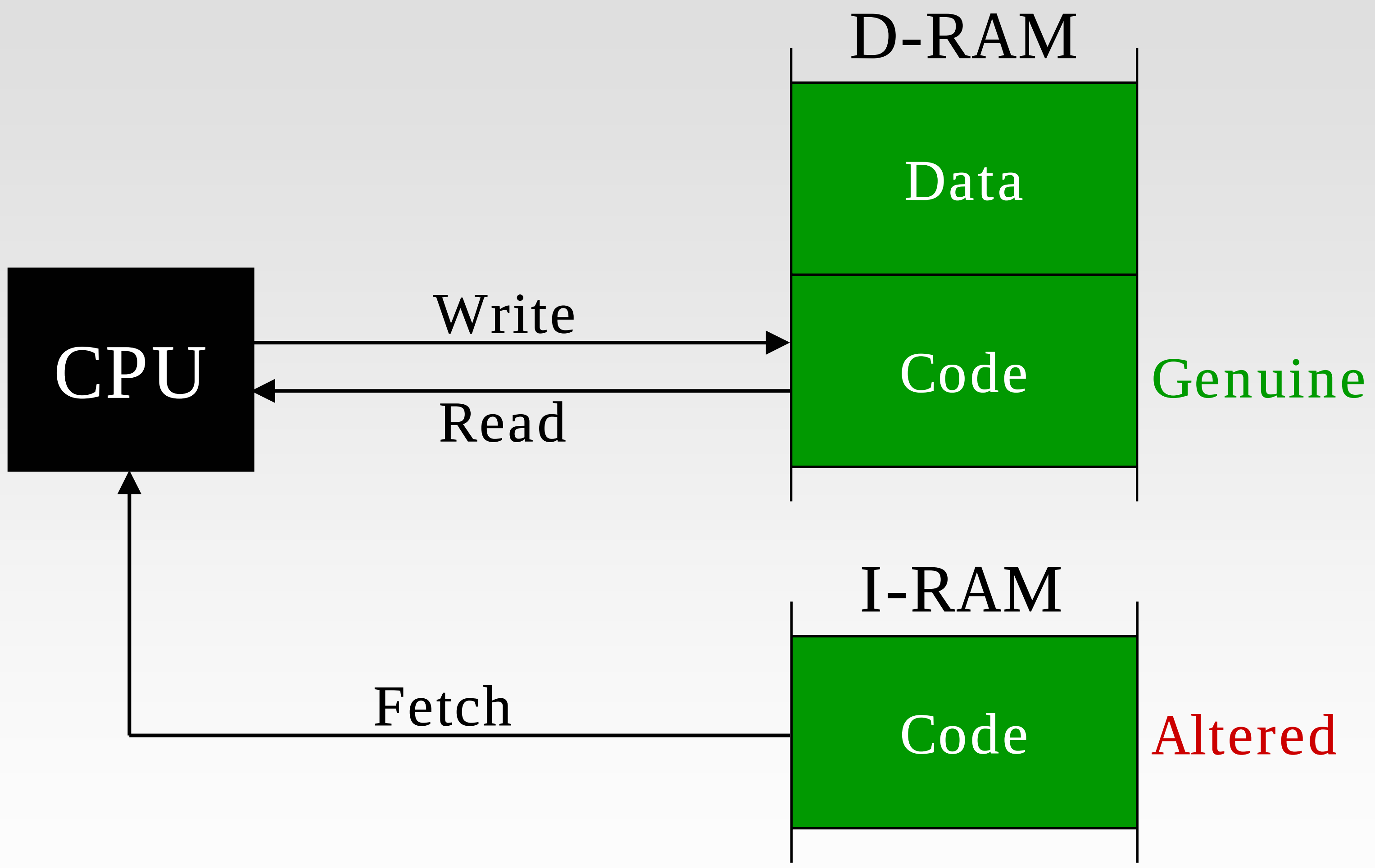
[Wurster *et al.* 2005]

# Page-Replication Attack



[Wurster *et al.* 2005]

# Page-Replication Attack



[Wurster *et al.* 2005]

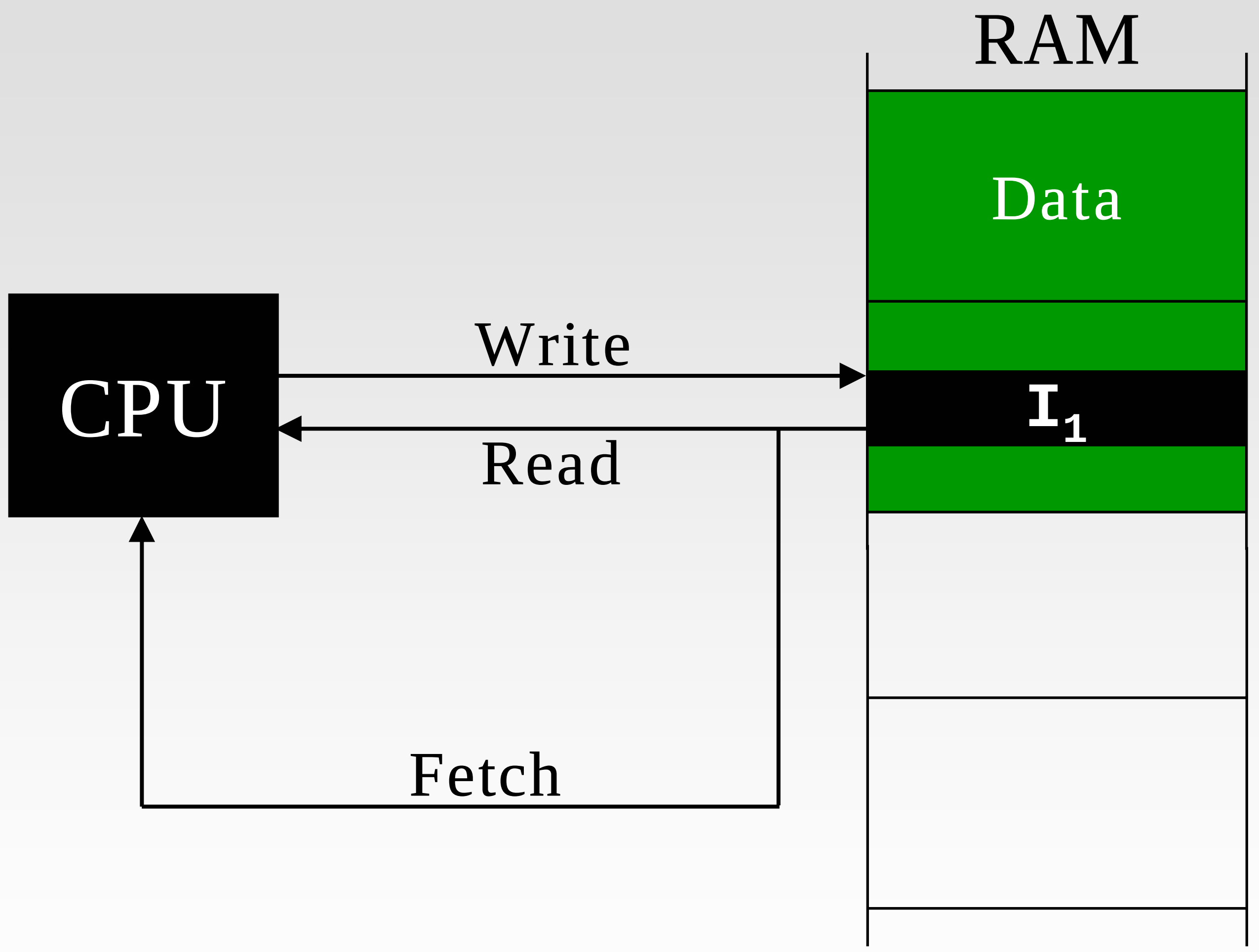
## Attack Detection

Observation:

Writes to code affect program differently depending upon memory architecture

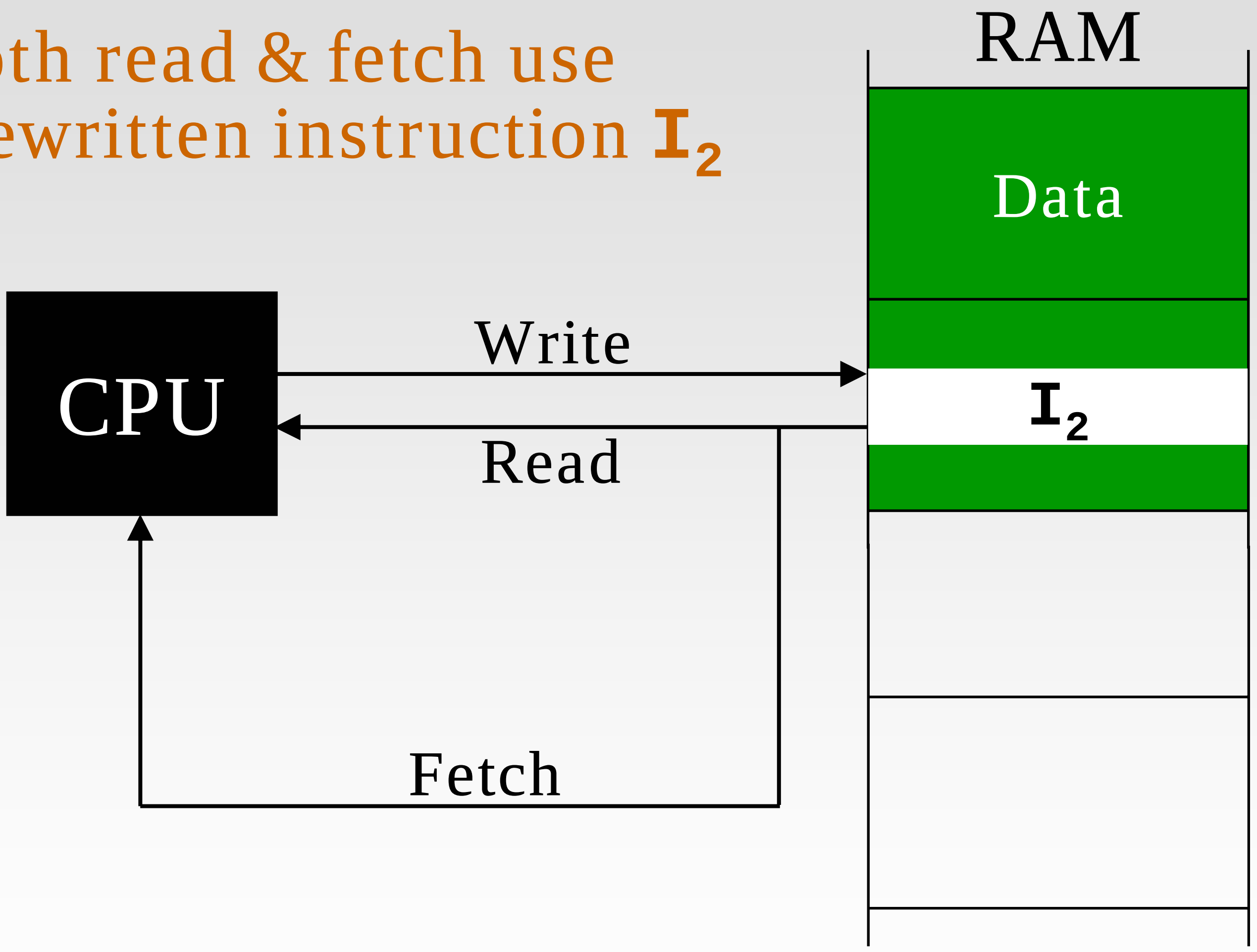
Use self-modifying code to detect page-replication attack

# Self-Modifying Code

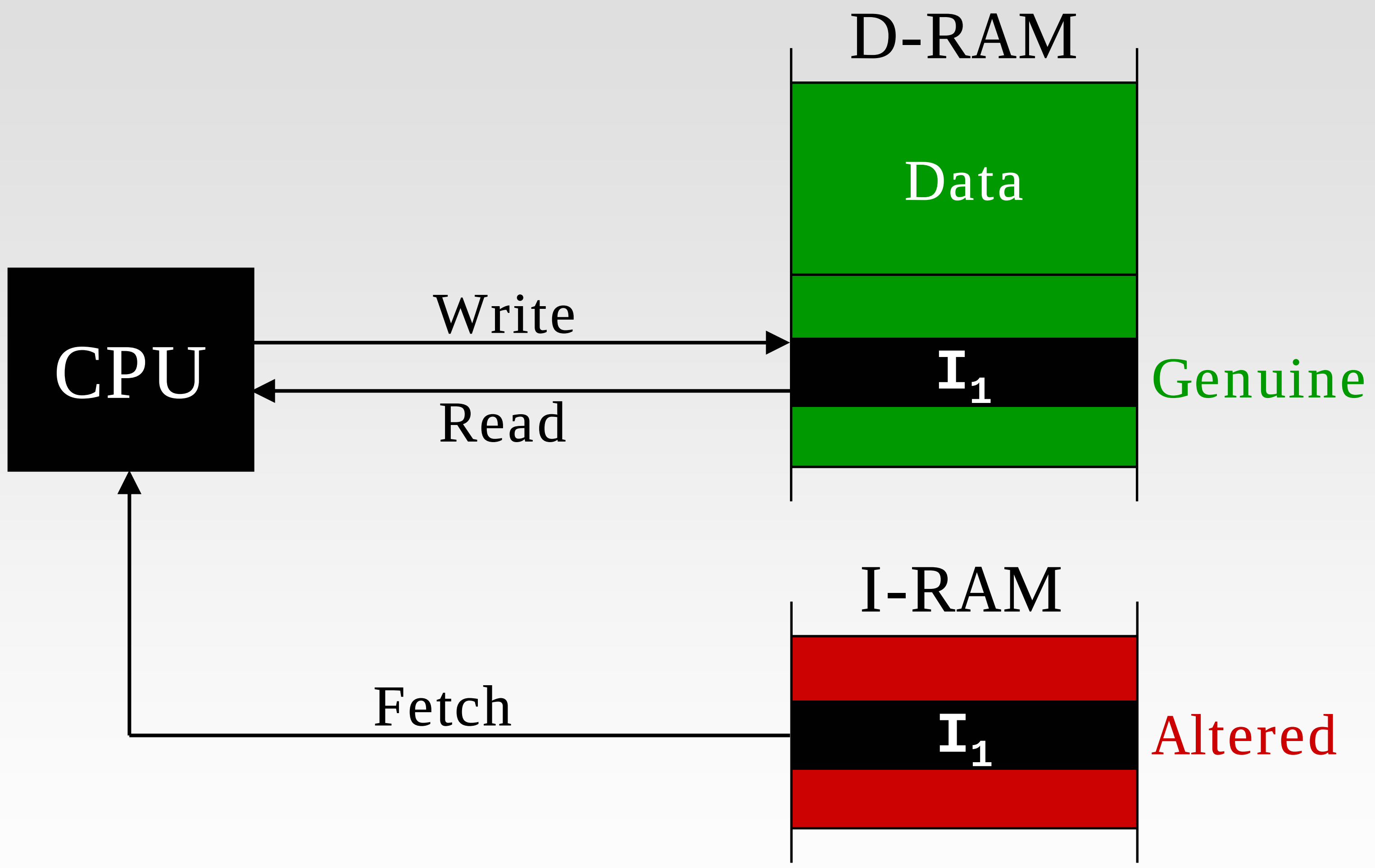


# Self-Modifying Code

Both read & fetch use
the rewritten instruction $\mathbf{I_2}$

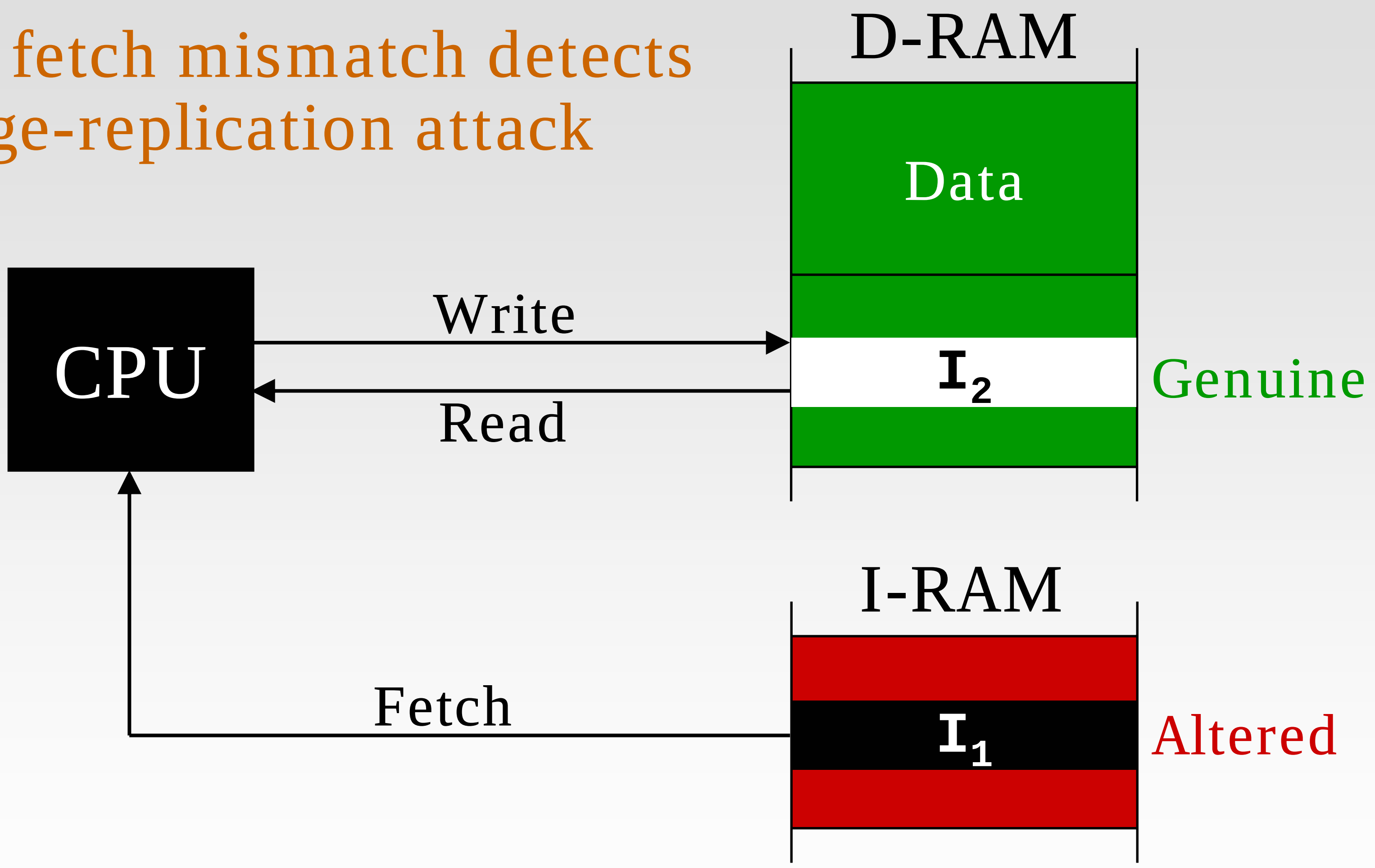


# Attack Detection



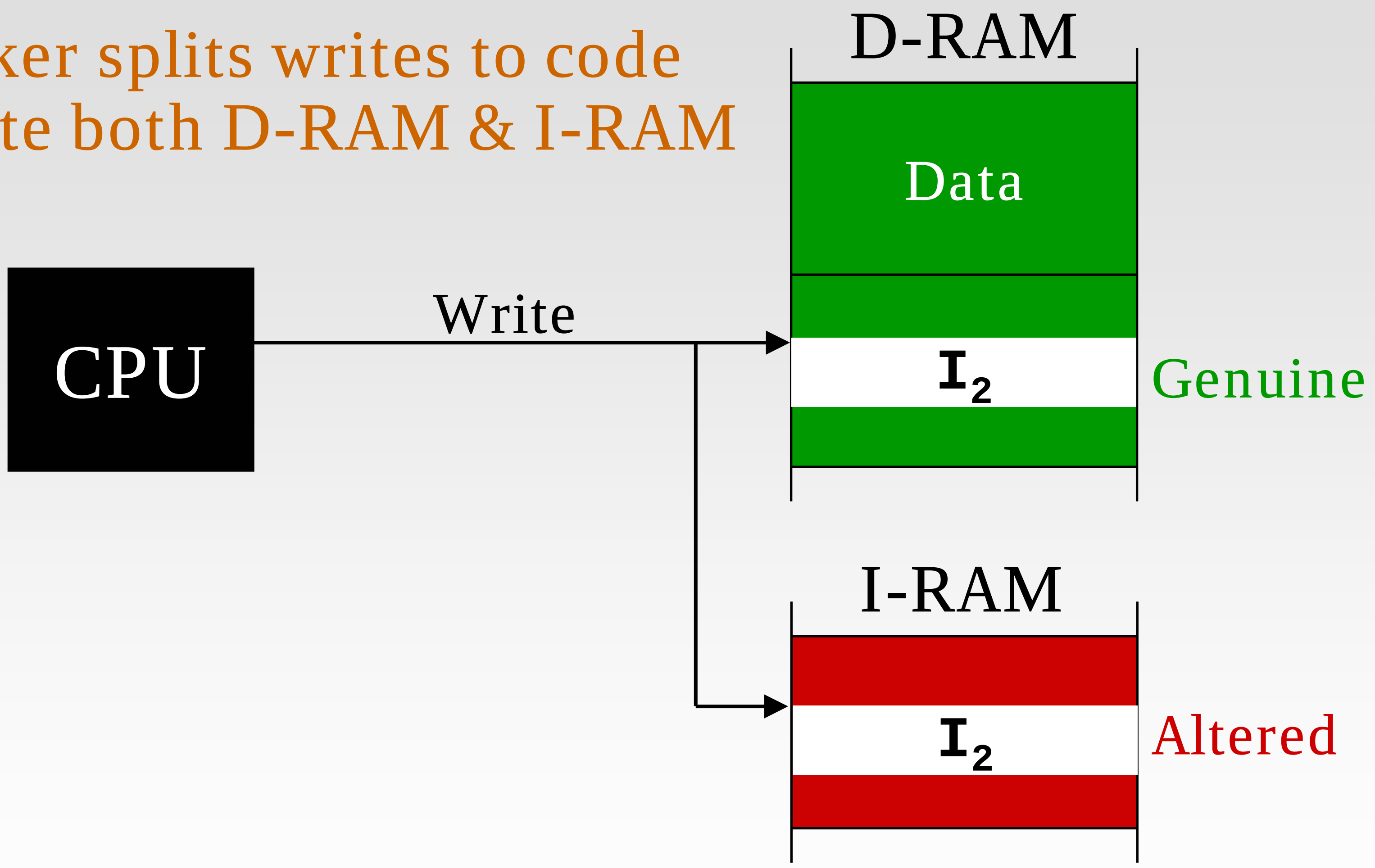
# Attack Detection

Read / fetch mismatch detects page-replication attack



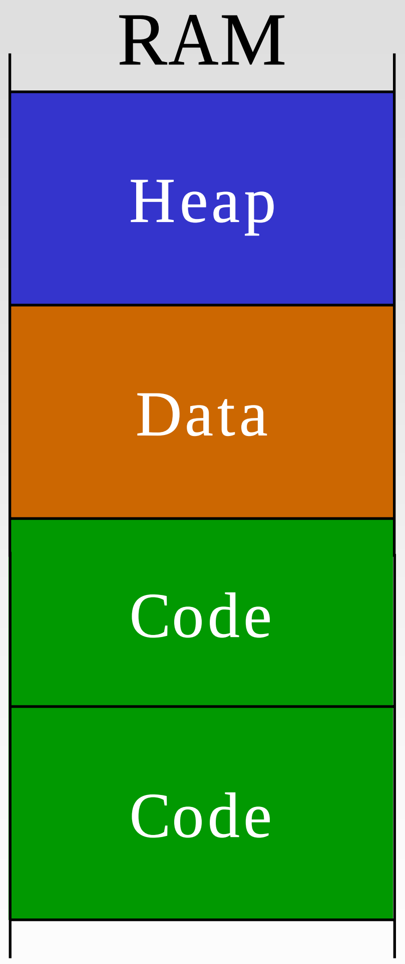
# Resistance to Attack

Attacker splits writes to code to update both D-RAM & I-RAM



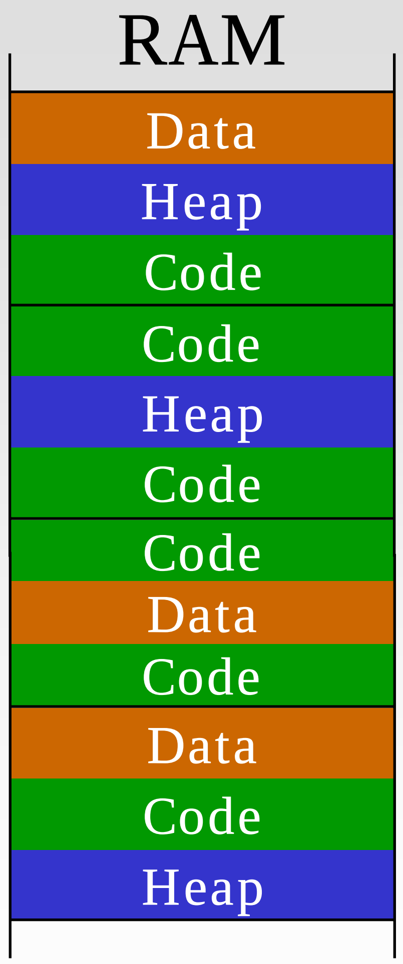
# Resistance to Attack

- Split writes requires attacker to emulate writes to code
  - Efficiently done via memory page protection bits



## Resistance to Attack

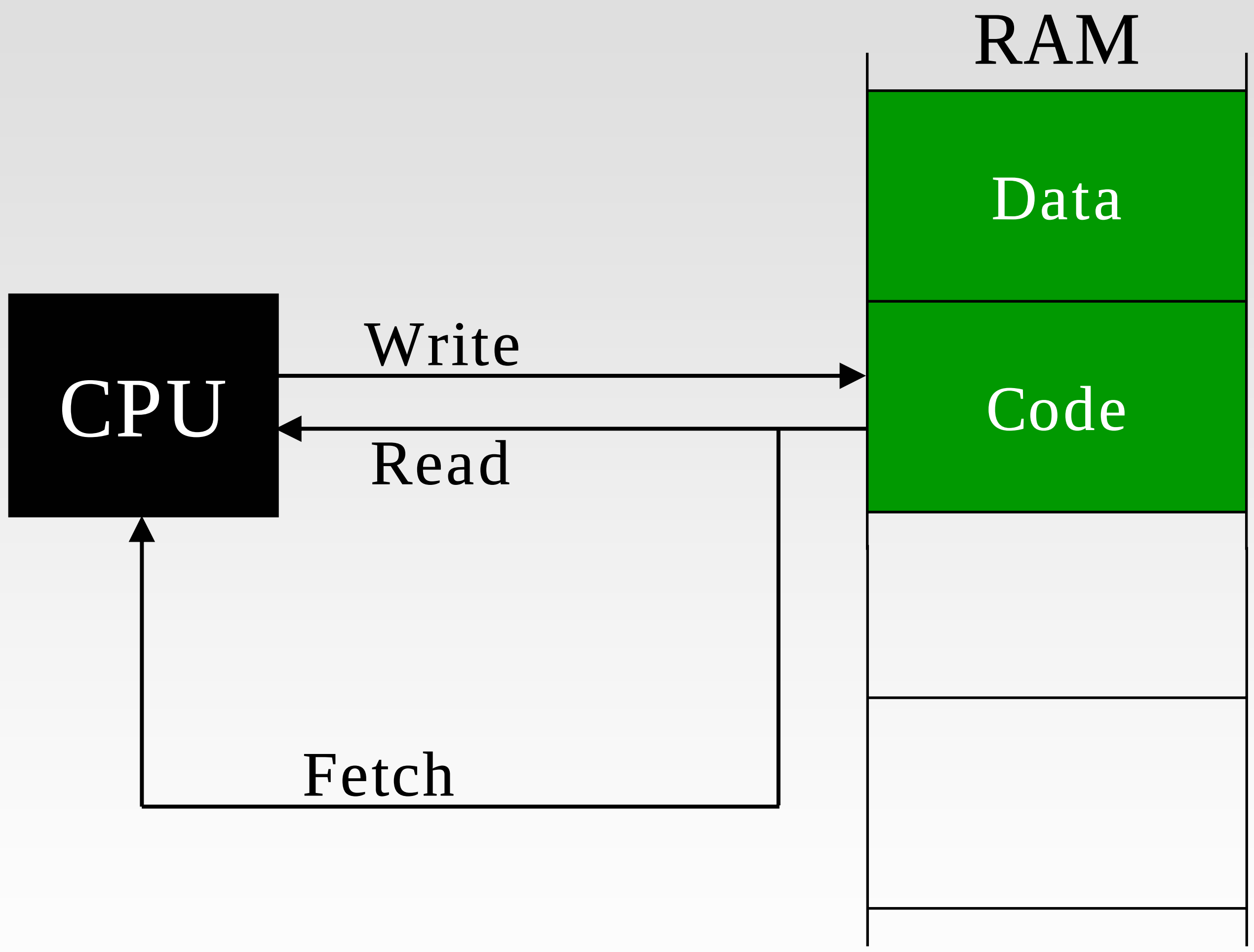
- Split writes requires attacker to emulate writes to code
  - Efficiently done via memory page protection bits
- Make code writes appear identical to data writes
  - Interleave code and data
  - Successful attack requires emulation of all writes



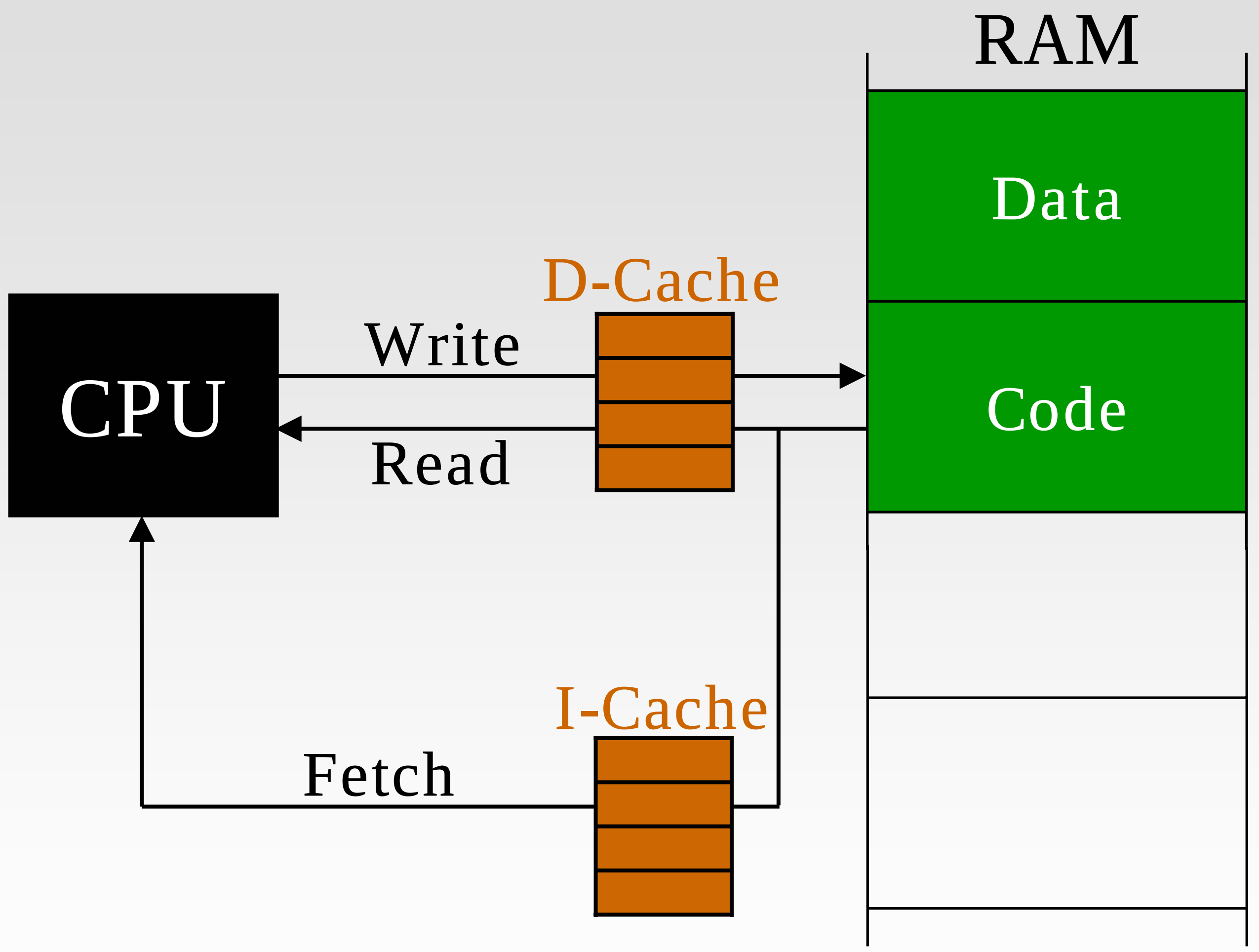
## Drawbacks

- Increases debugging complexity
  - Add self-modifying code in final development stage
- Requires writable code pages
  - Use alternative attack detection/prevention techniques
- Harvard caches must be kept consistent

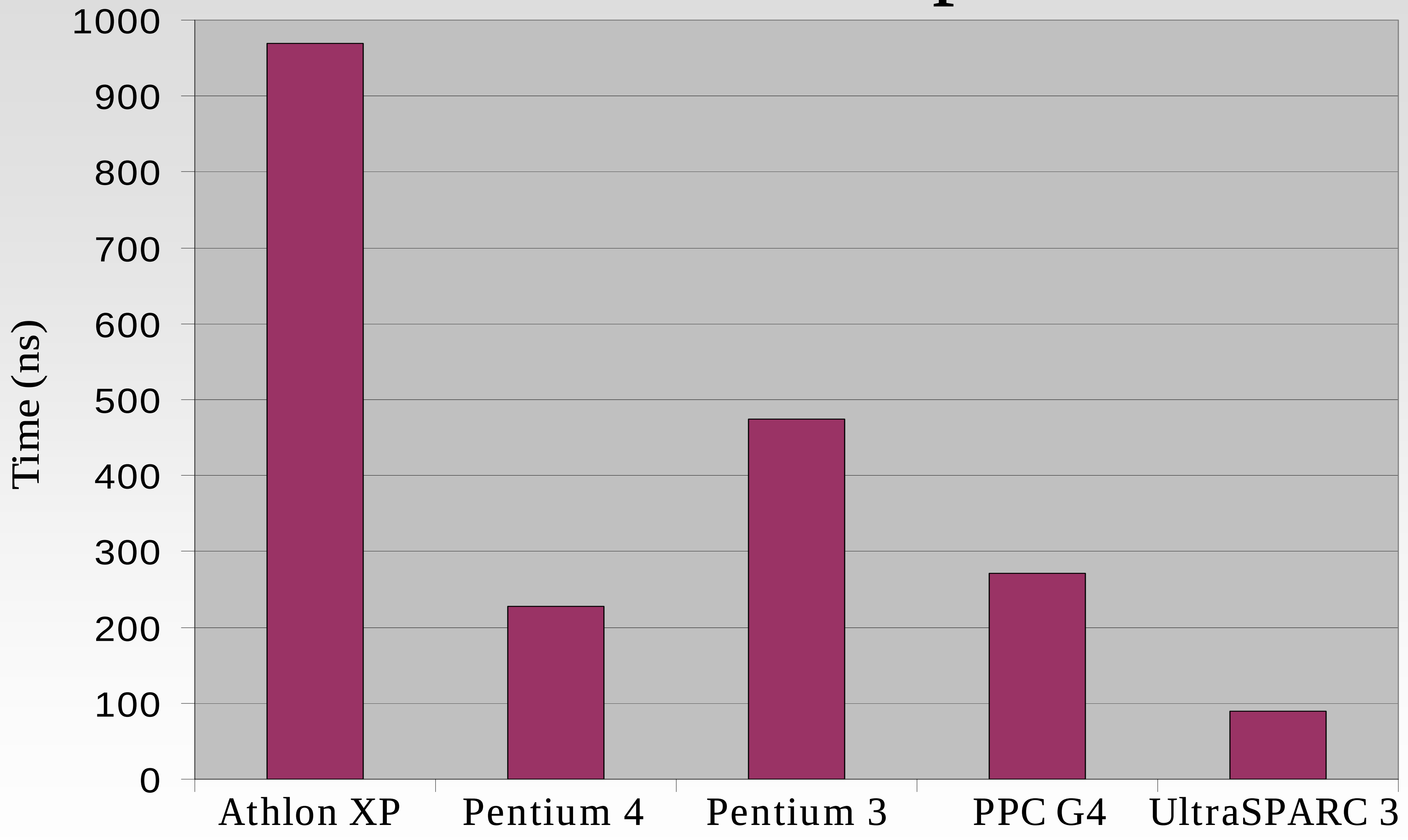
# Cache Coherency



## Cache Coherency



# Performance Impact



## Conclusions

Self-modifying code ...

... detects page-replication attacks ...

... efficiently ...

... in a way robust up to emulation attacks ...

... restoring the previous viability of self-checksumming.

## Questions?

Contact the authors:

| | |
|---|---|
| Jonathon T. Giffin | **giffin@cs.wisc.edu** |
| Mihai Christodorescu | **mihai@cs.wisc.edu** |
| Louis Kruger | **lpkruger@cs.wisc.edu** |

Computer Sciences Department
University of Wisconsin