Side channel example: The Cold-Boot Attack

Lest We Remember

Cold-Boot Attacks Against Disk Encryption

J. Alex Halderman, Seth D. Schoen, Nadia Heninger,
William Clarkson, William Paul, Joseph A. Calandrino,
Ariel J. Feldman, Jacob Appelbaum, Edward W. Felten

Data Theft Threat

OS Access Control

Attacker's Computer

How Safe is a Stolen Laptop?

Unlocked
Unencrypted
No protection

Locked
Unencrypted
Minimal protection

Locked
Encrypted
Industry best practice

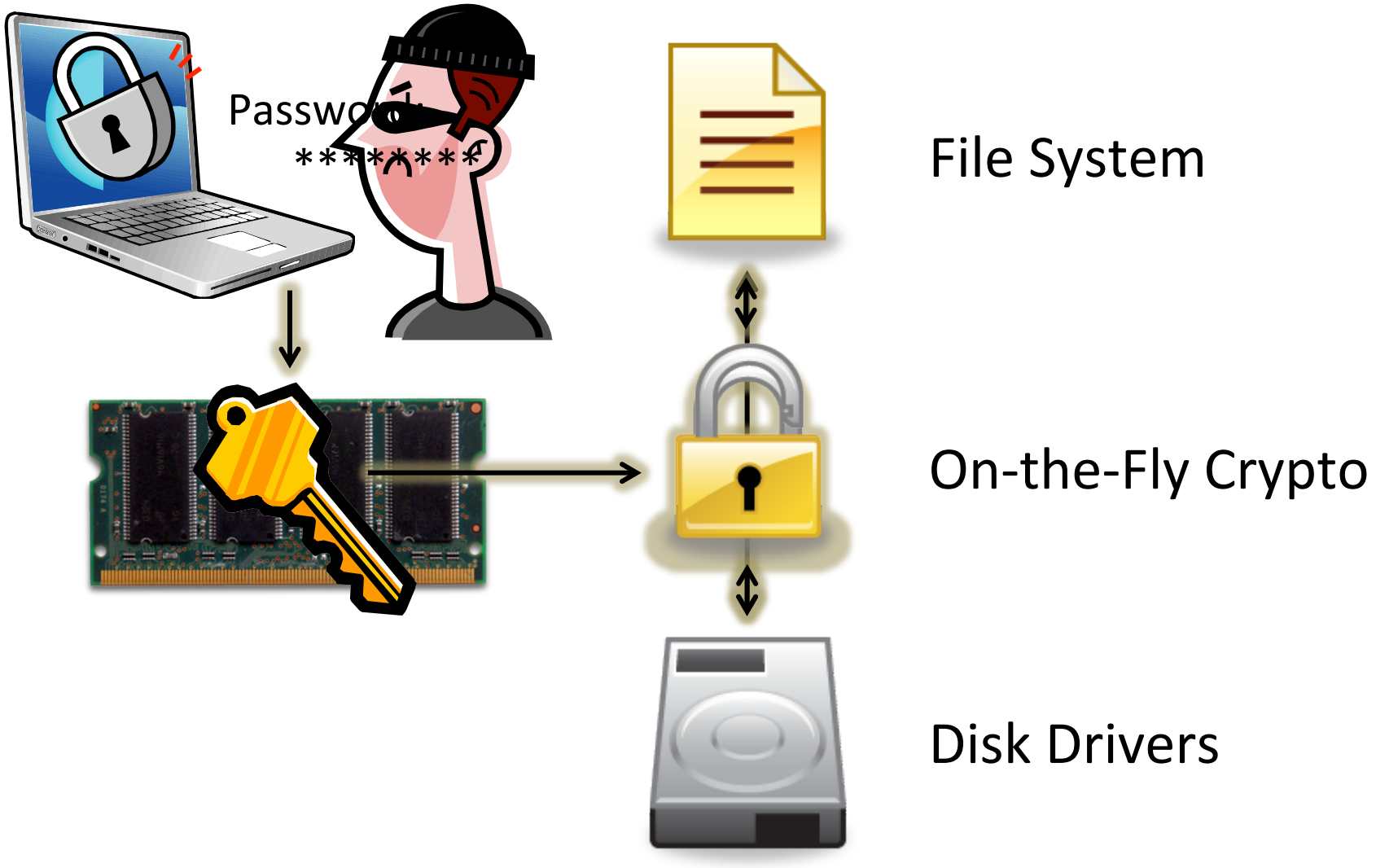
How Safe is a Stolen Laptop?

In 2007: **119,000** encrypted laptops with sensitive data lost at major US airports

Were they *really* protected?

Locked
Encrypted
Industry best practice

Disk Encryption Defense

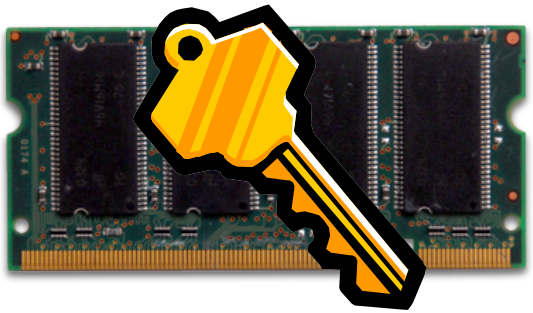


Common Attack Scenario

Security Assumptions

The encryption is strong

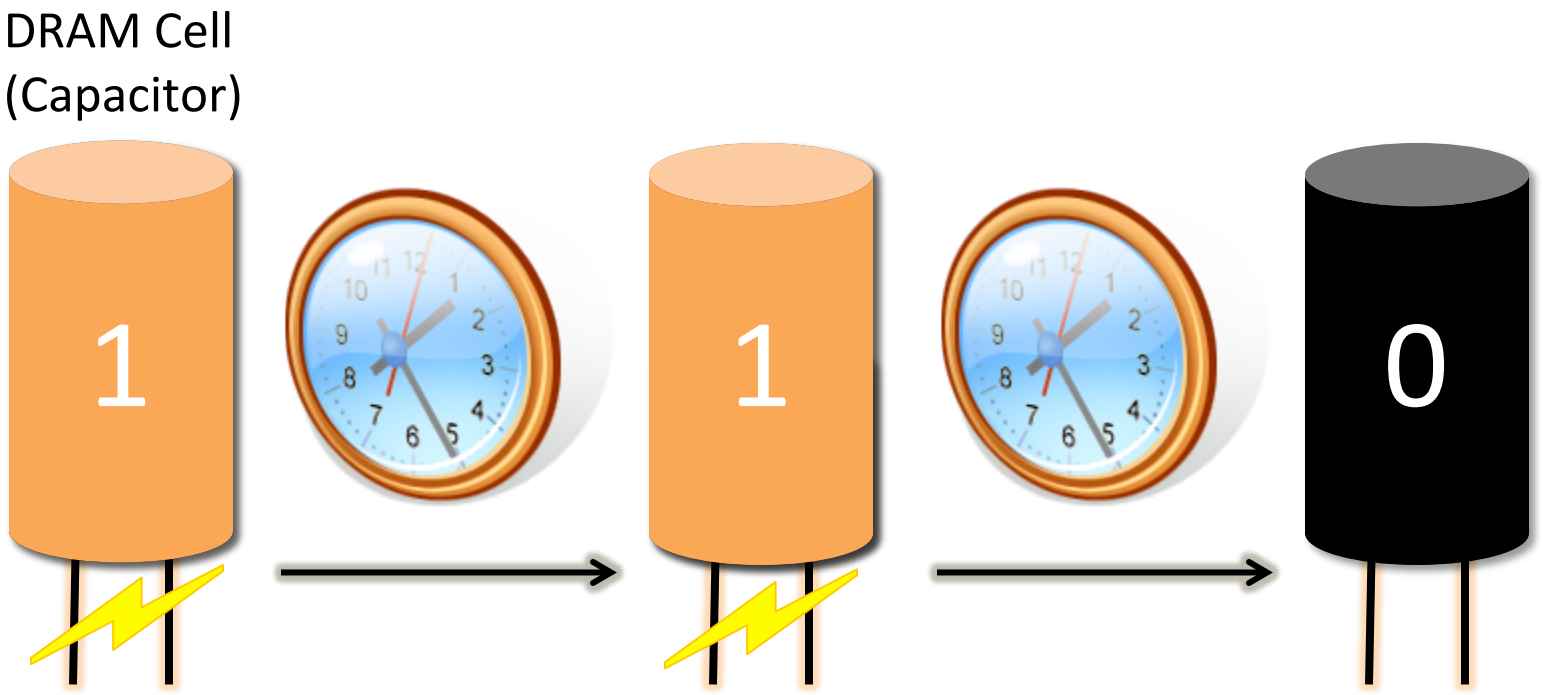
The OS protects the key in RAM



…the attacker might reboot to circumvent the OS, but since RAM is volatile, the key will be lost…

Right…?

Dynamic RAM Volatility

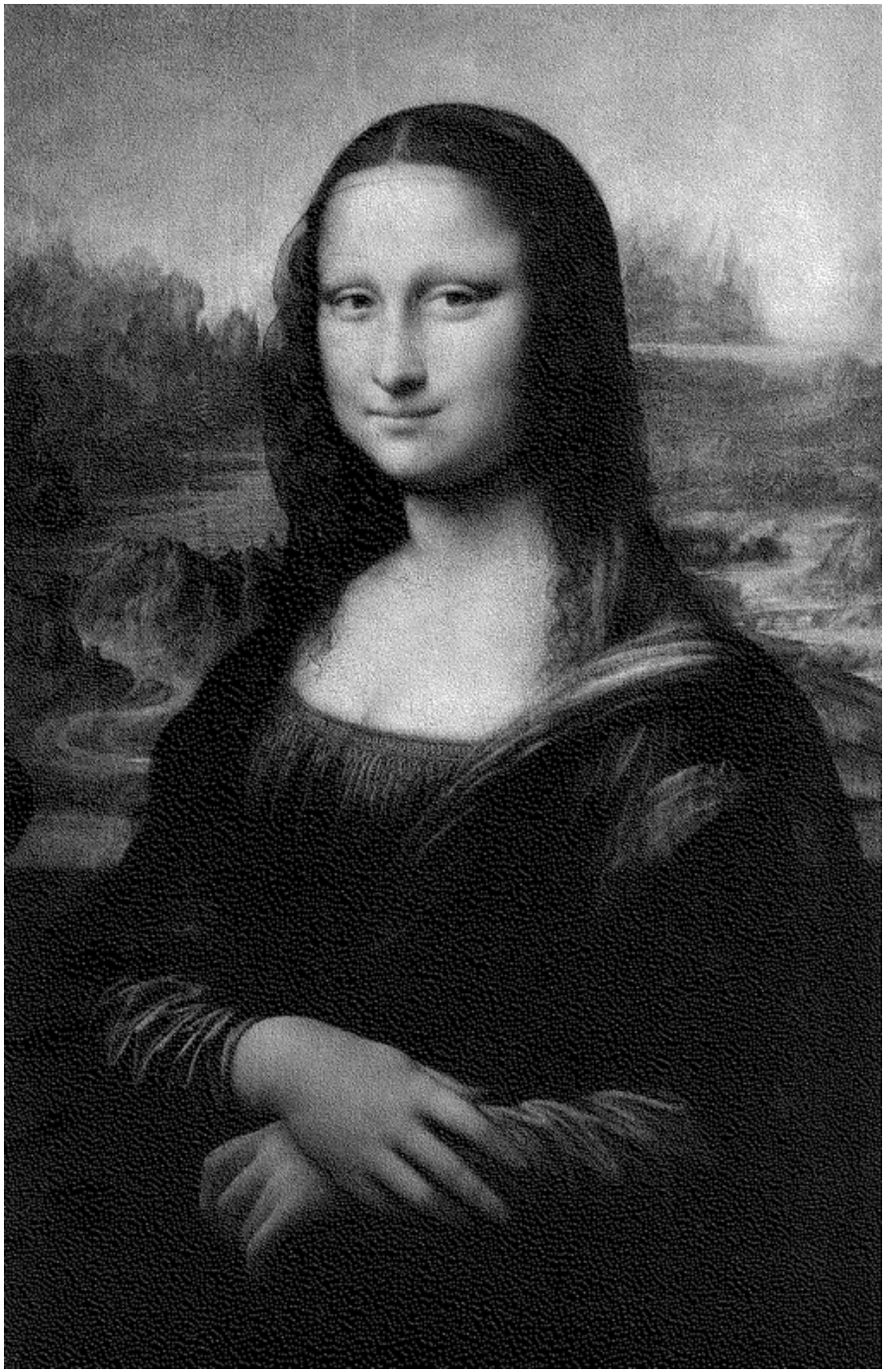


Security Assumptions:

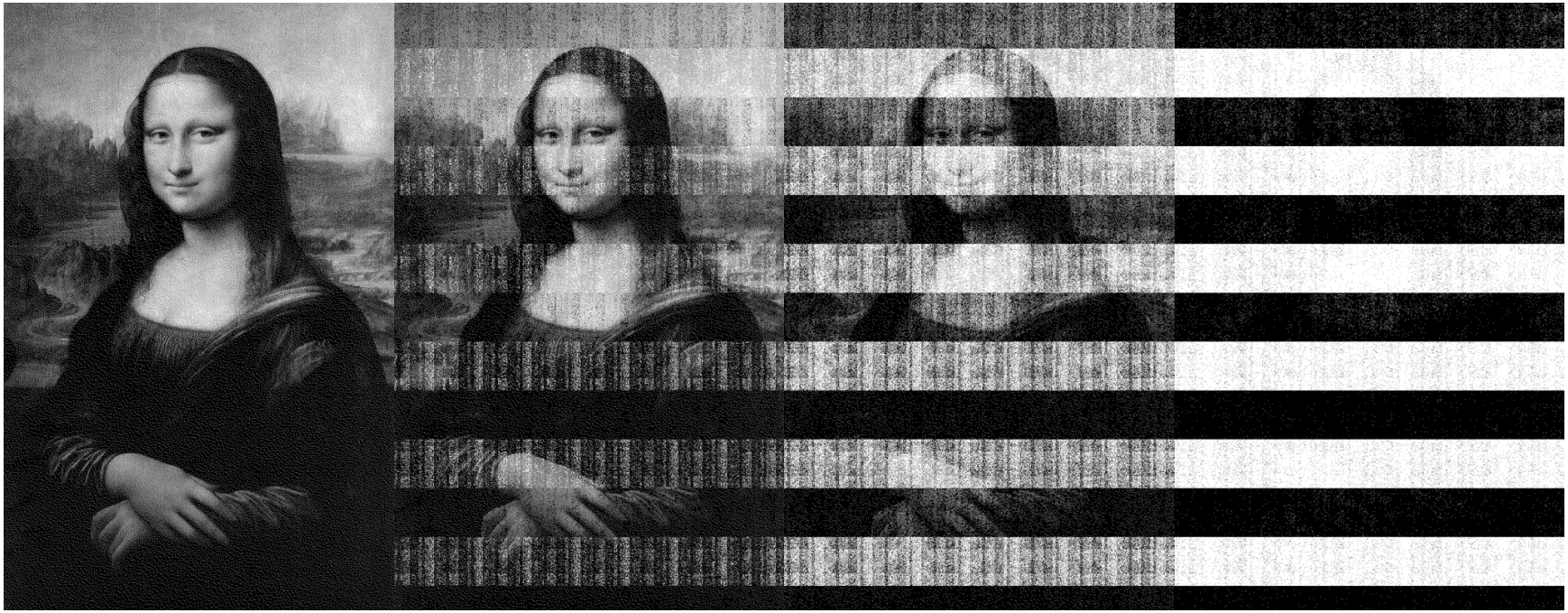
What if we don't refresh?

Data fades almost instantaneously without refresh

Any residual data is difficult to recover

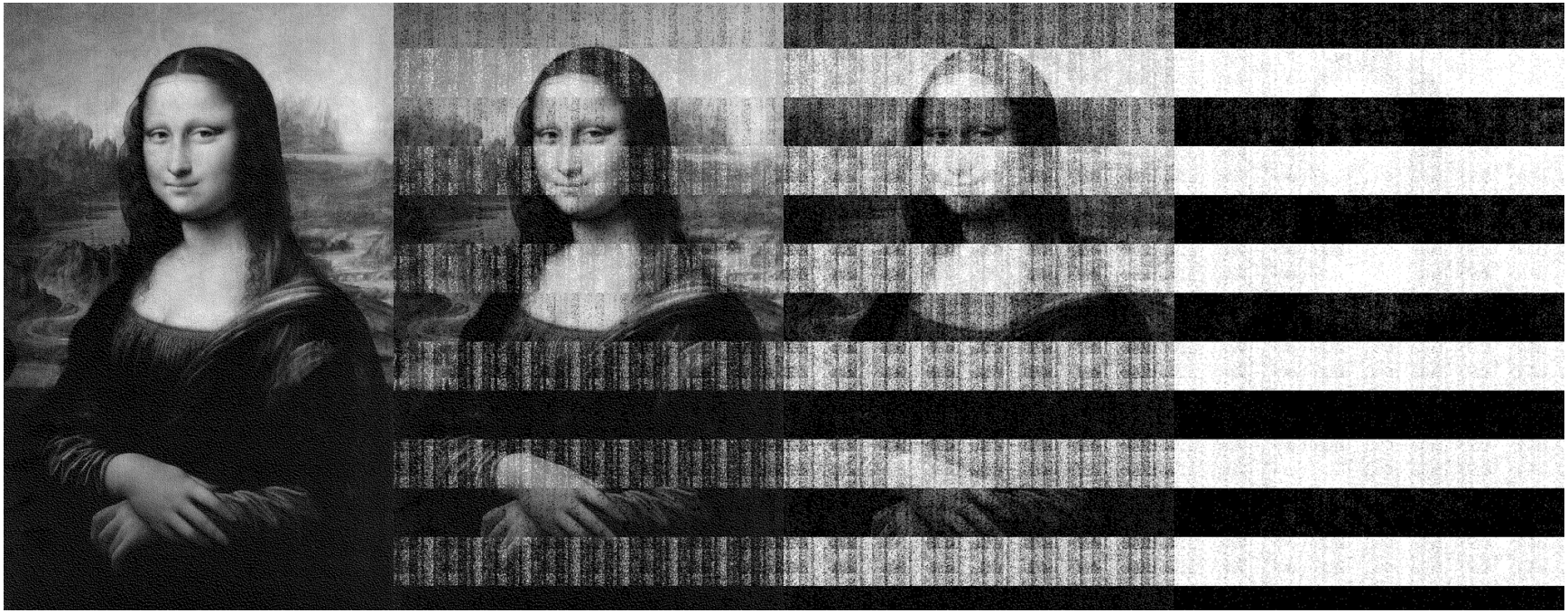


Decay After Cutting Power



5 secs 30 secs 60 secs 300 secs

Decay After Cutting Power



5 secs 30 secs 60 secs 300 secs

~~DRAM data disappears almost instantaneously without refresh~~

Gradual

Unidirectional

Predictable

Capturing Residual Data

~~Any residual data is difficult to recover~~

Residual data can be captured easily, with no special equipment

Complication

Booting full OS overwrites large areas of RAM

Solution

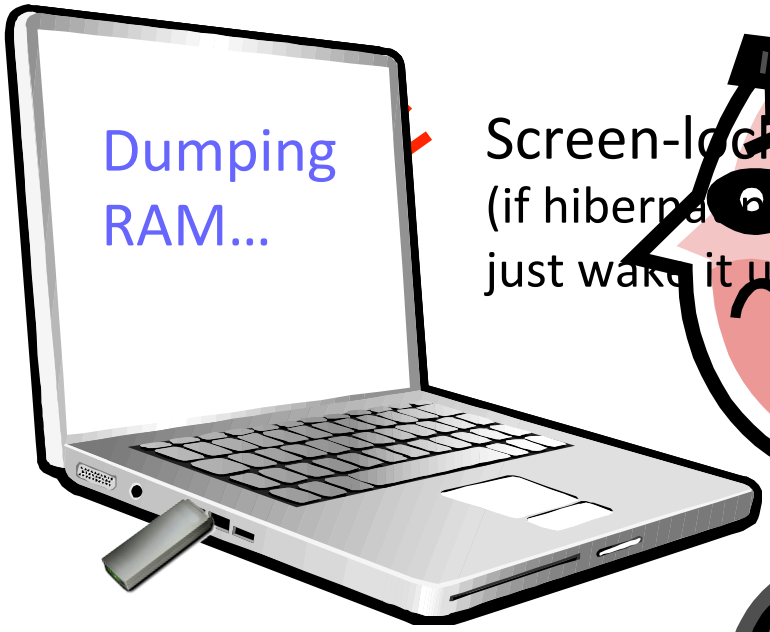
Boot a small low-level program to write out (dump) memory content

Implementations

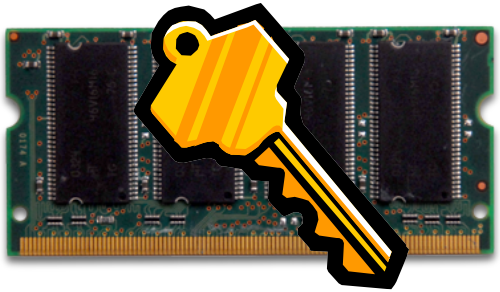
PXE Dump (9 KB)	**EFI Dump** (10 KB)	**USB Dump** (22 KB)

Delivering the Attack

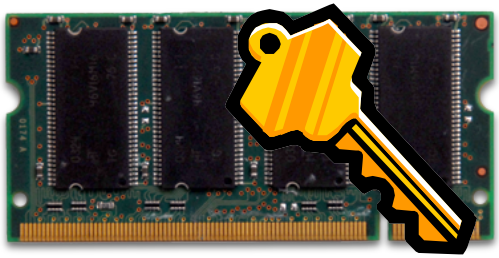
Basic Cold-Boot Attack



Screen-locked machine
(if hibernating/sleeping,
just wake it up!)



What if the BIOS clears RAM?



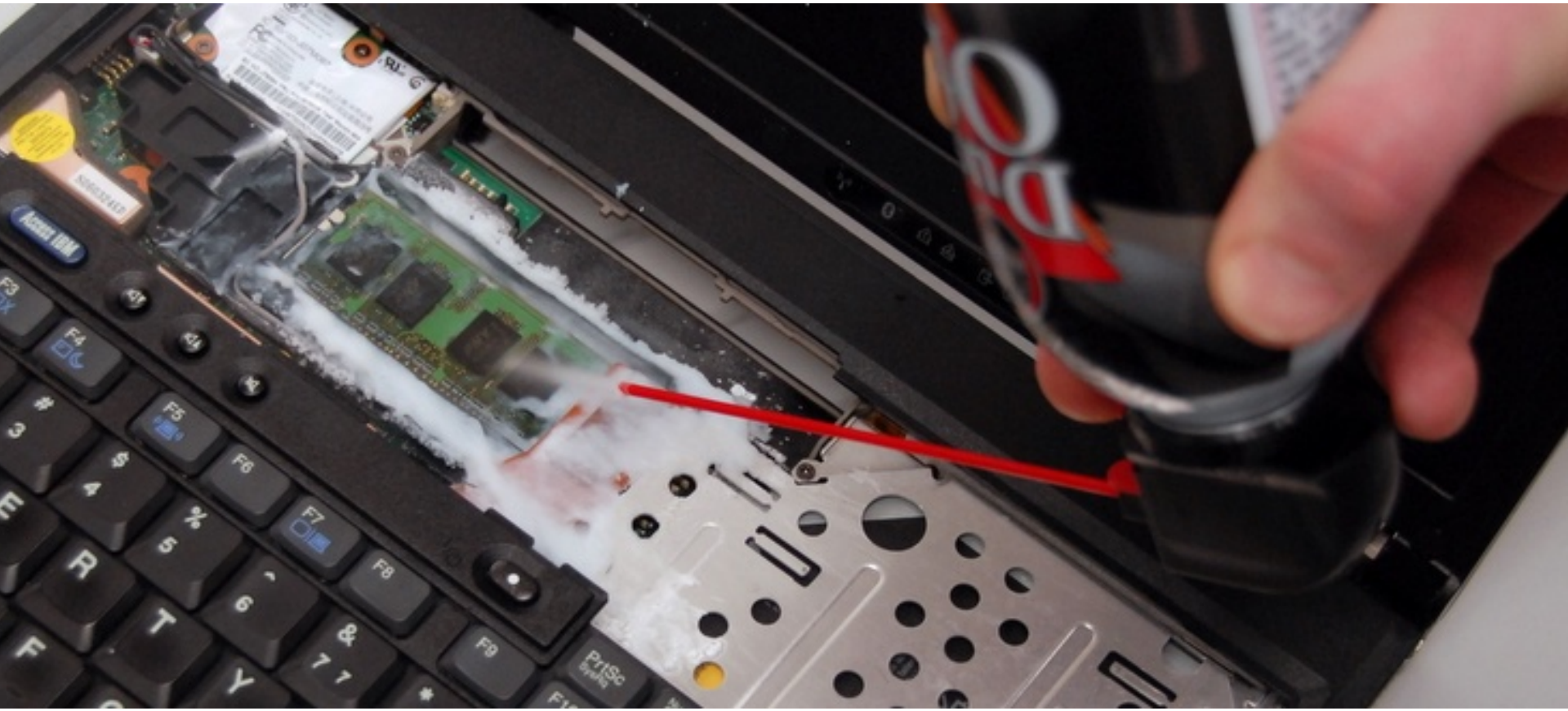
Common in machines that support ECC RAM

Advanced Cold-Boot Attack

Victim's Computer

Attacker's Computer

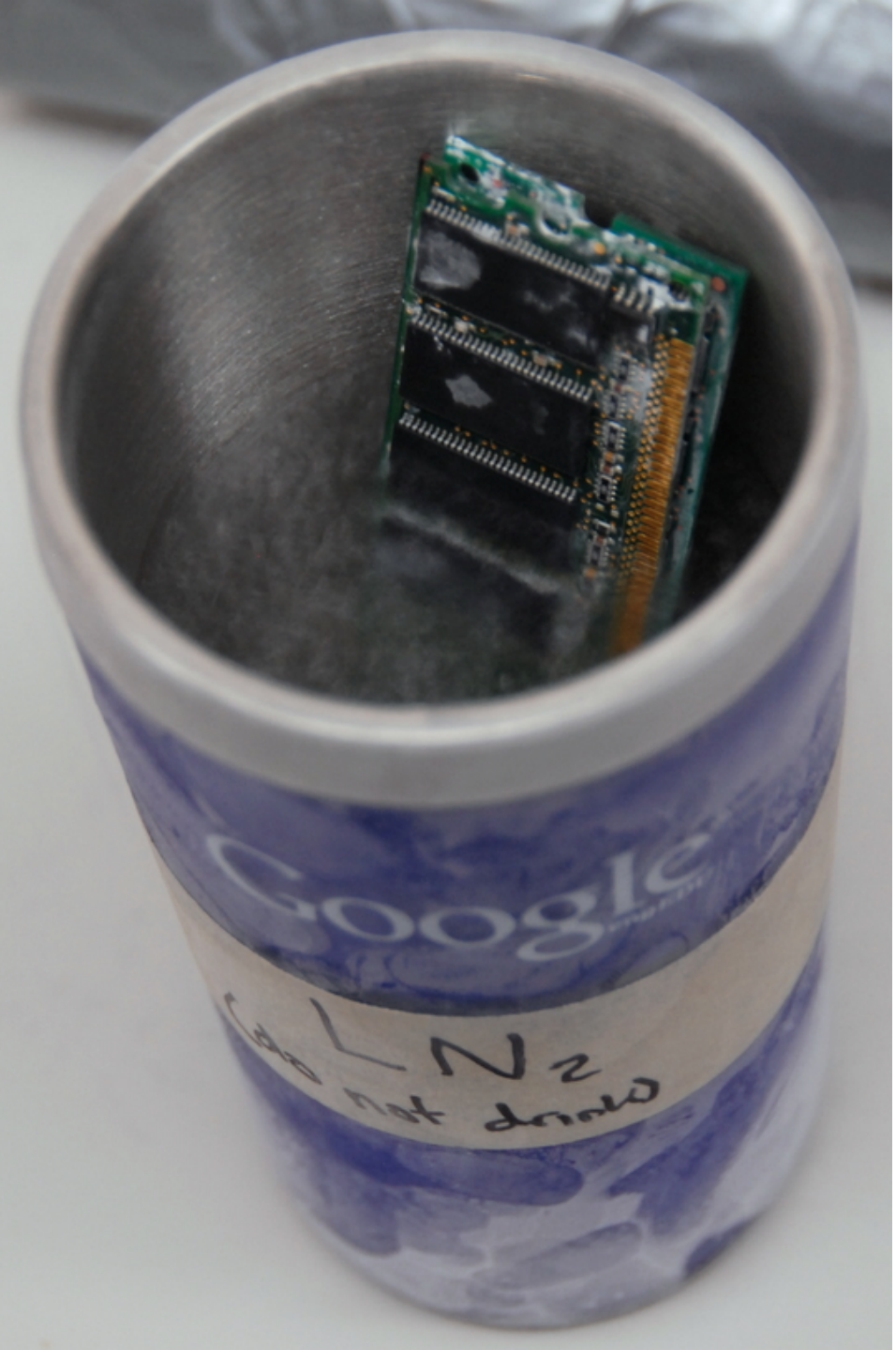
Slowing Decay By Cooling



-50°C

< 0.2% decay after **1 minute**


Esc
F1
F2
F3
Tab
START/STOP
SET
MODE
SPLIT/RESET
SELECT
SUN MON TUE WED THU FRI SAT
6:48
29
SEC
MIN
GOWalkin
by SPORTLINE



Even Cooler

Liquid Nitrogen **-196°C**

< 0.17% decay after **1 hour**

Not necessary in practice

Dealing with Bit Errors

Some bit errors inevitable, especially without cooling (increasing with memory density)

Naïve Approach

Given corrupted K', find K:

Brute-force search over low Hamming distance to K'

e.g. 256-bit key with 10% error:

$> 2^{56}$ guesses (too slow!)

Insight

Most programs store precomputed derivatives of K (e.g. key schedules)

These derivatives contain redundancy; we can treat them as error correcting codes

Recovering Other Keys

AES, DES (key schedules)

LRW tweak keys (multiplication tables)

RSA private keys (primes P and Q)

Efficiency vs. Security

Finding Keys

Previous Methods

- Entropy methods
  - But, most high entropy data isn't key material
- Looking for data structure
  - But, need to know internal details about software

Our Method

- Focuses on key schedules
  - Precise and fully automatic, even with decay
  - Doesn't require analysis of target program

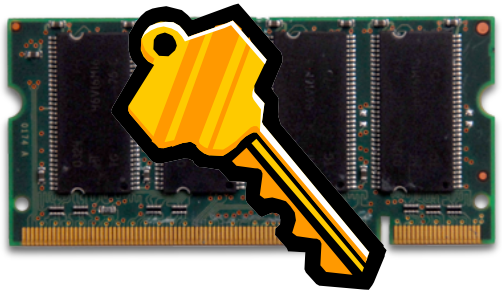
Finding AES Key Schedules

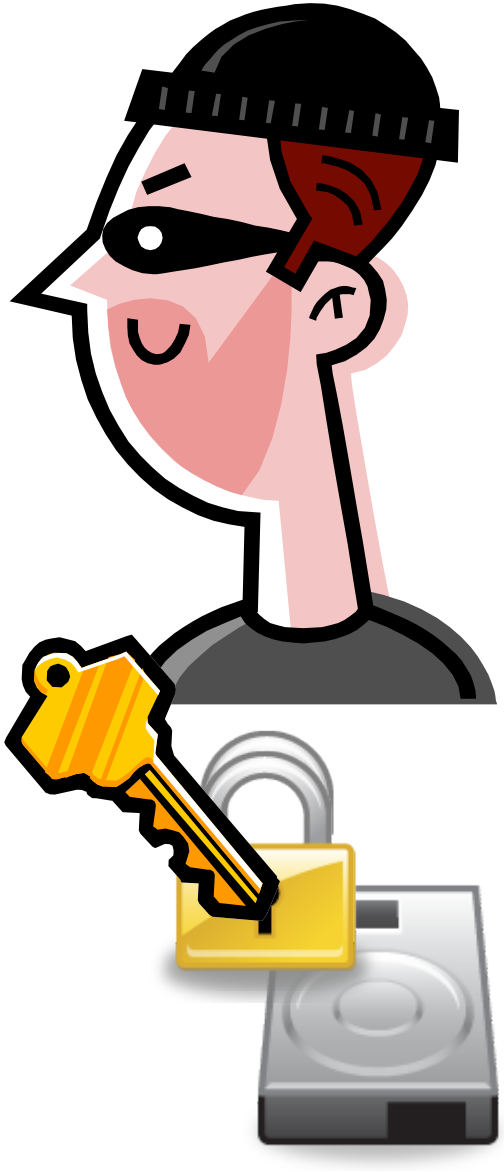
Iterate through each byte of memory –

1. Treat following region as an AES key schedule
2. For each word in the “schedule”:
   - Calculate correct value, assuming other bytes correct
   - Take Hamming distance to observed value
3. If total distance is low, output the key

Steps of Cold-Boot Attack

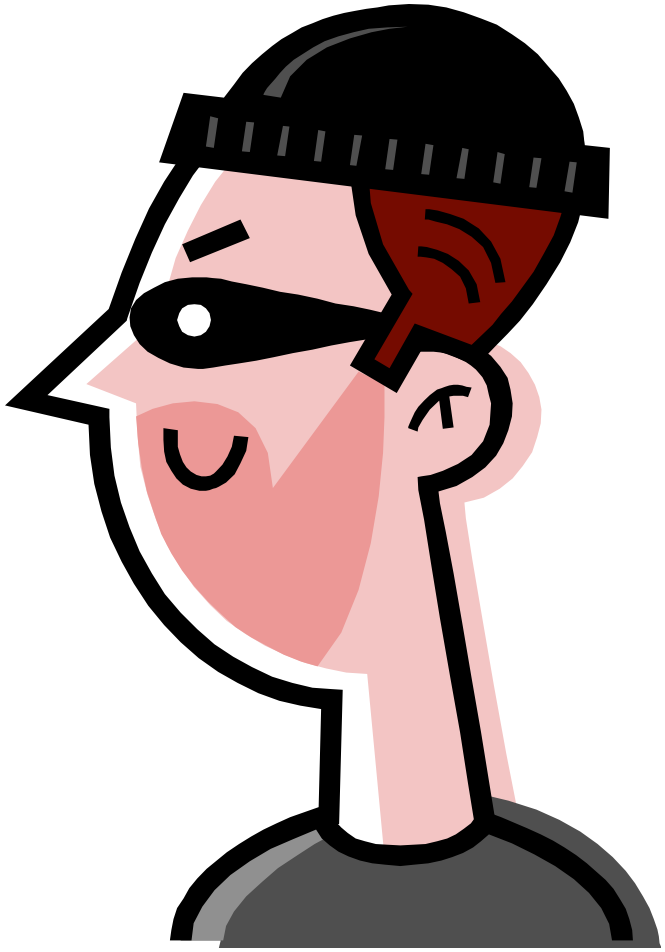
1. Extract memory
2. Locate keys in memory
3. Reconstruct decayed keys
4. Decrypt hard drive



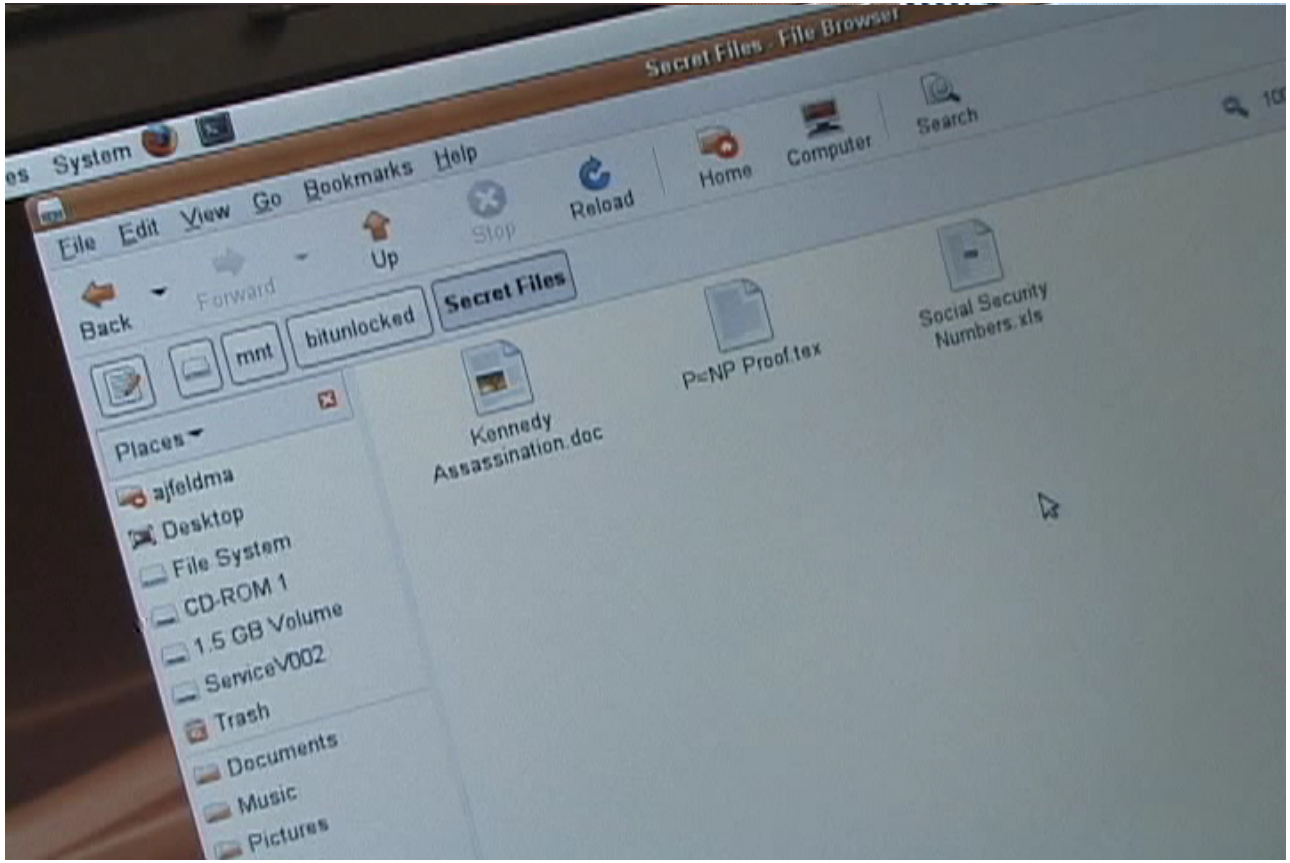


Demonstrated Attacks

- Windows BitLocker
- Mac OS FileVault
- Linux dm-crypt
- Linux LoopAES
- TrueCrypt



“BitLocker, Meet Bit*Un*locker”



Demonstration of fully automated attack:

Connect USB drive, reboot, and browse files

Countermeasures

No Magic Bullet

Possible Mitigations

- Encrypt key in memory when screen-locked
- Avoid precomputation
- Fully encrypted memory
- Trusted Platform Module (TPM)

Broader Lessons

- Mistaken assumptions can compromise the security of otherwise good crypto
- Abstractions (e.g., hardware/software divide) can hide critical behaviors and requirements
- Ultimately, security is an empirical property of systems – experiments can support or overturn theory

Covert channel example: EC2

Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds, by

Ristenpart et al.

A simplified model of public cloud computing

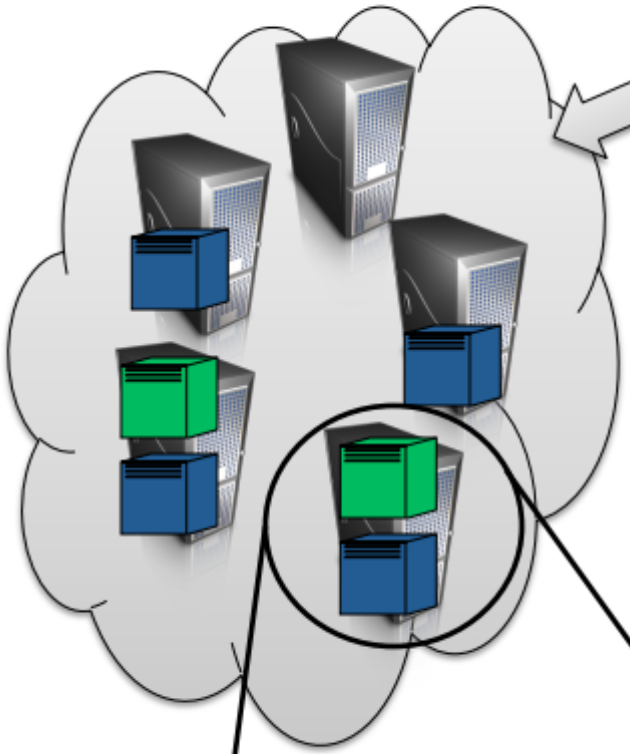
Users run Virtual Machines (VMs) on cloud provider's infrastructure

User A

virtual machines (VMs)

User B

virtual machines (VMs)



Owned/operated by cloud provider

Windows Azure

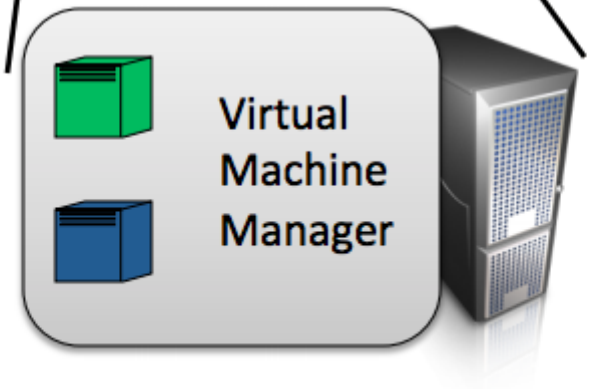
MOSSO

amazon web services

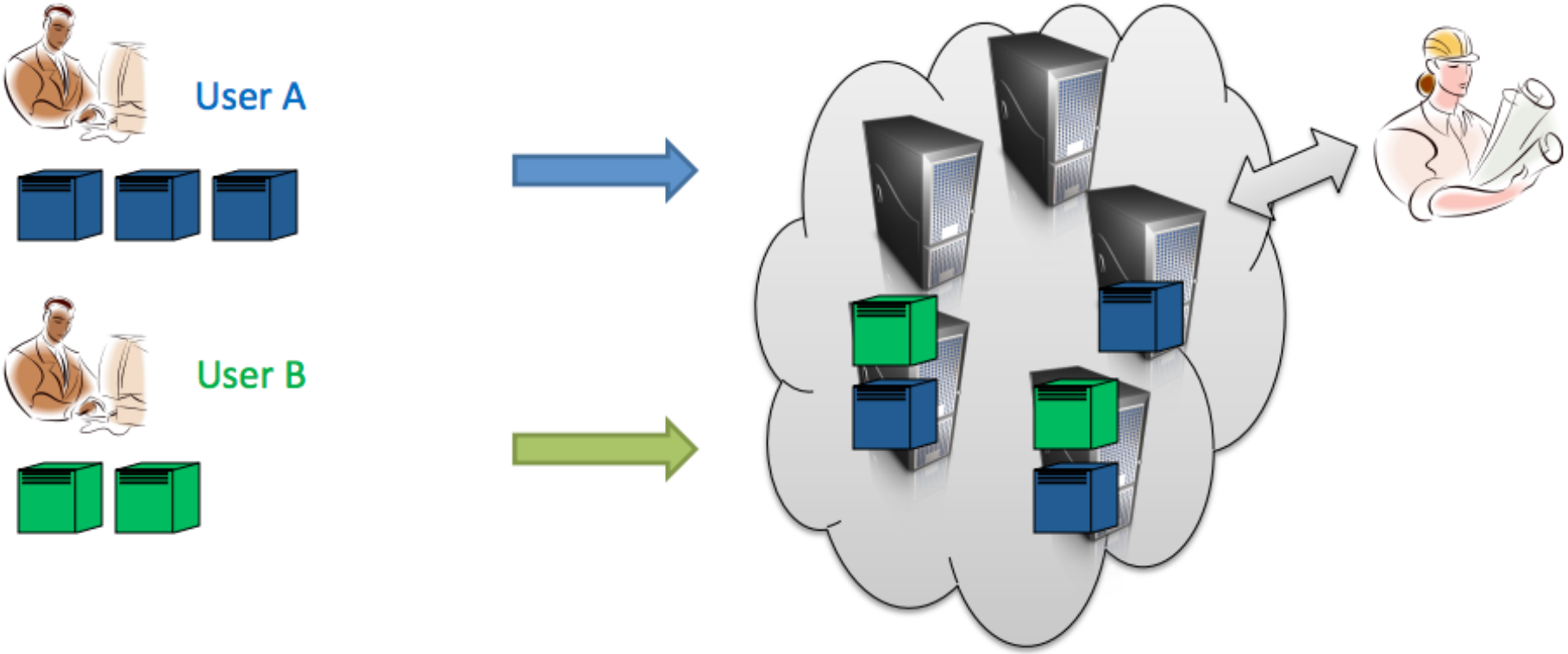
Multitenancy (users share physical resources)

Virtual Machine Manager (VMM) manages physical server resources for VMs

To the VM should look like dedicated server



Trust models in public cloud computing



Users must trust third-party provider to

not spy on running VMs / data

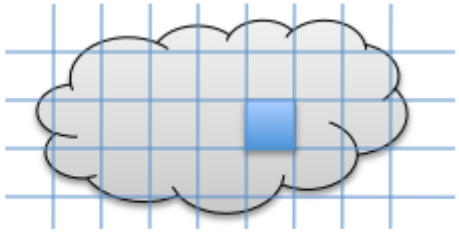
secure infrastructure from external attackers

secure infrastructure from internal attackers

Outline of a more damaging approach:

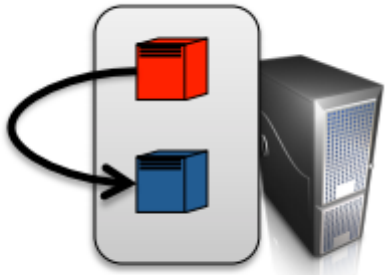
1) Cloud cartography

map internal infrastructure of cloud
map used to locate targets in cloud



2) Checking for co-residence

check that VM is on same server as target
- network-based co-residence checks
- efficacy confirmed by covert channels



3) Achieving co-residence

brute forcing placement
instance flooding after target launches

4) Location-based attacks

side-channels, DoS, escape-from-VM

Placement vulnerability: attackers can knowingly achieve co-residence with target

An Exploration of L2 Cache Covert Channels in Virtualized Environments

Yunjing Xu, Michael Bailey, Farnam Jahanian
University of Michigan
Kaustubh Joshi, Matti Hitunen, Rick Schlichting
AT&T Labs Research

Motivation

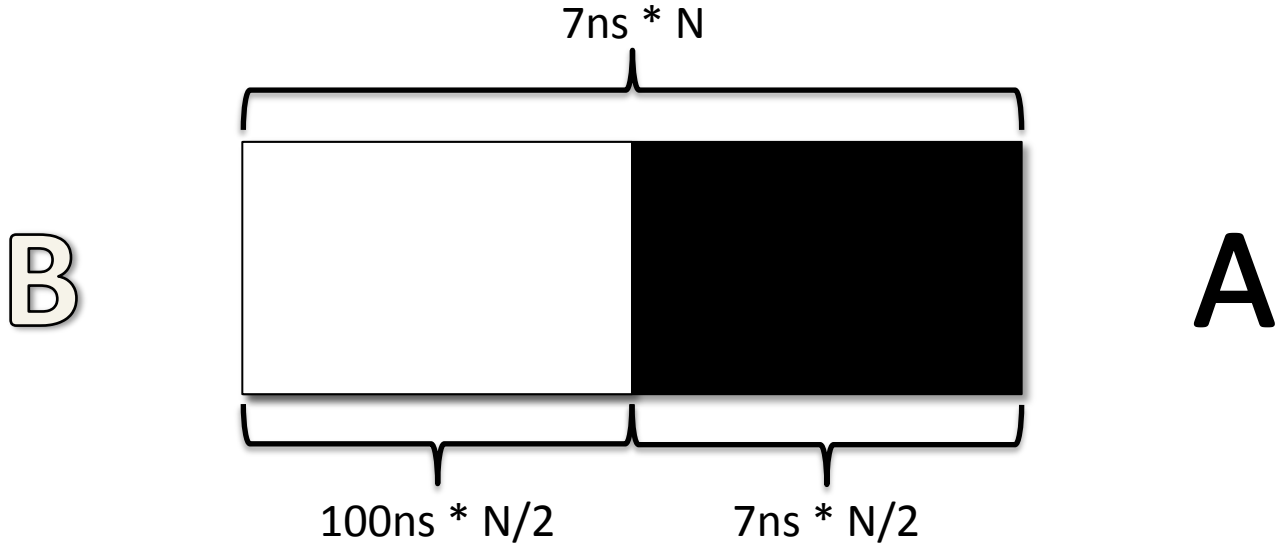
- The cloud environment (re)introduces shared hardware
- While powerful, this can be abused by attackers for information leakage
- Ristenpart et al. pioneering work in 2009
  - Forced VM co-location in EC2
  - Cross-VM information leakage
    - Covert channels
    - Side channels

Motivation

- Cross-VM covert channel using shared L2 cache
  - 0.2bps: proof-of-concept example
- What is the maximum achievable bit rate in practice over cross-VM covert channels using L2 cache?
- What factors influence the bit rate achievable in this scenario?

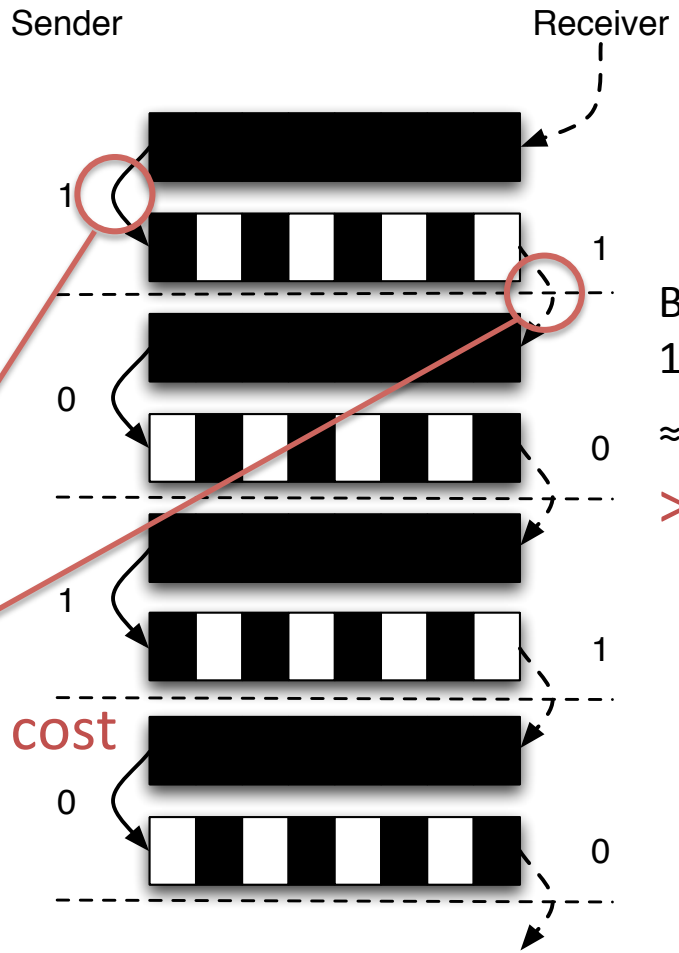
Cross-VM L2 Cache Covert Channel

- **Processes on different VMs sharing a L2 cache can covertly communicate by manipulating the cache content**



I_2 = log 2 = 1bit
I_n = log n

Bit Rate Estimation

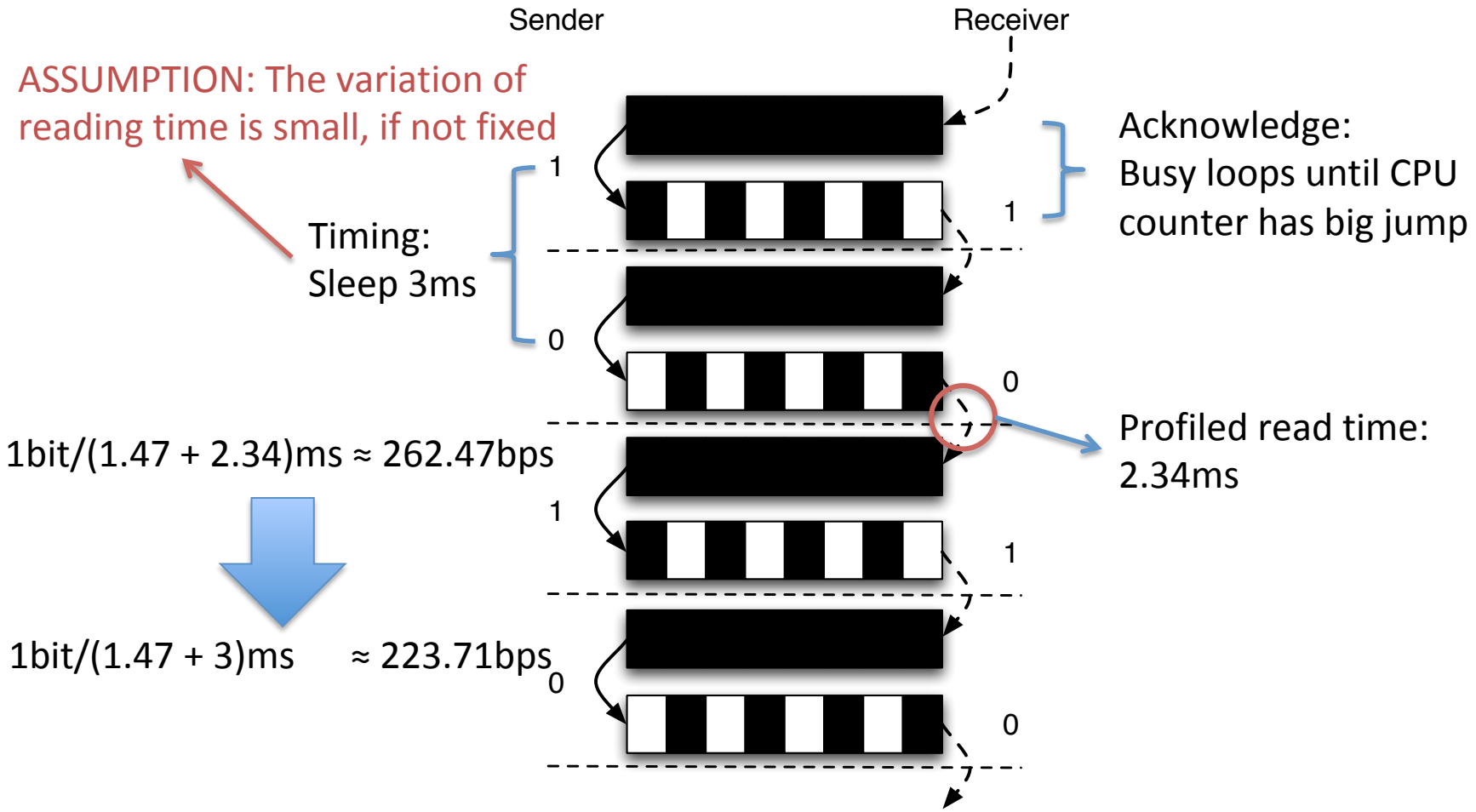


IMPLEMENTATION
on lab machines:
Write 1bit: 1.47ms
Read 1bit: 2.34ms

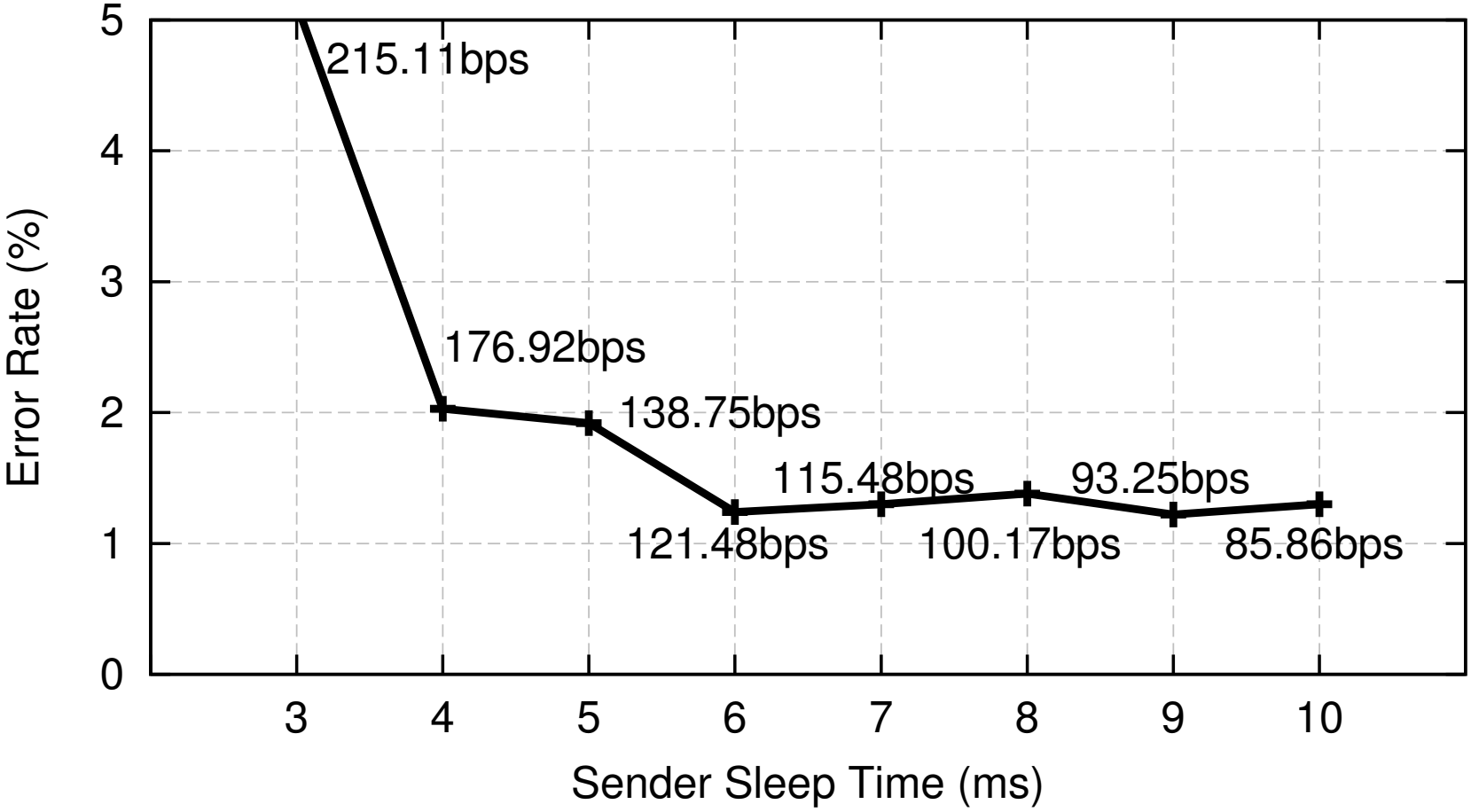
Bit rate:
1bit/(1.47ms + 2.34ms)
≈ 262.47bps
>> 0.2bps

No Synchronization cost

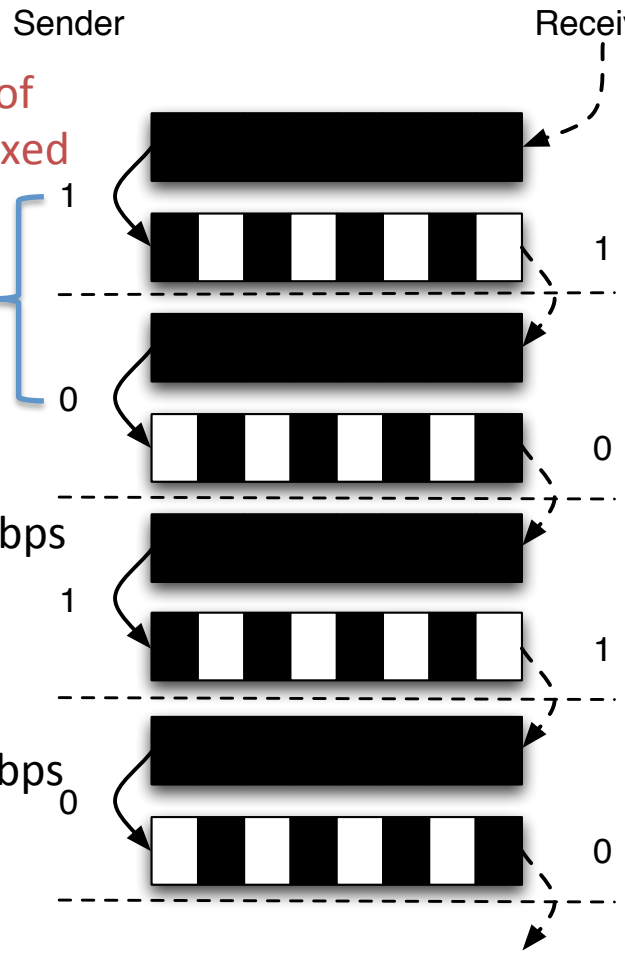
Cost of Synchronization



Sleep time vs. Bit rate



Cost of Synchronization



ASSUMPTION: The variation of reading time is small, if not fixed

Timing:
Sleep 3ms

Acknowledge:
Busy loops until CPU counter has big jump

1bit/(1.47 + 2.34)ms ≈ 262.47bps

1bit/(1.47 + 3)ms ≈ 223.71bps

ASSUMPTION:
Sender and receiver share the physical CPU

CPU Caps

- 40% hard limit on EC2 small instances
  - Cannot communicate in 60% of time

$$1\text{bit}/(1.47\text{ms} + 3\text{ms})^{*} \approx 223.71\text{bps}$$

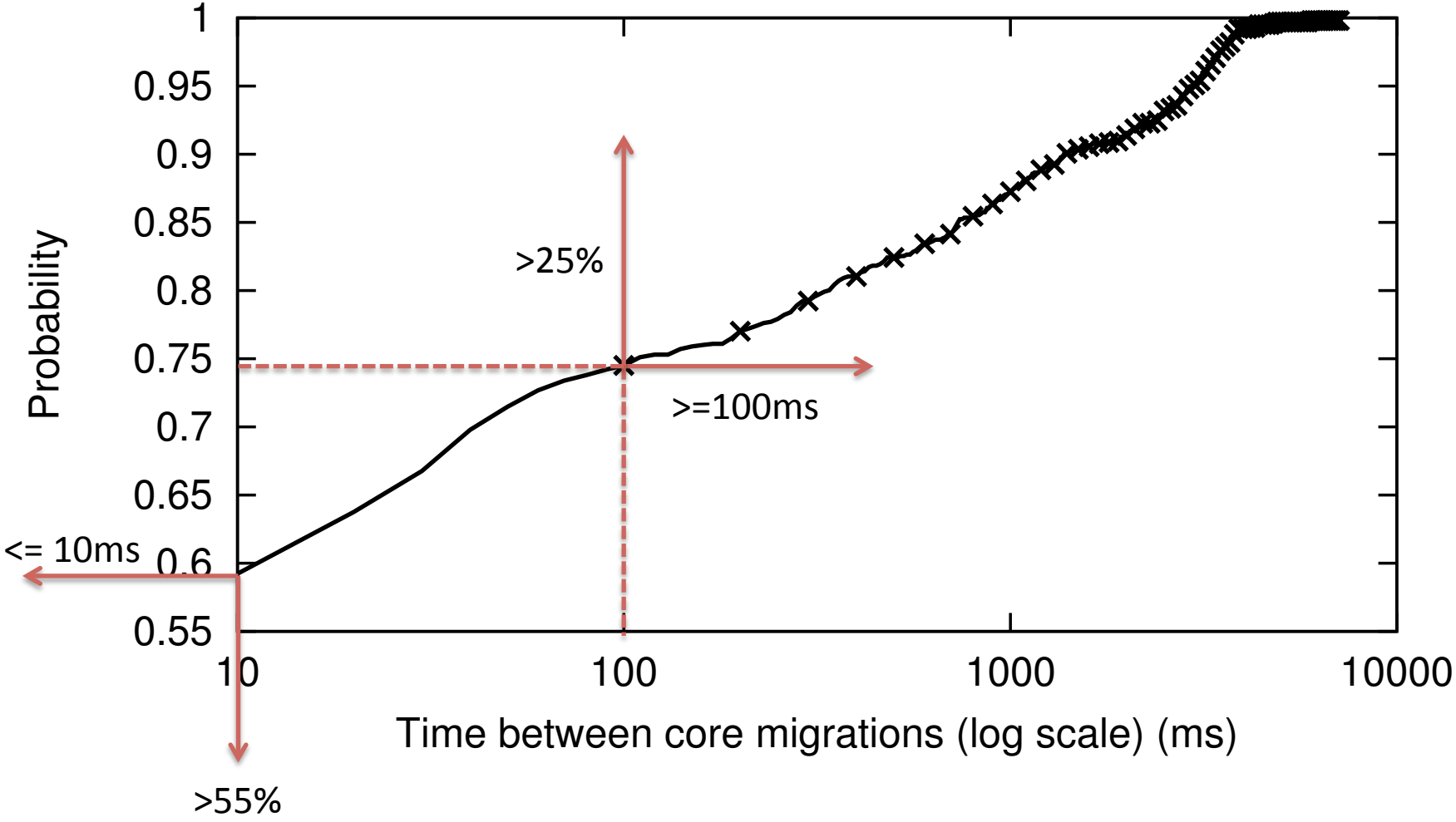
$$1\text{bit}/(1.47\text{ms} + 3\text{ms}) * 40\% \approx 89.48\text{bps}$$

* Refer to the paper for the numbers on EC2

Core Migration

- EC2 VCPUs are not pinned on physical CPU
  - CPU counter trick requires shared CPU
  - Two physical CPUs may not share L2 cache
- EC2's core migration policy is unclear
  - Empirical data

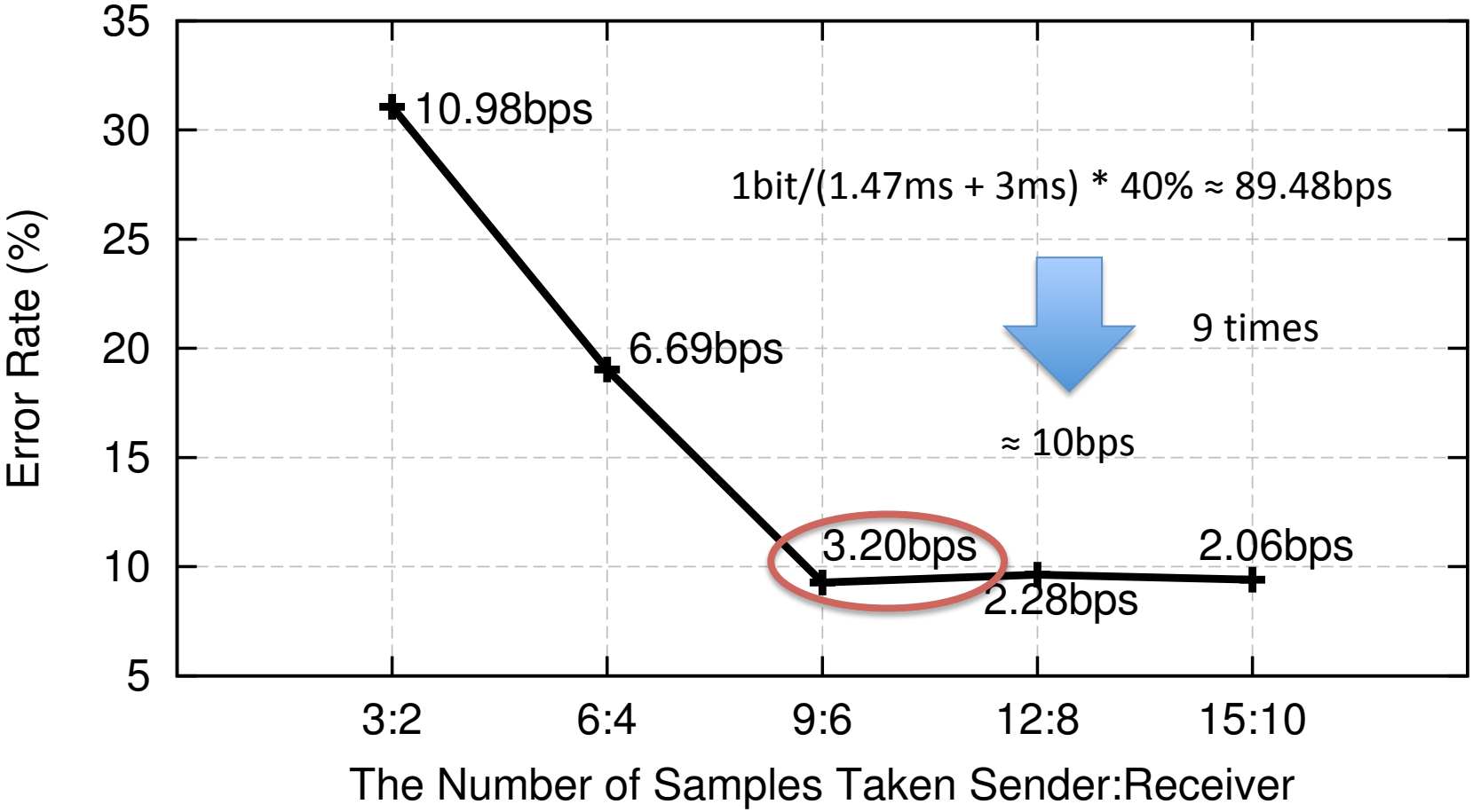
Profile the Core Migration



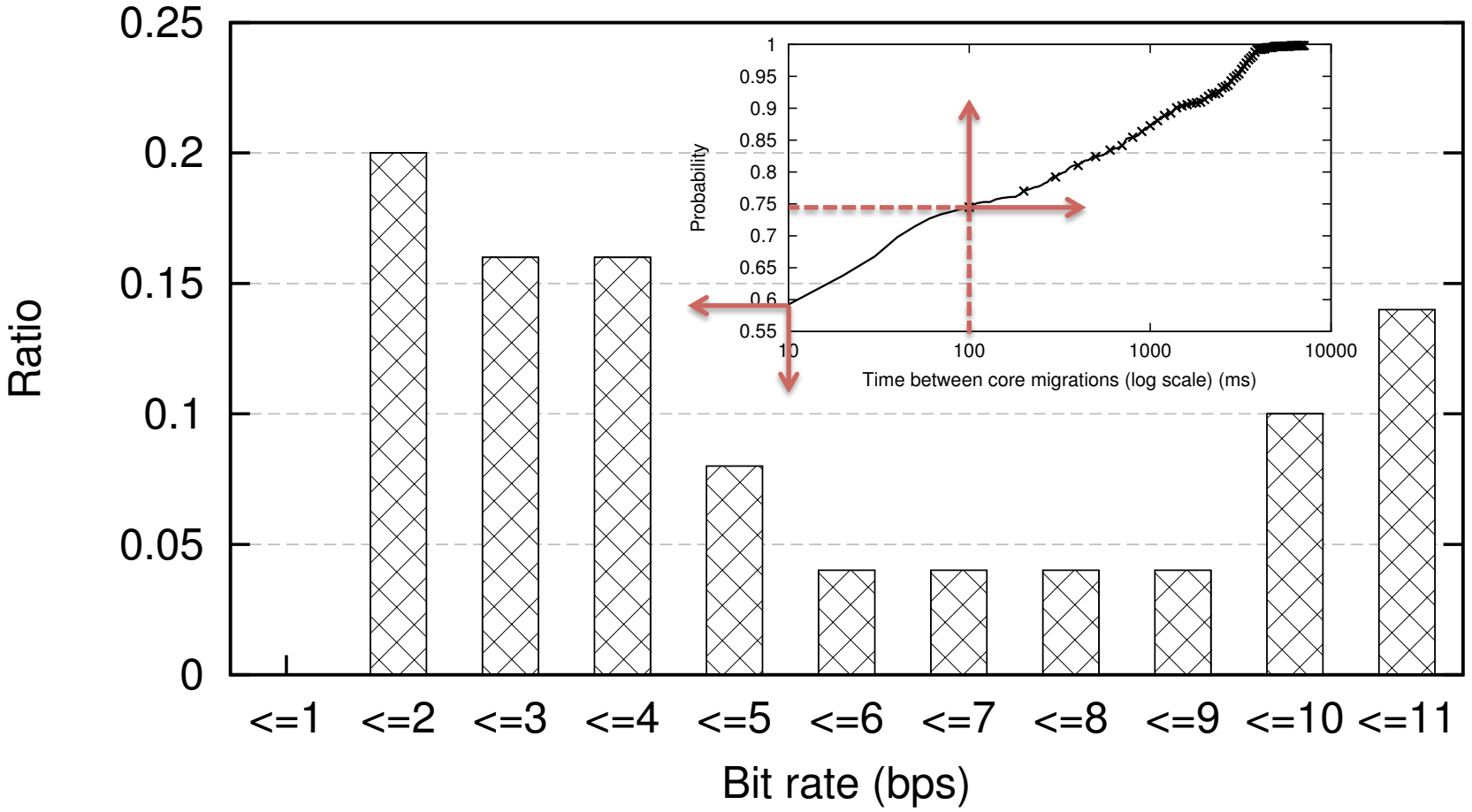
Sharing resistant Synchronization

- Ristenpart et al.
  - Take multiple samples to make a decision
  - Revert to the beginning if core migrated
- Challenges of this approach:
  - Effectiveness of the CPU counter technique
  - Repeat: how many times?
- A protocol is detailed on the paper
  - Use L2 cache itself to indicate activities
  - Explore optimal number of repeats

The Number of Samples Taken



Bit Rate Distribution



Conclusion and Future Work

- Questions to be answered
  - Maximum achievable bit rate
  - Factors influence the bit rate
- A better picture
  - Sleep time little impact if selected right
  - 40% CPU cap: ~200bps -> ~90bps
  - Core migration: ~90bps -> 10bps max, 3bps average
- Data exfiltration using covert channels on EC2
  - No documents leaked out to wikileaks this way
  - Leak small secrets like private keys
- Future work
  - Other cloud providers

Flipping Bits in Memory Without Accessing Them

DRAM Disturbance Errors

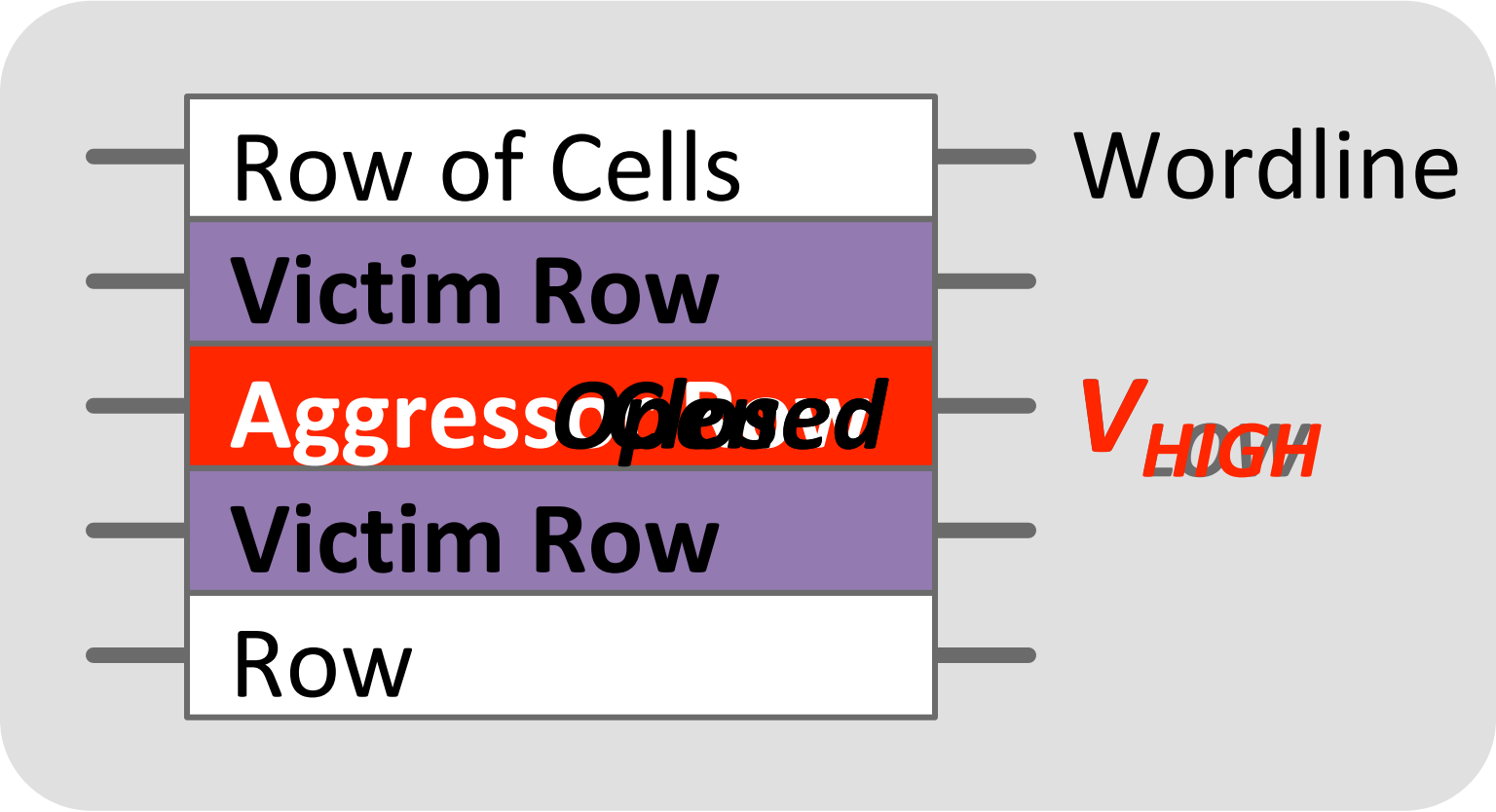
Yoongu Kim

Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee,

Donghyuk Lee, Chris Wilkerson, Konrad Lai, Onur Mutlu

Carnegie Mellon SAFARI intel

DRAM Chip

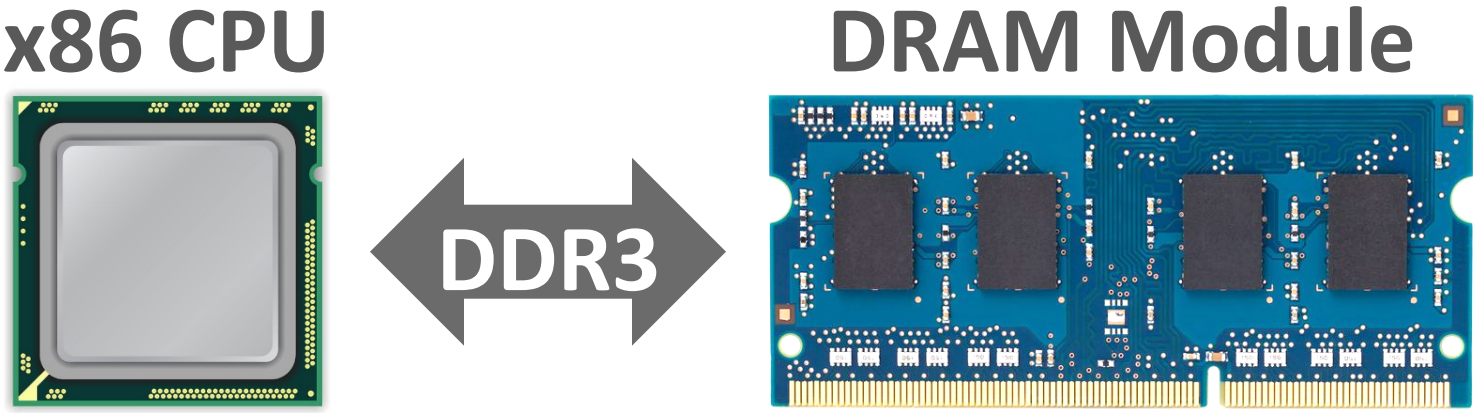


Repeatedly opening and closing a row induces ***disturbance errors*** *in adjacent rows*

Quick Summary of Paper

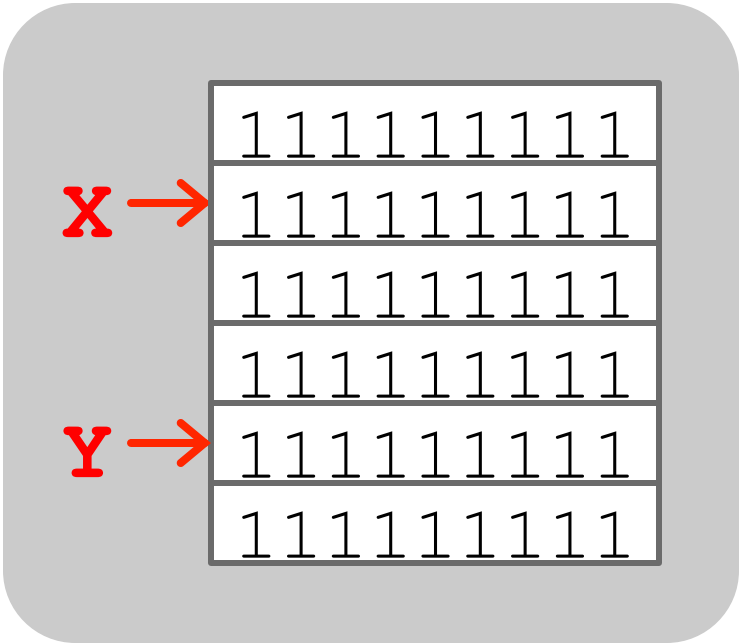
- *We expose the **existence** and **prevalence** of disturbance errors in DRAM chips of today*
  - 110 of 129 modules are vulnerable
  - Affects modules of 2010 vintage or later
- *We characterize the **cause** and **symptoms***
  - Toggling a row accelerates charge leakage in adjacent rows: *row-to-row coupling*
- *We prevent errors using a **system-level** approach*
  - Each time a row is closed, we refresh the charge stored in its adjacent rows with a low probability

How to Induce Errors



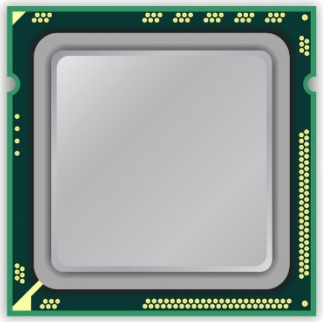
1. Avoid ***cache hits***
   - Flush **X** from cache

2. Avoid ***row hits*** to **X**
   - Read **Y** in another row



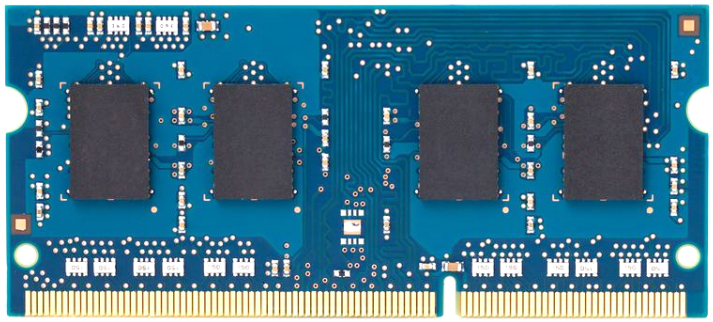
How to Induce Errors

x86 CPU

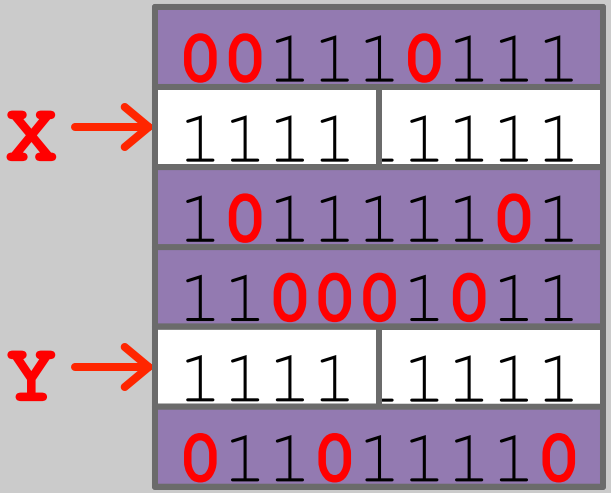


DDR3

DRAM Module



```
loop:
   mov (X), %eax
   mov (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp loop
```



Security Implications

- *Breach of memory protection*
  - OS page (4KB) fits inside DRAM row (8KB)
  - Adjacent DRAM row → Different OS page
- *Vulnerability: disturbance attack*
  - By accessing its own page, a program could corrupt pages belonging to another program
- *We constructed a proof-of-concept*
  - Using only user-level instructions

Exploiting the DRAM rowhammer bug to gain kernel privileges

How to cause and exploit
single bit errors

Mark Seaborn and Thomas Dullien

Kernel exploit

- x86 page tables entries (PTEs) are **dense and trusted**
  - They control access to physical memory
  - A bit flip in a PTE's physical page number can give a process access to a different physical page
- Aim of exploit: Get access to a page table
  - Gives access to all of physical memory
- Maximise chances that a bit flip is useful:
  - Spray physical memory with page tables
  - Check for useful, repeatable bit flip first

x86-64 Page Table Entries (PTEs)

- Page table is a 4k page containing array of 512 PTEs
- Each PTE is 64 bits, containing:

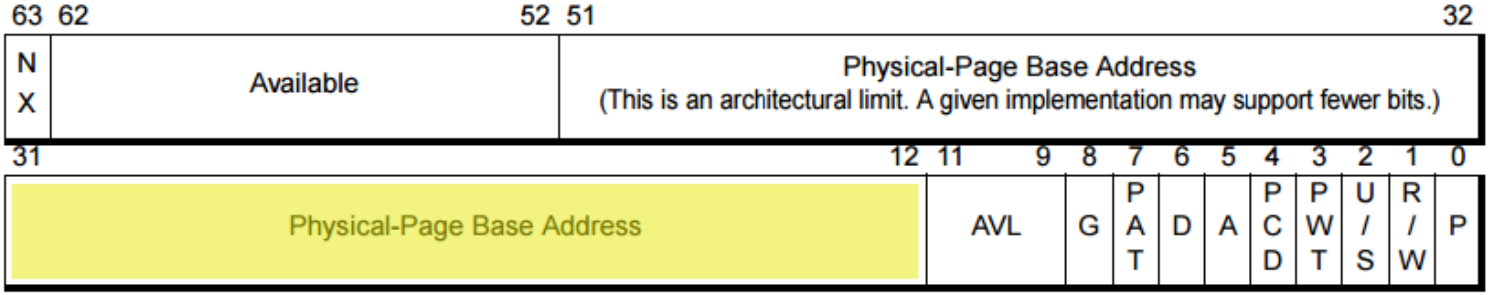
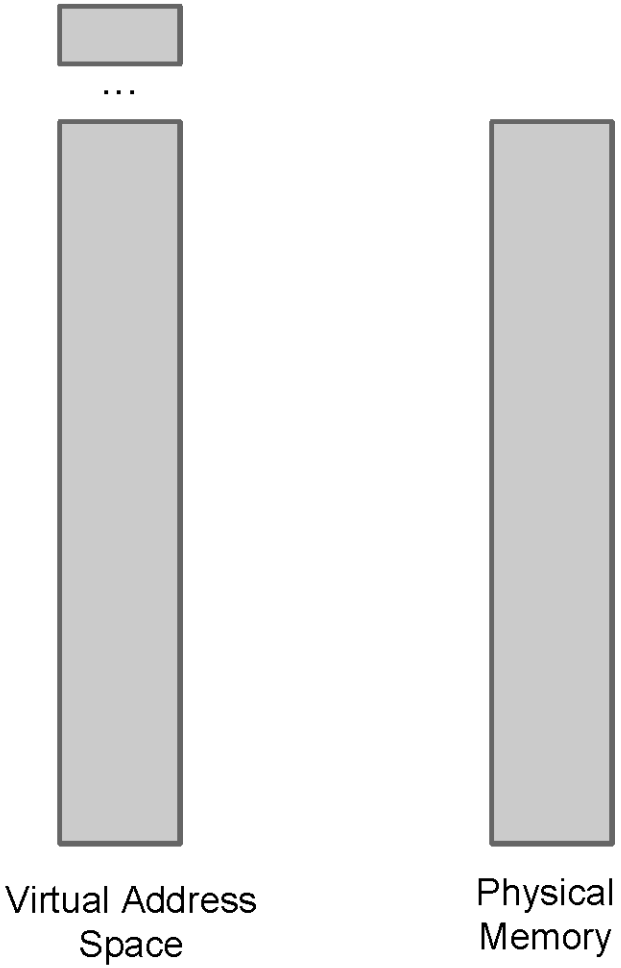
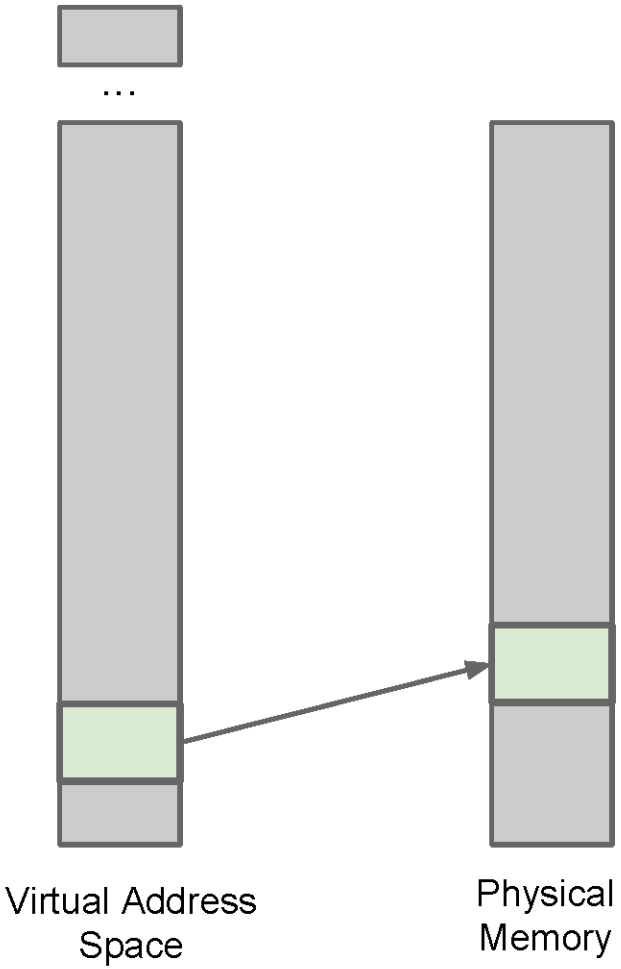


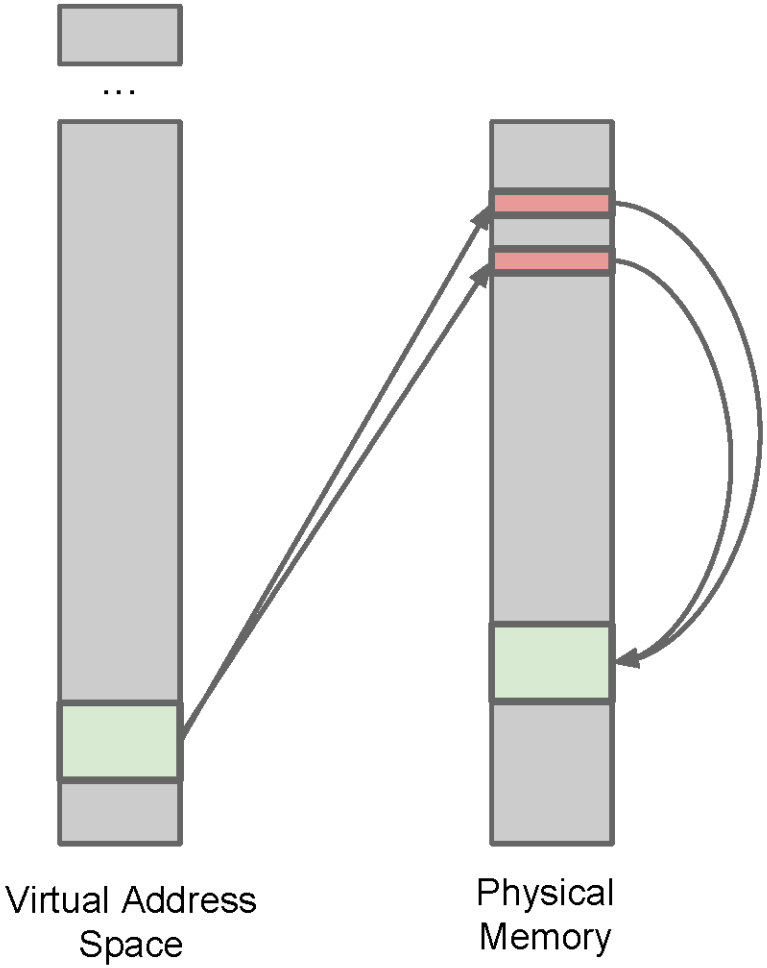
Figure 5-21. 4-Kbyte PTE—Long Mode

- Could flip:
  - "Writable" permission bit (RW): 1 bit → 2% chance
  - Physical page number: 20 bits on 4GB system → 31% chance

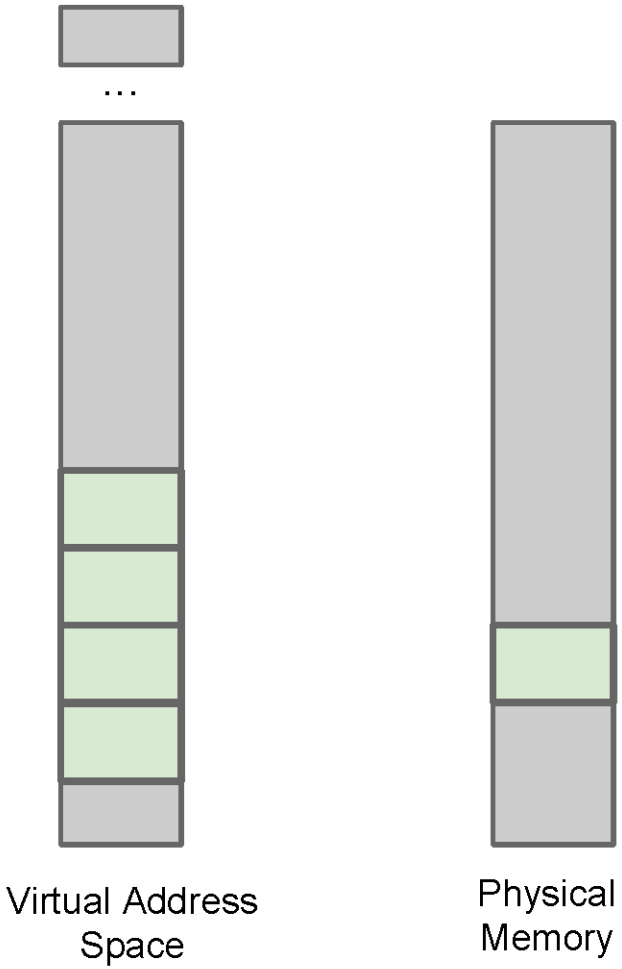

...
Virtual Address Space
Physical Memory



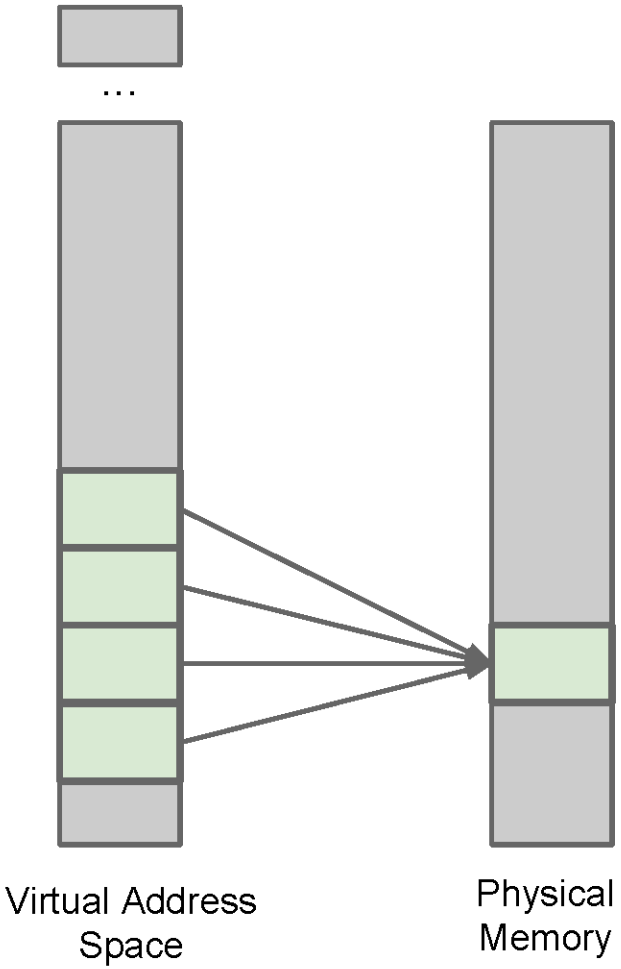
What happens when we map a file with read-write permissions?



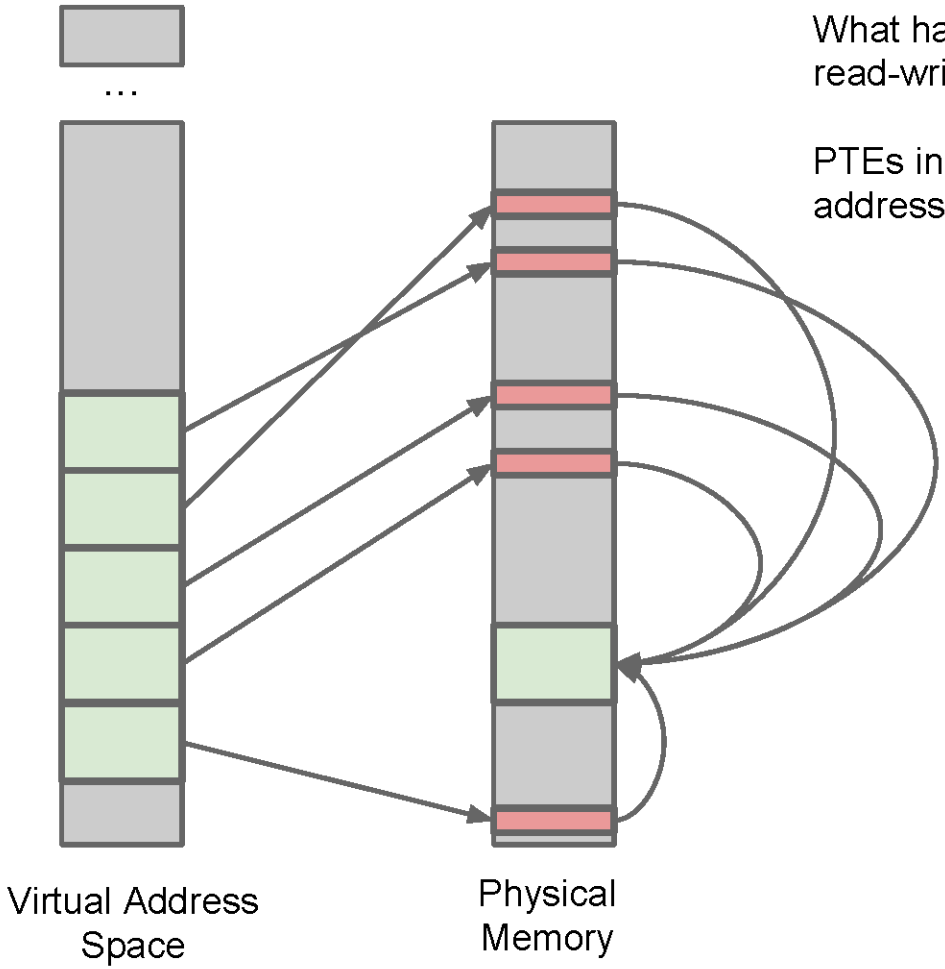
What happens when we map a file with read-write permissions? Indirection via page tables.



What happens when we repeatedly map a file with read-write permissions?

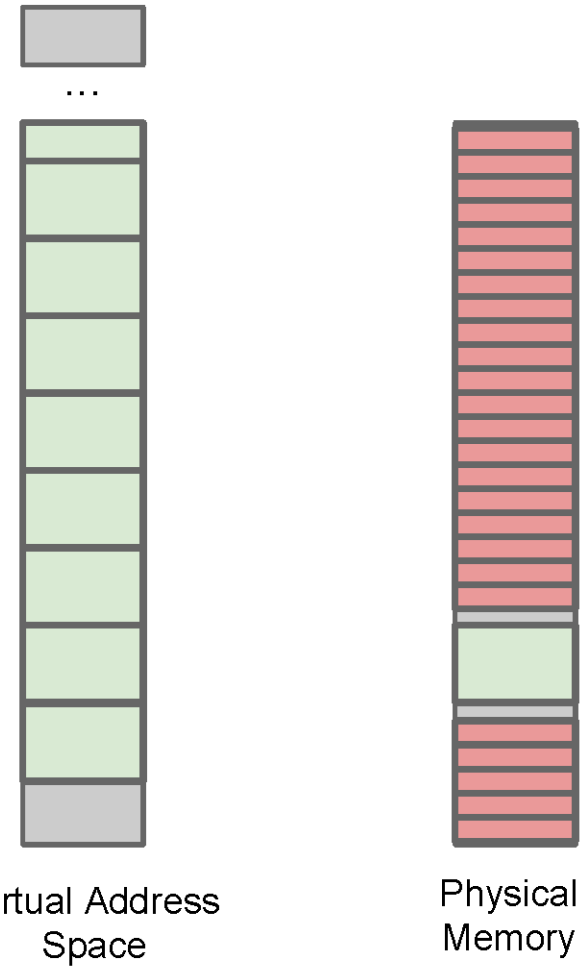


What happens when we repeatedly map a file with read-write permissions?



What happens when we repeatedly map a file with read-write permissions?

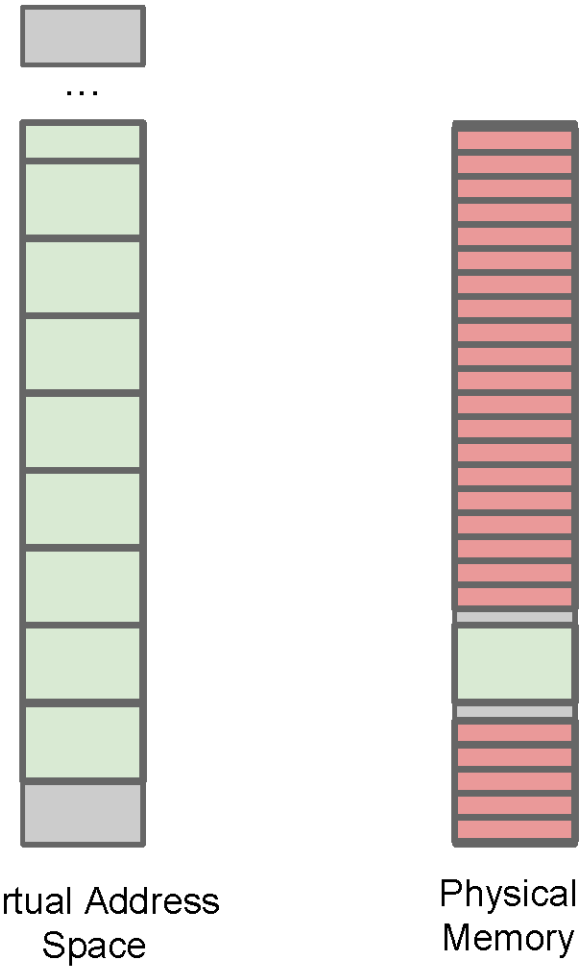
PTEs in physical memory help resolve virtual addresses to physical pages.



What happens when we repeatedly map a file with read-write permissions?

PTEs in physical memory help resolve virtual addresses to physical pages.

We can fill physical memory with PTEs.

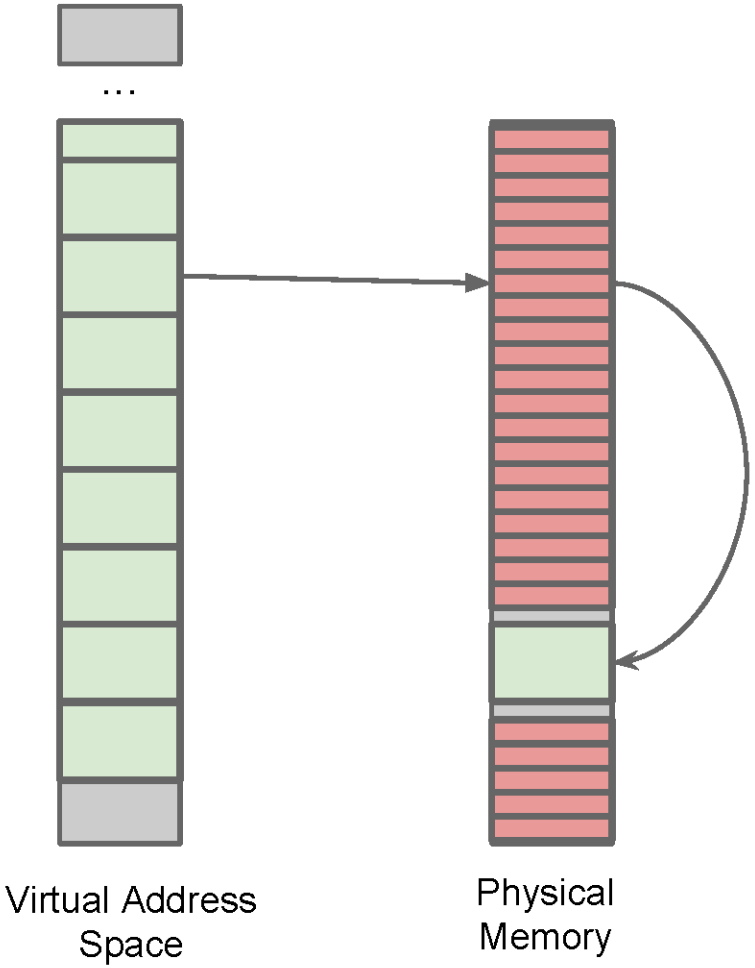


What happens when we repeatedly map a file with read-write permissions?

PTEs in physical memory help resolve virtual addresses to physical pages.

We can fill physical memory with PTEs.

Each of them points to pages in the same physical file mapping.



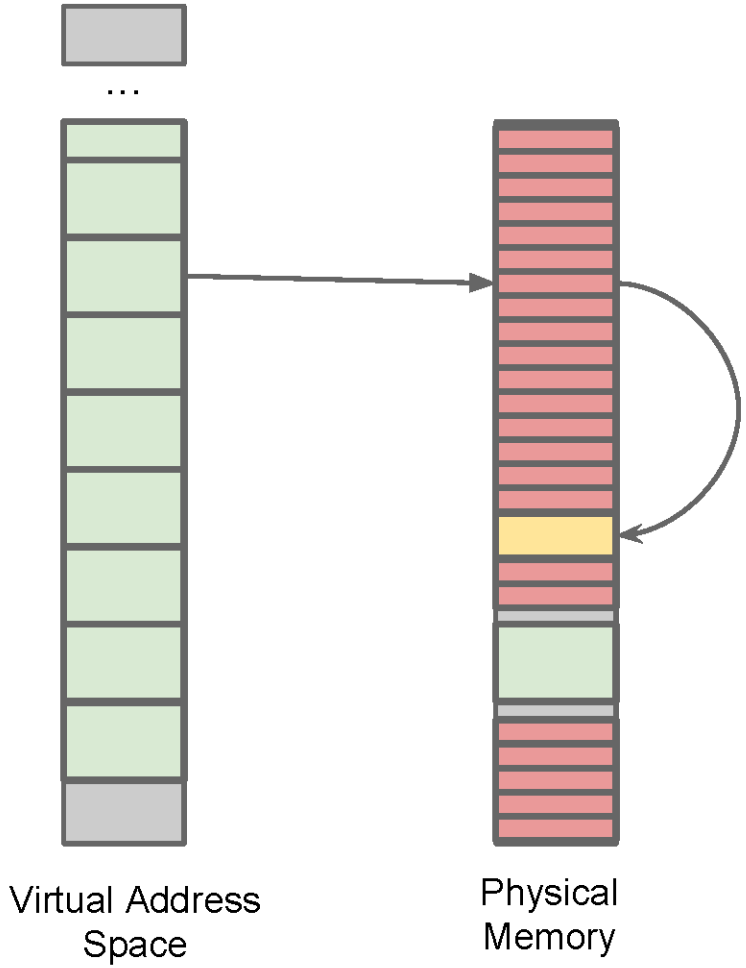
What happens when we repeatedly map a file with read-write permissions?

PTEs in physical memory help resolve virtual addresses to physical pages.

We can fill physical memory with PTEs.

Each of them points to pages in the same physical file mapping.

If a bit in the right place in the PTE flips ...



What happens when we repeatedly map a file with read-write permissions?

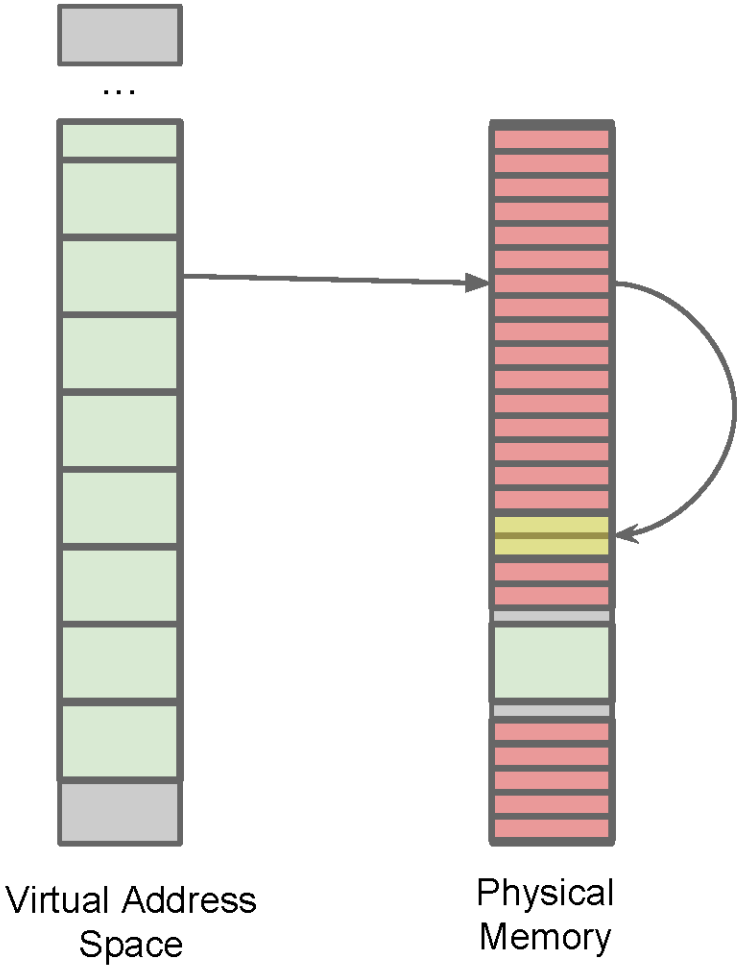
PTEs in physical memory help resolve virtual addresses to physical pages.

We can fill physical memory with PTEs.

Each of them points to pages in the same physical file mapping.

If a bit in the right place in the PTE flips …

… the corresponding virtual address now points to a wrong physical page - with RW access.



What happens when we repeatedly map a file with read-write permissions?

PTEs in physical memory help resolve virtual addresses to physical pages.

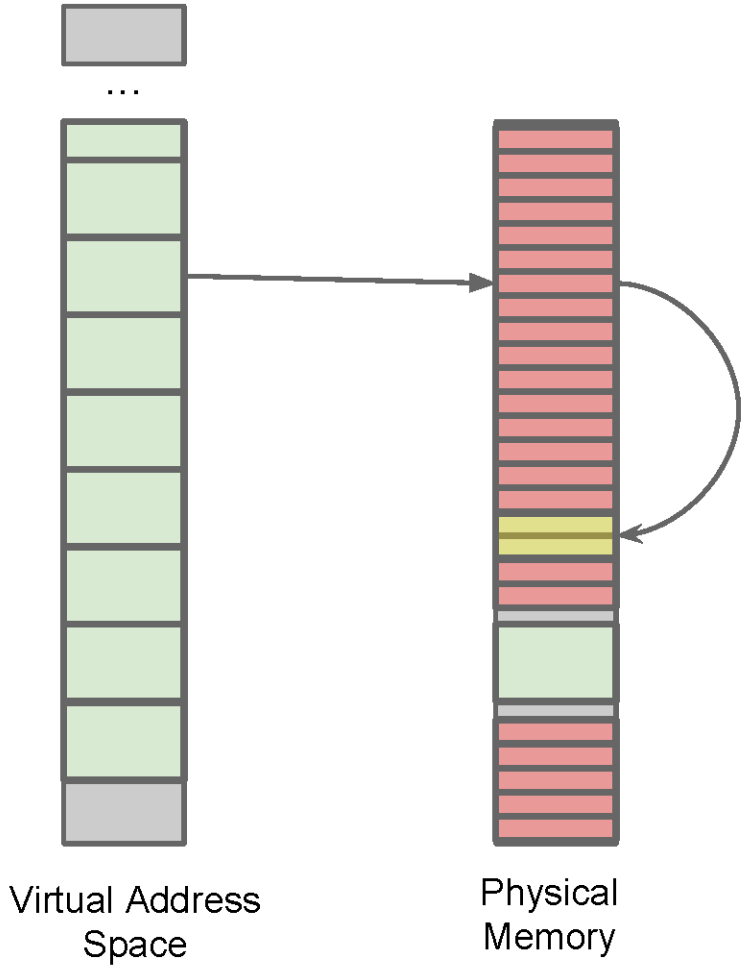
We can fill physical memory with PTEs.

Each of them points to pages in the same physical file mapping.

If a bit in the right place in the PTE flips …

… the corresponding virtual address now points to a wrong physical page - with RW access.

Chances are this wrong page contains a page table itself.



What happens when we repeatedly map a file with read-write permissions?

PTEs in physical memory help resolve virtual addresses to physical pages.

We can fill physical memory with PTEs.

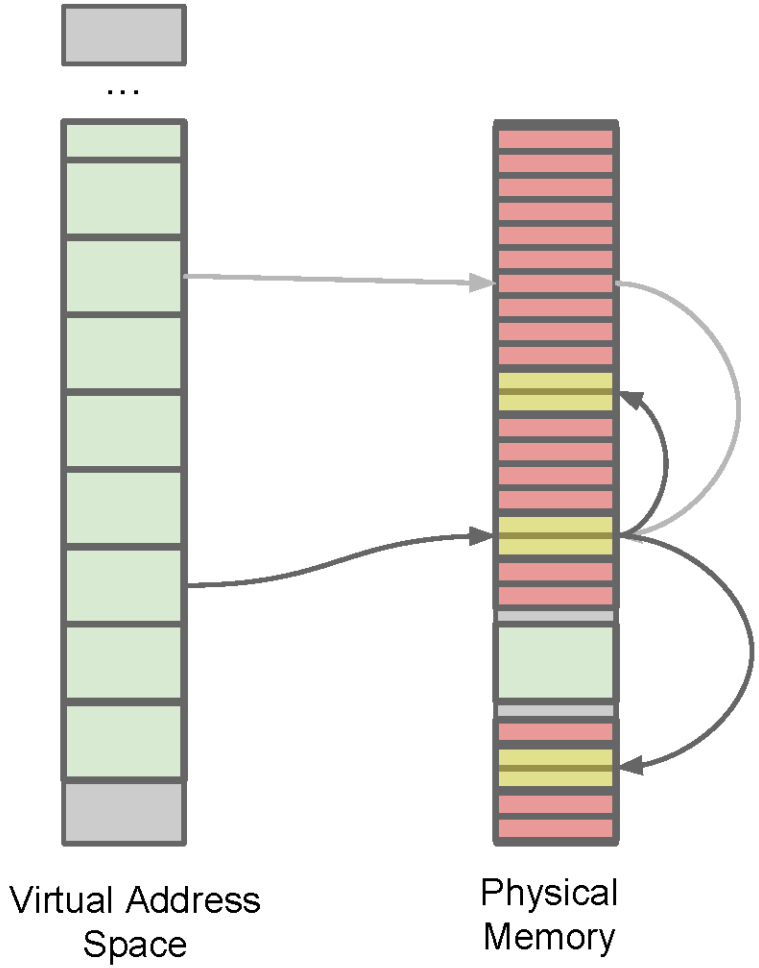
Each of them points to pages in the same physical file mapping.

If a bit in the right place in the PTE flips …

… the corresponding virtual address now points to a wrong physical page - with RW access.

Chances are this wrong page contains a page table itself.

An attacker that can read / write page tables …



What happens when we repeatedly map a file with read-write permissions?

PTEs in physical memory help resolve virtual addresses to physical pages.

We can fill physical memory with PTEs.

Each of them points to pages in the same physical file mapping.

If a bit in the right place in the PTE flips …

… the corresponding virtual address now points to a wrong physical page - with RW access.

Chances are this wrong page contains a page table itself.

An attacker that can read / write page tables can use that to map **any** memory read-write.

Exploit strategy

Privilege escalation in 7 easy steps …

1. Allocate a large chunk of memory
2. Search for locations prone to flipping
3. Check if they fall into the “right spot” in a PTE for allowing the exploit
4. Return that particular area of memory to the operating system
5. Force OS to re-use the memory for PTEs by allocating massive quantities of address space
6. Cause the bitflip - shift PTE to point into page table
7. Abuse R/W access to all of physical memory

In practice, there are many complications.