The House is Built on Sand

Exploiting Hardware Glitches and Side Channels in Perfect Software

Herbert Bos

Vrije Universiteit Amsterdam

Outline of the talk

- Begin
- Middle
- End

VUSec
https://vusec.net

VUSec

https://vusec.net

Erik Bosman

Kaveh Razavi

Victor van der Veen

Cristiano Giuffrida

Andrei Tatar

Ben Gras

Pietro Frigo

Dennis Andriesse

Lucian Cojocar

Radhesh Konoth


31337
Microsoft Bounty Program
One Microsoft Way, Redmond, WA 98052
Pay to the order of
James Forshaw
$ 100,000
One Hundred Thousand and 00/100
Dollars USD
Memo
$100,000 Microsoft


PWN2FEST
PWN2FEST
$300,000
PwnFest'16


$215,000 Pwn2Own


I need a new terrace

VUSec
https://vusec.net

Erik Bosman

Kaveh Razavi

Victor van der Veen

Cristiano Giuffrida

Andrei Tatar

Ben Gras

Pietro Frigo

Dennis Andriesse

Lucian Cojocar

Radhesh Konoth

VUSec
https://vusec.net

Exploit students

Erik Bosman

Kaveh Razavi

Victor van der Veen

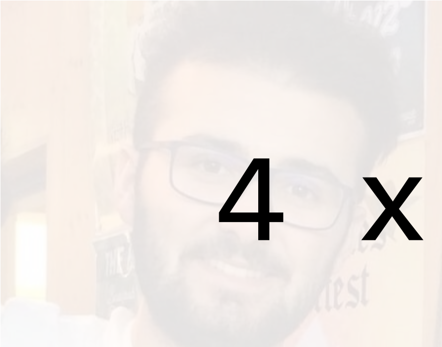
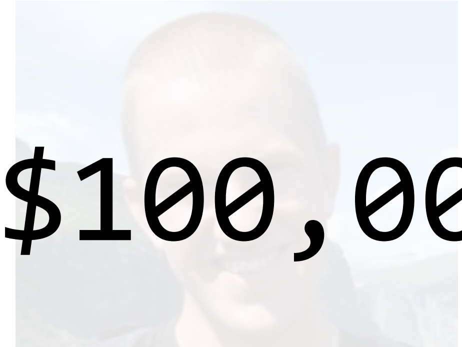
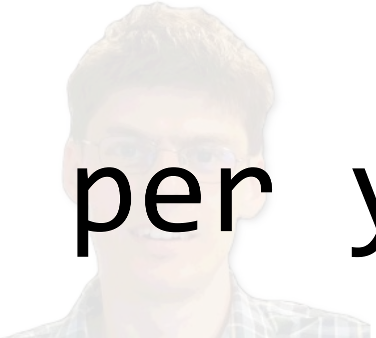
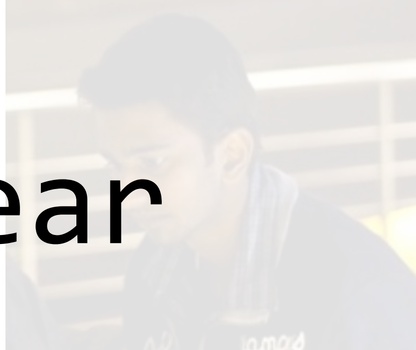
Cristiano Giuffrida

Andrei Tatar

Make lots of money

Terrace!

Ben Gras

Pietro Frigo

Dennis Andriesse

Lucian Cojocar

Radhesh Konoth


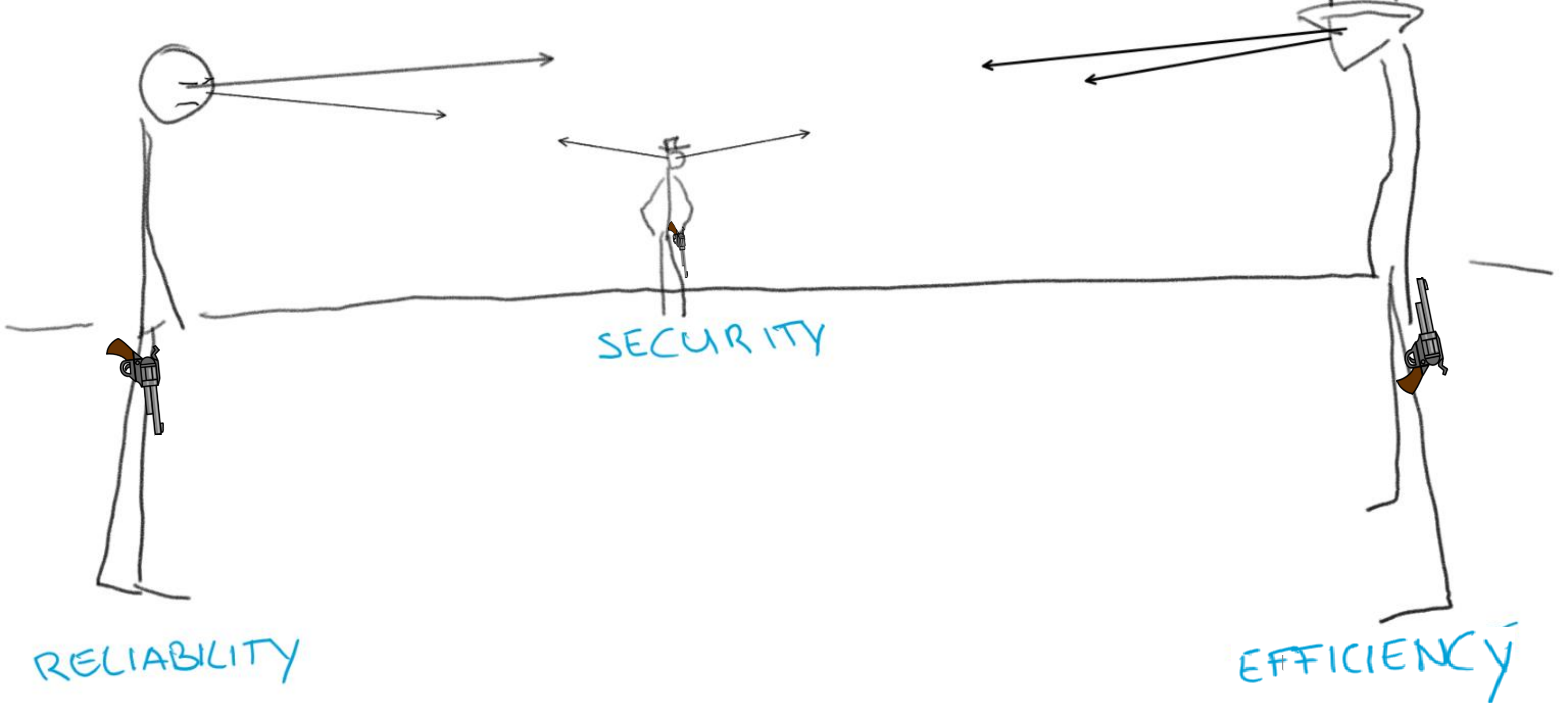
4 x $100,000 per year

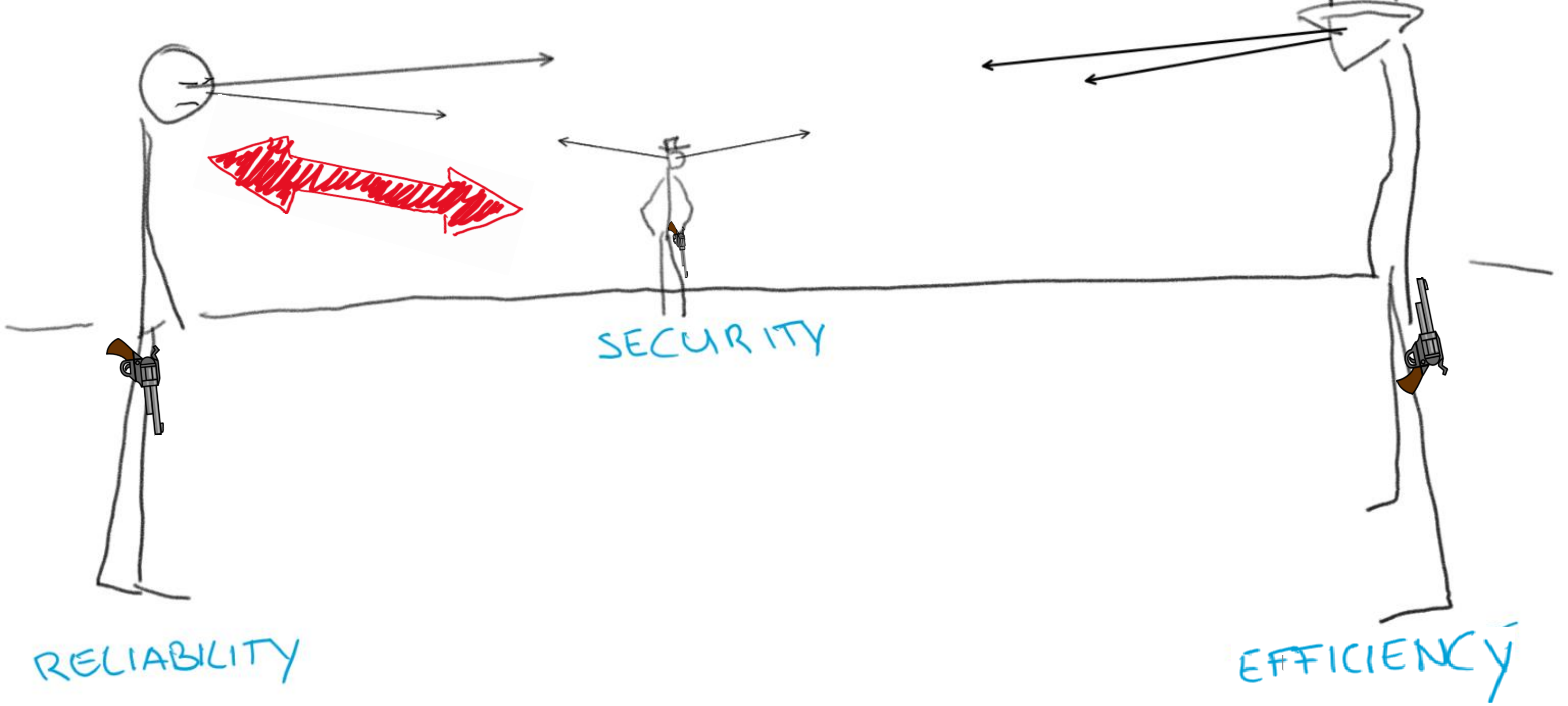
Three observations

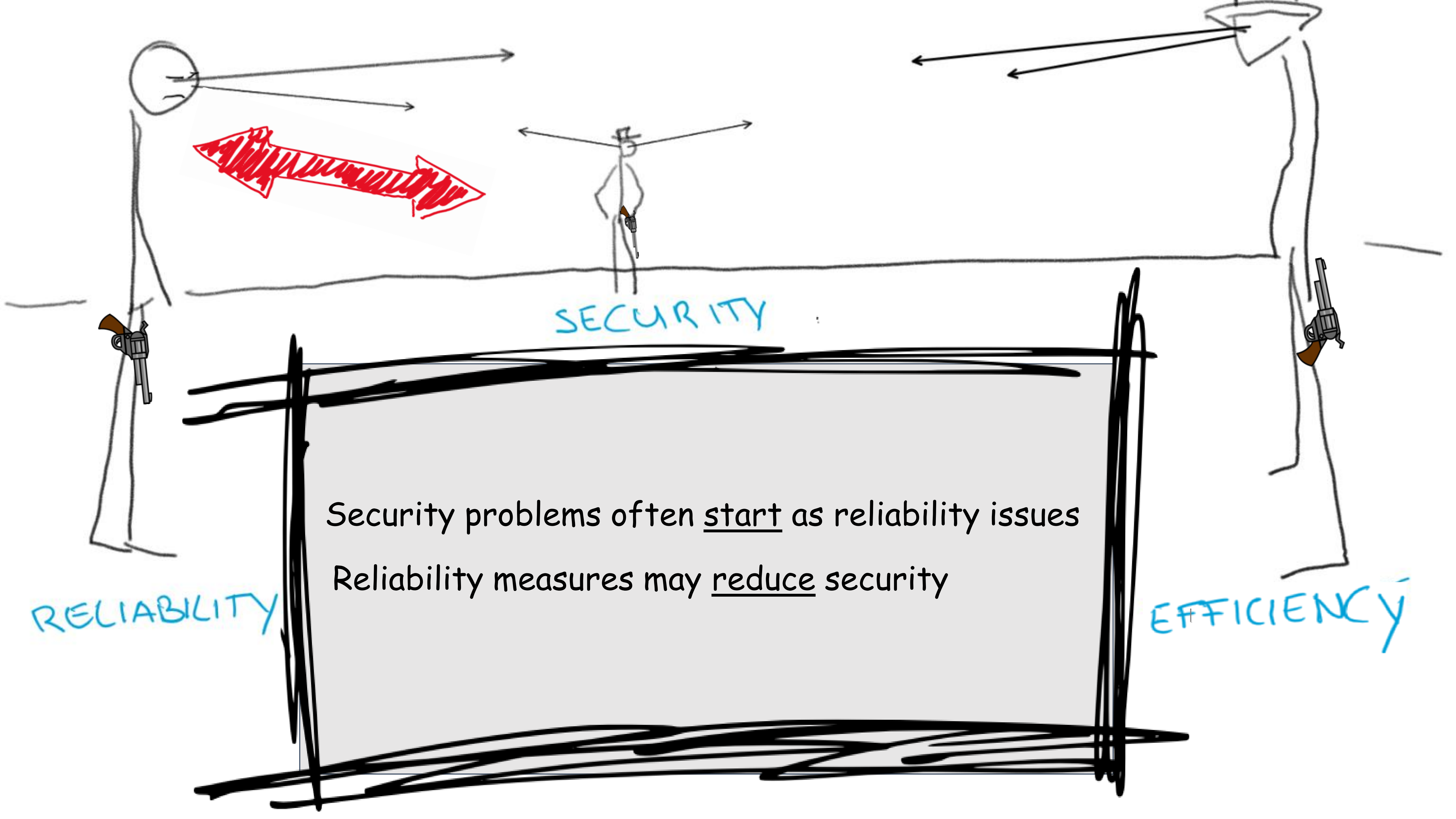
Two observations

Observation #1

The awkward relation between security, reliability and efficiency

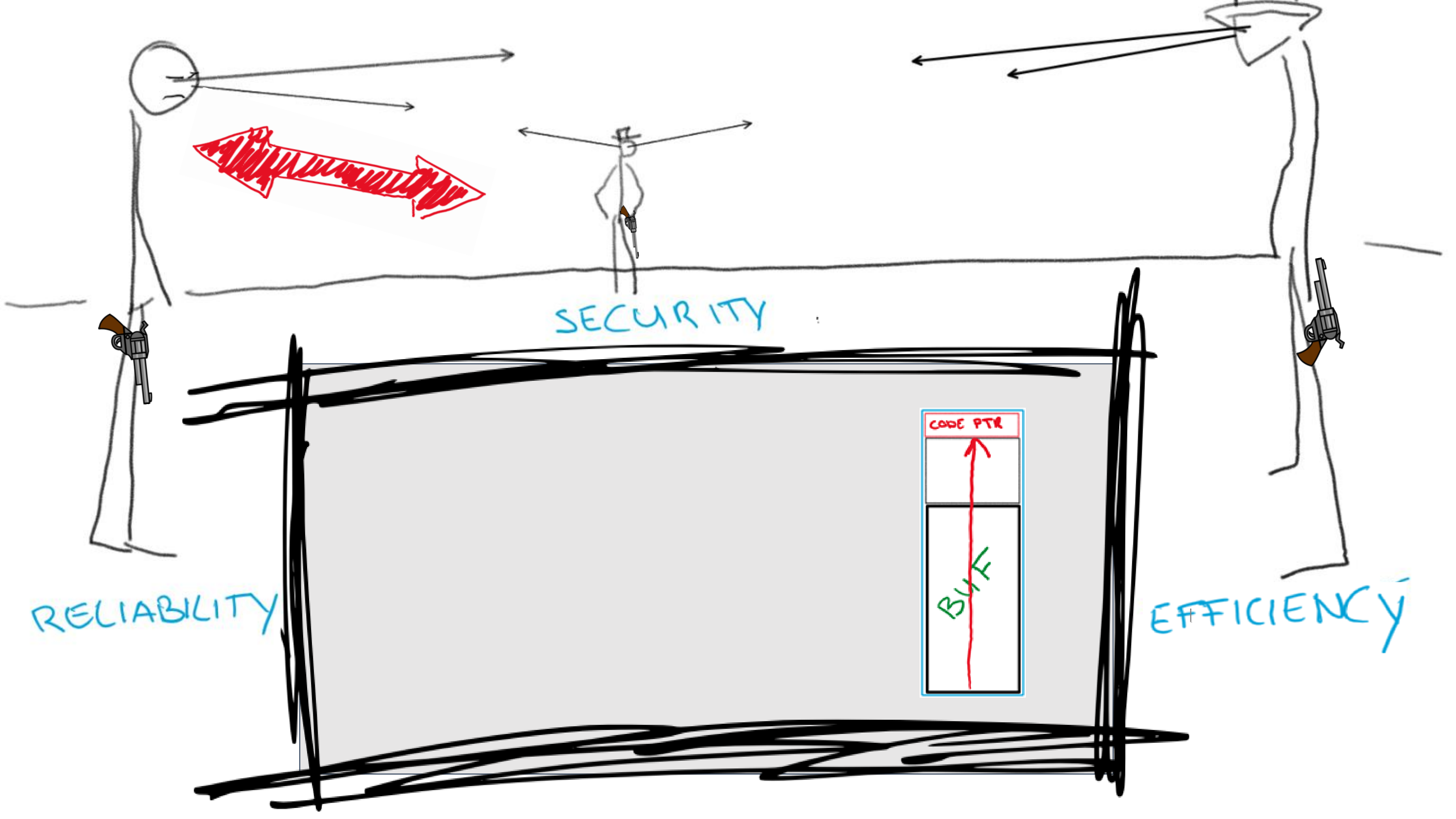

SECURITY
RELIABILITY
EFFICIENCY

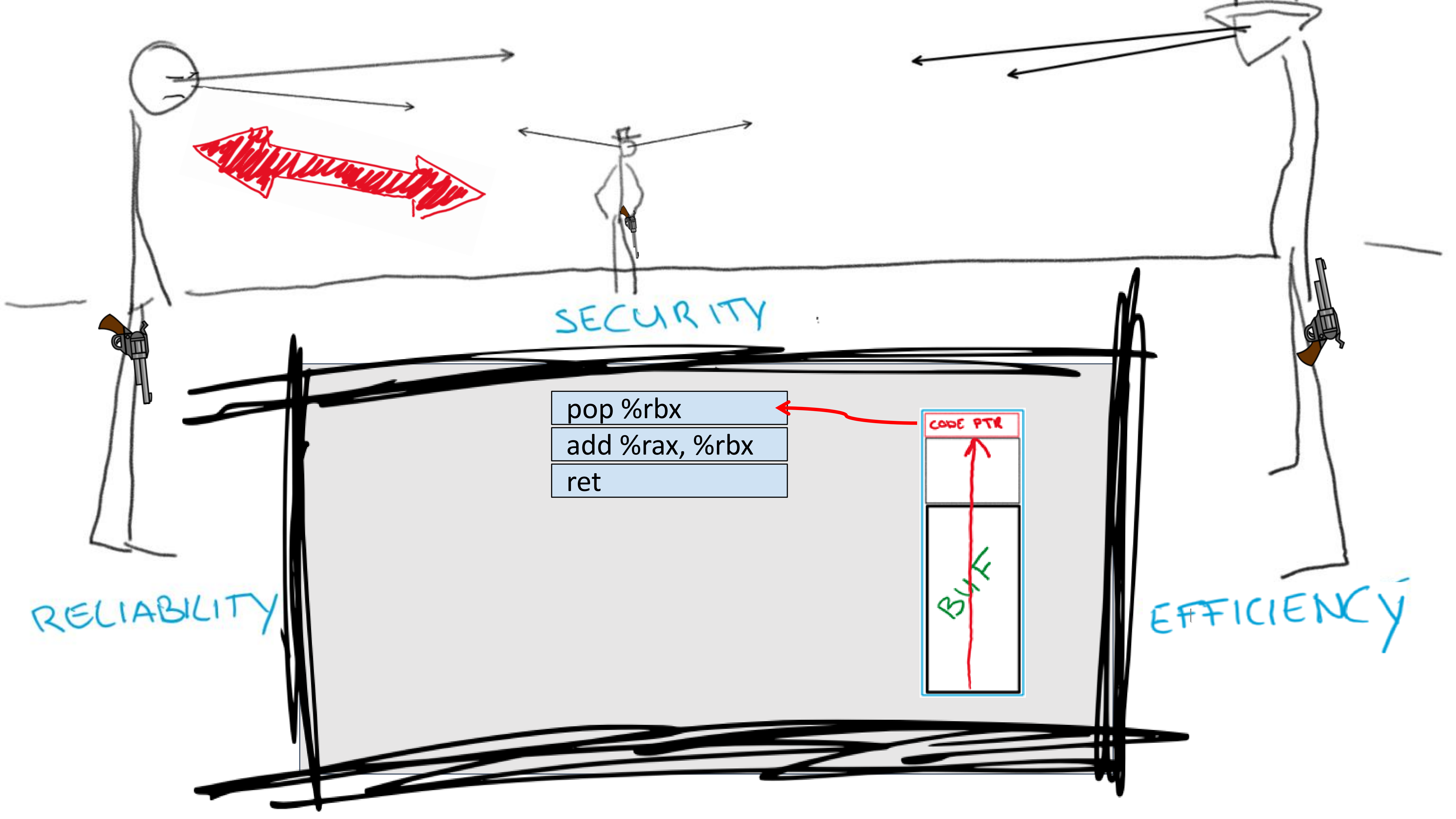

SECURITY
RELIABILITY
EFFICIENCY

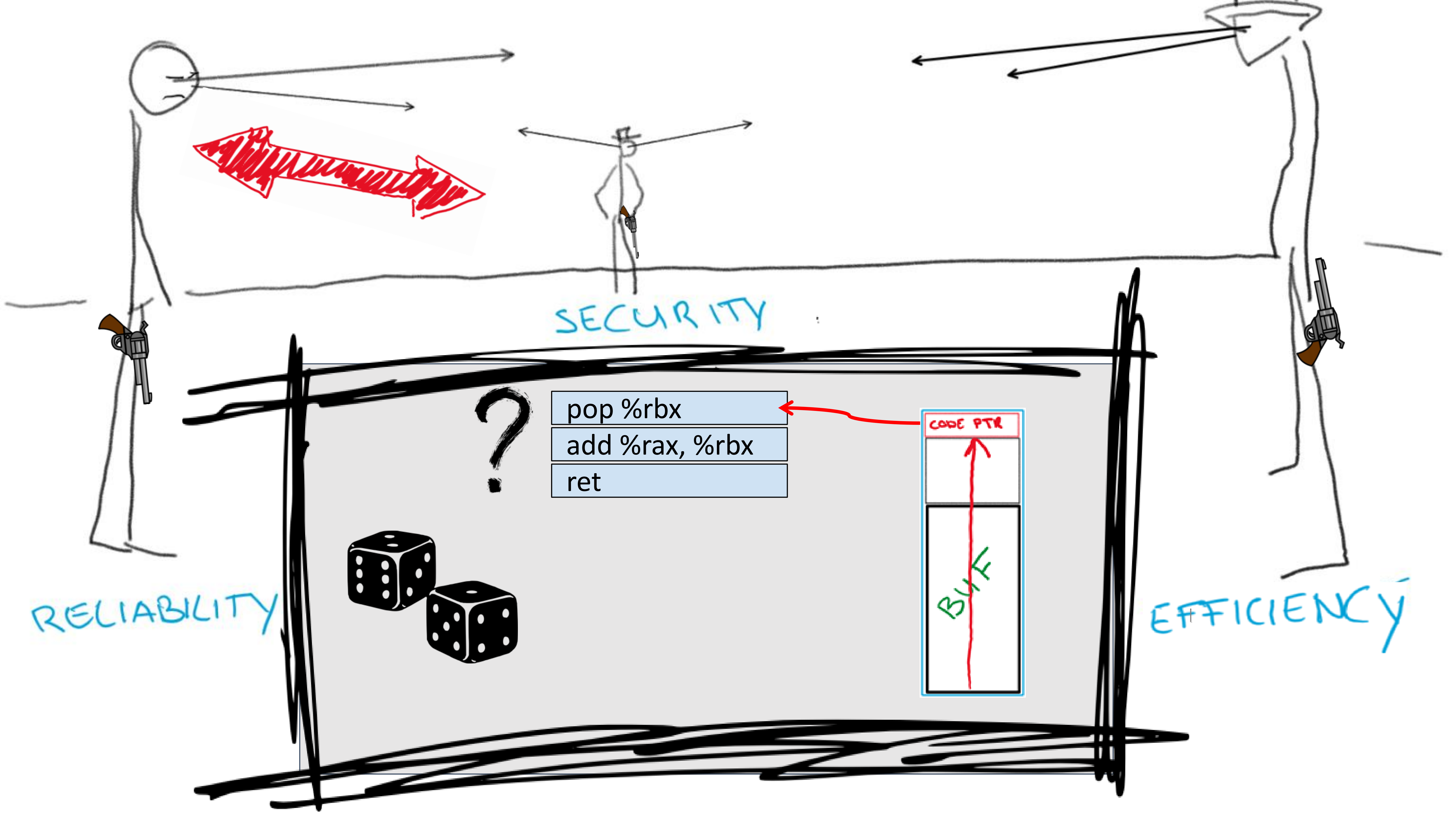


Security problems often start as reliability issues

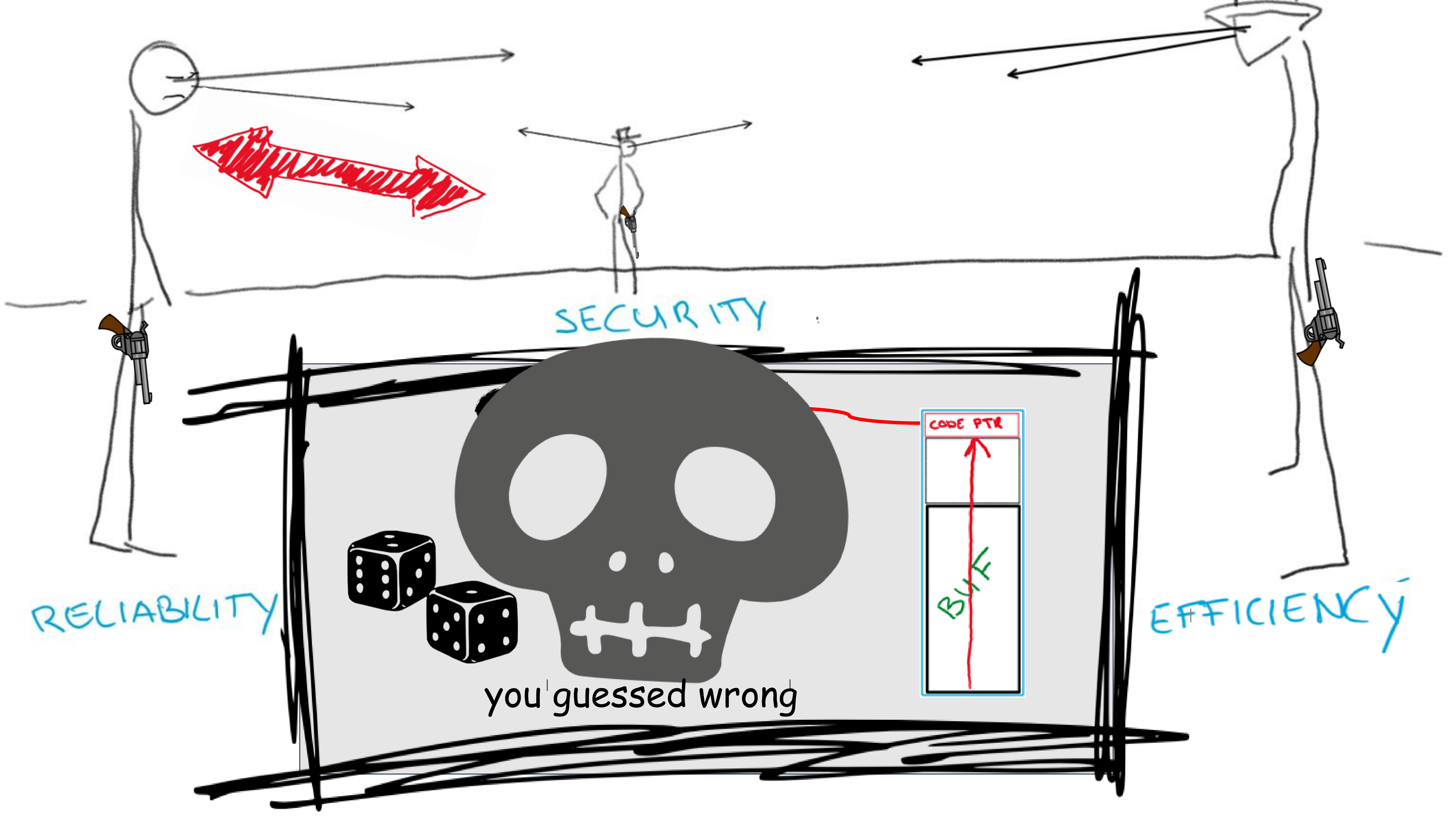
Reliability measures may reduce security


SECURITY
RELIABILITY
EFFICIENCY
CODE PTR
BUF


SECURITY
RELIABILITY
EFFICIENCY
pop %rbx
add %rax, %rbx
ret
CODE PTR
BUF

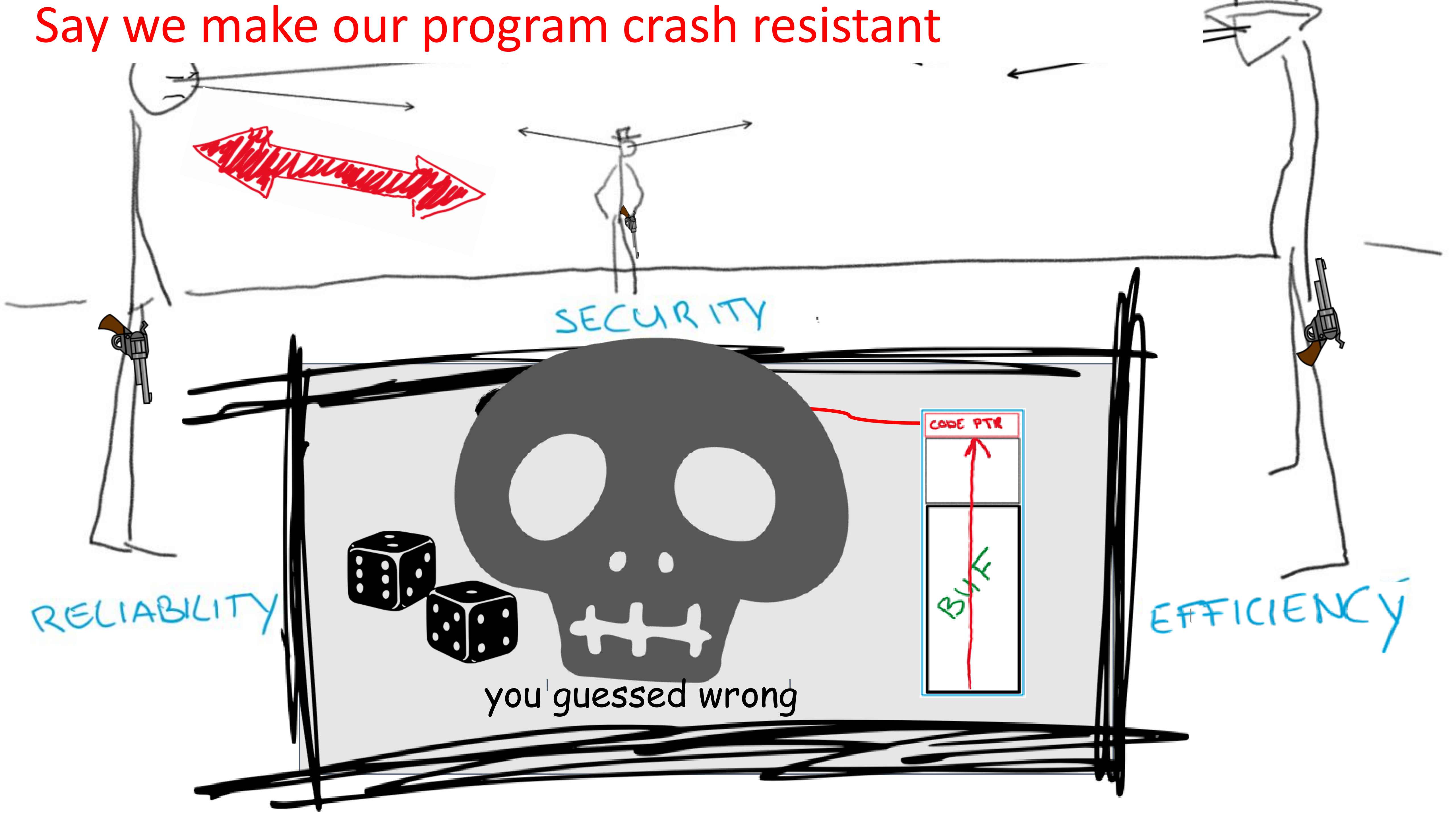


SECURITY
pop %rbx
add %rax, %rbx
ret
CODE PTR
BUF
RELIABILITY
EFFICIENCY

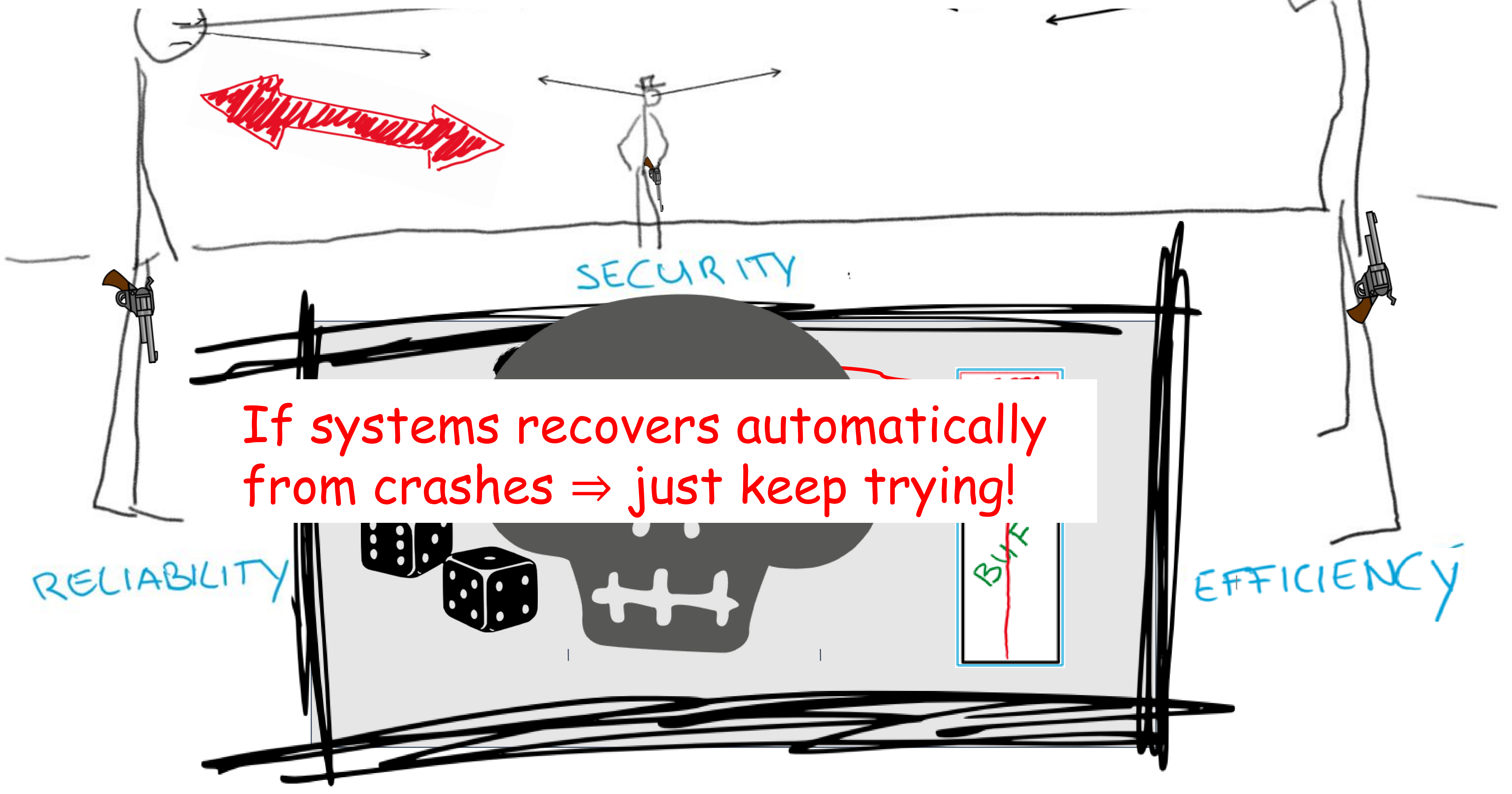


SECURITY
RELIABILITY
EFFICIENCY
CODE PTR
BUF
you guessed wrong

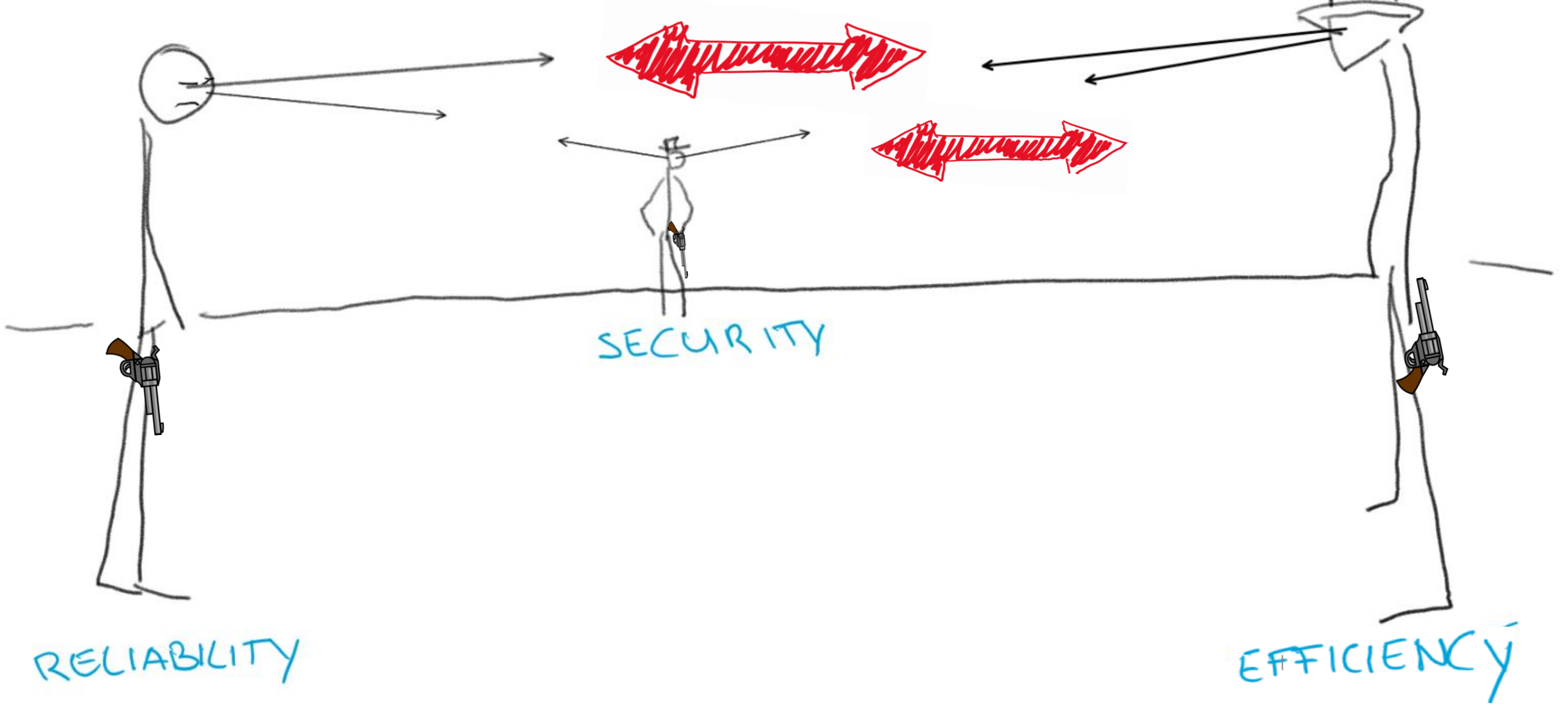
Say we make our program crash resistant

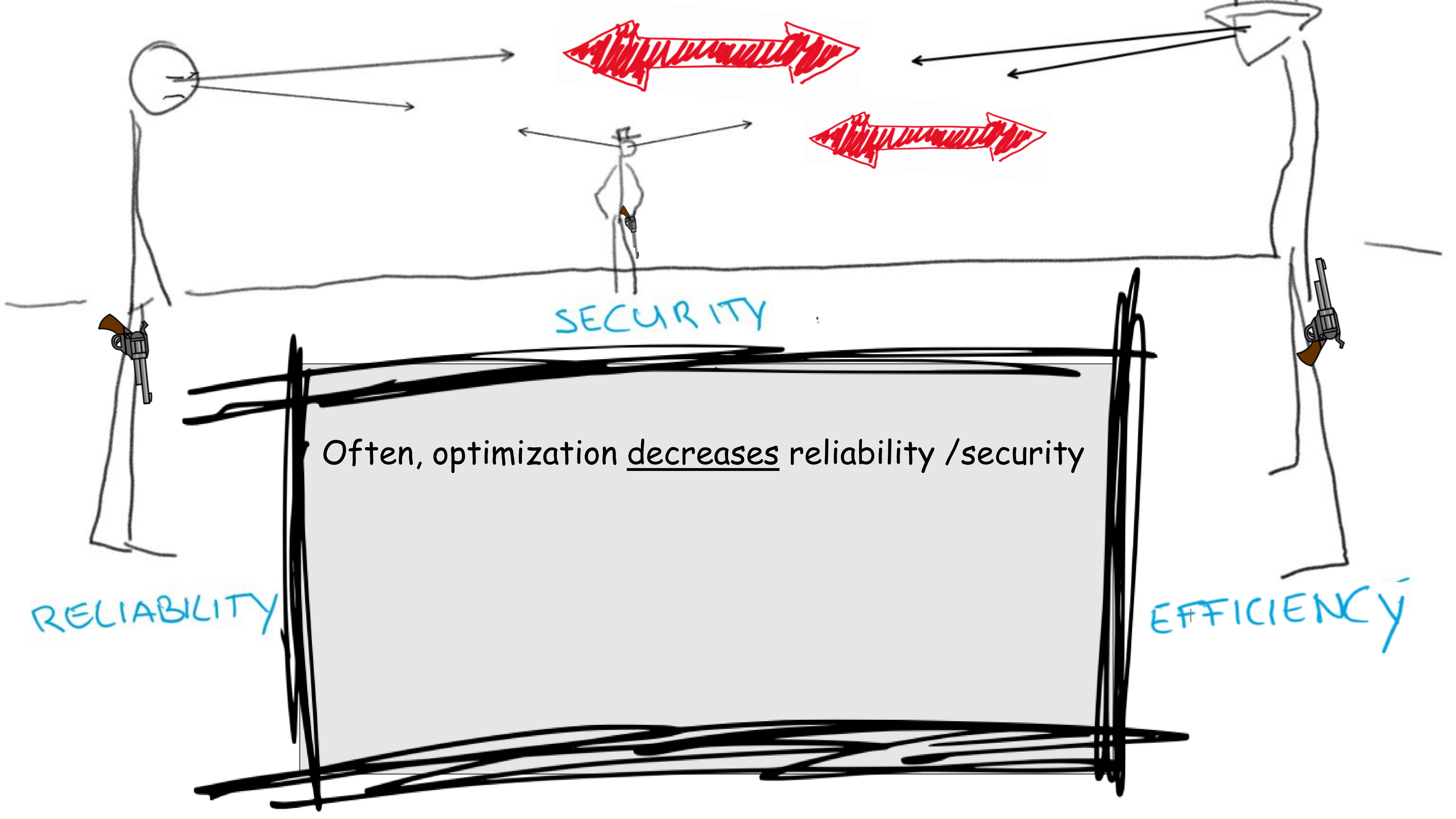


Say we make our program crash resistant

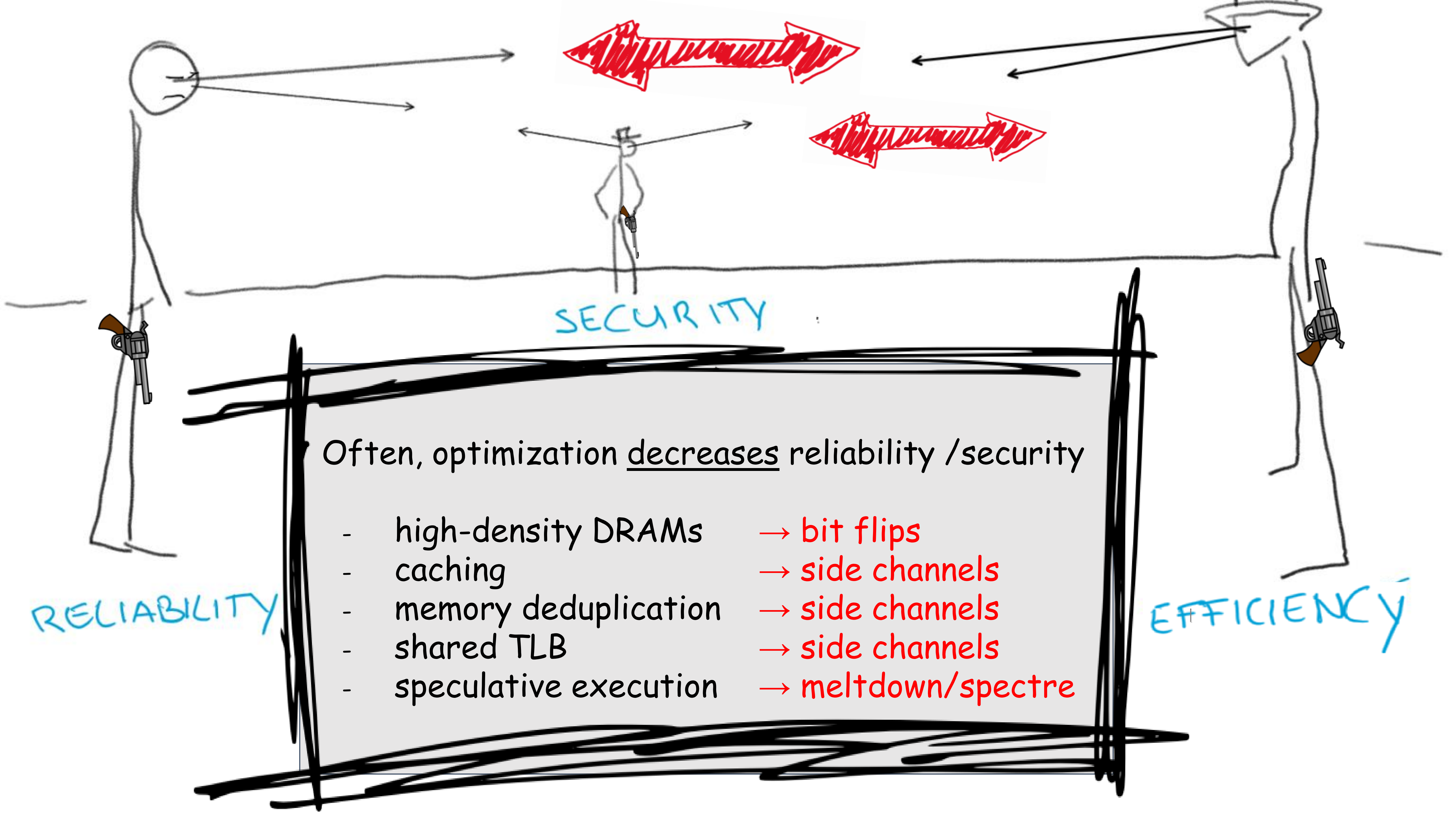


If systems recovers automatically from crashes ⇒ just keep trying!


SECURITY
RELIABILITY
EFFICIENCY



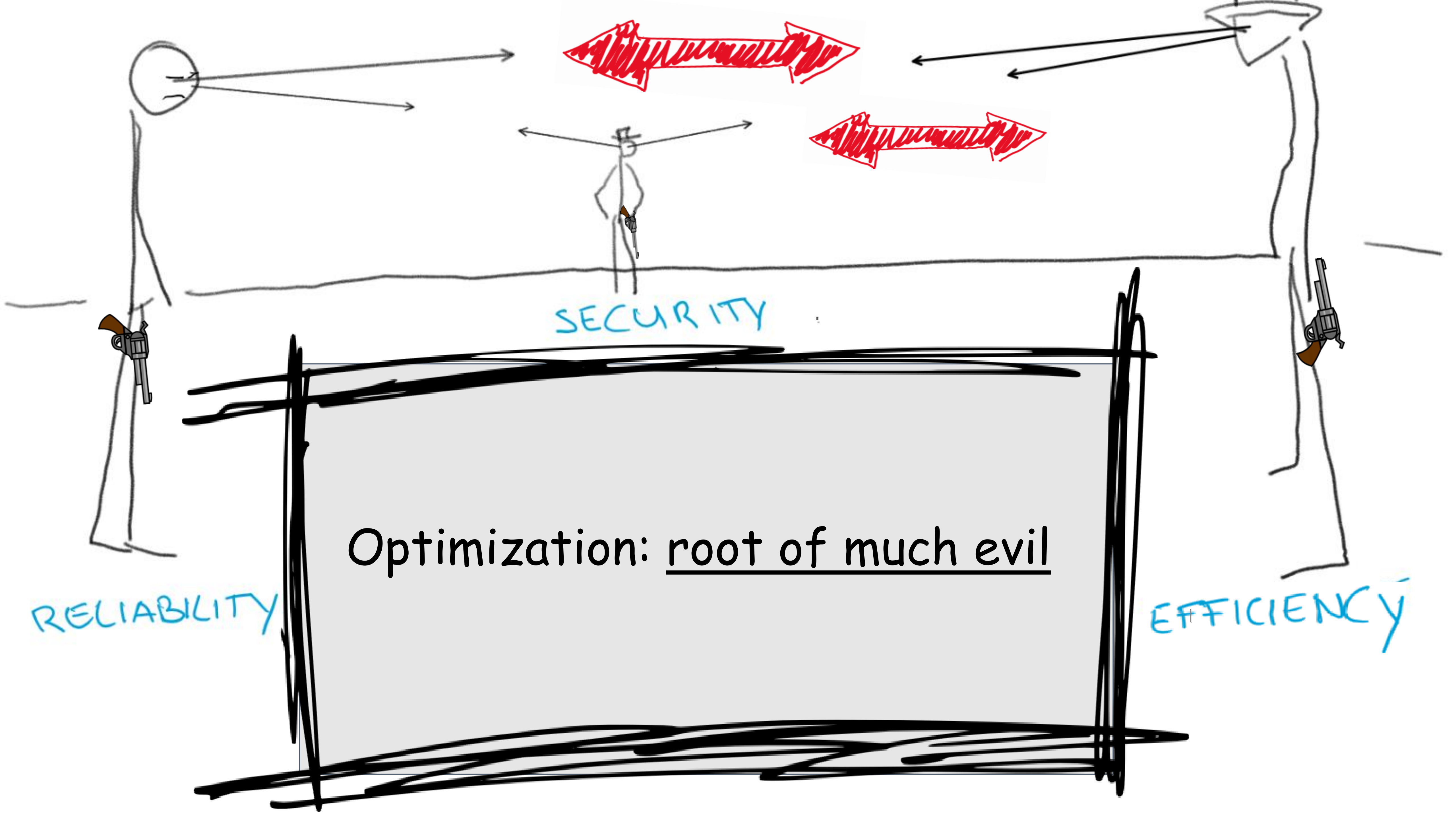
Often, optimization <u>decreases</u> reliability /security



Often, optimization decreases reliability /security

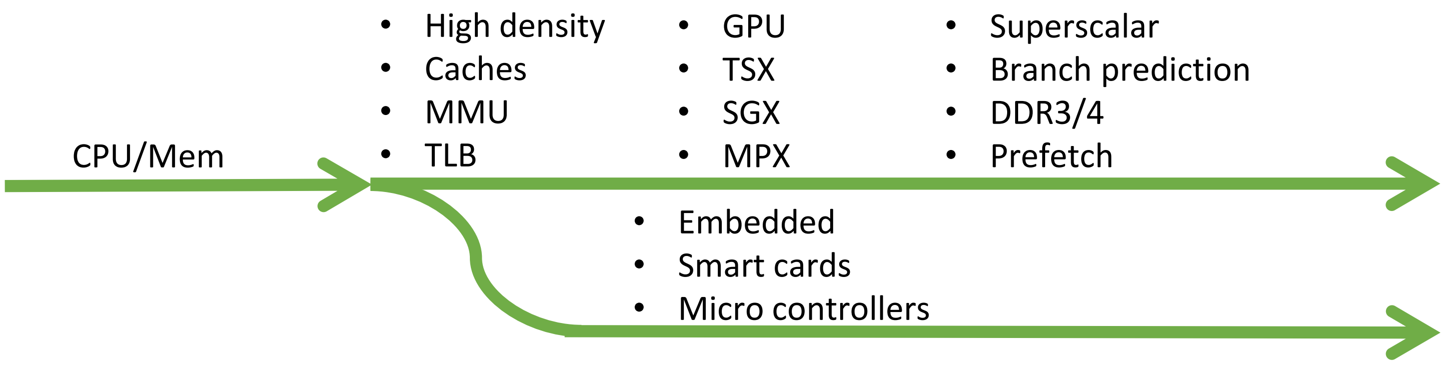
- high-density DRAMs → bit flips
- caching → side channels
- memory deduplication → side channels
- shared TLB → side channels
- speculative execution → meltdown/spectre

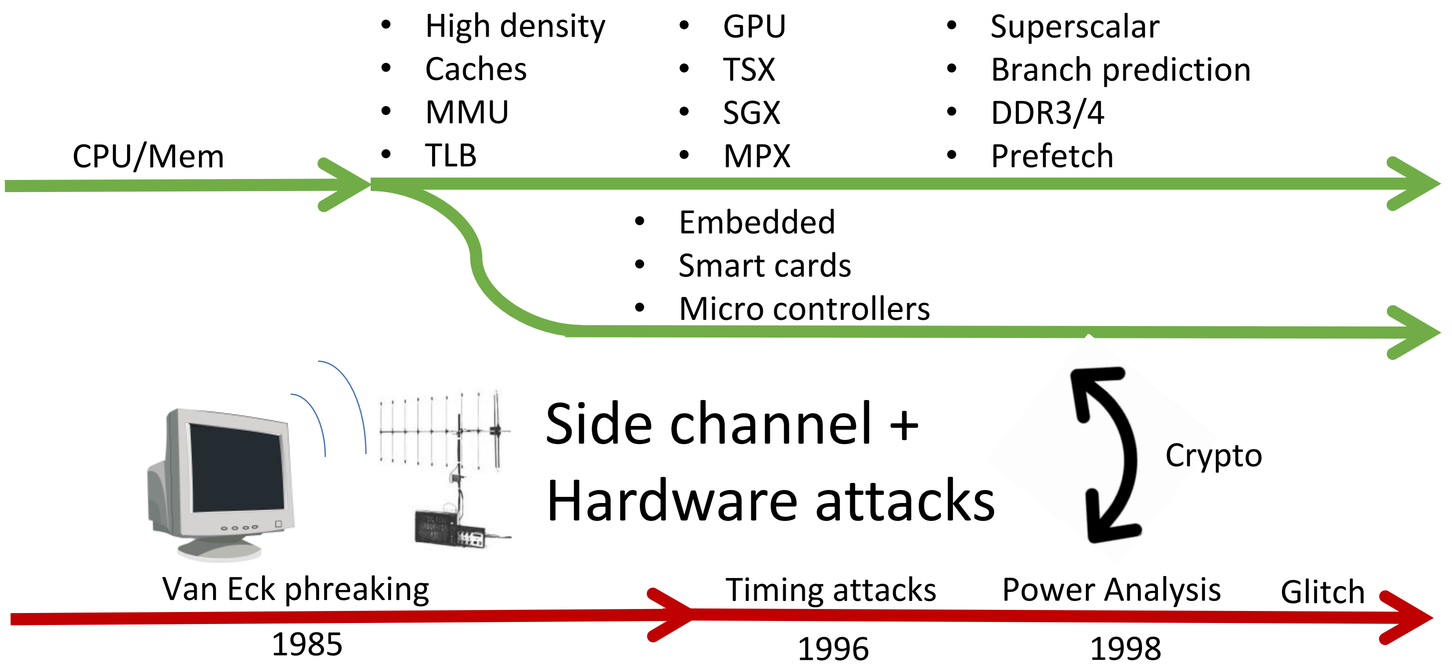
Optimization: root of much evil

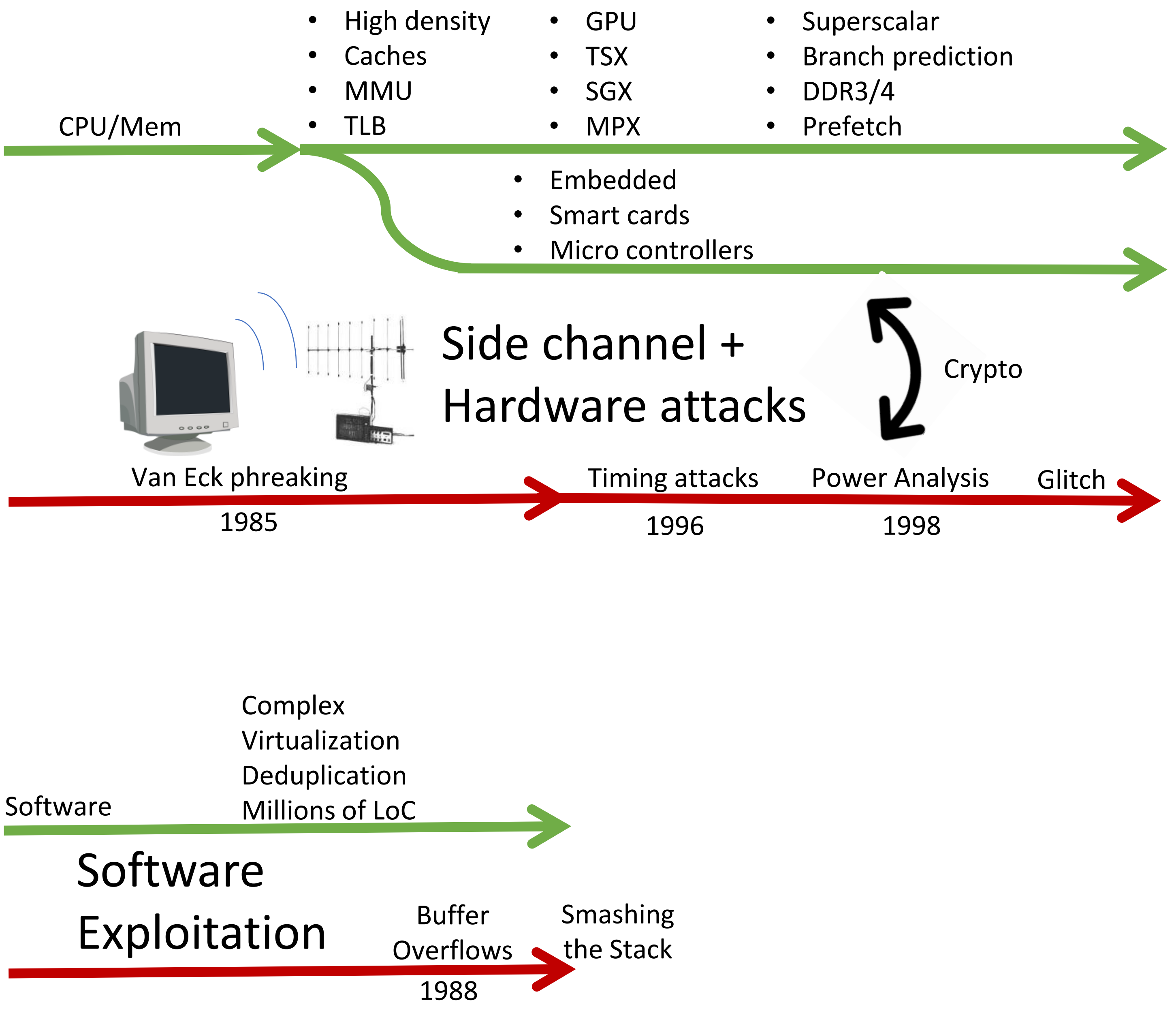


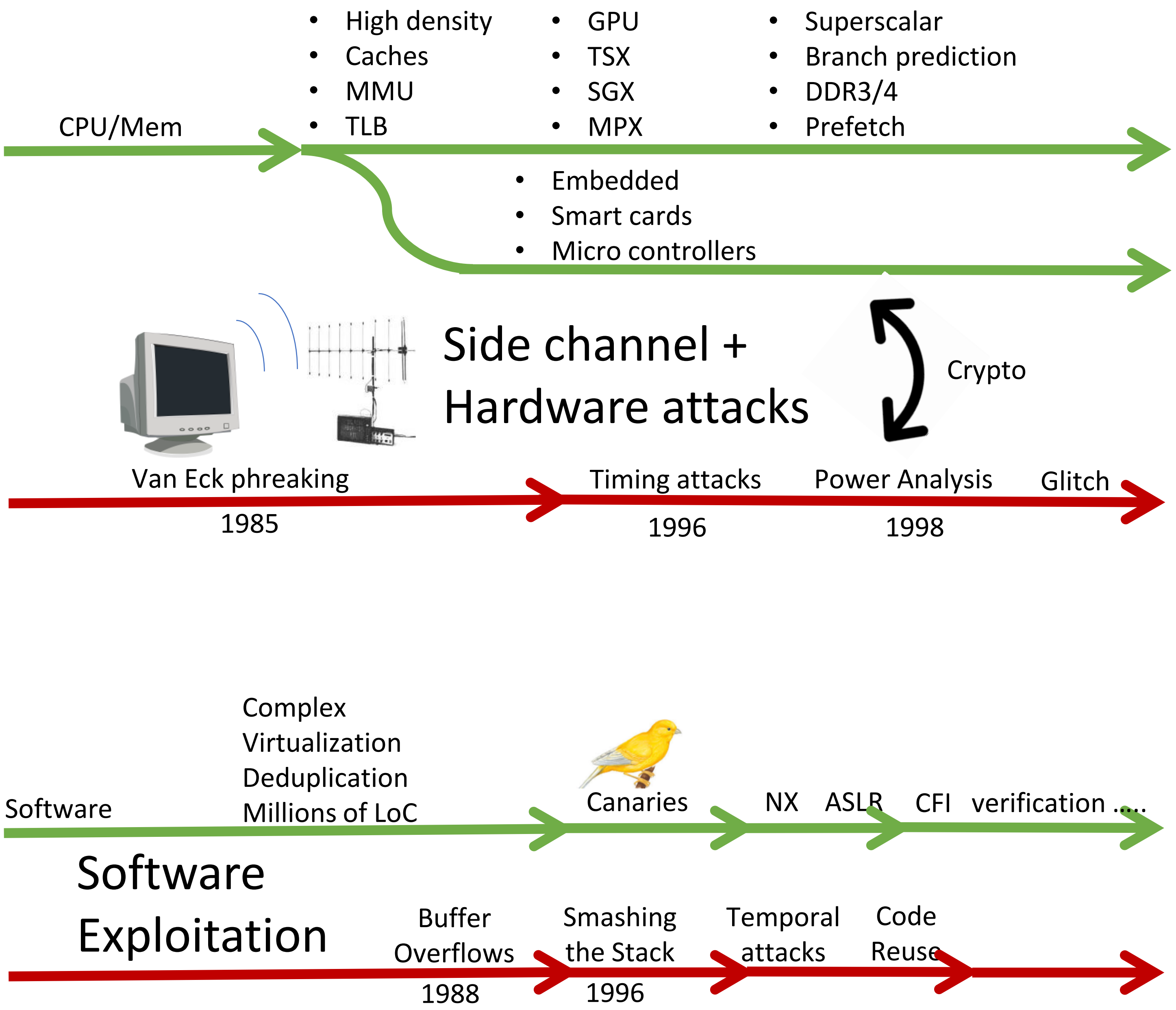
Observation #2

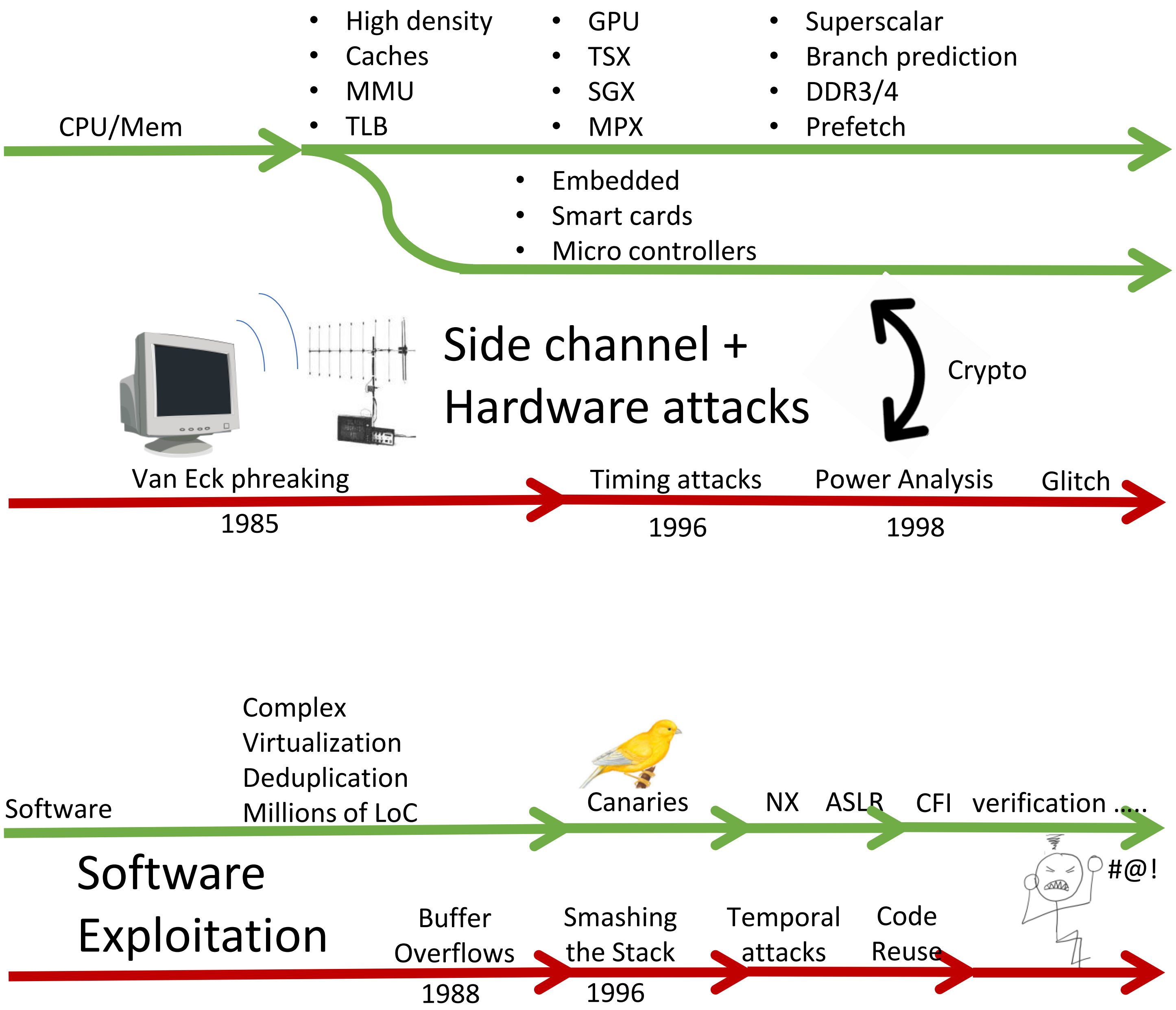
Physical attacks and software exploitation: colliding worlds


CPU/Mem
High density
Caches
MMU
TLB
GPU
TSX
SGX
MPX
Superscalar
Branch prediction
DDR3/4
Prefetch
Embedded
Smart cards
Micro controllers

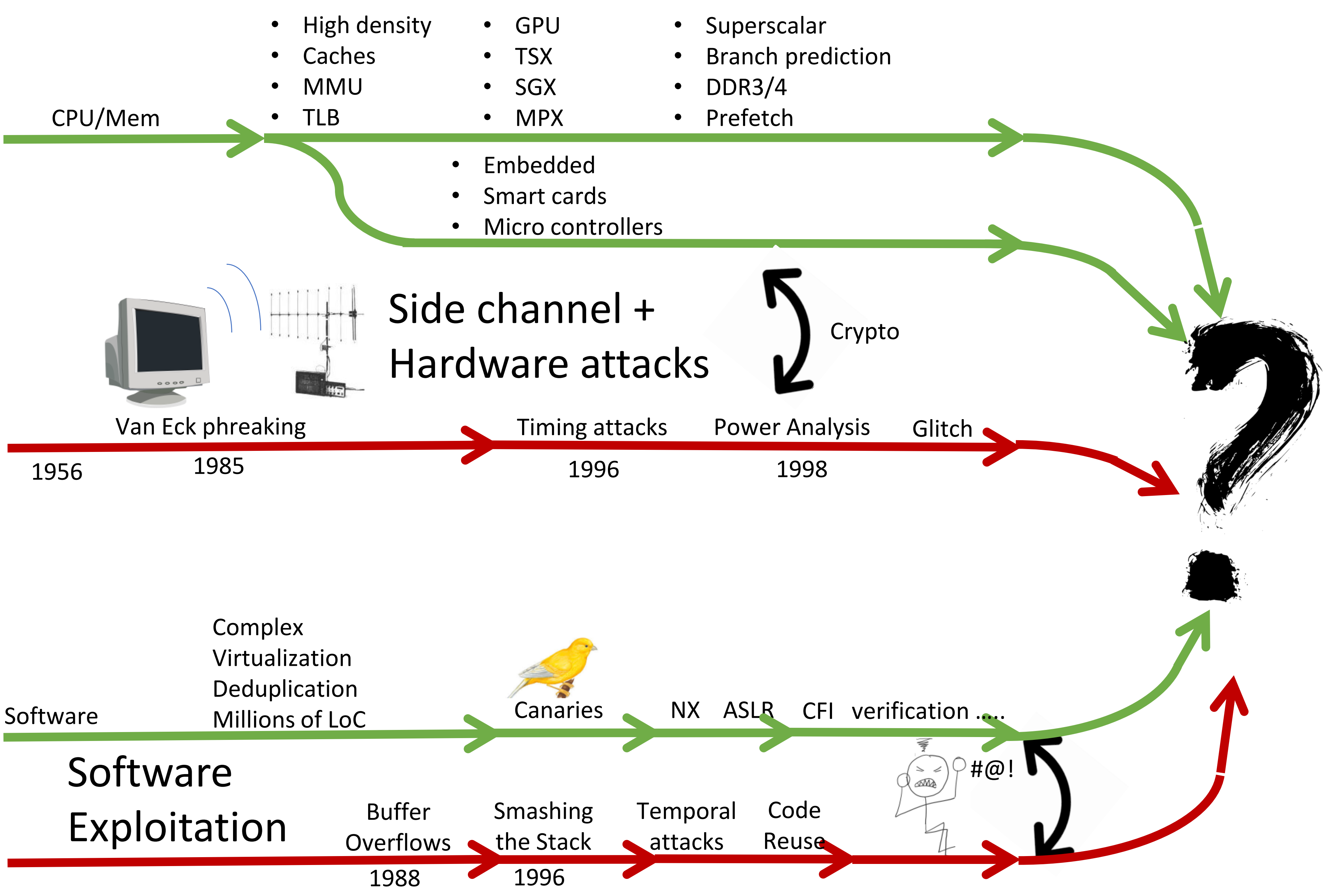

High density
Caches
MMU
TLB
GPU
TSX
SGX
MPX
Superscalar
Branch prediction
DDR3/4
Prefetch
CPU/Mem
Embedded
Smart cards
Micro controllers
Side channel +
Hardware attacks
Crypto
Van Eck phreaking
1985
Timing attacks
1996
Power Analysis
1998
Glitch


CPU/Mem
High density
Caches
MMU
TLB
GPU
TSX
SGX
MPX
Superscalar
Branch prediction
DDR3/4
Prefetch
Embedded
Smart cards
Micro controllers
Side channel +
Hardware attacks
Crypto
Van Eck phreaking
1985
Timing attacks
1996
Power Analysis
1998
Glitch
Software
Complex
Virtualization
Deduplication
Millions of LoC
Software
Exploitation
Buffer
Overflows
1988
Smashing
the Stack


CPU/Mem
High density
Caches
MMU
TLB
GPU
TSX
SGX
MPX
Superscalar
Branch prediction
DDR3/4
Prefetch
Embedded
Smart cards
Micro controllers
Side channel +
Hardware attacks
Crypto
Van Eck phreaking
1985
Timing attacks
1996
Power Analysis
1998
Glitch
Software
Complex
Virtualization
Deduplication
Millions of LoC
Canaries
NX
ASLR
CFI
verification .....
Software
Exploitation
Buffer
Overflows
1988
Smashing
the Stack
1996
Temporal
attacks
Code
Reuse



CPU/Mem
High density
Caches
MMU
TLB
GPU
TSX
SGX
MPX
Superscalar
Branch prediction
DDR3/4
Prefetch
Embedded
Smart cards
Micro controllers
Side channel +
Hardware attacks
Crypto
Van Eck phreaking
1985
Timing attacks
1996
Power Analysis
1998
Glitch
Complex
Virtualization
Deduplication
Millions of LoC
Software
Canaries
NX
ASLR
CFI
verification .....
#@!
Software
Exploitation
Buffer
Overflows
1988
Smashing
the Stack
1996
Temporal
attacks
Code
Reuse



CPU/Mem
High density
Caches
MMU
TLB
GPU
TSX
SGX
MPX
Superscalar
Branch prediction
DDR3/4
Prefetch
Embedded
Smart cards
Micro controllers
Side channel +
Hardware attacks
Crypto
Van Eck phreaking
Timing attacks
Power Analysis
Glitch
1956
1985
1996
1998
Complex
Virtualization
Deduplication
Millions of LoC
Software
Canaries
NX
ASLR
CFI
verification .....
#@!
Software
Exploitation
Buffer
Overflows
Smashing
the Stack
Temporal
attacks
Code
Reuse
1988
1996


WHY WOULD I CARE?!

2010

Security problems are caused by

- **Software bugs**, and
- **Configuration bugs**

2018

Even if the software is perfect

- and well-configured

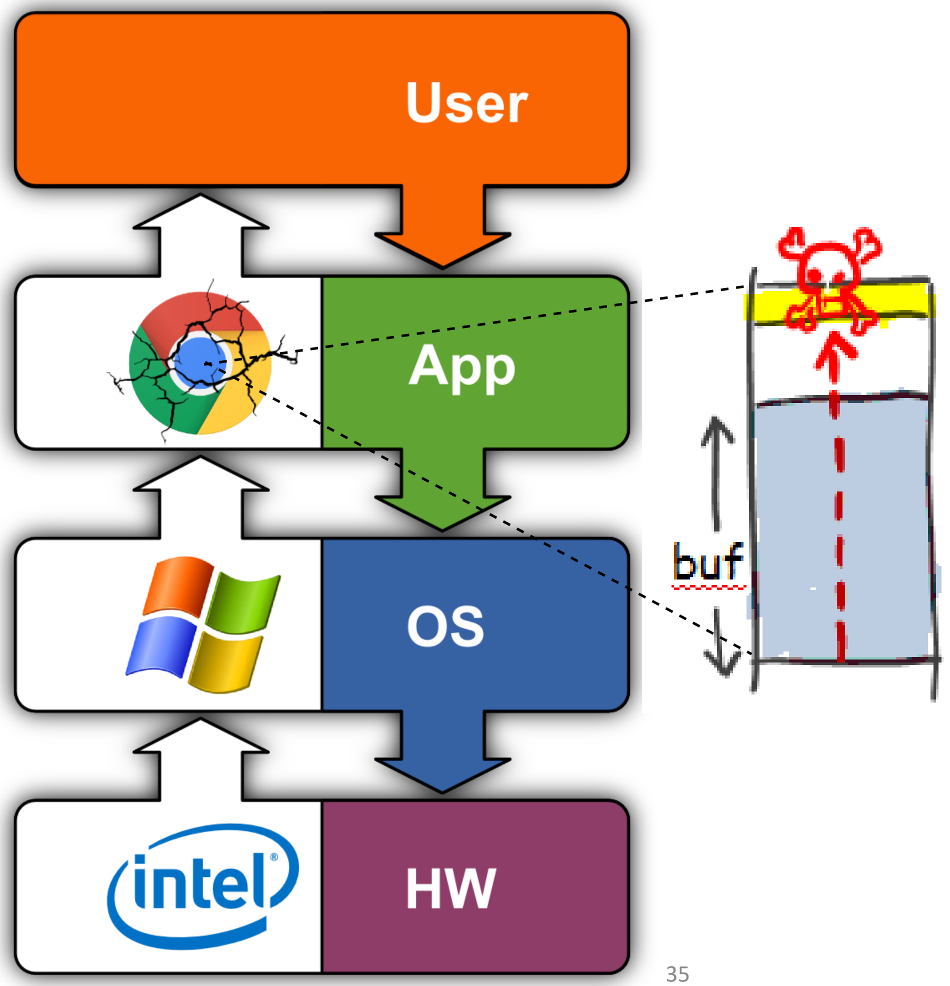
it is **still vulnerable!**

What does that mean for

formally verified systems?

Software Exploitation:

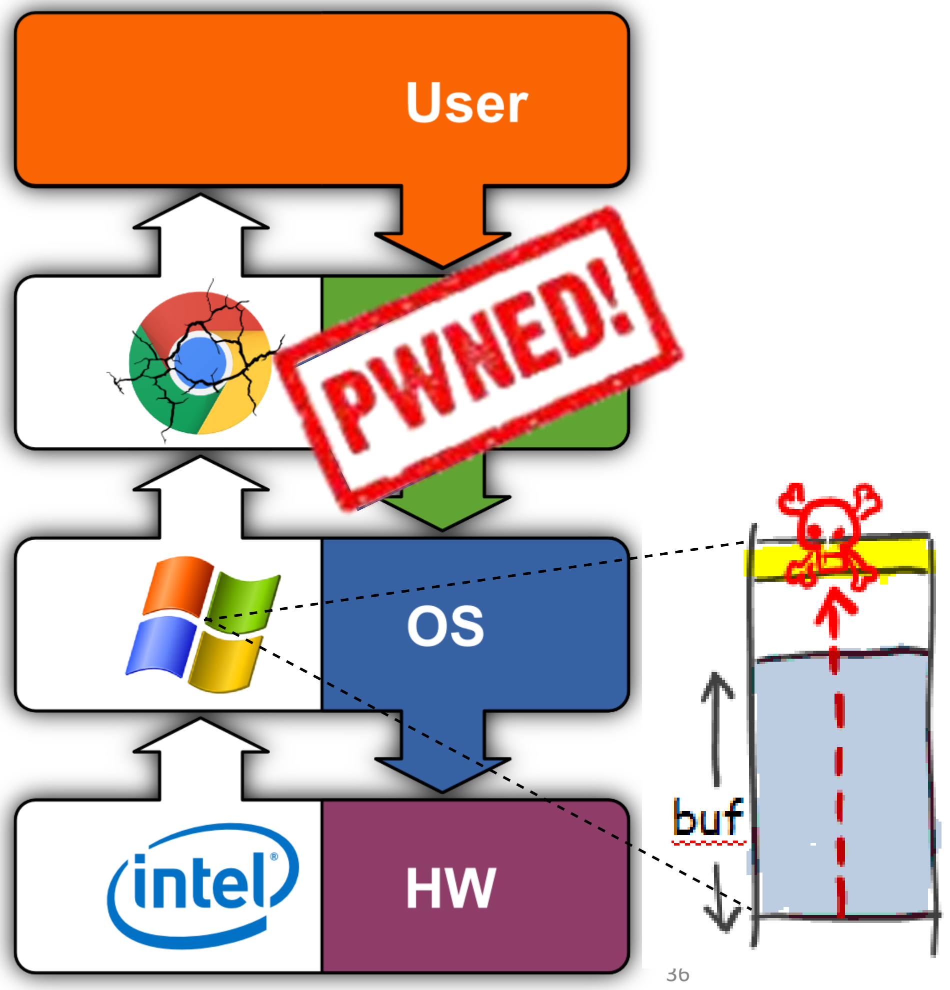
2010



Software Exploitation:

2010

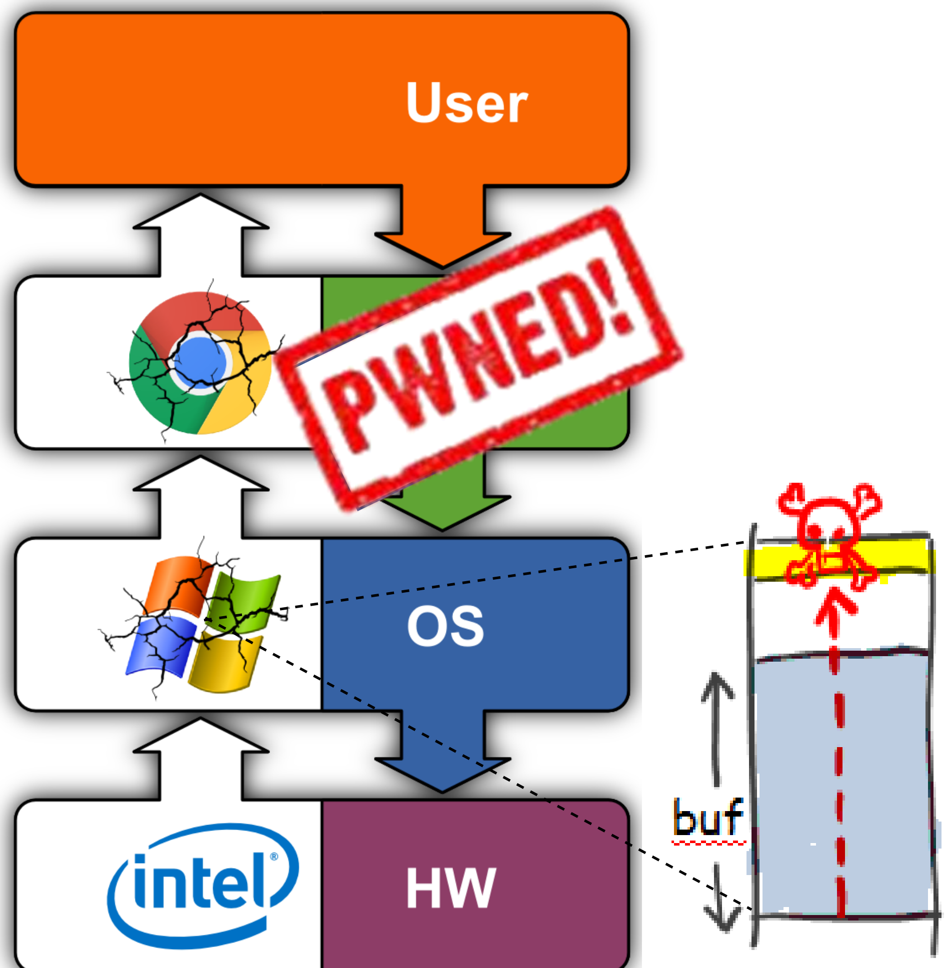
Attacker Exploits Vulnerable Software



Software Exploitation:

2010

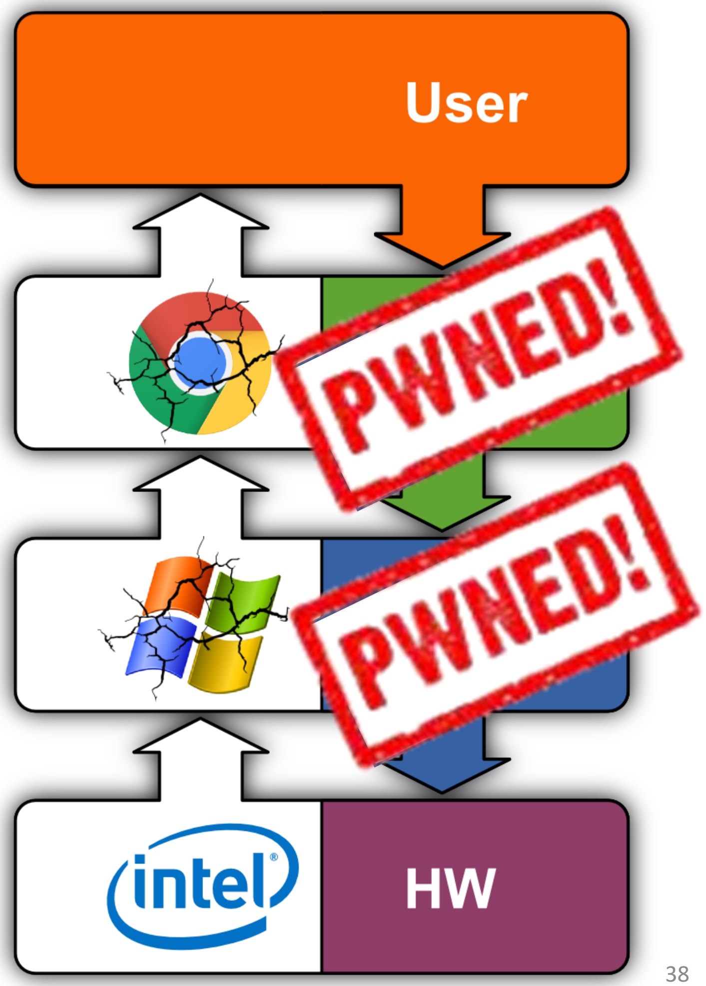
Attacker Exploits Vulnerable Software



Software Exploitation:

2010

Attacker Exploits Vulnerable Software

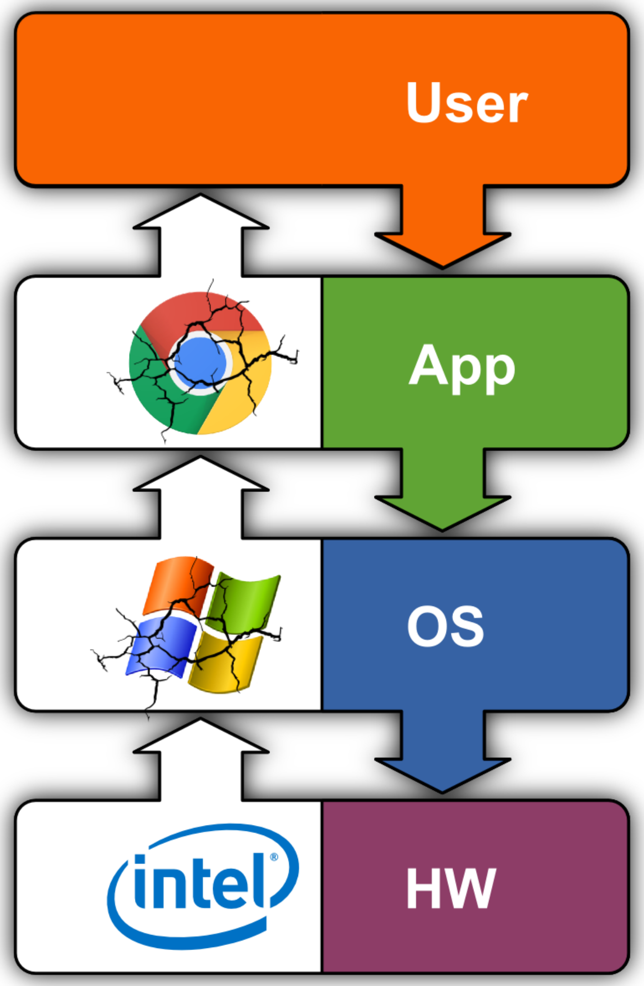


Software Exploitation:

2010

Exploits difficult

- Hardening

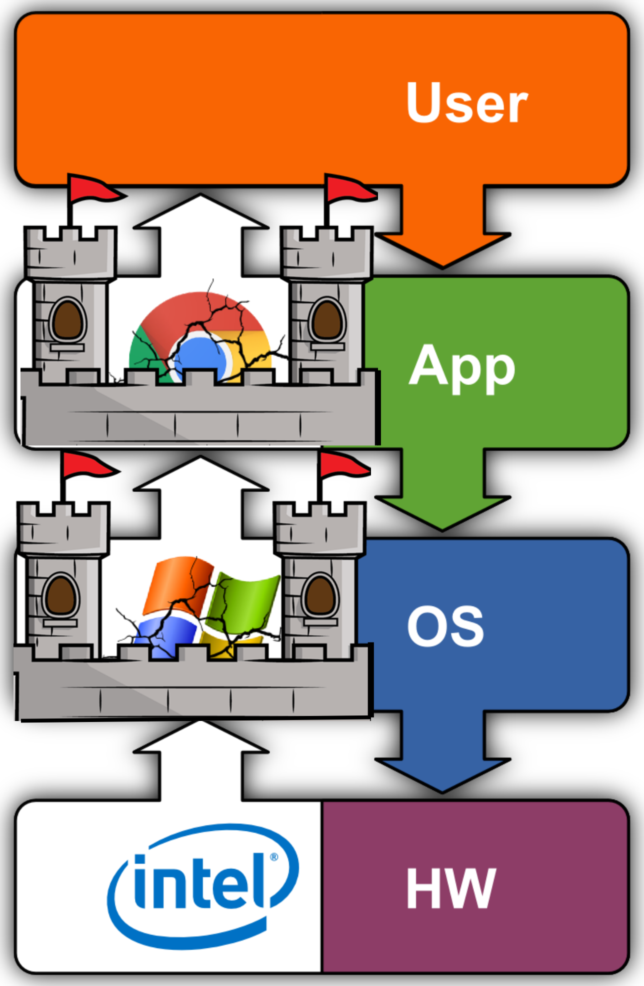


Software Exploitation:

2010

Exploits difficult

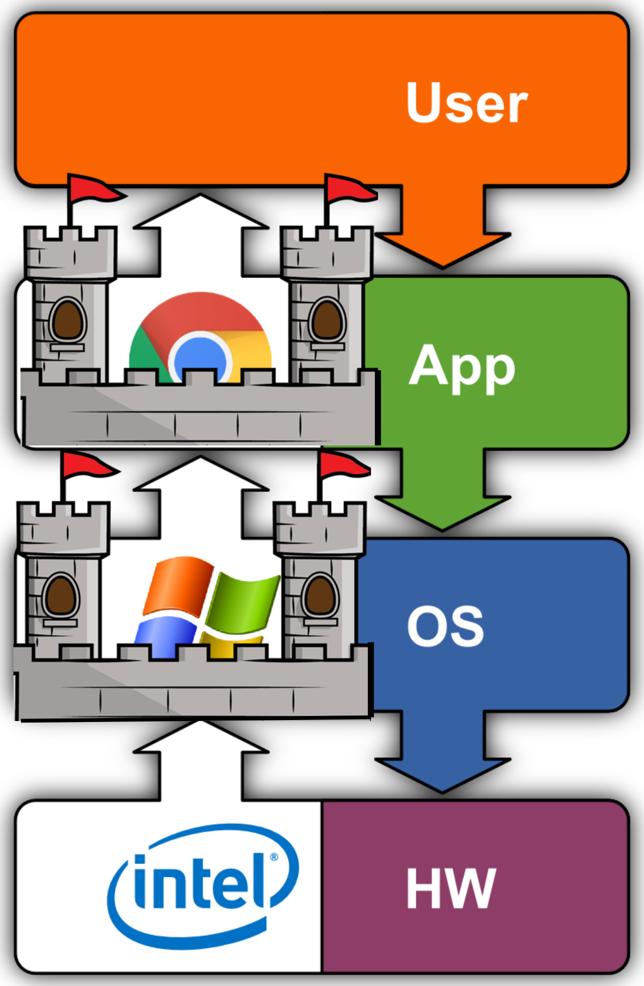
- Hardening



Software Exploitation:

2010

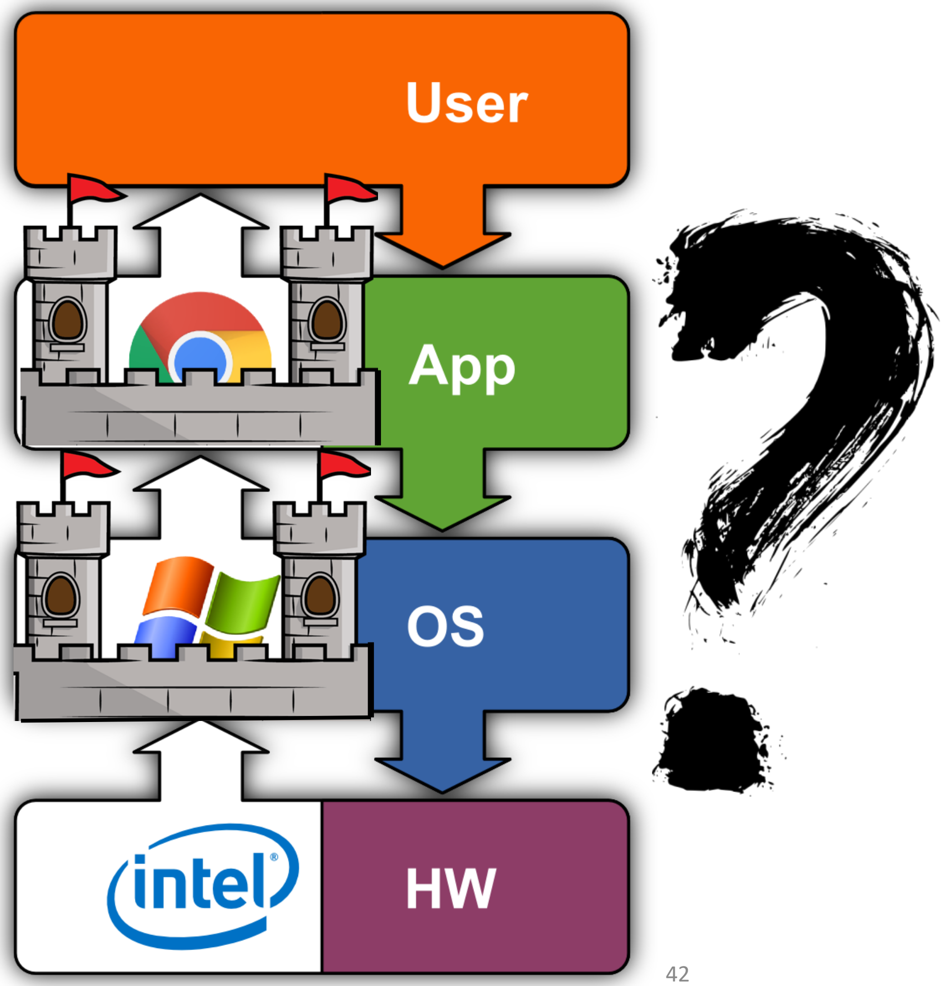
Exploits difficult
- Hardening
- Verification



Software Exploitation:

2018

How to Find Memory R/W Primitives?



Sharing

is efficient

Sharing

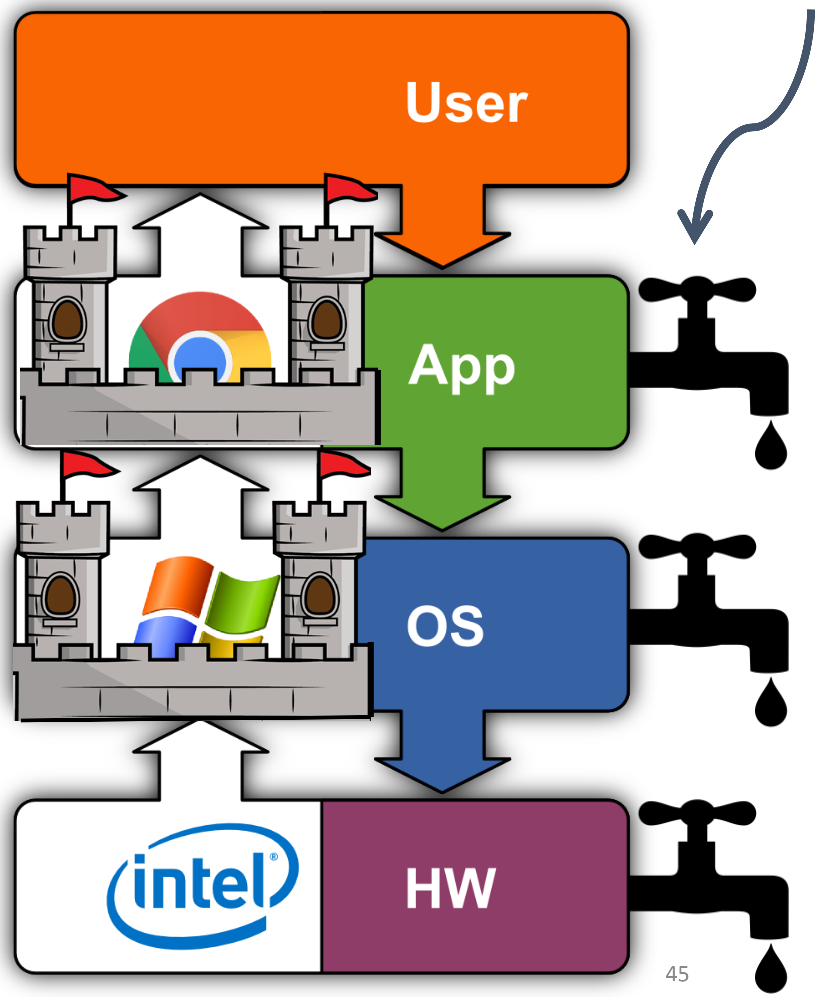
is not caring

Software Exploitation:

2018

Memory R: Hw/Sw Side Channels

memory read: side channels



Glitches

from software

2018

Memory W?

Glitches

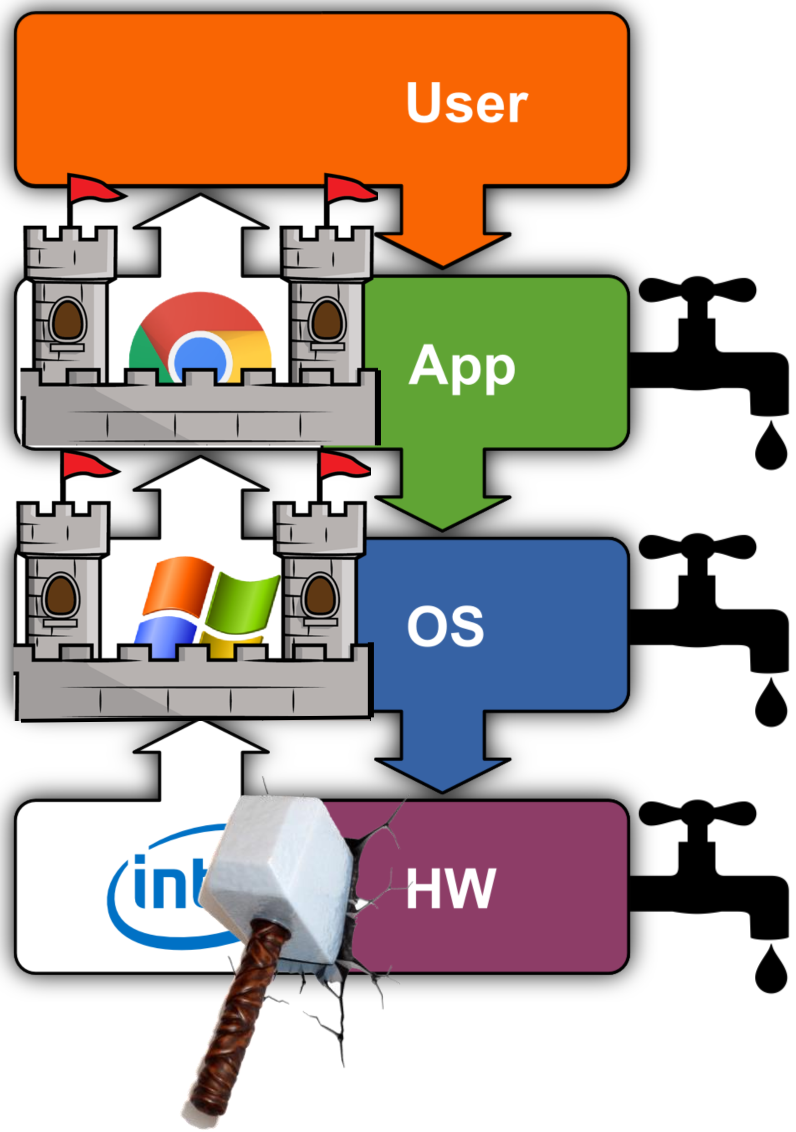
from software

memory read:
side channels

Software Exploitation:

2018

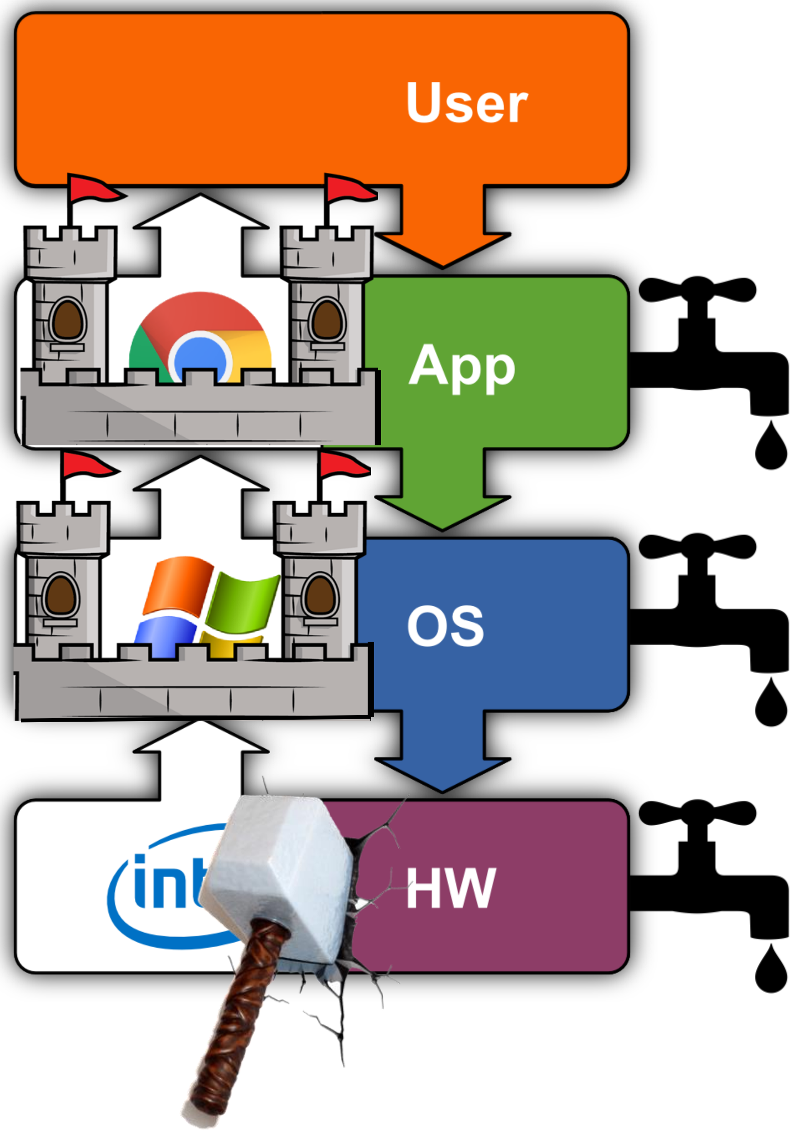
Memory W: Hardware Glitches



Software Exploitation:

2018

Goal: Controllable from Software

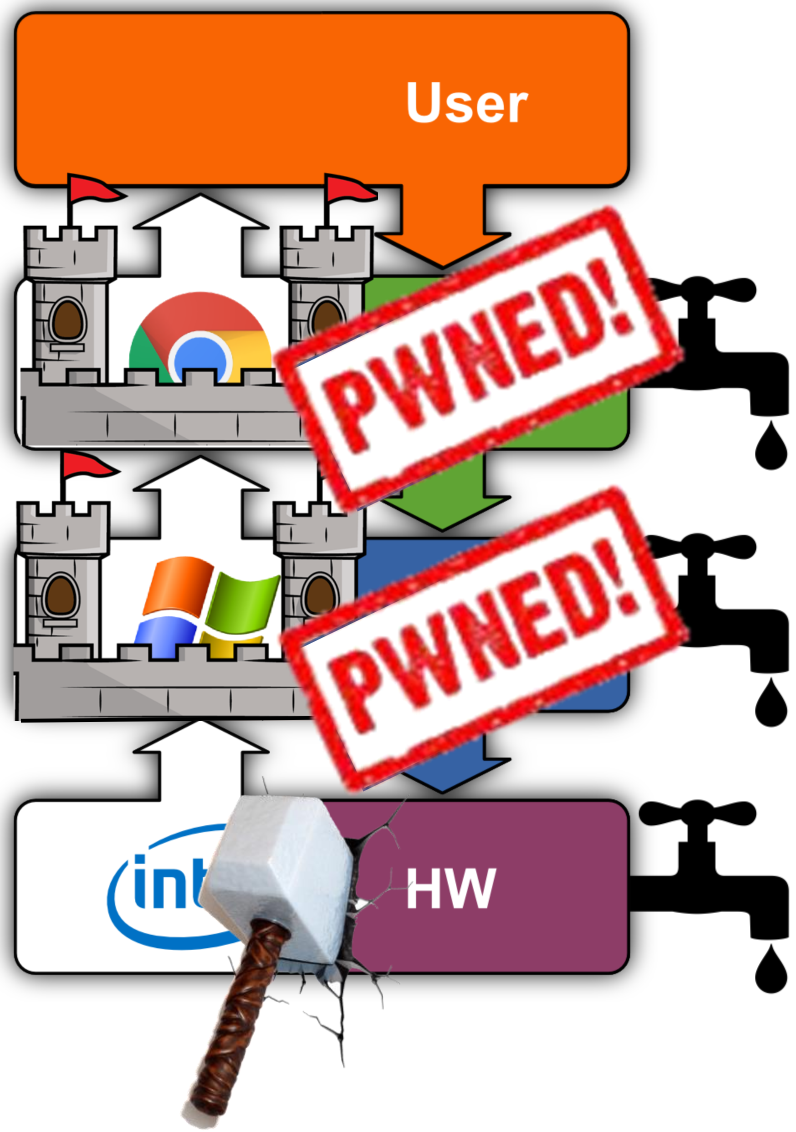


Software Exploitation:

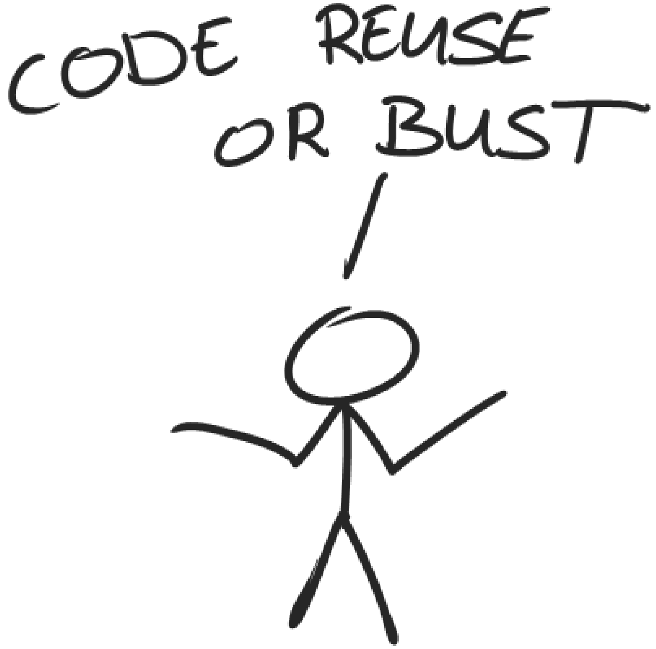
2018

Memory RW:

Back to reliable Exploits!



Past 10 years

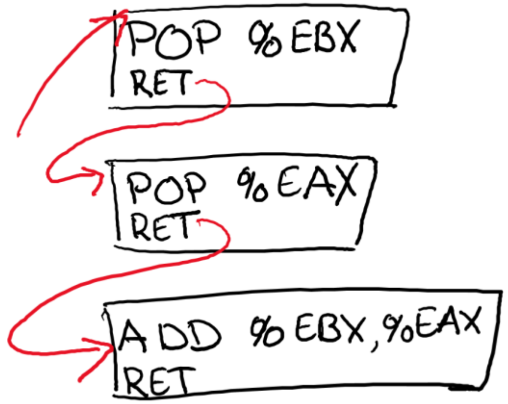


Code reuse

ROP

Small snippets of code ending with a RET

Can be chained together



Crucial requirements

Need: to find address of code (and data)

Need: bugs

This is getting harder

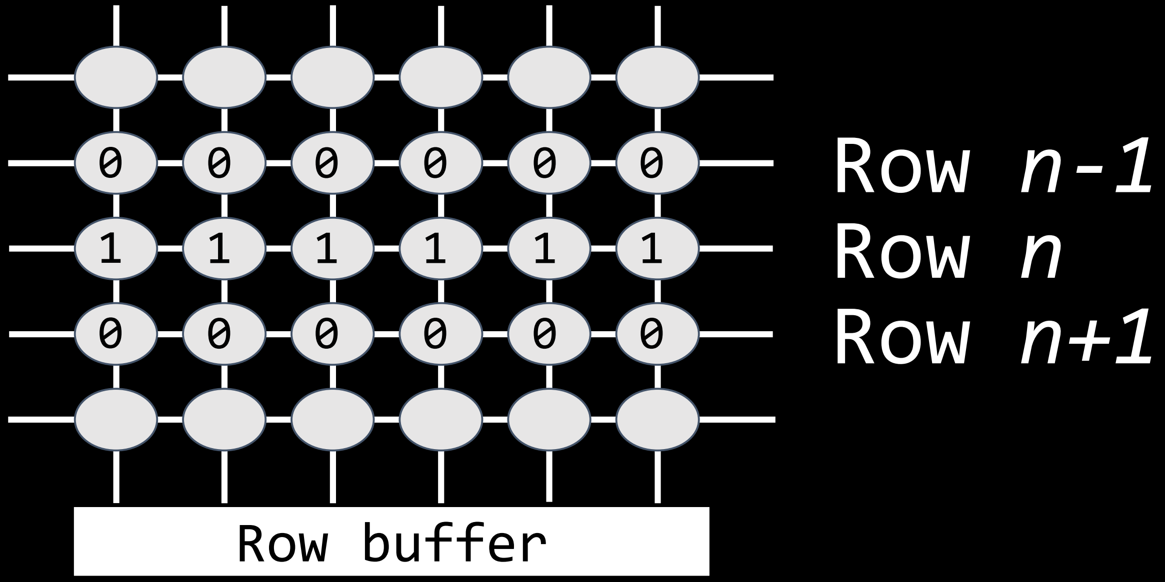
Want to do this

without the software bugs

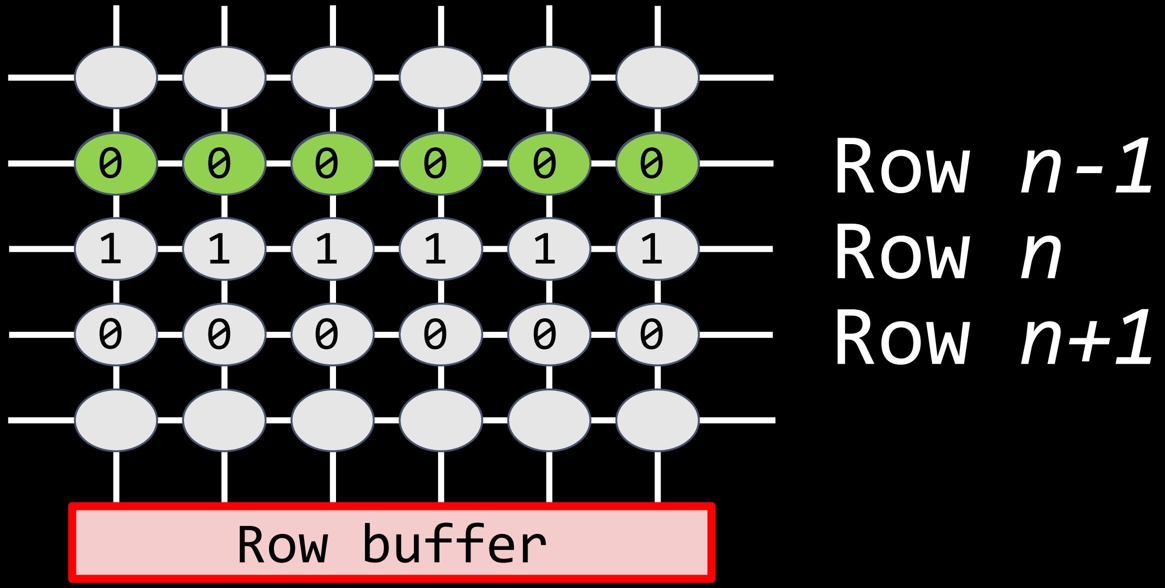
The rise and rise and rise

Rowhammer

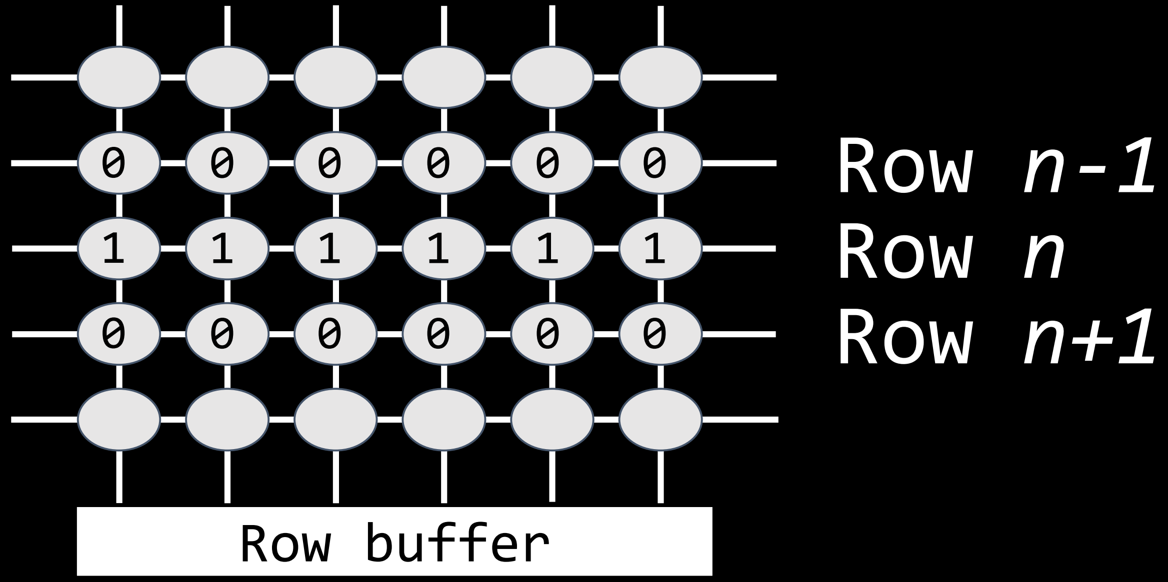
DRAM



DRAM

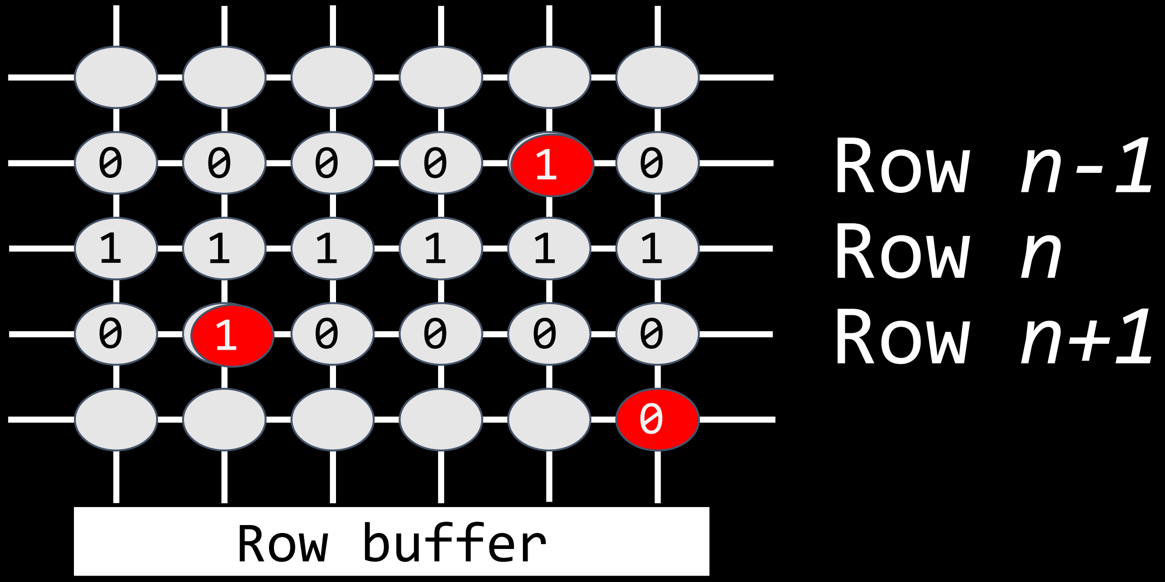


DRAM needs periodic refresh



DRAM needs periodic refresh

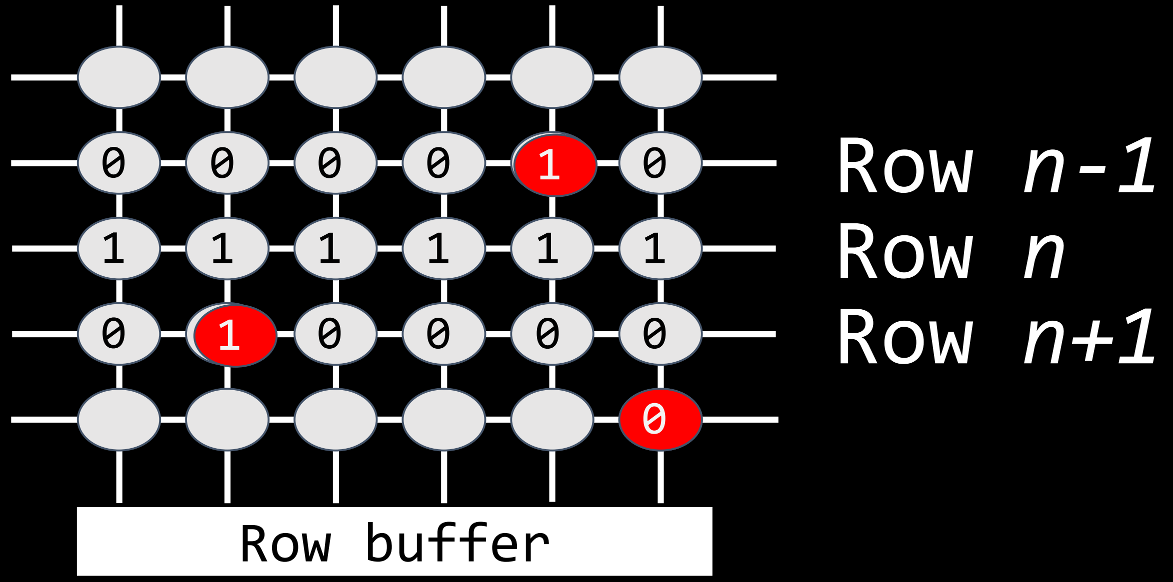
Or else:



Charge leakage causes bit flips

Reliability problem!

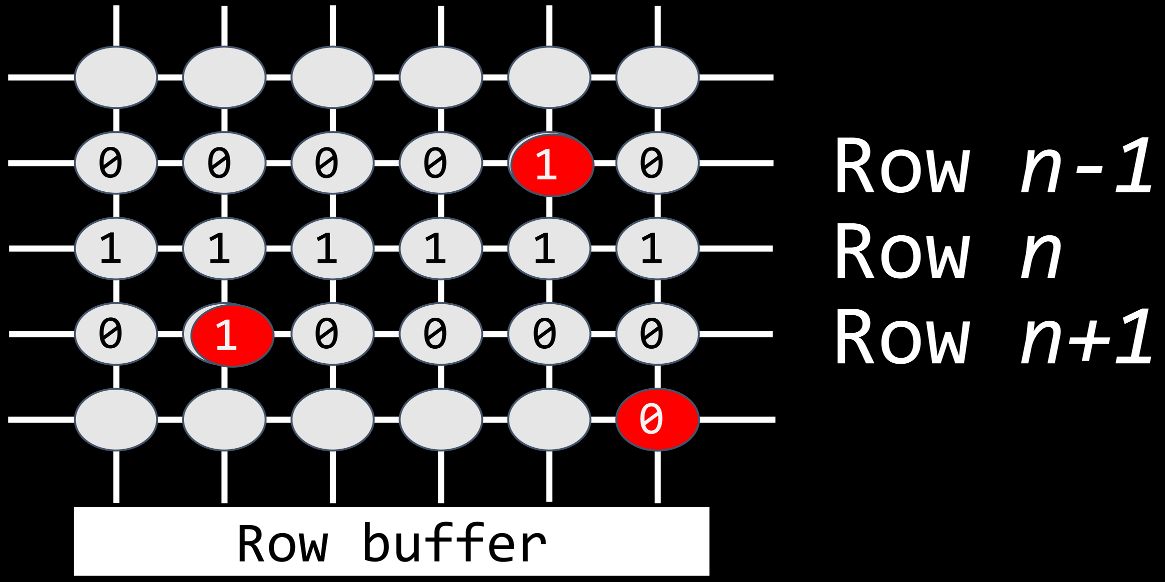
Or else:



Charge leakage causes bit flips

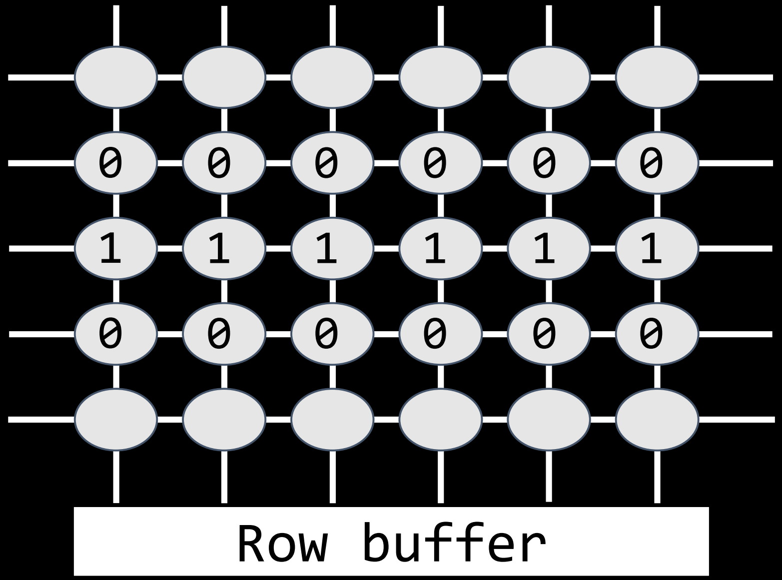
But wait!

Or else:



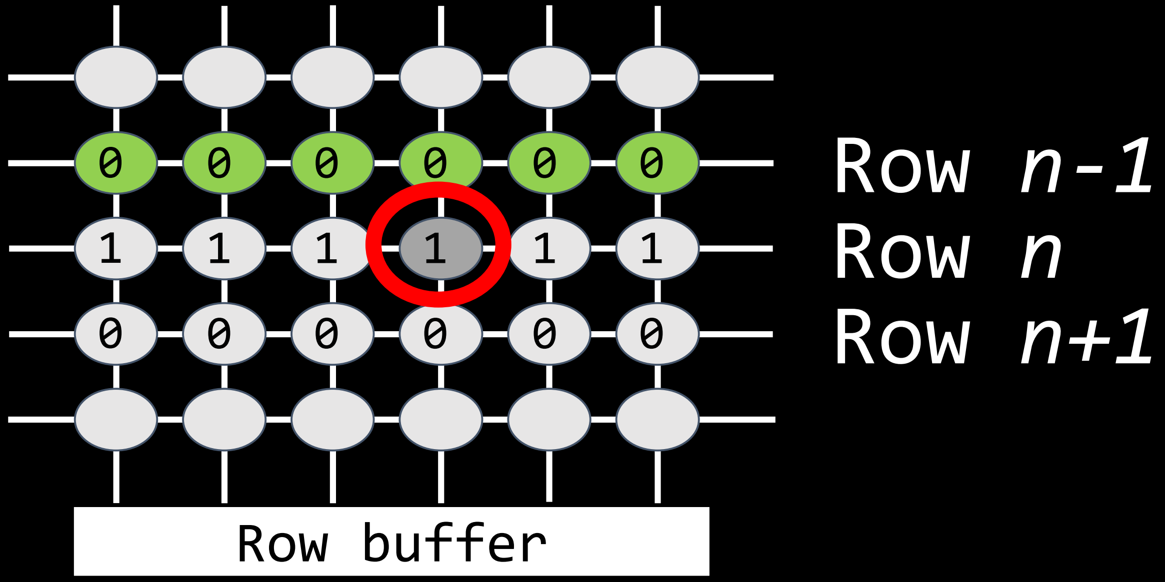
Charge leakage causes bit flips

Rowhammer

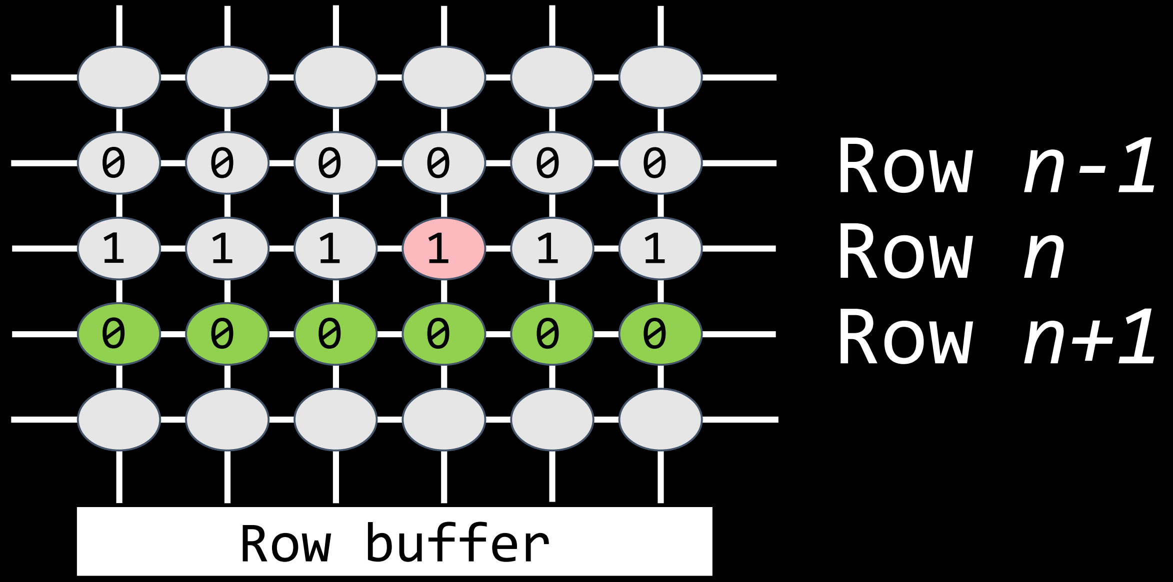


Row *n-1*
Row *n*
Row *n+1*

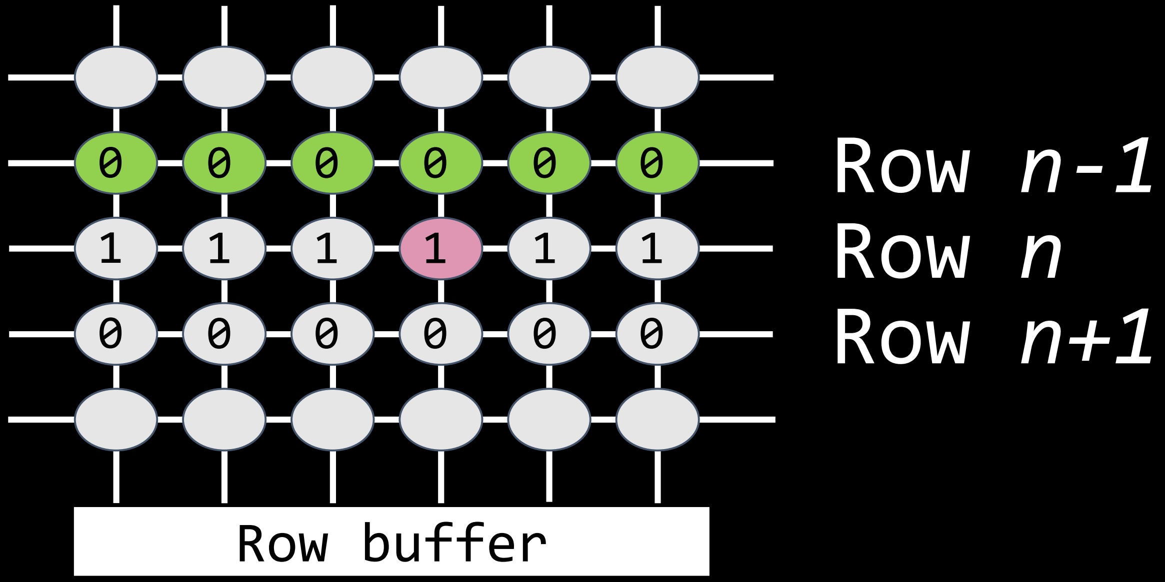
Rowhammer



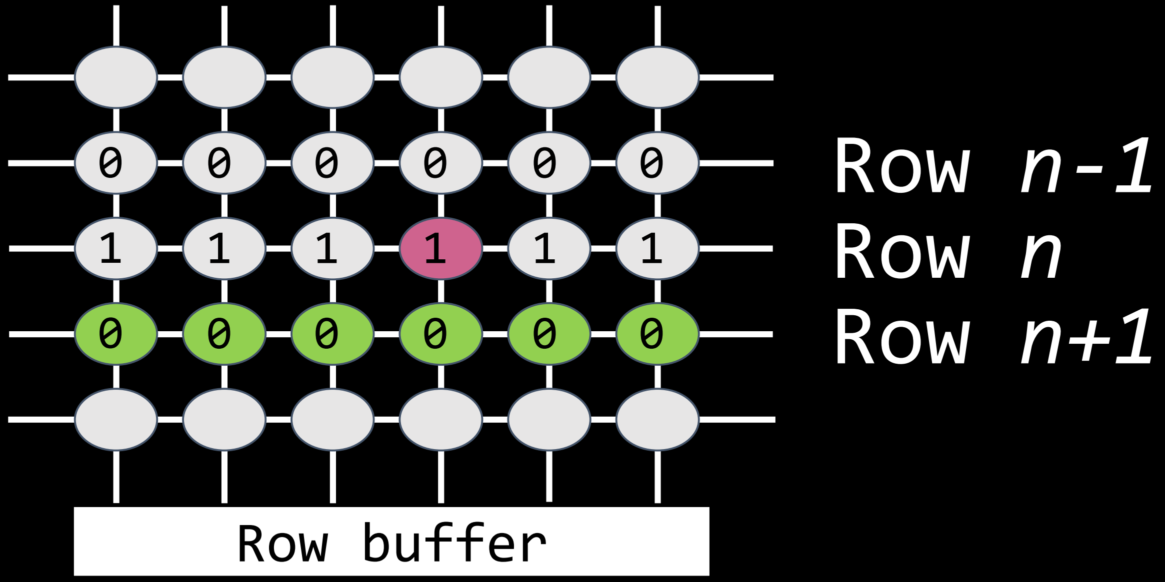
Rowhammer



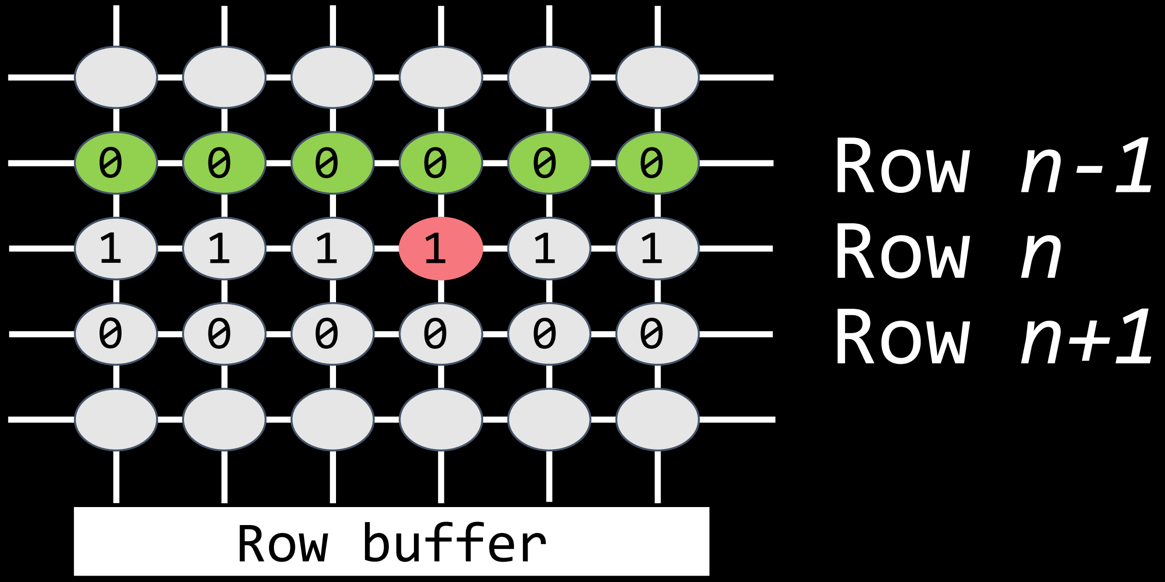
Rowhammer



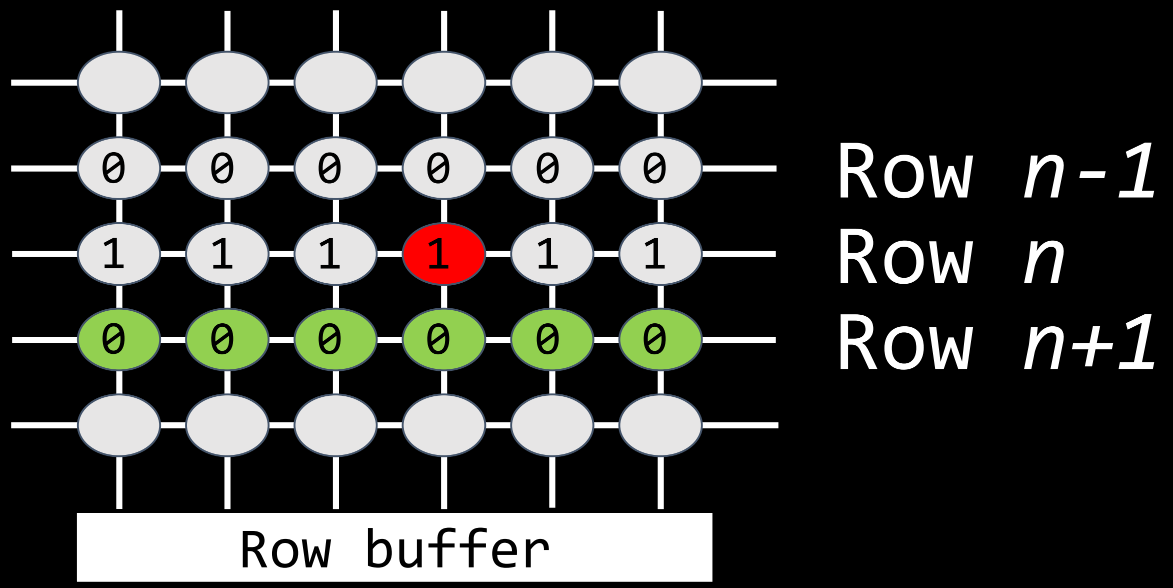
Rowhammer



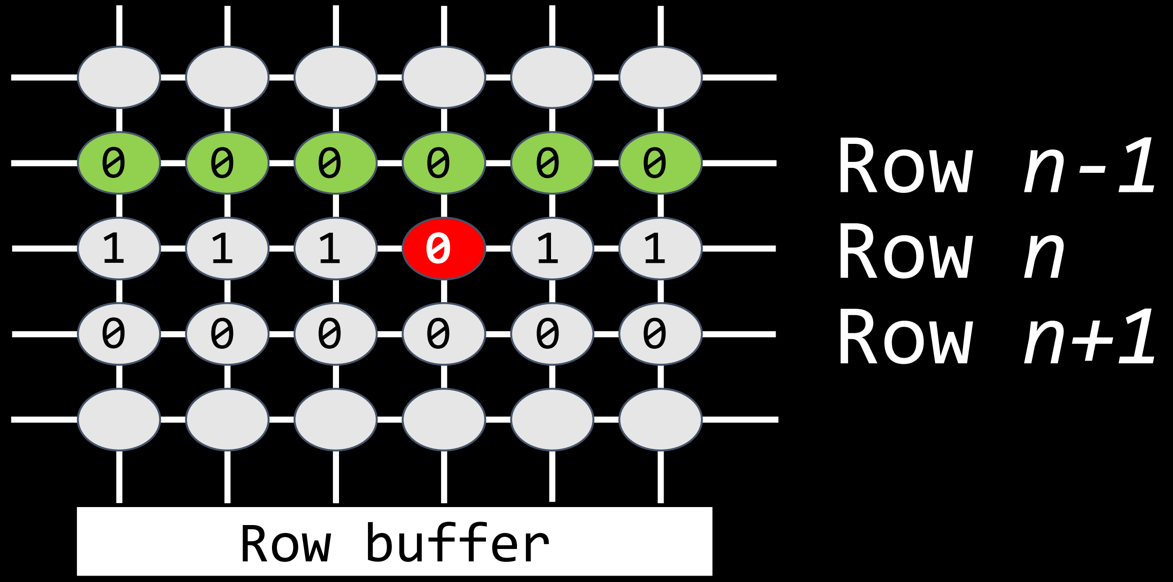
Rowhammer



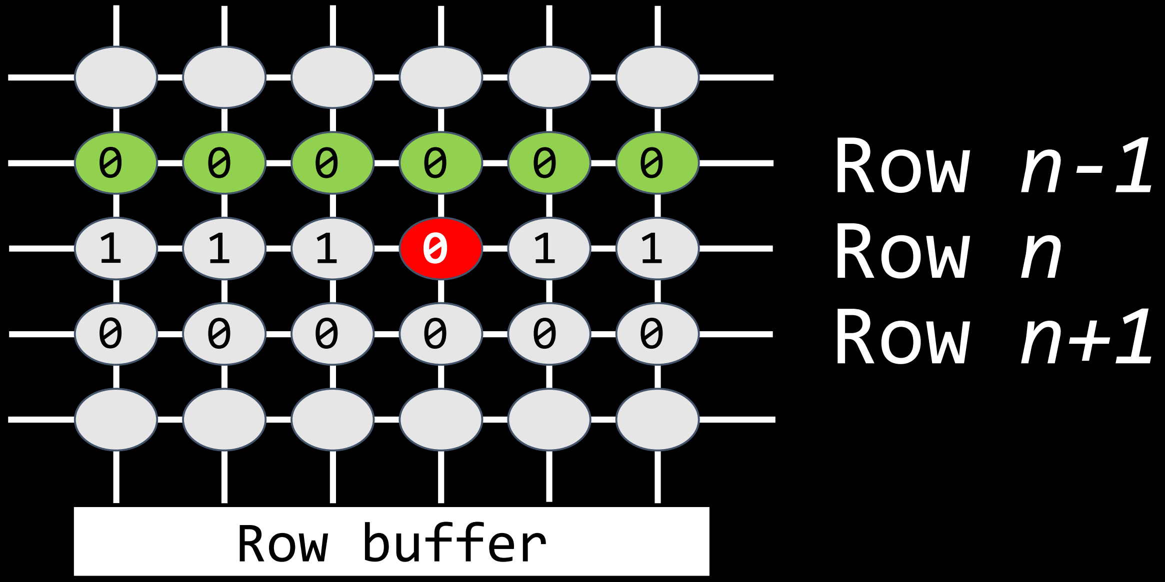
Rowhammer



Rowhammer

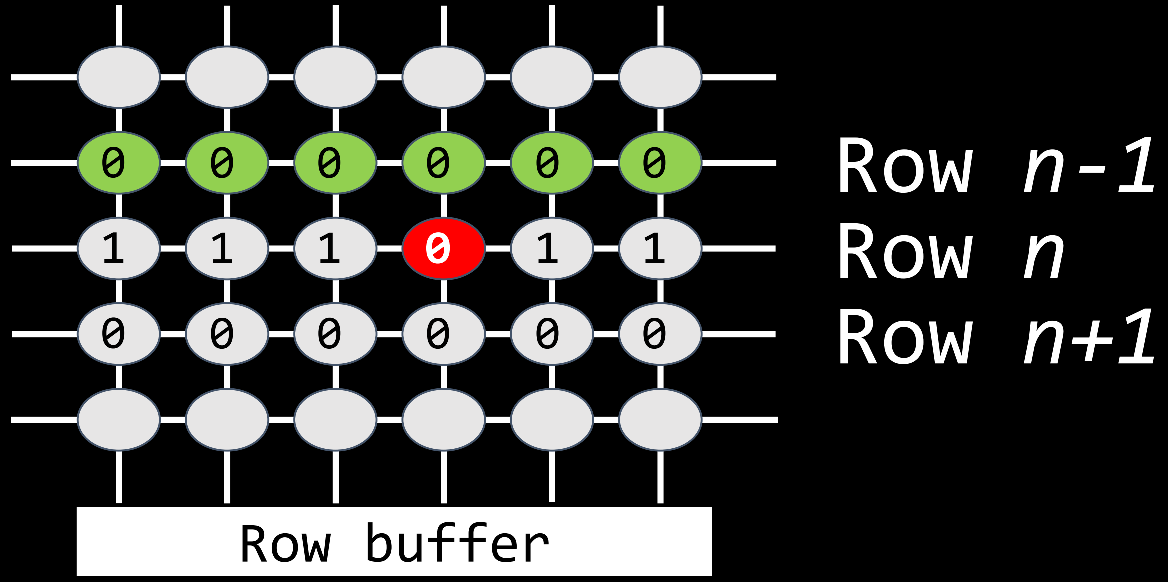


Rowhammer



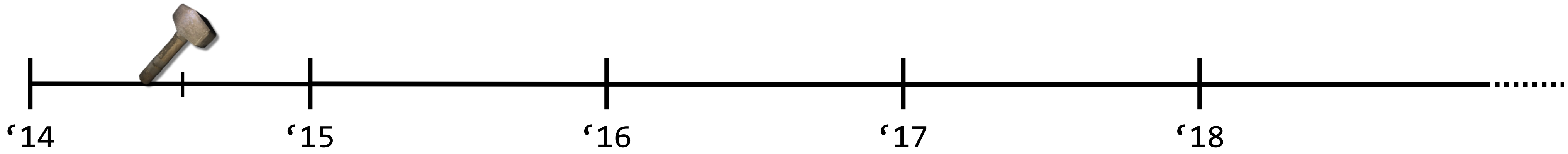
Don't know in advance which flips, but if it flips once, it will flip again

Rowhammer: security problem



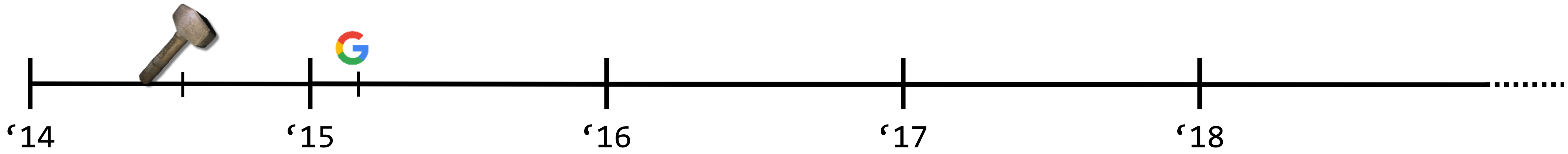
Root cause: efficiency fetish

Rowhammer Evolution



[1] CMU finds first bit flip (2014)

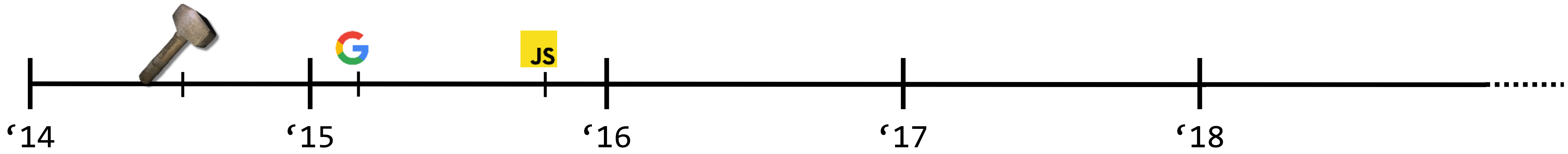
Rowhammer Evolution



[1] CMU finds first bit flip (2014)

[2] Google Project Zero: 1st Rowhammer root Exploit (flipping PTEs)

Rowhammer Evolution

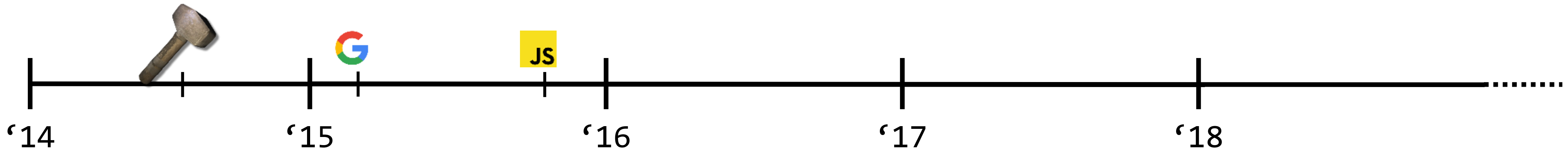


[1] CMU finds first bit flip (2014)

[2] Google Project Zero: 1st Rowhammer root Exploit (flipping PTEs)

[3] Rowhammer.js: 1st RH bit flip in JavaScript

Rowhammer Evolution



[1] CMU finds first bit flip (2014)

[2] Google Project Zero: 1st Rowhammer root Exploit (flipping PTEs)

[3] Rowhammer.js: 1st RH bit flip in JavaScript

Can we do this on Edge from Javascript in realistic settings?

Goal 1

Bug-free Exploitation in Browsers

Dedup Est Machina

Published at IEEE S&P 2016

Won **Pwnie Award** at Black HAT 2016

"Most Innovative Research"

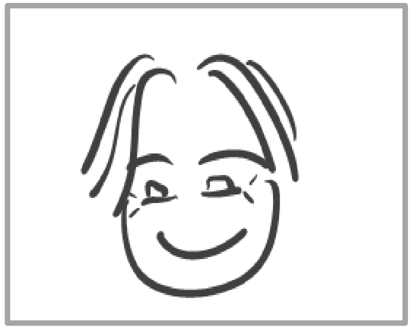
Exploit of MS Edge browser on Windows 10 from JavaScript

...without relying on a single software bug

Erik Bosman

Kaveh razavi

Herbert Bos



Cristiano Giuffrida

Memory deduplication

(software side channel)

Memory deduplication

(software side channel)

+

Rowhammer

(hardware glitch)

Dedup Est Machina

Memory deduplication

(software side channel)

+

Rowhammer

(hardware glitch)

⬇

Exploit MS Edge without software bugs

(from JavaScript)

Remember

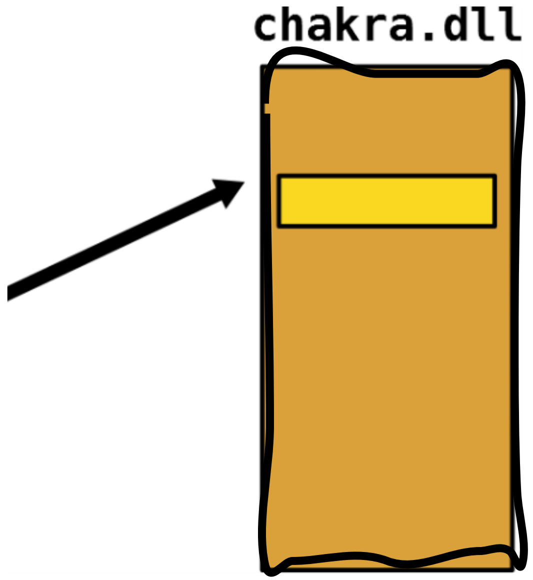
Crucial:

need to find address of code and data

Dedup Est Machina: Overview

Memory deduplication

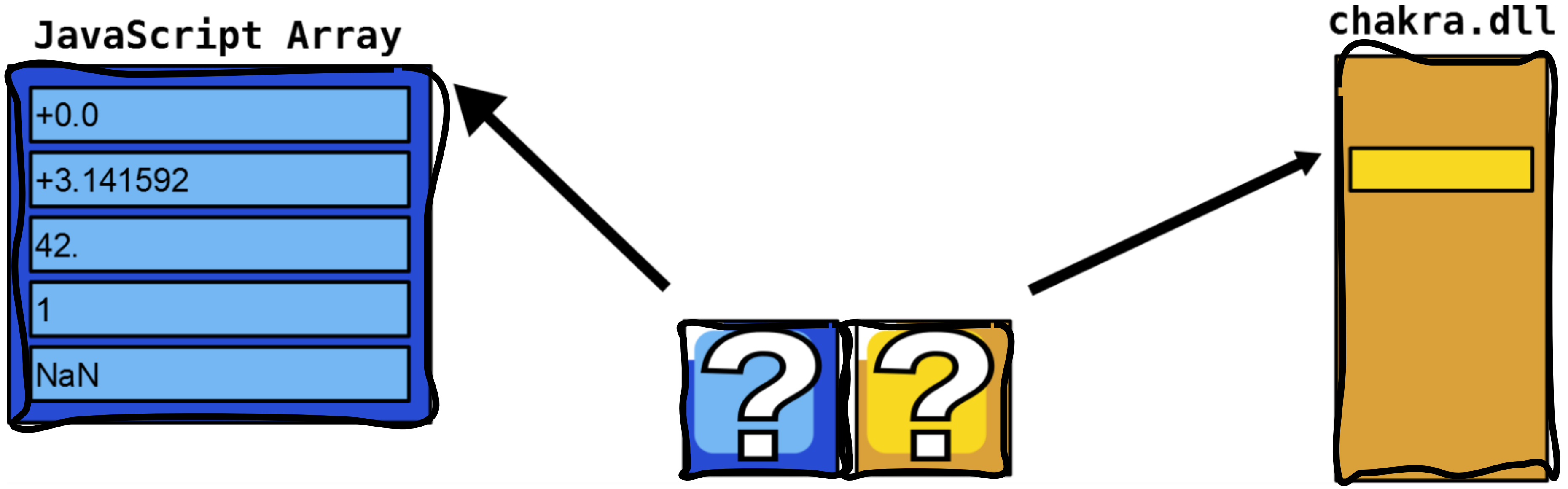
Leak randomized heap and code pointers



Dedup Est Machina: Overview

Memory deduplication

Leak randomized heap and code pointers

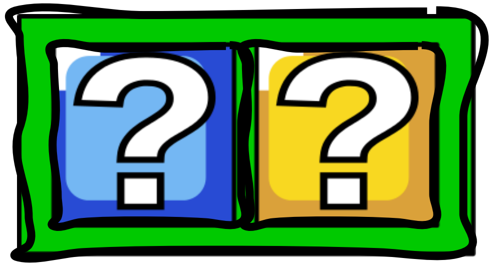


Dedup Est Machina: Overview

Memory deduplication

Leak randomized heap and code pointers

Create a fake JavaScript object



Dedup Est Machina: Overview

Memory deduplication

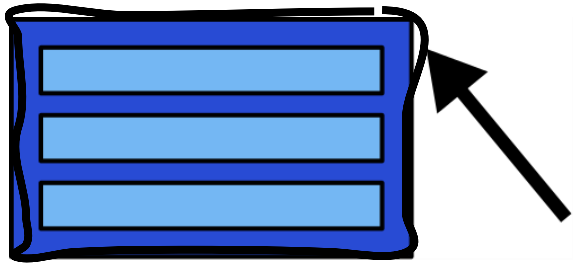
Leak randomized heap and code pointers

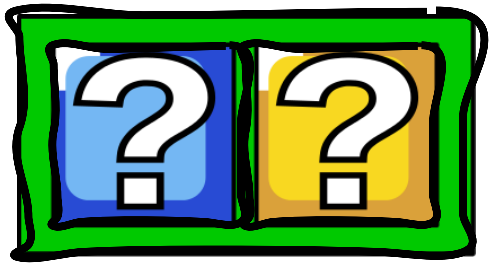
Create a fake JavaScript object

+

Rowhammer

Create a reference to our fake object





Dedup Est Machina: Overview

Memory deduplication

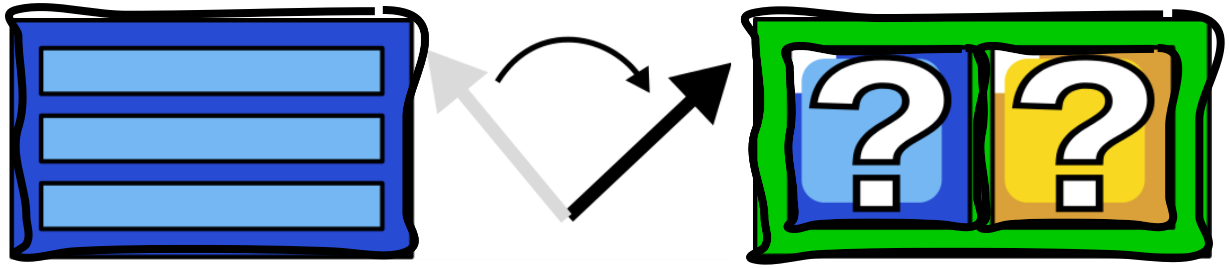
Leak randomized heap and code pointers

Create a fake JavaScript object

+

Rowhammer

Create a reference to our fake object



Dedup Est Machina: Overview

Memory deduplication

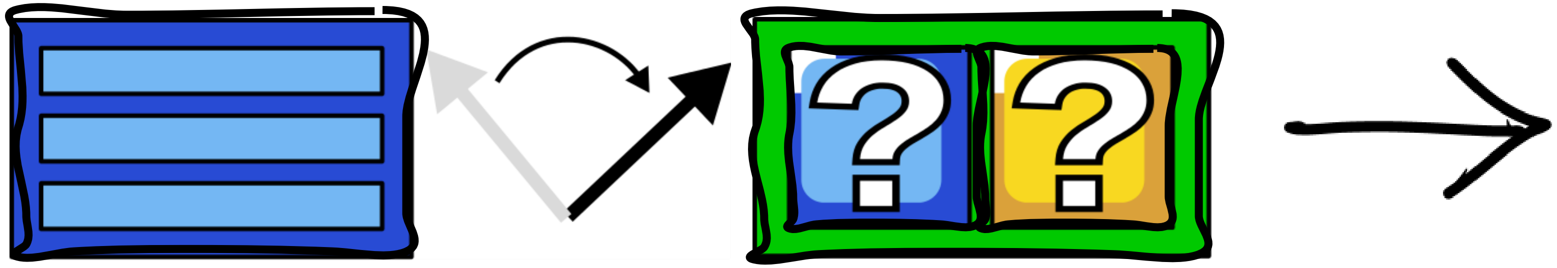
Leak randomized heap and code pointers

Create a fake JavaScript object

+

Rowhammer

Create a reference to our fake object



Memory Deduplication

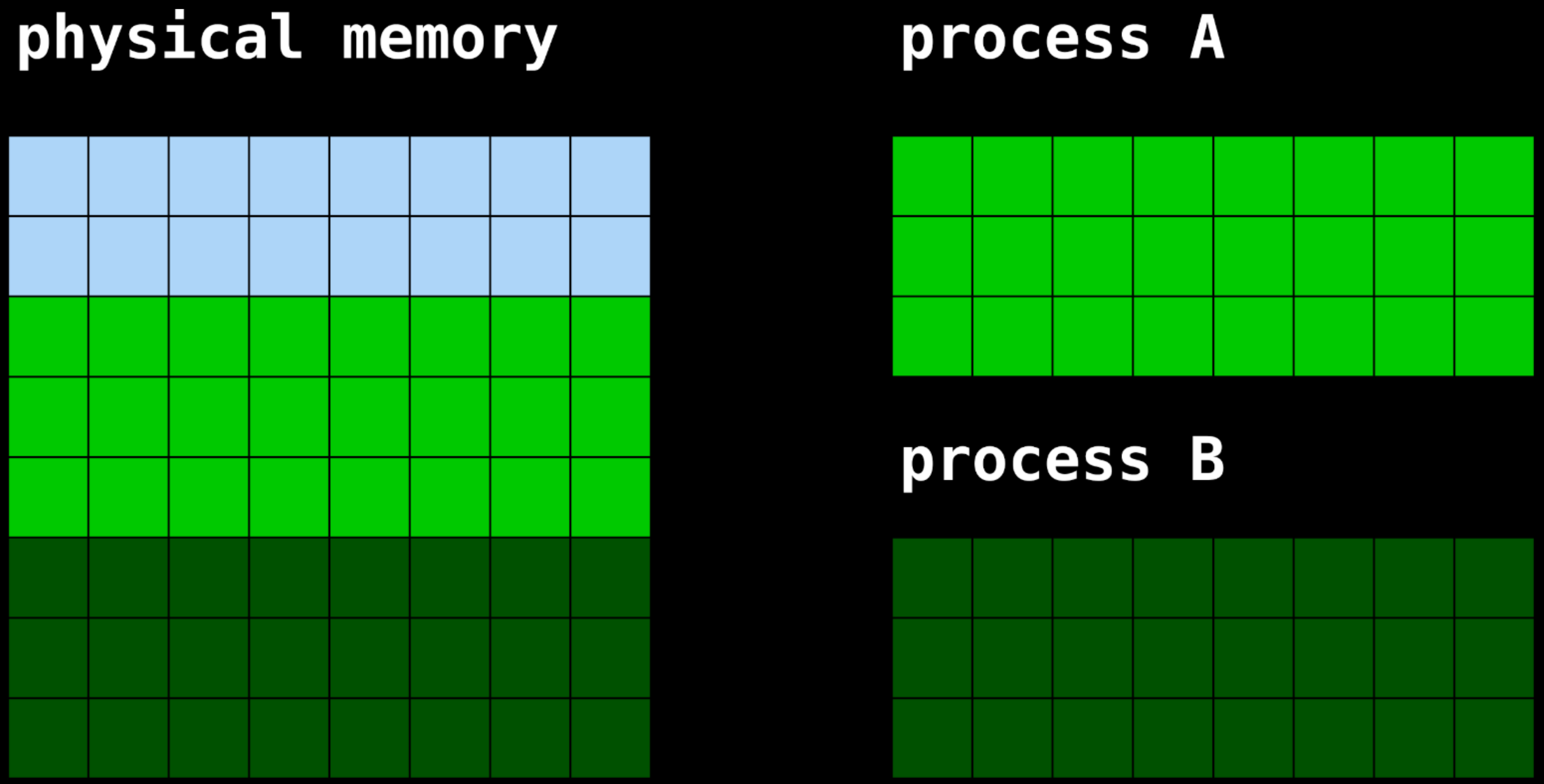
An *efficiency* measure to reduce physical memory usage

Common in virtualization environments

Enabled by **default on Windows**

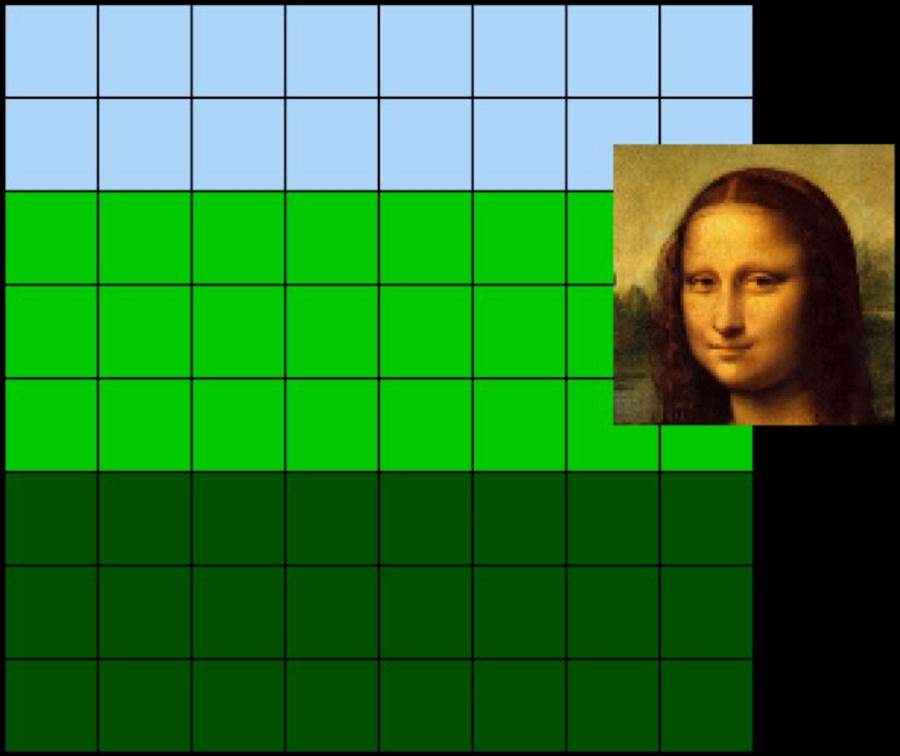
- Windows 8.1
- Windows 10

Memory Deduplication: Mechanics

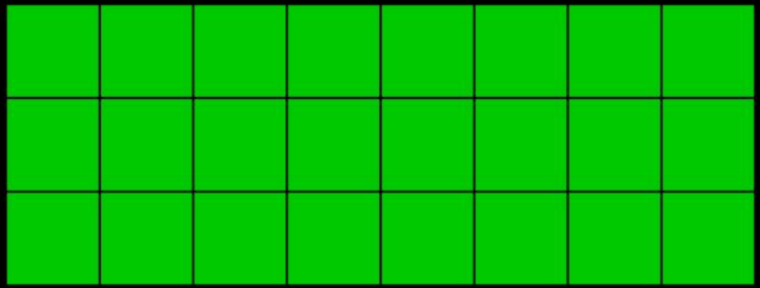


Memory Deduplication: Mechanics

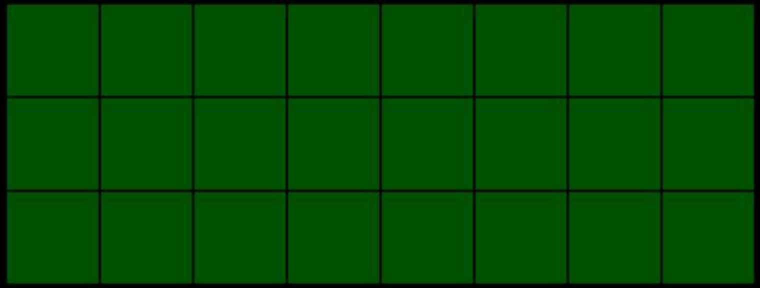
physical memory



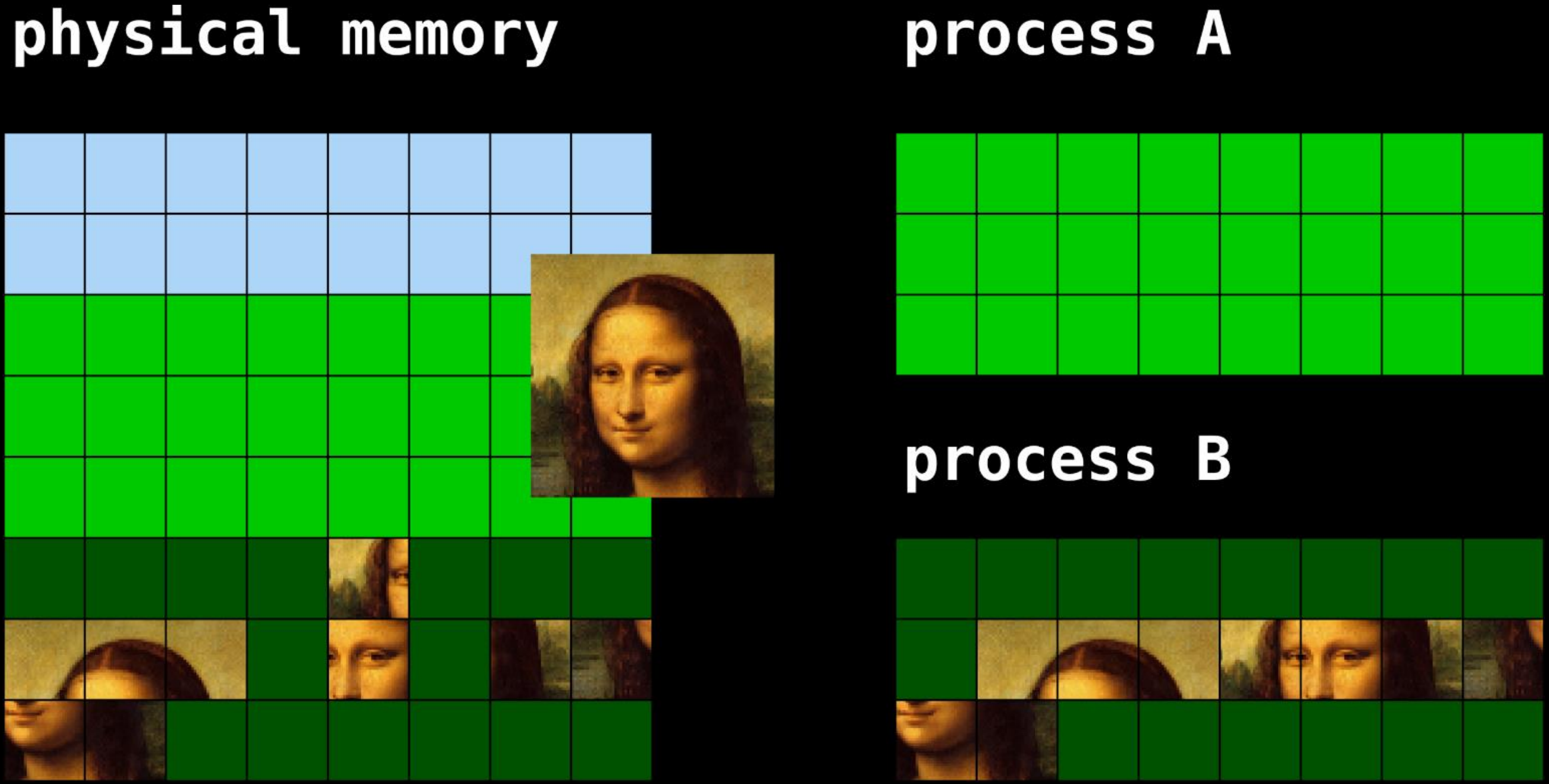
process A



process B

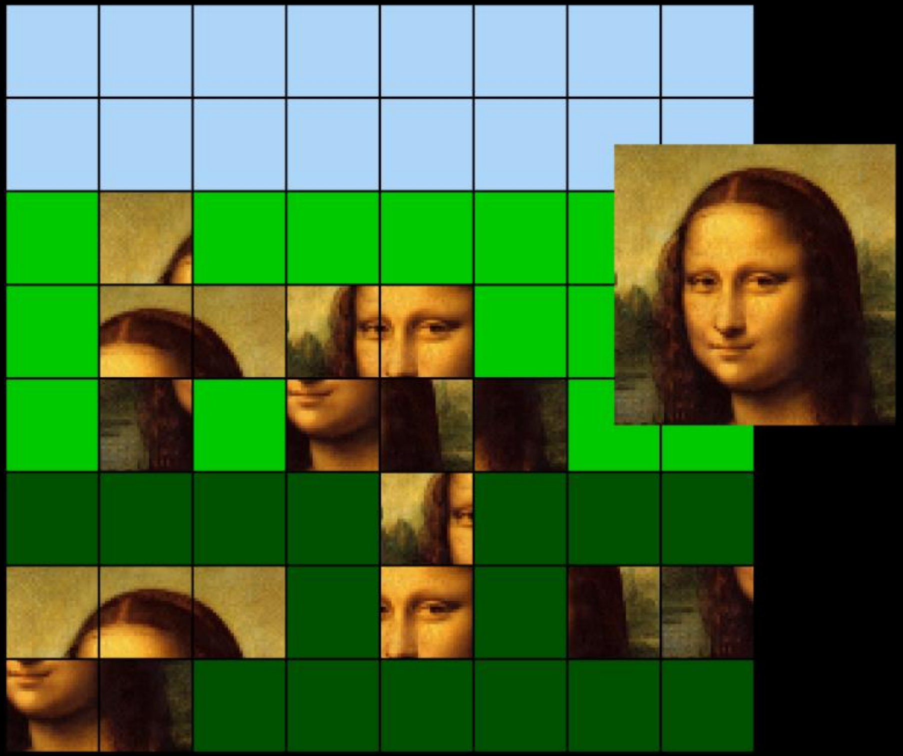


Memory Deduplication: Mechanics

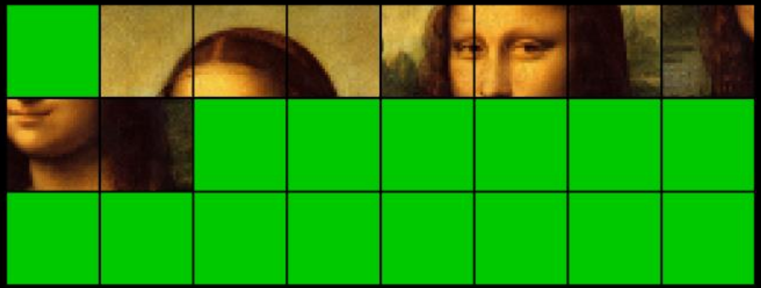


Memory Deduplication: Mechanics

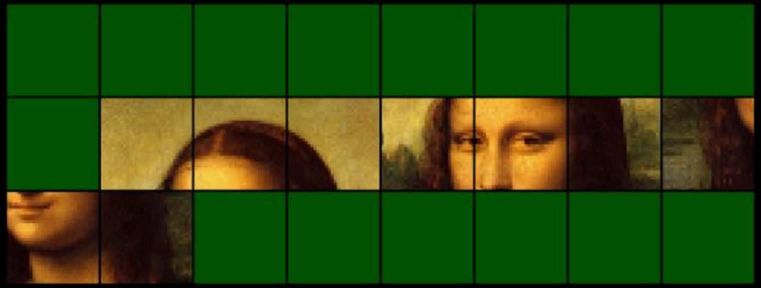
physical memory



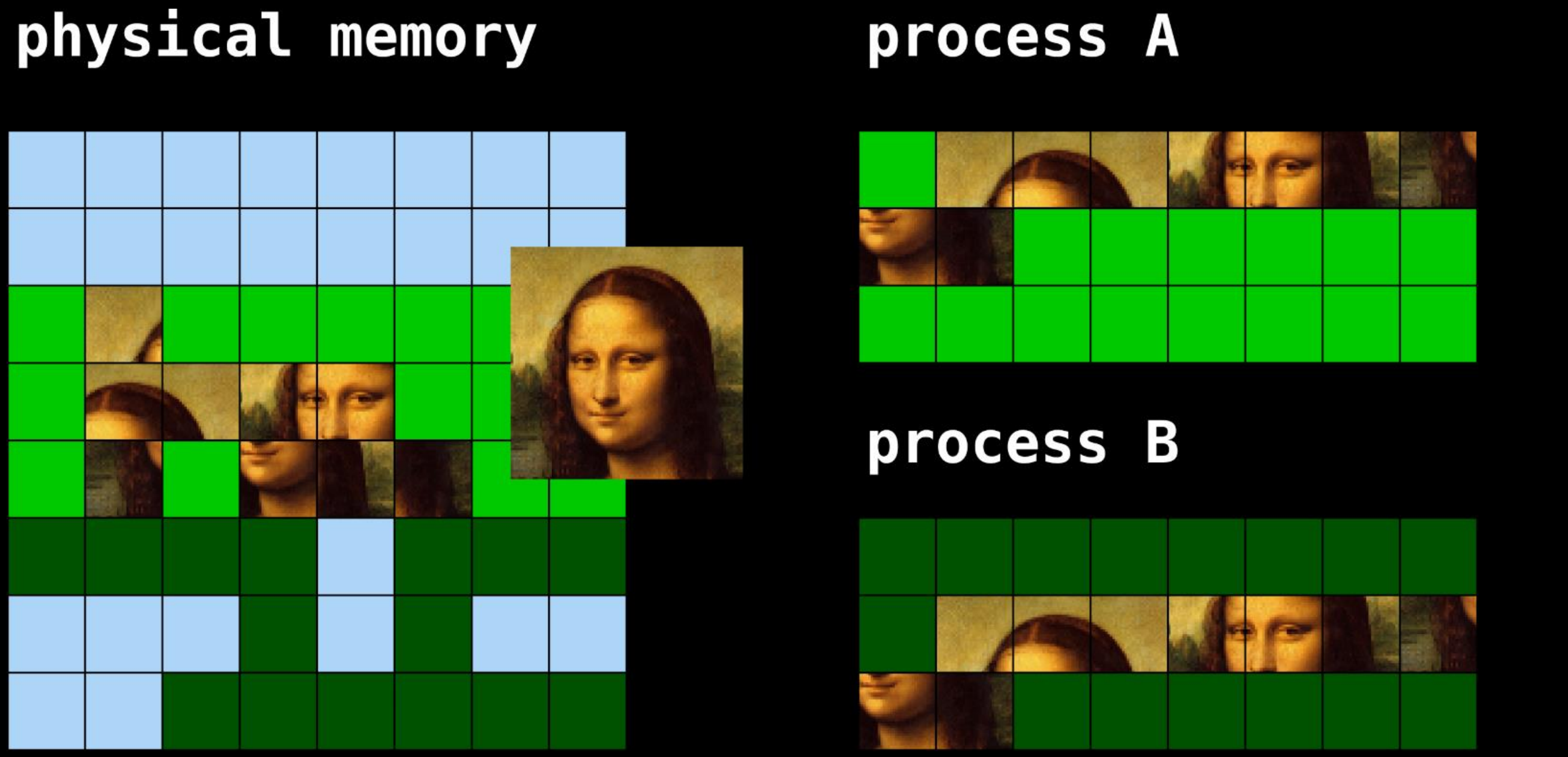
process A



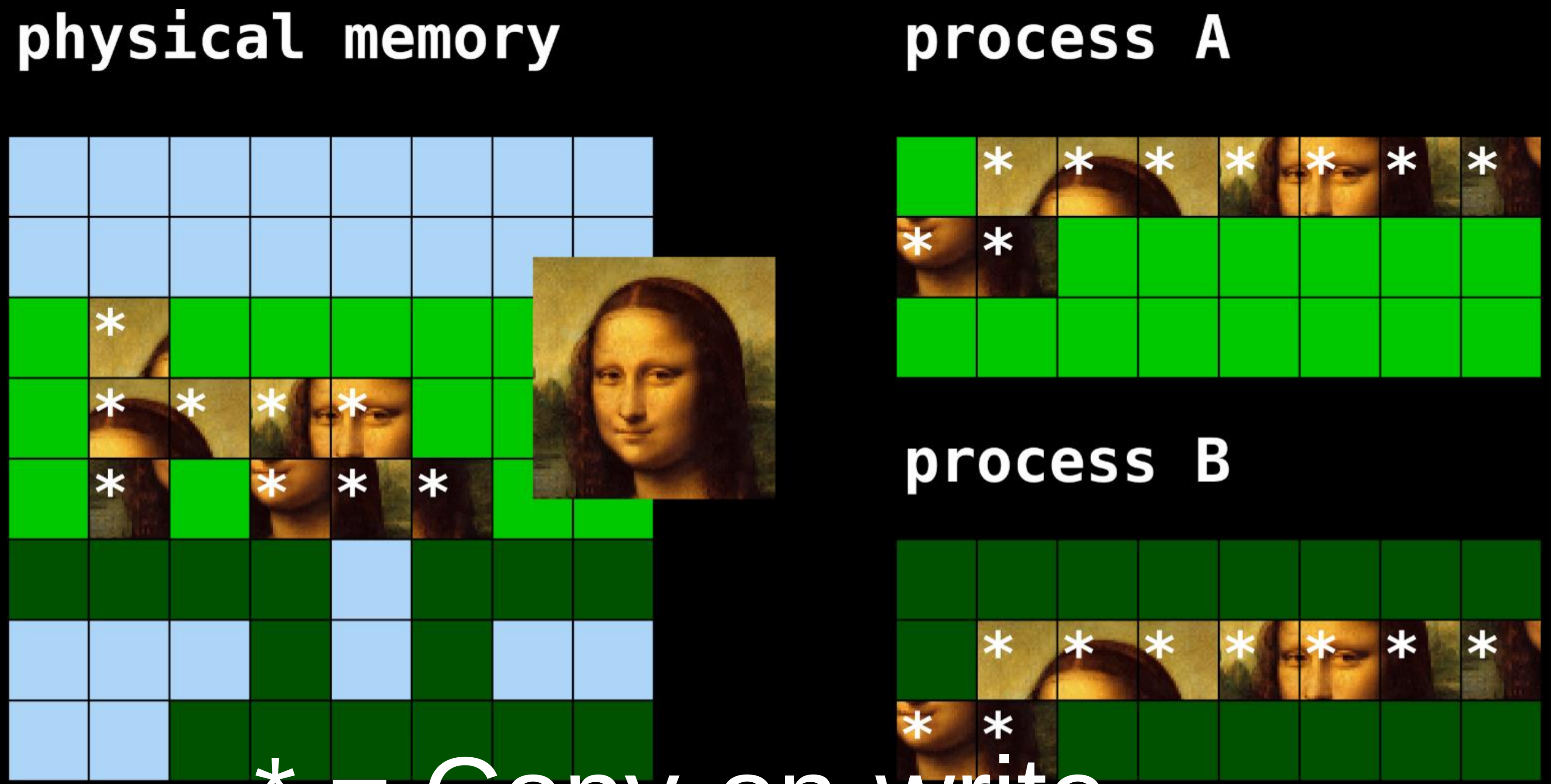
process B



Memory Deduplication: Mechanics



Memory Deduplication: Mechanics



* = Copy-on-write

DEDUPLICATION
WORKS ACROSS
SECURITY BOUNDARIES

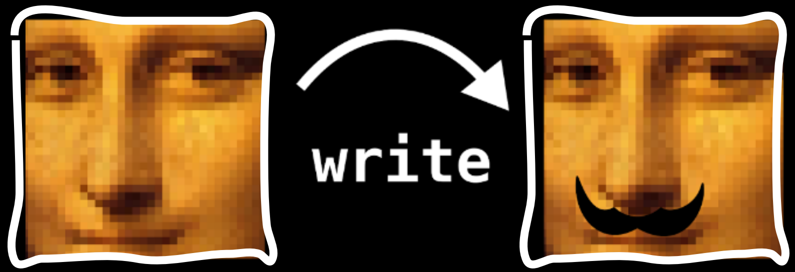
SIDE CHANNEL?

Memory Deduplication: Timing Side Channel

normal write

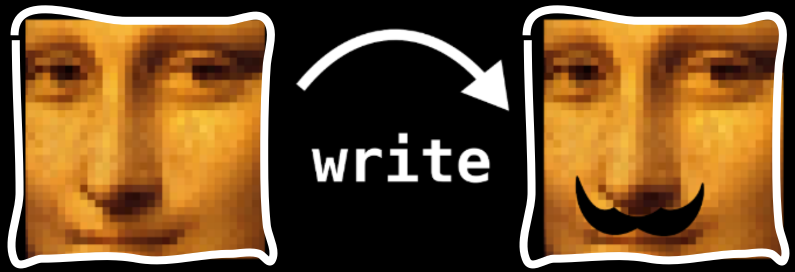
Memory Deduplication:
Timing Side Channel

normal write



Memory Deduplication:
Timing Side Channel

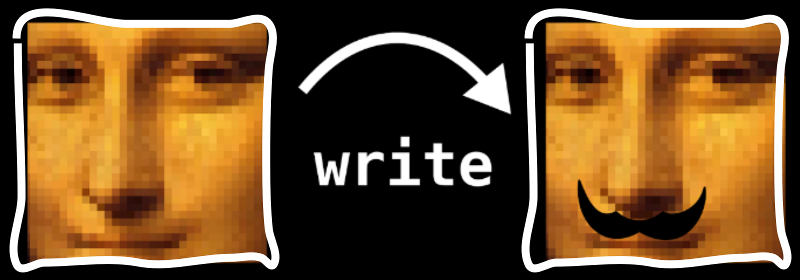
normal write



copy on write (due to deduplication)

Memory Deduplication: Timing Side Channel

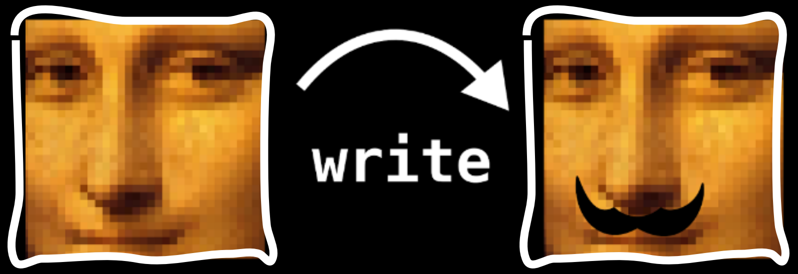
normal write



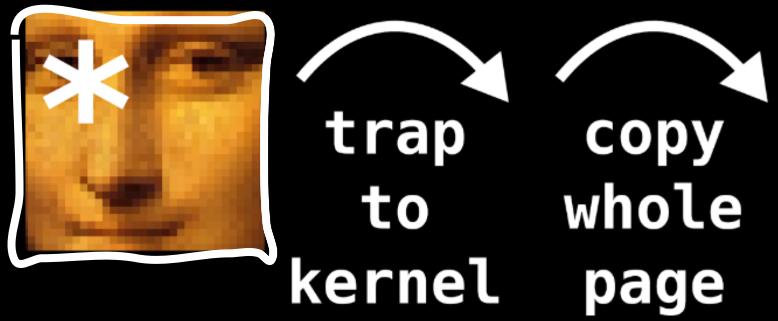
copy on write (due to deduplication)

Memory Deduplication: Timing Side Channel

normal write

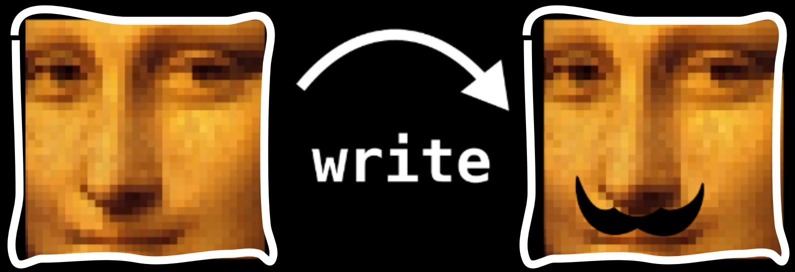


copy on write (due to deduplication)

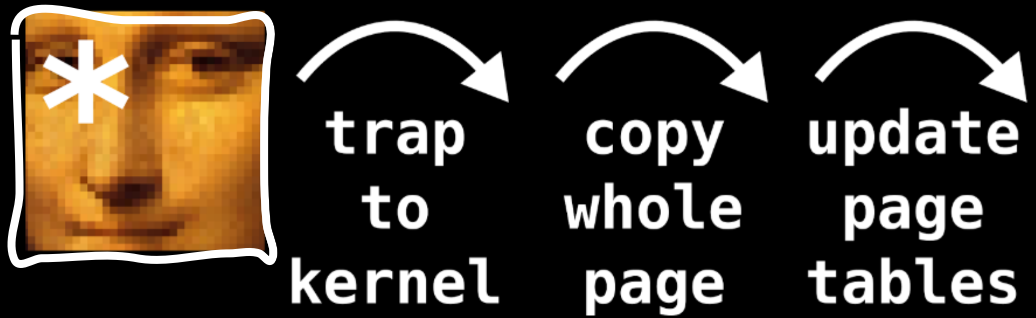


Memory Deduplication: Timing Side Channel

normal write

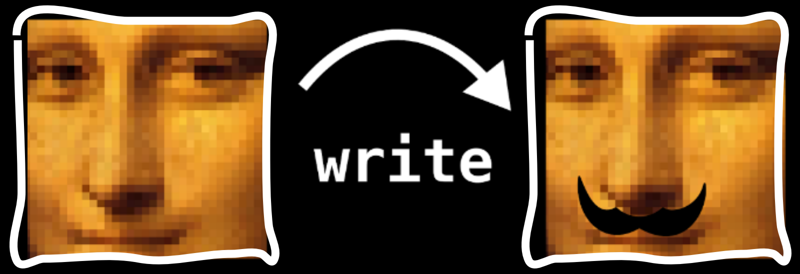


copy on write (due to deduplication)

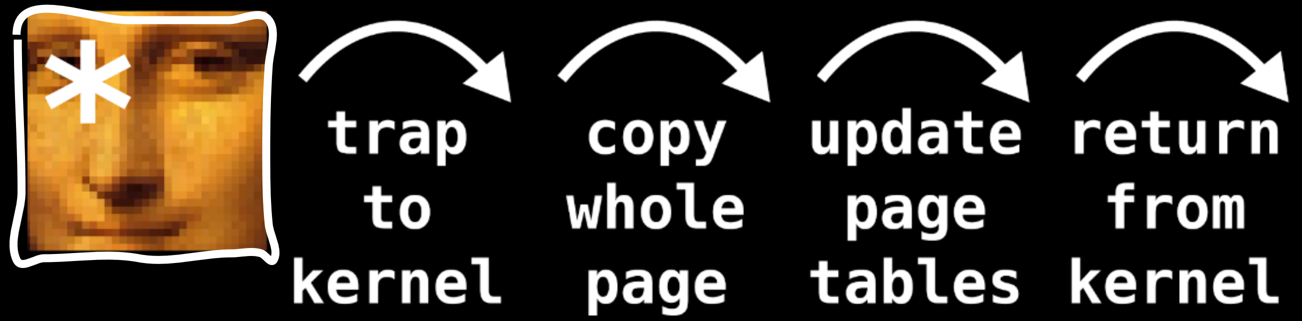


Memory Deduplication: Timing Side Channel

normal write

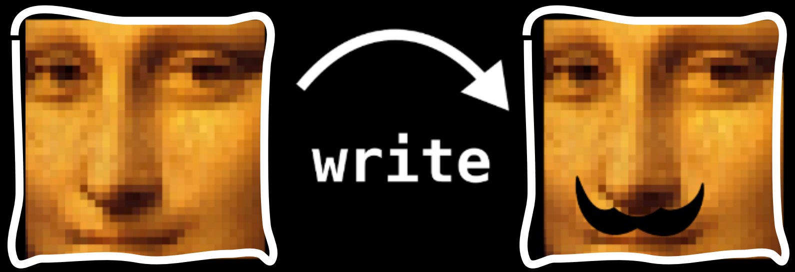


copy on write (due to deduplication)



Memory Deduplication: Timing Side Channel

normal write



copy on write (due to deduplication)

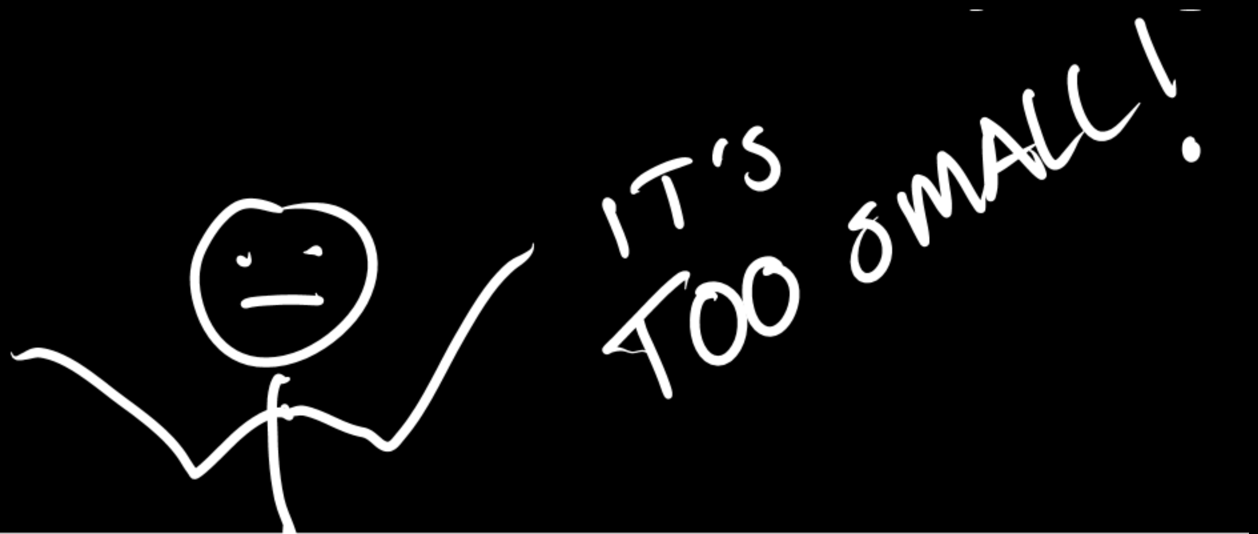
Memory Deduplication:
The Problem

“Can we generalize this to leaking ***arbitrary data*** *like randomized pointers?*

Dedup Est Machina: Challenges

Challenge 1:

The secret we want to leak does not span an entire memory page



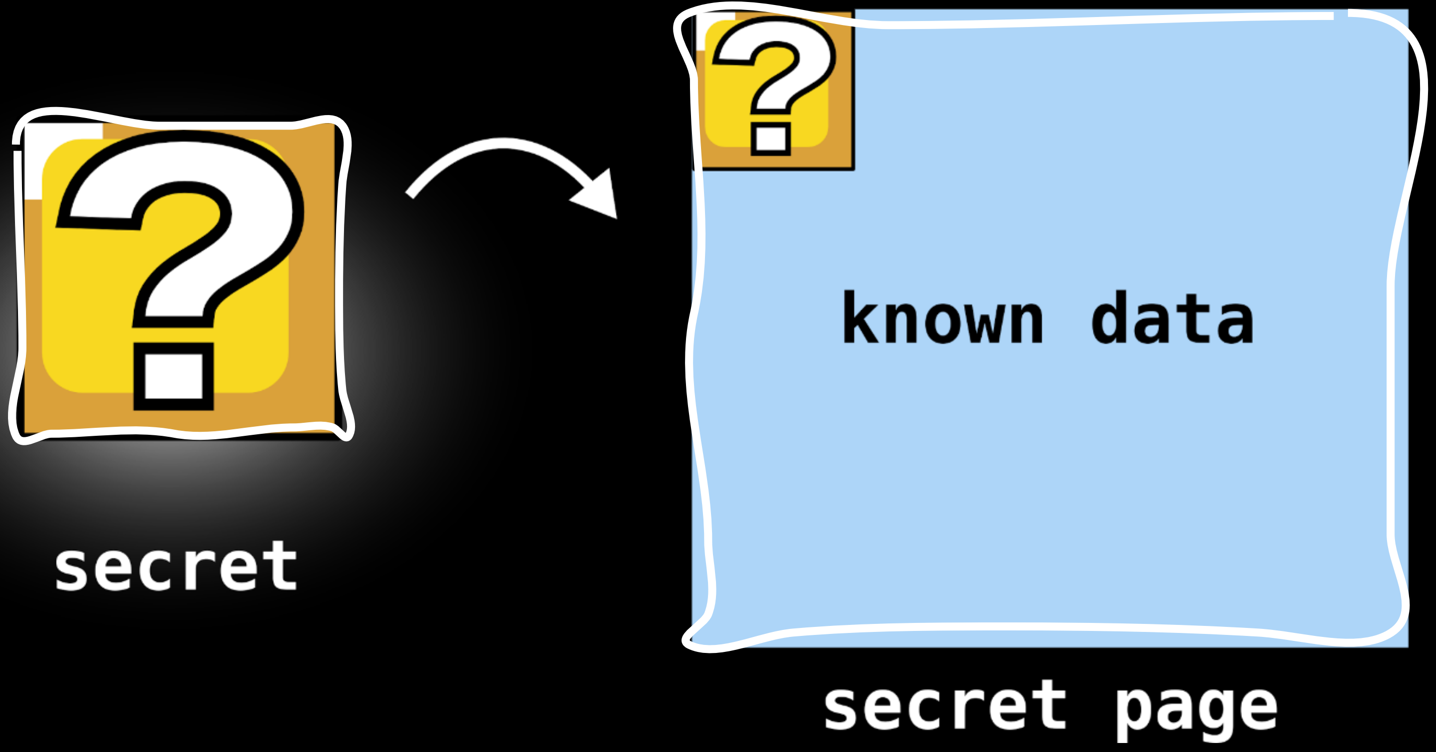
Dedup Est Machina: Challenges

Turning a secret into a page

secret

Dedup Est Machina: Challenges

Turning a secret into a page



Dedup Est Machina: Challenges

Challenge 2:

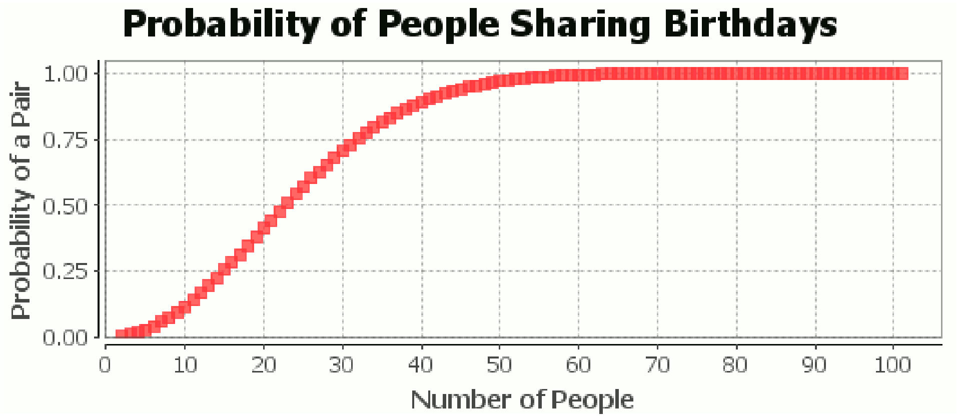
The secret to leak has too much entropy to leak it all at once

BIRTHDAY "PARADOX"

(NOT REALLY A PARADOX)

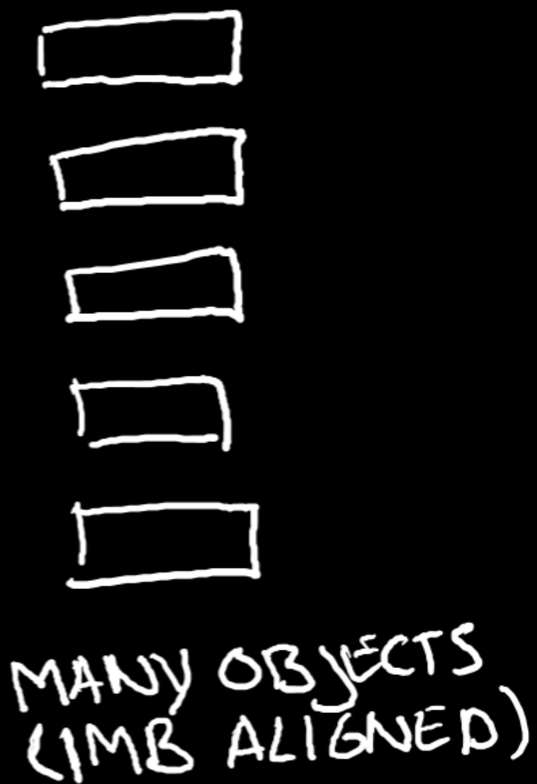
Only 23 people for a 50% same- birthday chance

You compare everyone with everyone else → **Any match suffices!**



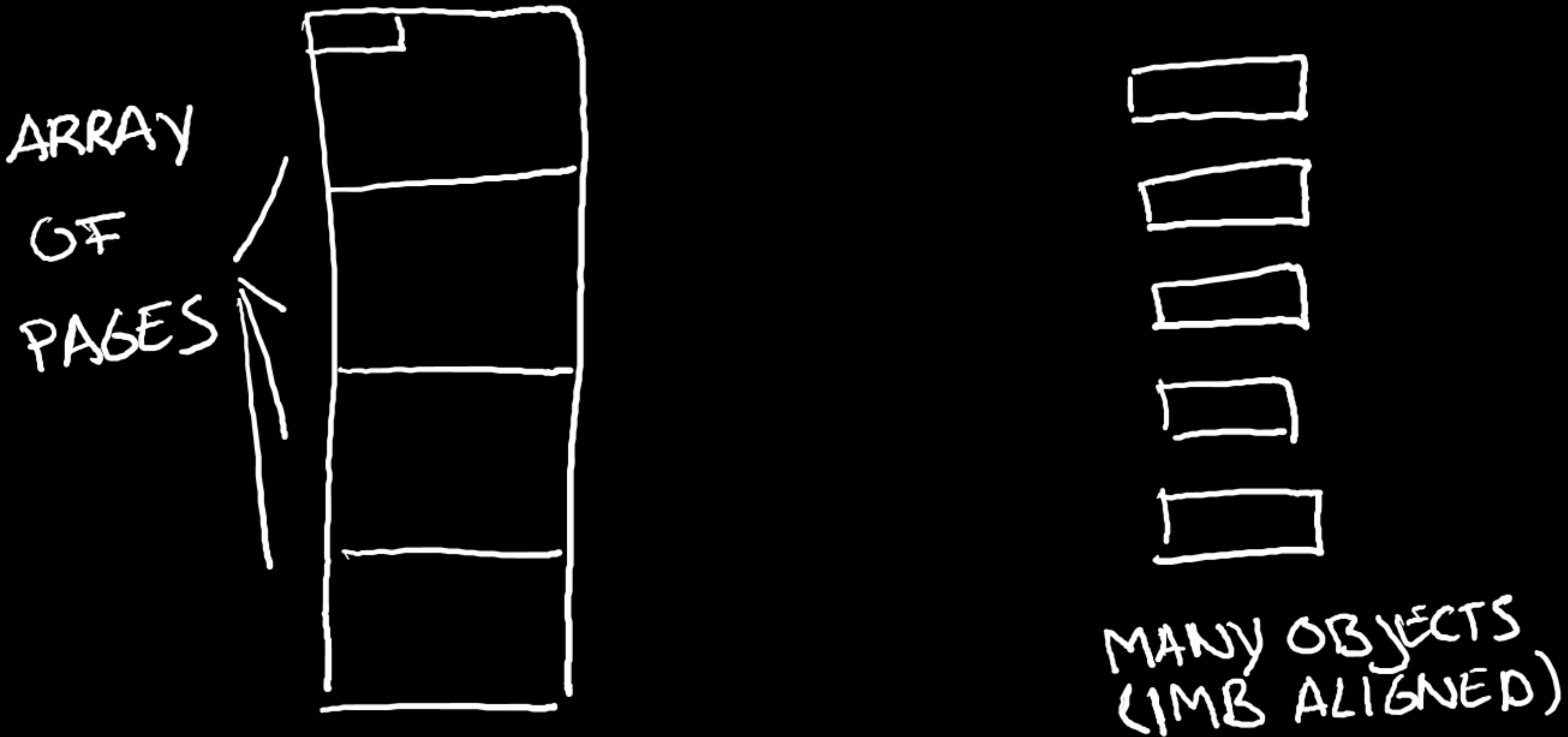
Dedup Est Machina: Leaking Heap Pointer

Create many Secret Pages



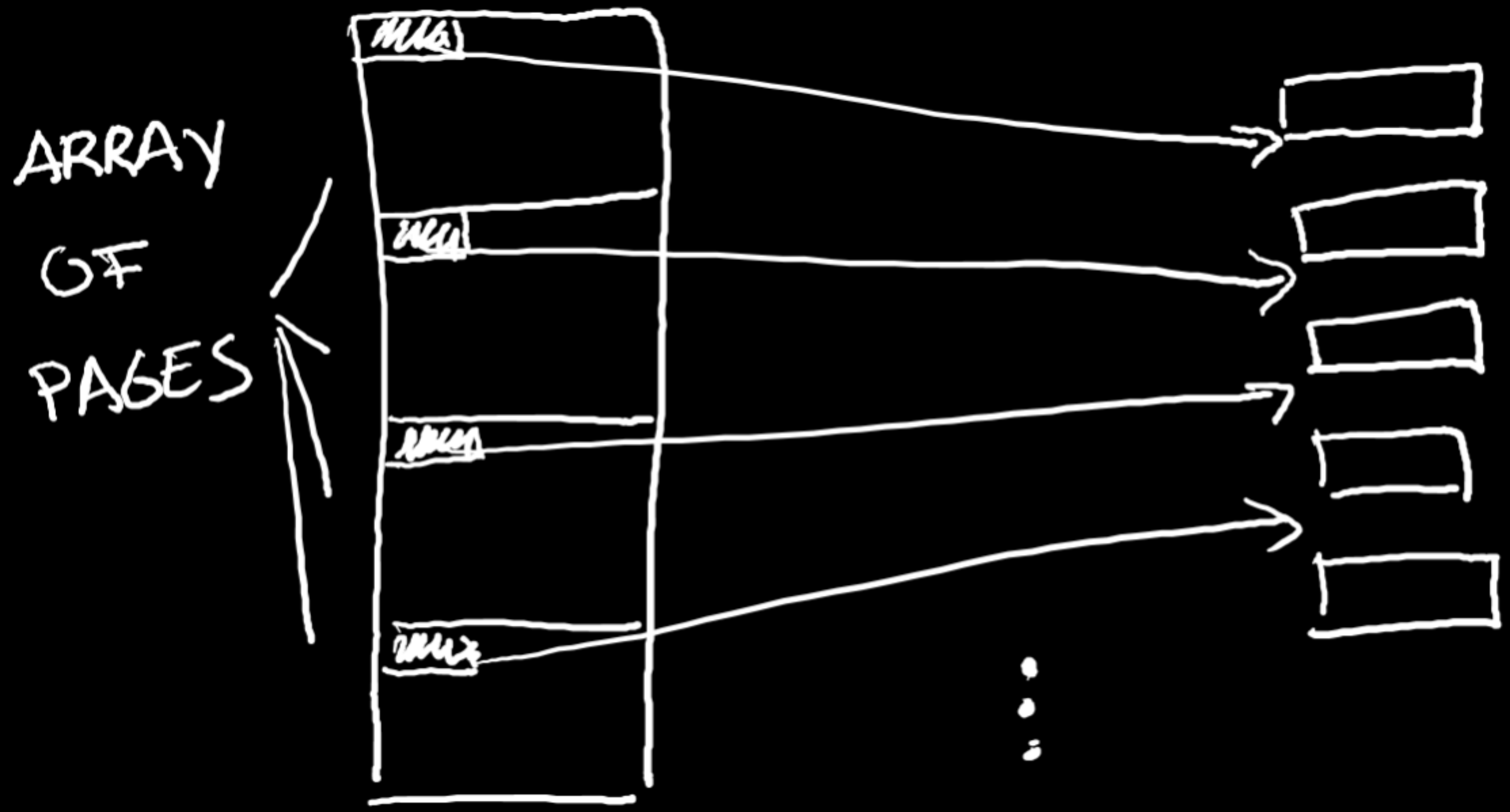
Dedup Est Machina: Leaking Heap Pointer

Create many Secret Pages



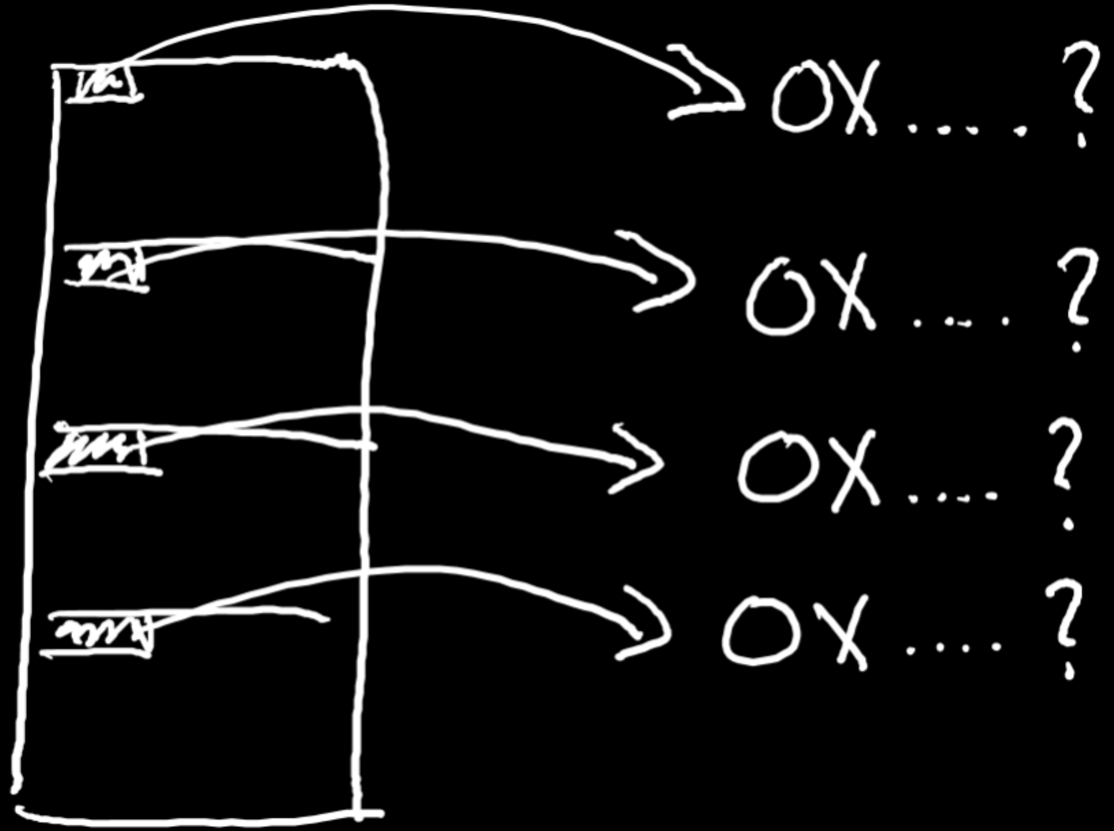
Dedup Est Machina:
Leaking Heap Pointer

Create many Secret Pages



Dedup Est Machina:
Leaking Heap Pointer

Create many guesses

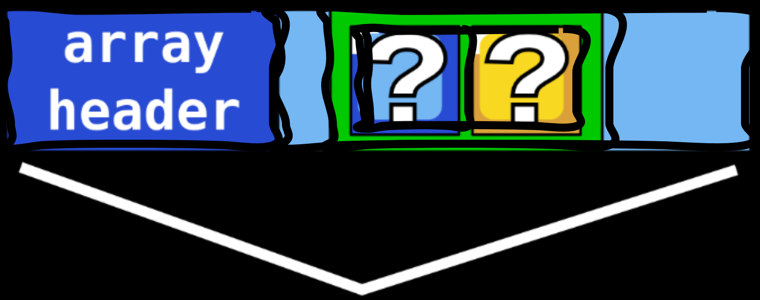


Dedup Est Machina:
Leaking Heap Pointer

If any deduplicated ➔ nailed it!

Dedup Est Machina: Creating a Fake Object

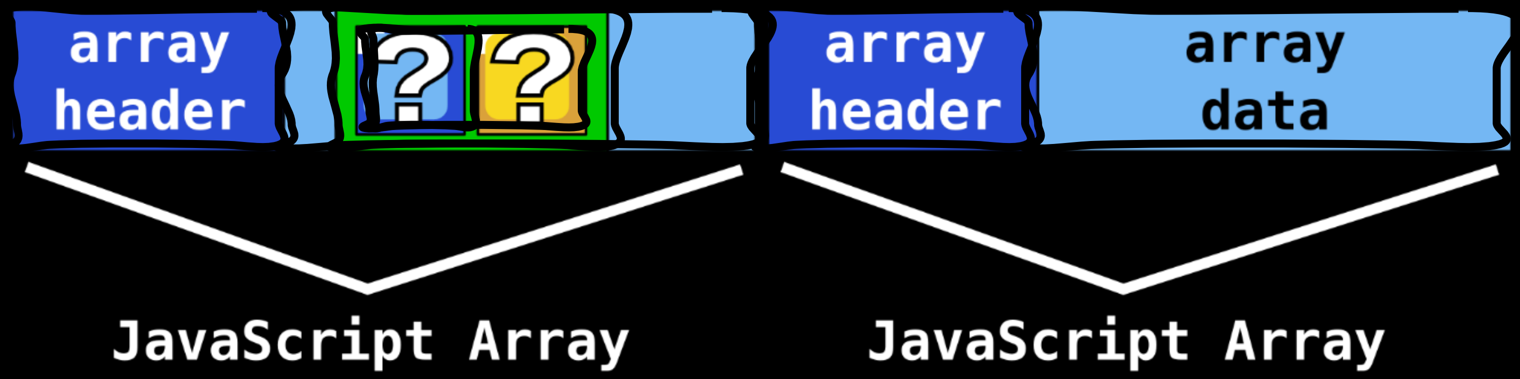
Fake JavaScript Uint8Array



JavaScript Array

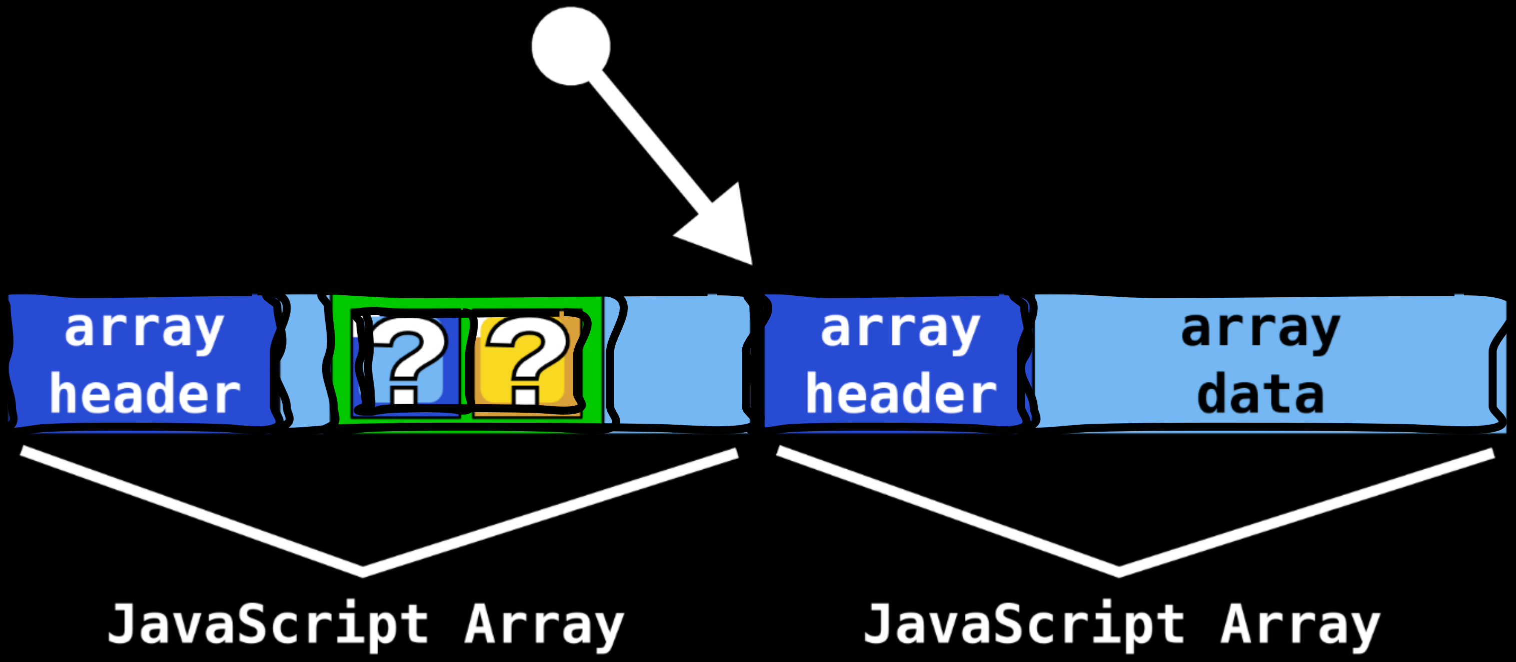
Dedup Est Machina: Creating a Fake Object

Fake JavaScript Uint8Array



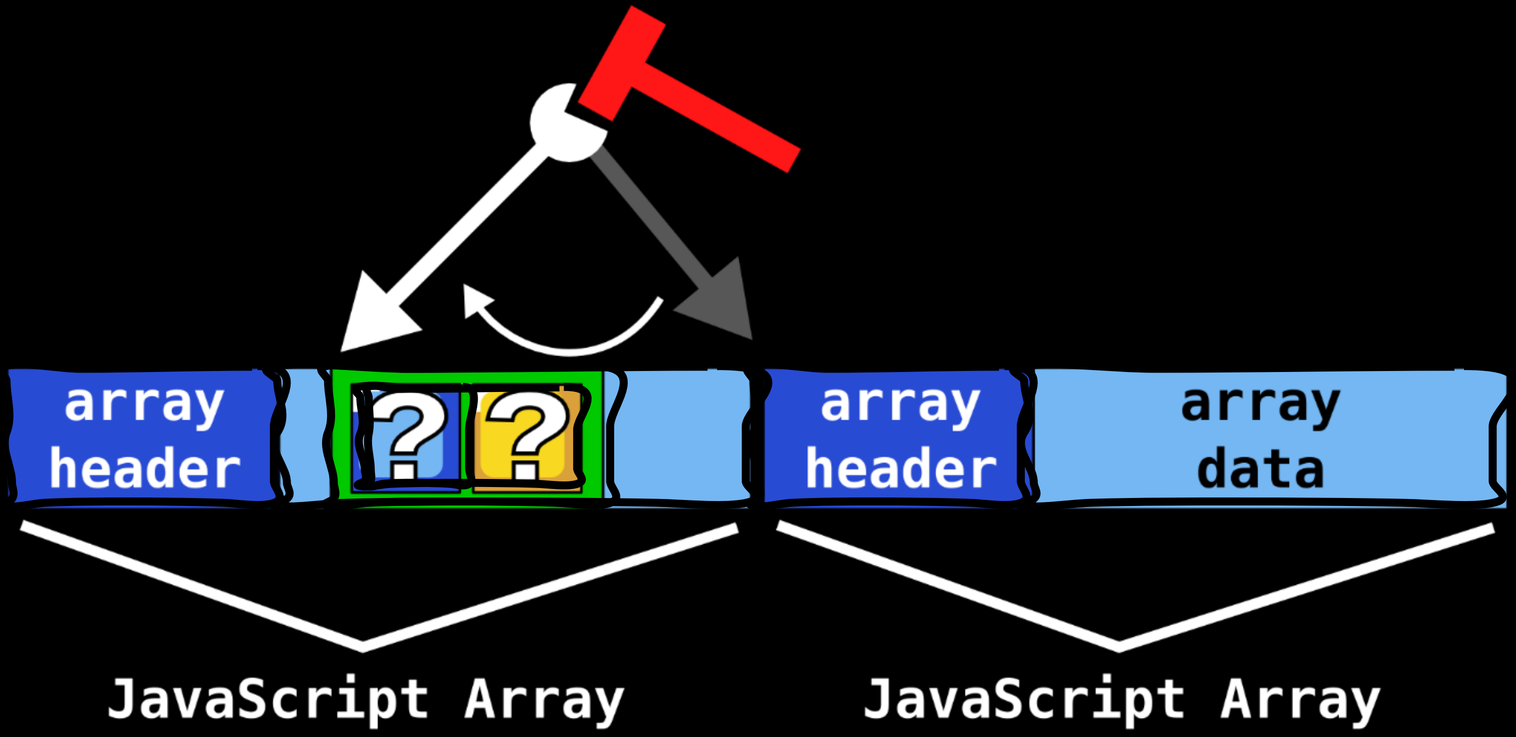
Dedup Est Machina: Creating a Fake Object

Fake JavaScript Uint8Array



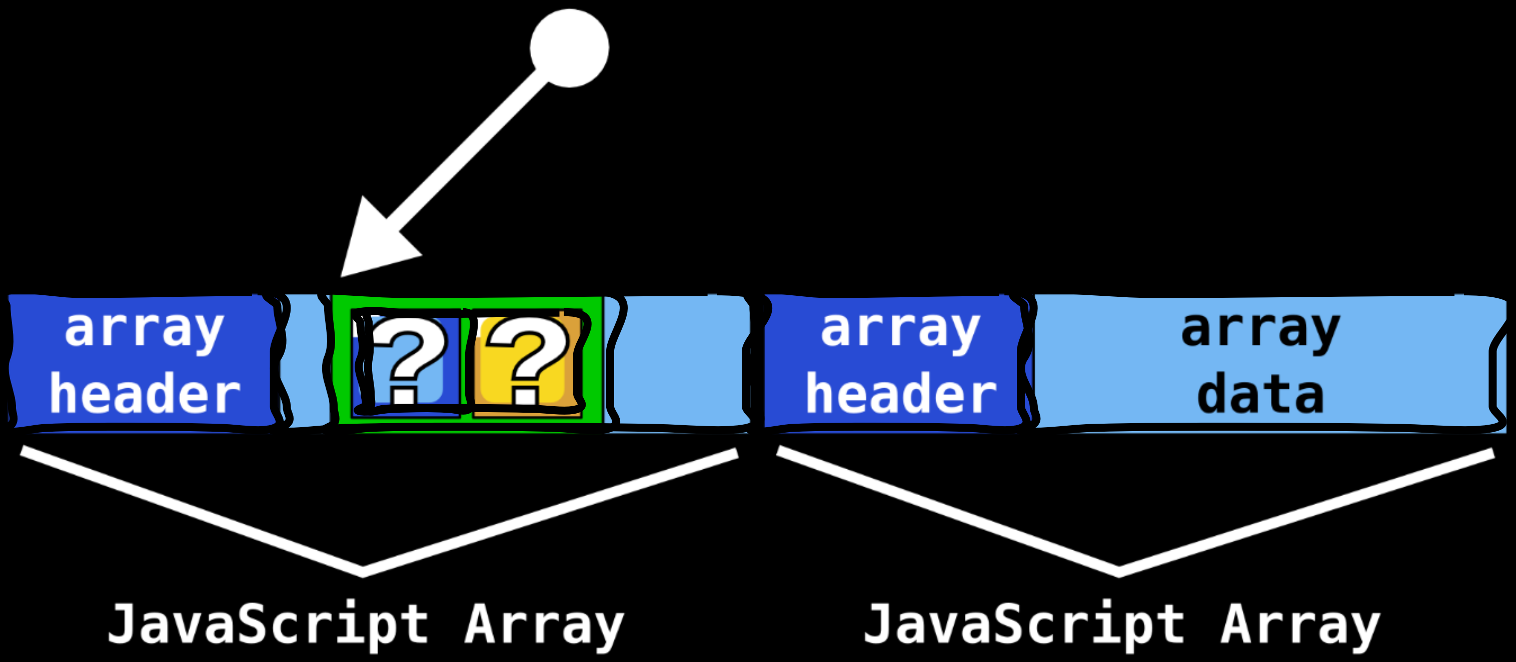
Dedup Est Machina: Creating a Fake Object

Pointer Pivoting



Dedup Est Machina: Creating a Fake Object

Pointer Pivoting



Cashing in...

Microsoft Bounty Program: $100,000

Cashing in...

Microsoft Bounty Program: $100,000

“Well, can you refrain from publishing?”

Cashing in...

Microsoft Bounty Program: $100,000

“Well, can you refrain from publishing?”

But, but, we observed the 90 days!

Cashing in…

Microsoft Bounty Progr m: $100,000

“Well, can you ref in om pu shing?

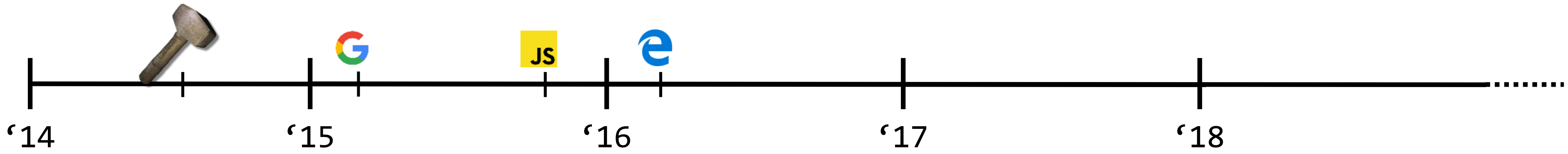
t, b , we o served the 90 days!

“Yes, well. Sorry!”

$0,-

Only the beginning

What else can we attack?

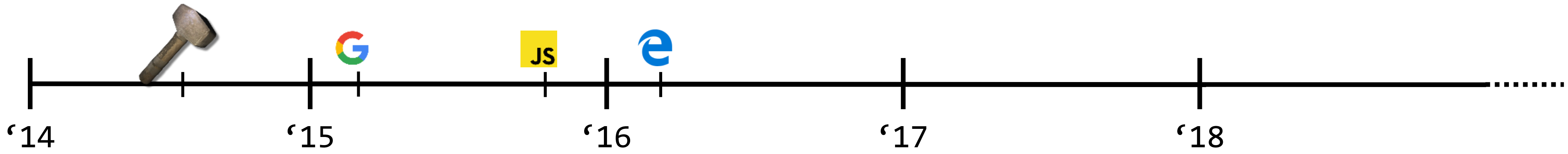


[1] CMU finds first bit flip (2014)

[2] Google Project Zero: 1st Rowhammer root Exploit (flipping PTEs)

[3] Rowhammer.js: 1st RH bit flip in JavaScript

[4] Dedup est Machina: Breaking Microsoft Edge’s sandbox



[1] CMU finds first bit flip (2014)

[2] Google Project Zero: 1st Rowhammer root Exploit (flipping PTEs)

[3] Ro

[4] De

What about the cloud?

Goal 2

Bug-free Exploitation in Clouds

Flip Feng Shui

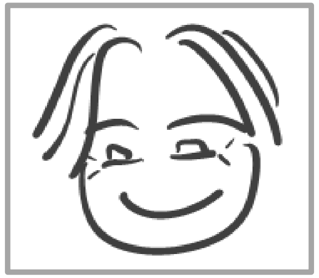
Ben Gras

Kaveh Razavi

Erik Bosman

Bart Preneel

Herbert Bos



Cristiano Giuffrida

USENIX Security
2017

Flip Feng Shui

Published at USENIX Security 2016

with Ben, Kaveh, Erik, Herbert, and Bart (KU Leuven)

System-wide exploits in public KVM clouds

...without relying on a single software bug

Flip Feng Shui: Overview

Rowhammer

(hardware glitch)

Flip Feng Shui: Overview

Rowhammer
(hardware glitch)
+
Memory deduplication
(physical memory massaging primitive)

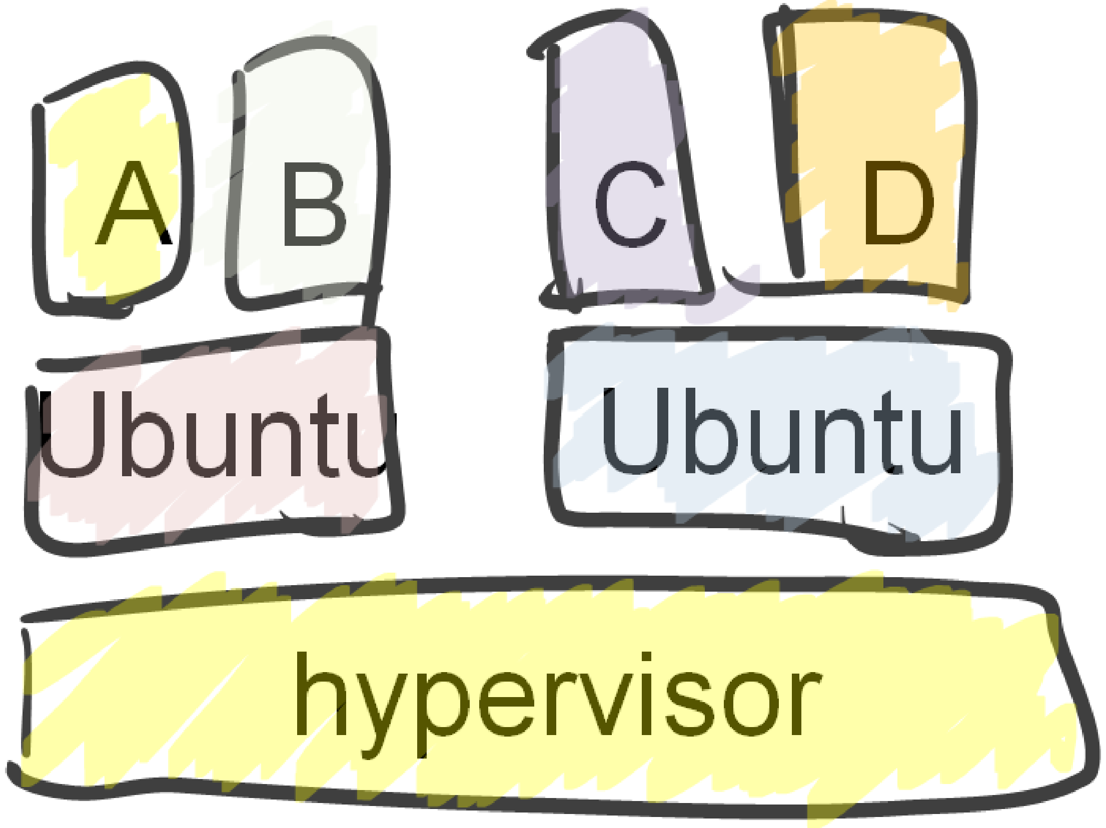
Flip Feng Shui:
Overview

Rowhammer
(hardware glitch)
+
Memory deduplication
(physical memory massaging primitive)

Cross-VM compromise in public Linux/KVM clouds
without software bugs

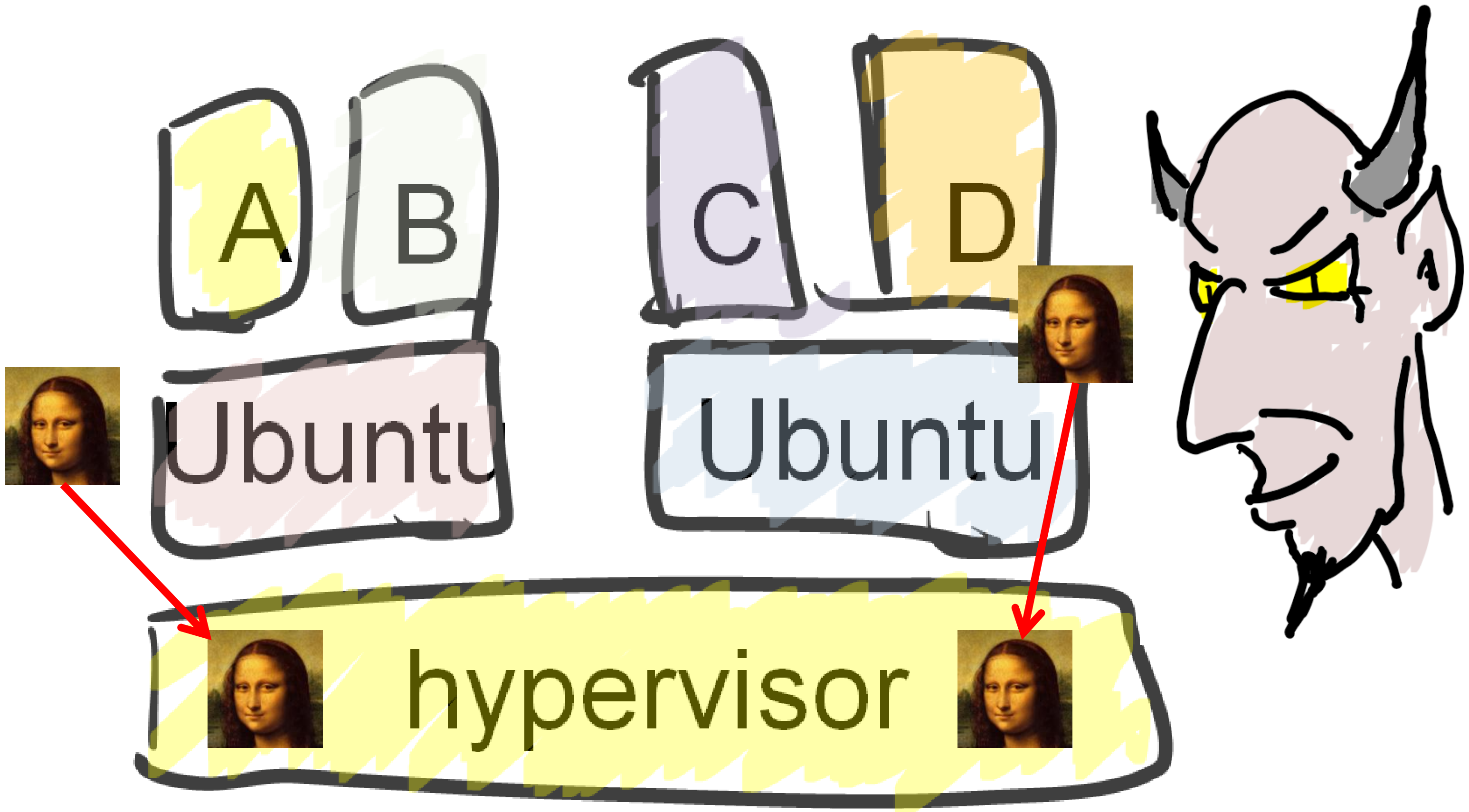
KVM / Clouds

KSM: Kernel Same-page Merging



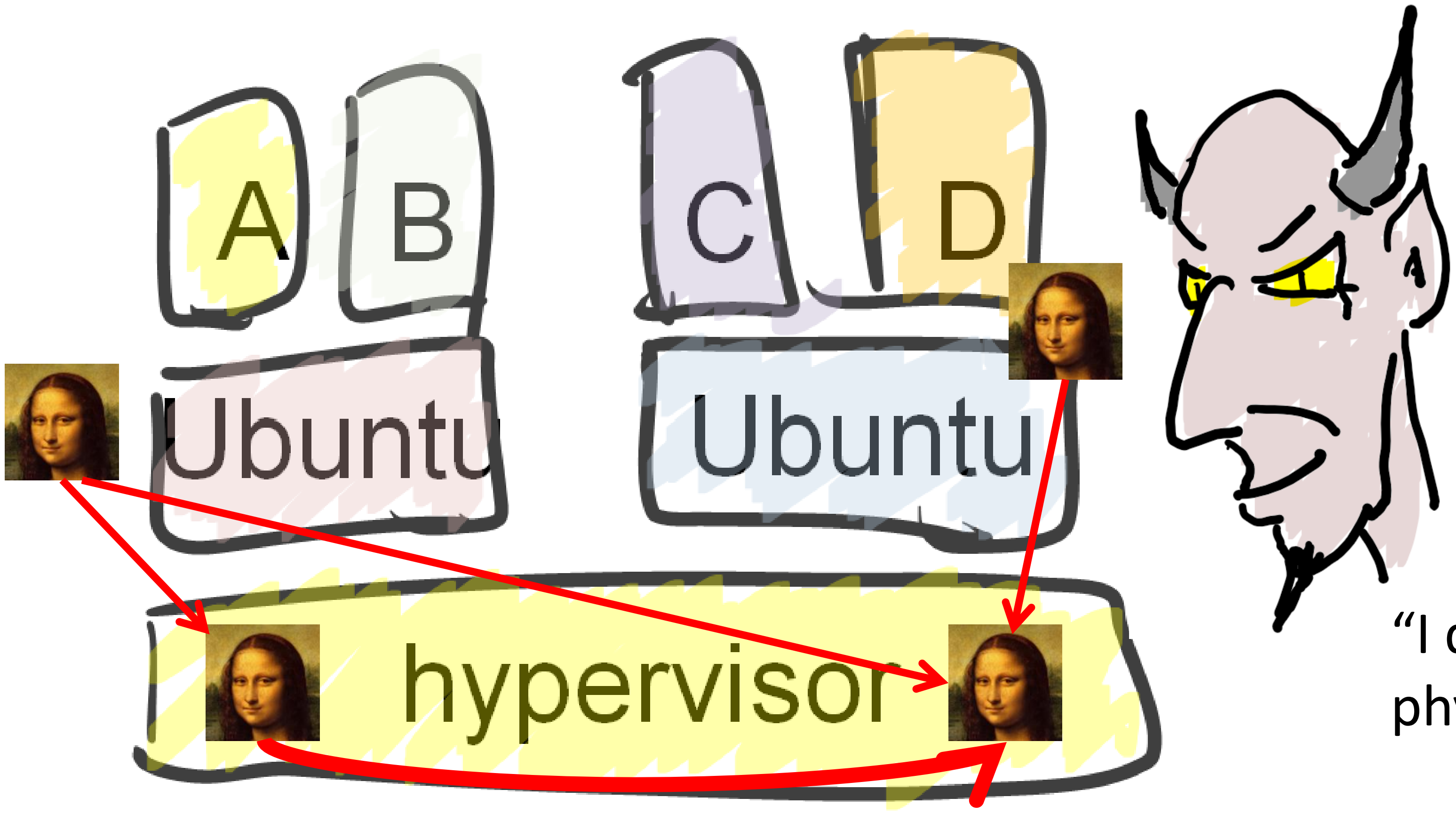
KVM / Clouds

KSM: Kernel Same-page Merging



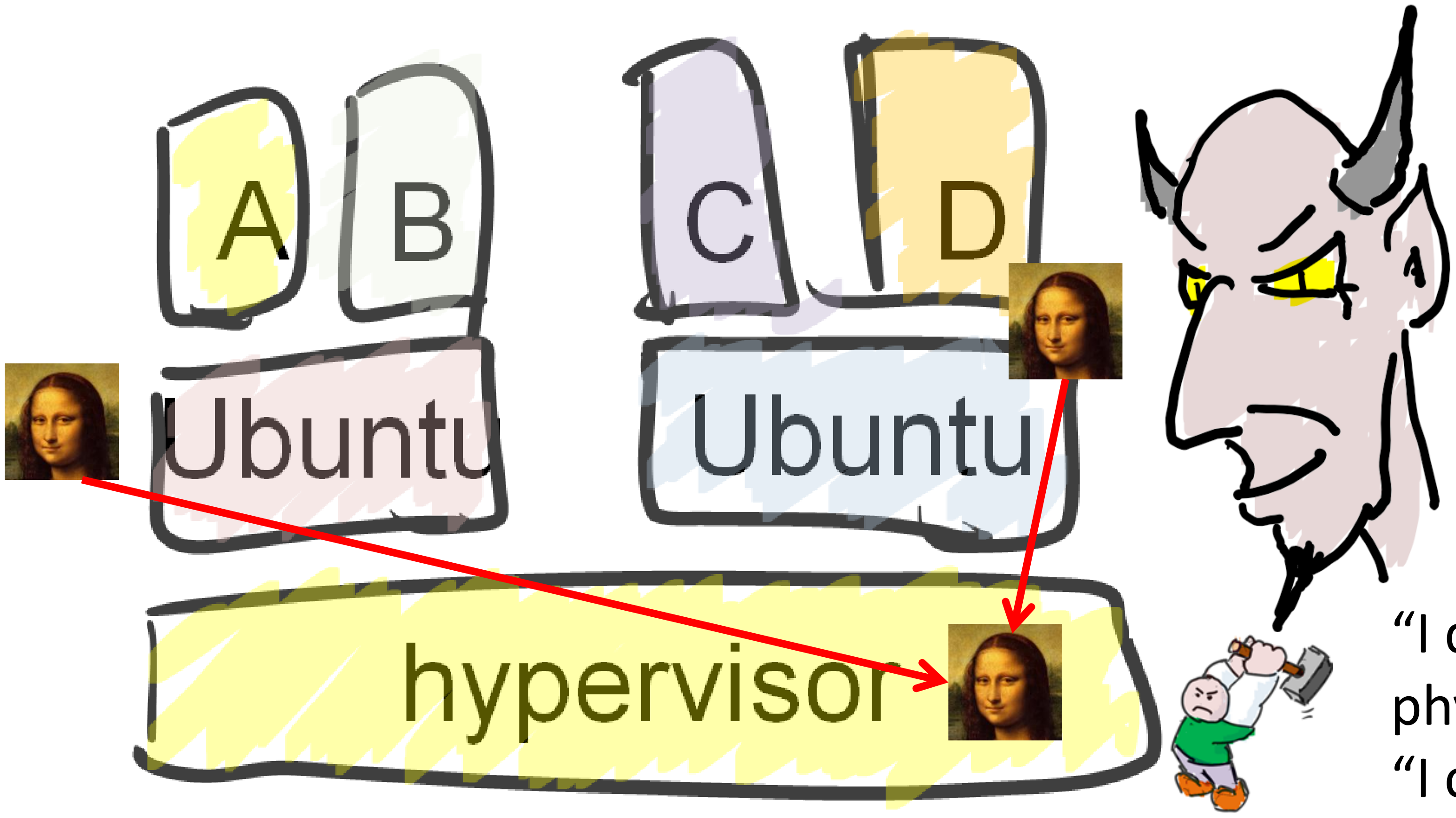
KVM / Clouds

KSM: Kernel Same-page Merging



“I determine the physical page!”

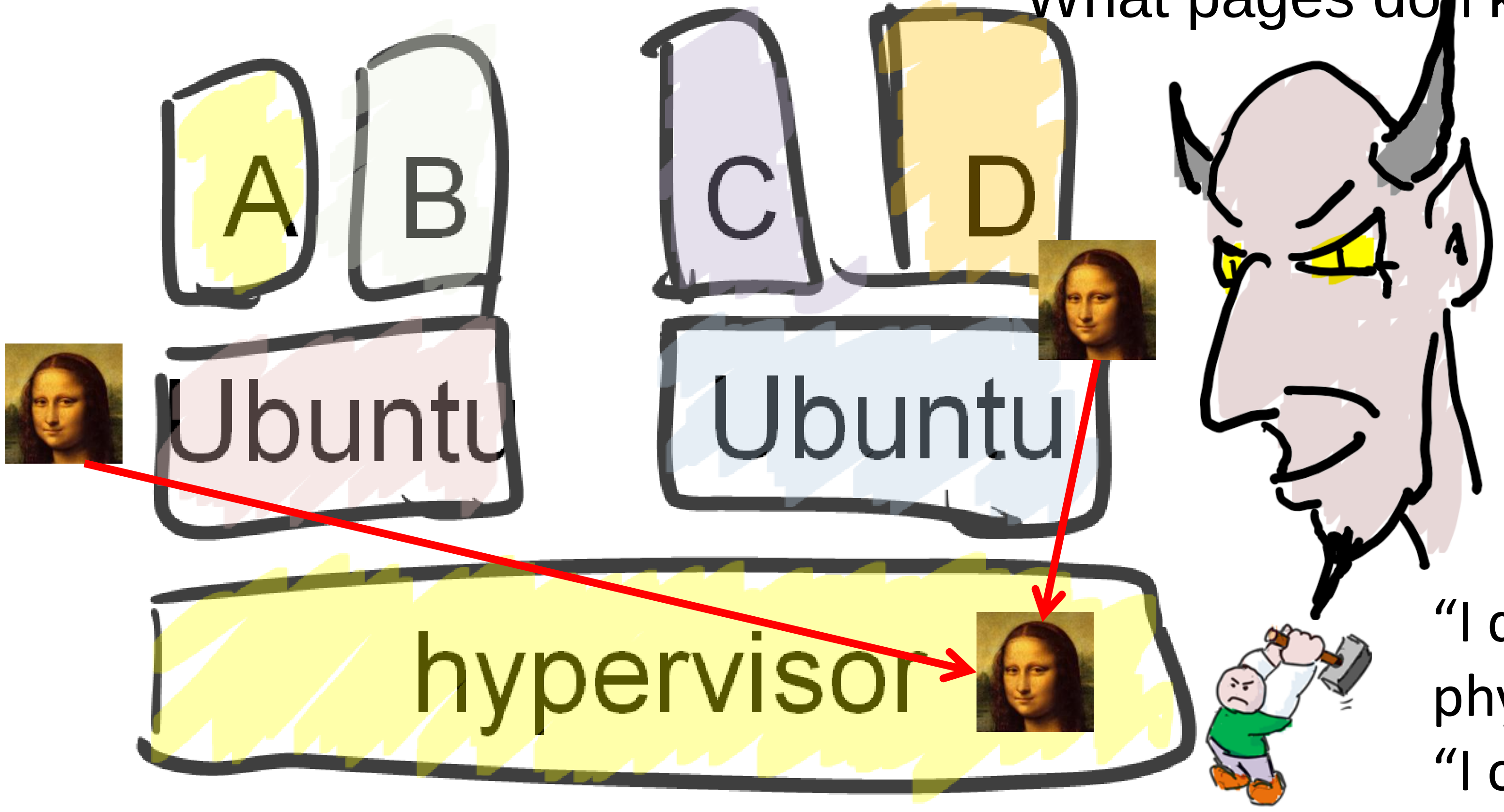
KVM / Clouds



"I determine the physical page!"
"I can hammer it!"

KVM / Clouds

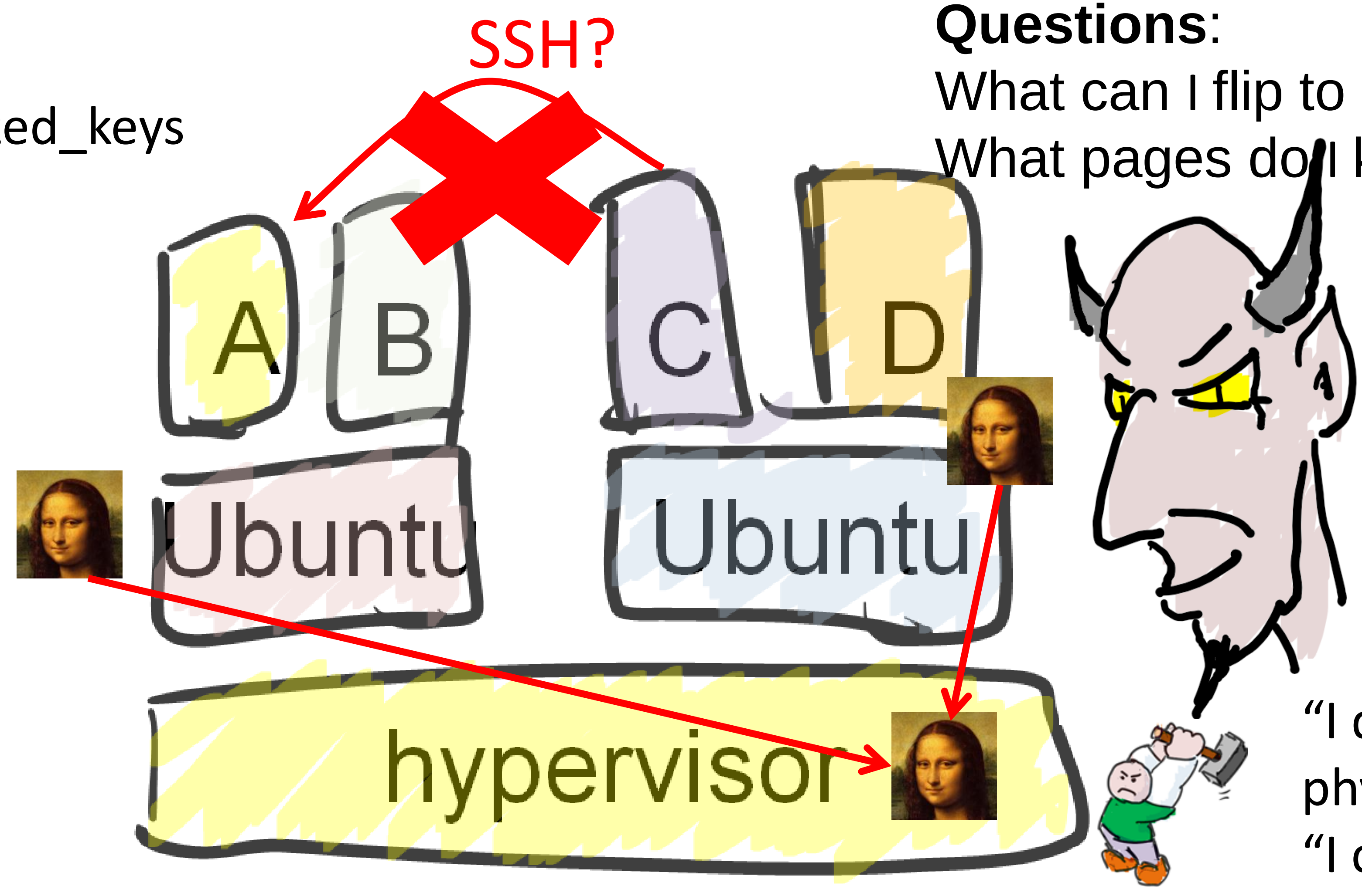
Questions:
What can I flip to gain access?
What pages do I know?



"I determine the physical page!"
"I can hammer it!"

KVM / Clouds

Check .authorized_keys



Questions:
What can I flip to gain access?
What pages do I know?

"I determine the physical page!"
"I can hammer it!"

Public keys are not secret

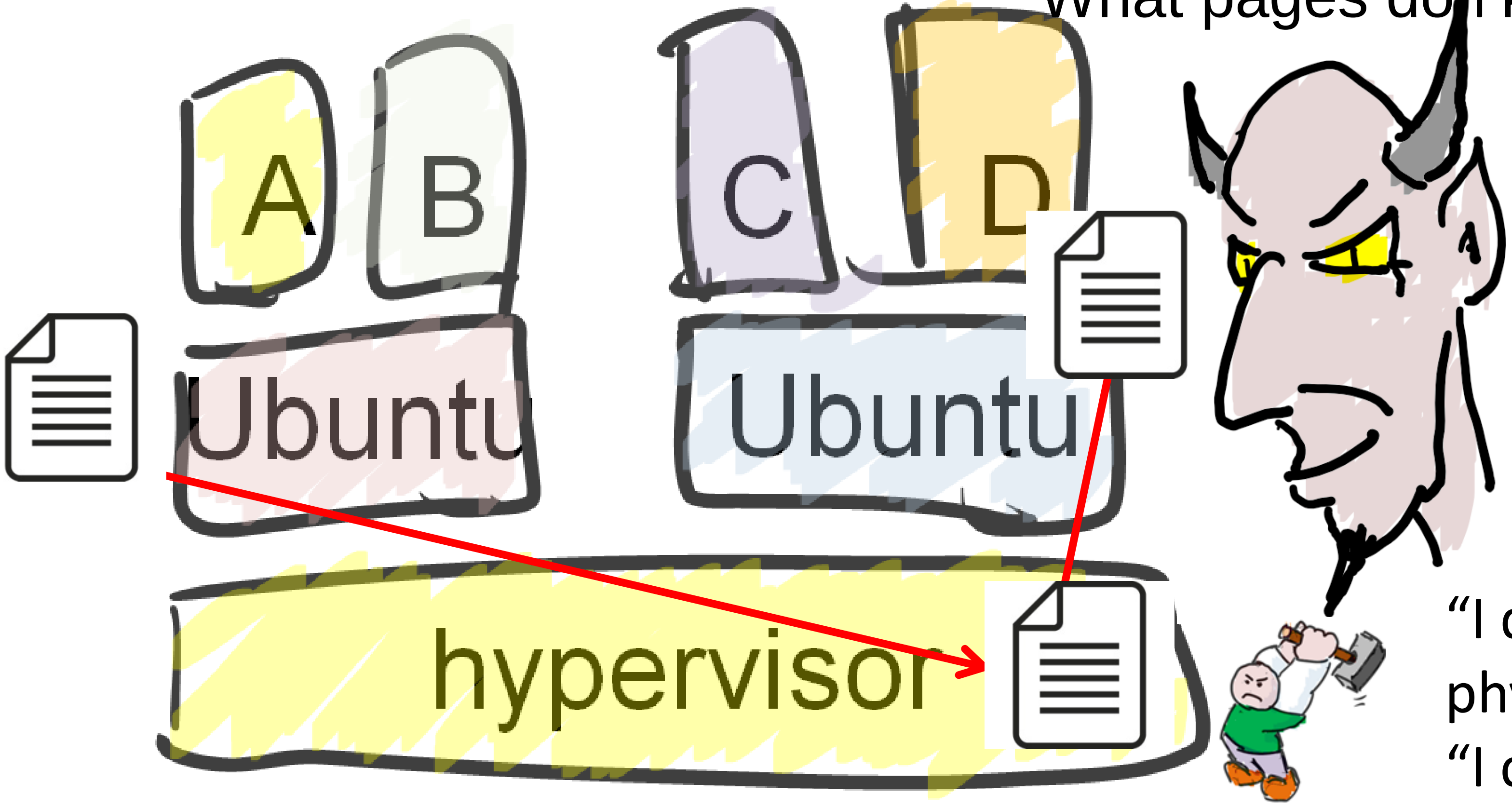
ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEA0ibAEmysI4o1zfb4dOJIyaN67pya8
AnOozVewilpv560jiagTzwrPG8bmK4GL3KEUc3lxZ/Xhj7RvdOD0qMAx0fFB
3r80ZSy1KlkIXwKumUY+YBMyn1xdMluWS/J4JWKBpuoOMNTGy7QdCPI
Hrt07OnwSxvZsoyTsh9QZ/eJv4qR0YaFkAHyH9Si2hTC/6G6CZdXkw93Ly
EtW1ykxxkSJB6JYwB8FsBMcXPvYJ5CiR30fKqo6GP+WTz1xbTbahLLO3
1mx/qSDntcXEYgfpw7Abi8W6LSkExFOxrsKir8QqZregznVeWPiht9kf4PT9
C3WOoDzA0aF1q+g1CJ1EhZow== joe@acme

So we know what is in memory

KVM / Clouds

We move it to a page susceptible to rowhammer

Questions:
What can I flip to gain access?
What pages do I know?



"I determine the physical page!"
"I can hammer it!"

Hammer Time!

A bit flips in the pub key

Makes a **weak key**

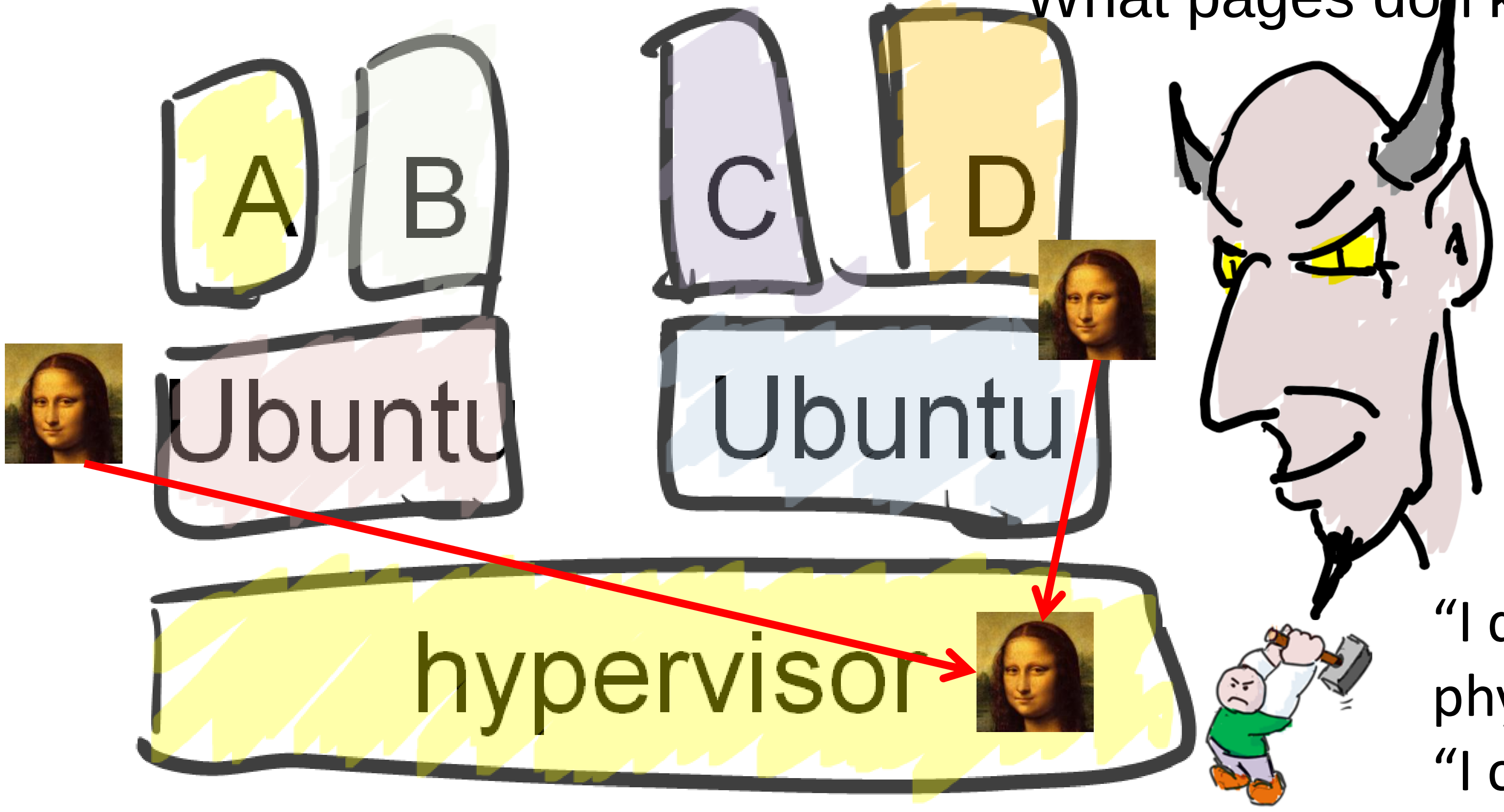
Easy to generate private key

$\Rightarrow$ We do this in minutes!

Better still?

[redacted]

Questions:
What can I flip to gain access?
What pages do I know?



How about updates (APT)?

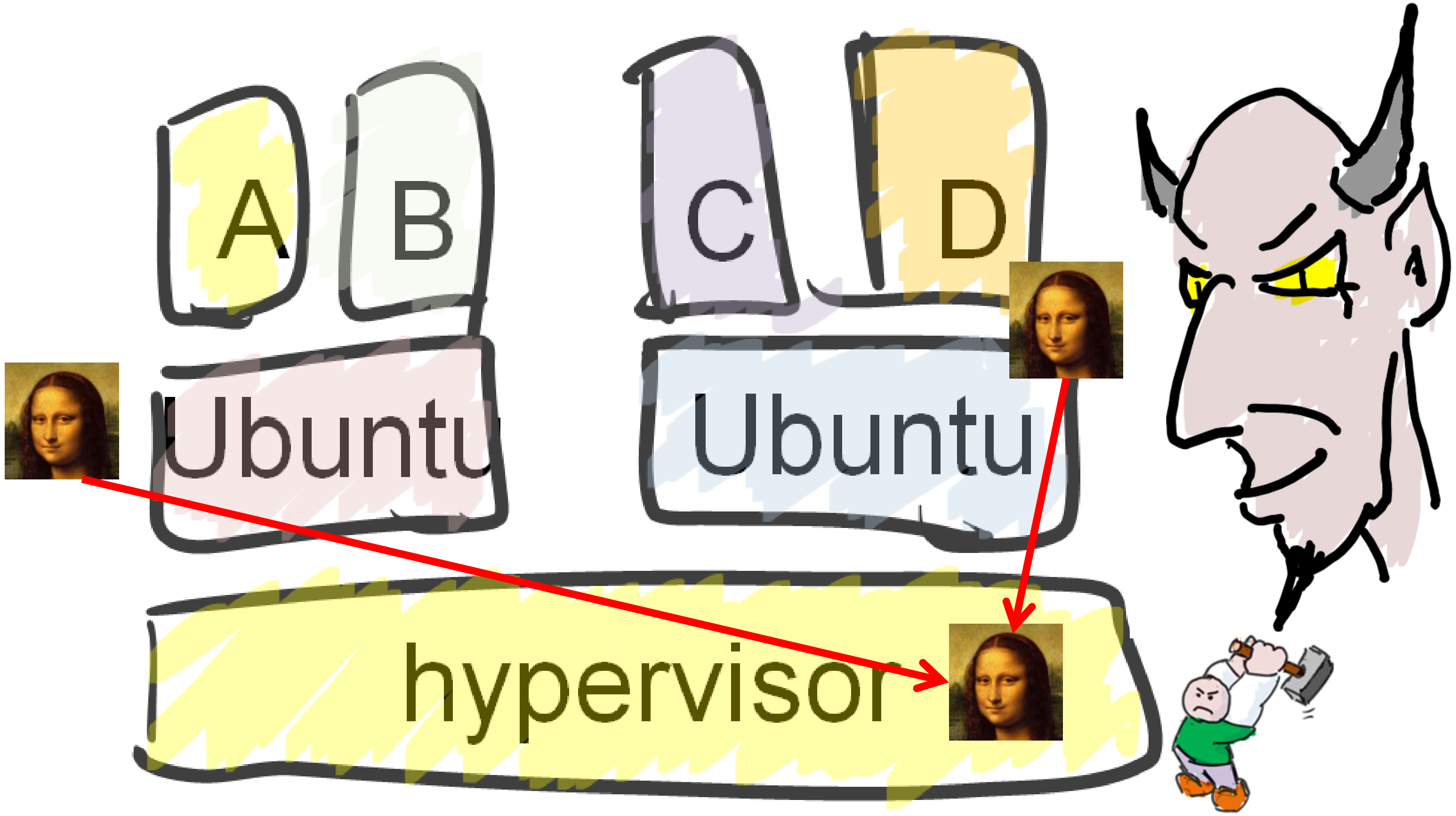
“I determine the physical page!”
“I can hammer it!”

debian.org

ubuntu.com

APT

`sources.list`: from which to install packages & updates

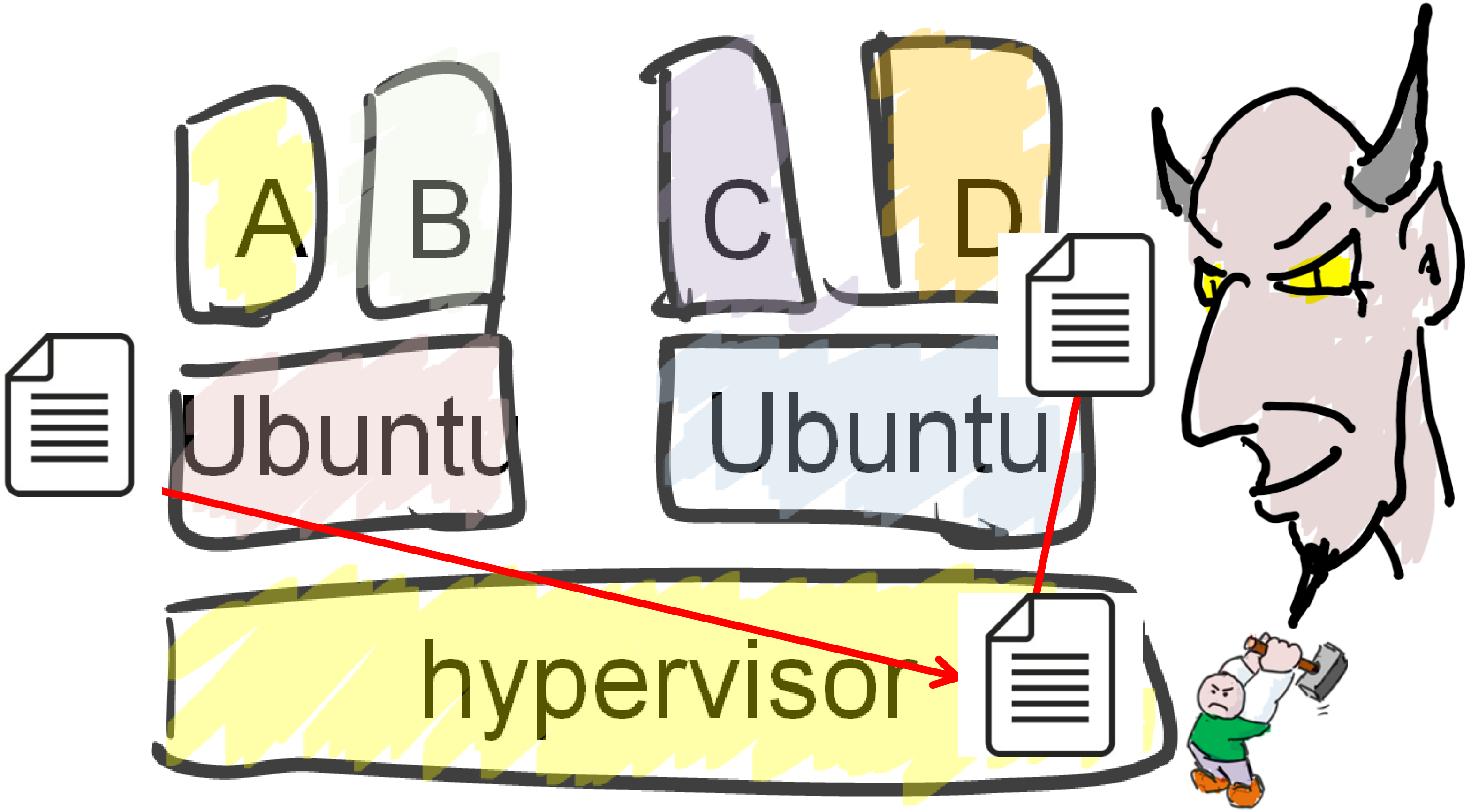


Using dedup, we move `sources.list` to page susceptible to rowhammer

debian.org
ubuntu.com

APT

`sources.list`: from which to install packages & updates



Hammer Time!

A bit flips…

Now we install from

ubunvu.com

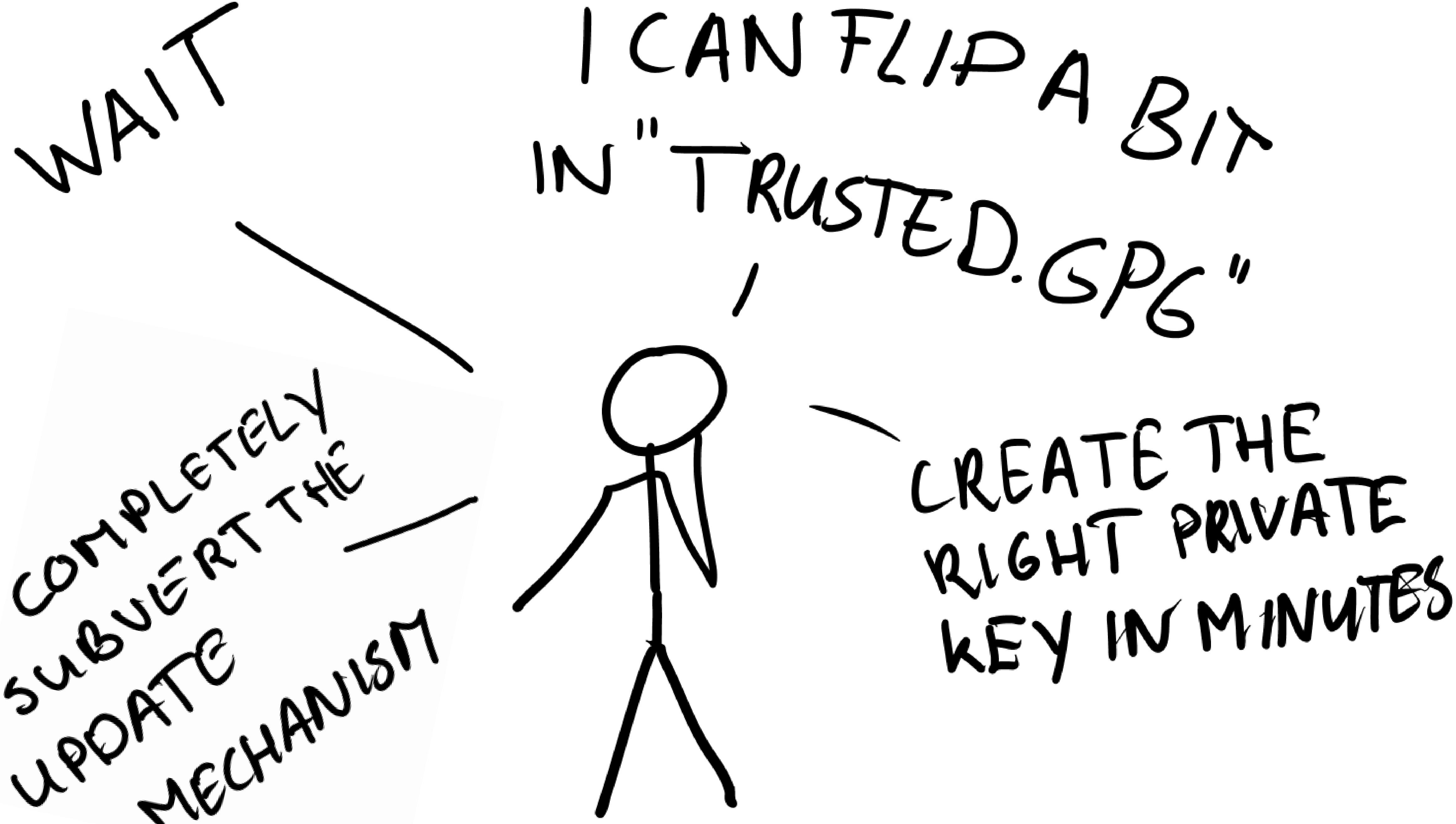
ucuntu.com

…

(which we own)

But fortunately, the packages are signed!

Public key of legitimate apt server in “trusted.gpg”

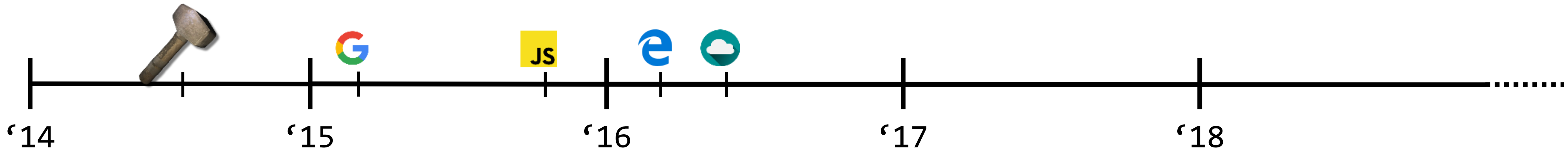

I CAN FLIP A BIT IN "TRUSTED.GPG"
CREATE THE RIGHT PRIVATE KEY IN MINUTES
COMPLETELY SUBVERT THE UPDATE MECHANISM
WAIT

BREAKING THE INTERNET

$0,-

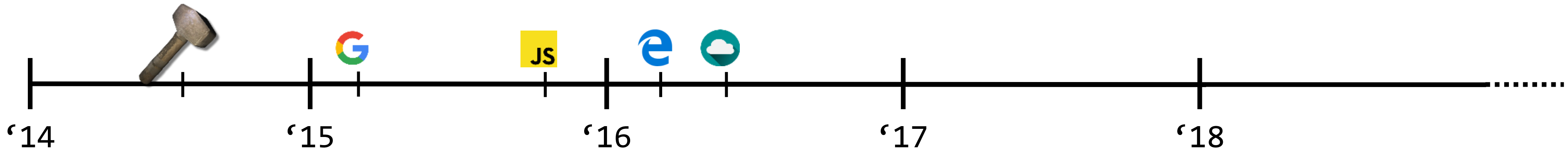
```
Root causes:
 -  unreliable DRAM
 -  push for efficiency (Dedup)
 -  bit flip not part of threat model
```

Rowhammer Evolution



[1] CMU finds first bit flip (2014)
[2] Google Project Zero: 1st Rowhammer root Exploit (flipping PTEs)
[3] Rowhammer.js: 1st RH bit flip in JavaScript
[4] Dedup est Machina: Breaking Microsoft Edge’s sandbox
[5] Flip Feng Shui: Breaking the cloud

Rowhammer Evolution



[1] CMU finds first bit flip (2014)
[2] Goo
[3] Row
[4] Ded
[5] Fli

Is this even possible on ARM?

Goal 3

Bug-free Exploitation on Phones

Drammer:
Deterministic Rowhammer Attacks on Mobile Platforms

CCS'16

Victor van der Veen[1], Yanick Fratantonio[2], Martina Lindorfer[2], Daniel Gruss[3], Clémentine Maurice[3], Giovanni Vigna[2], Herbert Bos[1], Kaveh Razavi[1], and Cristiano Giuffrida[1]

[1]Vrije Universiteit Amsterdam, [2]UC Santa Barbara, [3]TU Graz

We did PCs and clouds

Victor was looking for a project

“How about mobile phones?”

Overview

1. Memory Templating
   Scan memory for useful bit flips
2. Land sensitive data
   Store a crucial data structure on a vulnerable page
3. Reproduce the bit flip
   Modify the data structure and get root access

Overview

1. Memory Templating
   Scan memory for useful bit flips
2. Land sensitive data
   Store a crucial data structure on a vulnerable page
3. Reproduce the bit flip
   Modify the data structure and get root access

Rowhammer on ARM

None of the x86 techniques work

Rowhammer on ARM

None of the x86 techniques work

(We tried)

Rowhammer on ARM

None of the x86 techniques work

(We tried)

(Really hard)

Victor went to… Barbados …and Santa Barbara

“I will work on it there.”

Victor went to… Barbados
…and Santa Barbara

I was worried

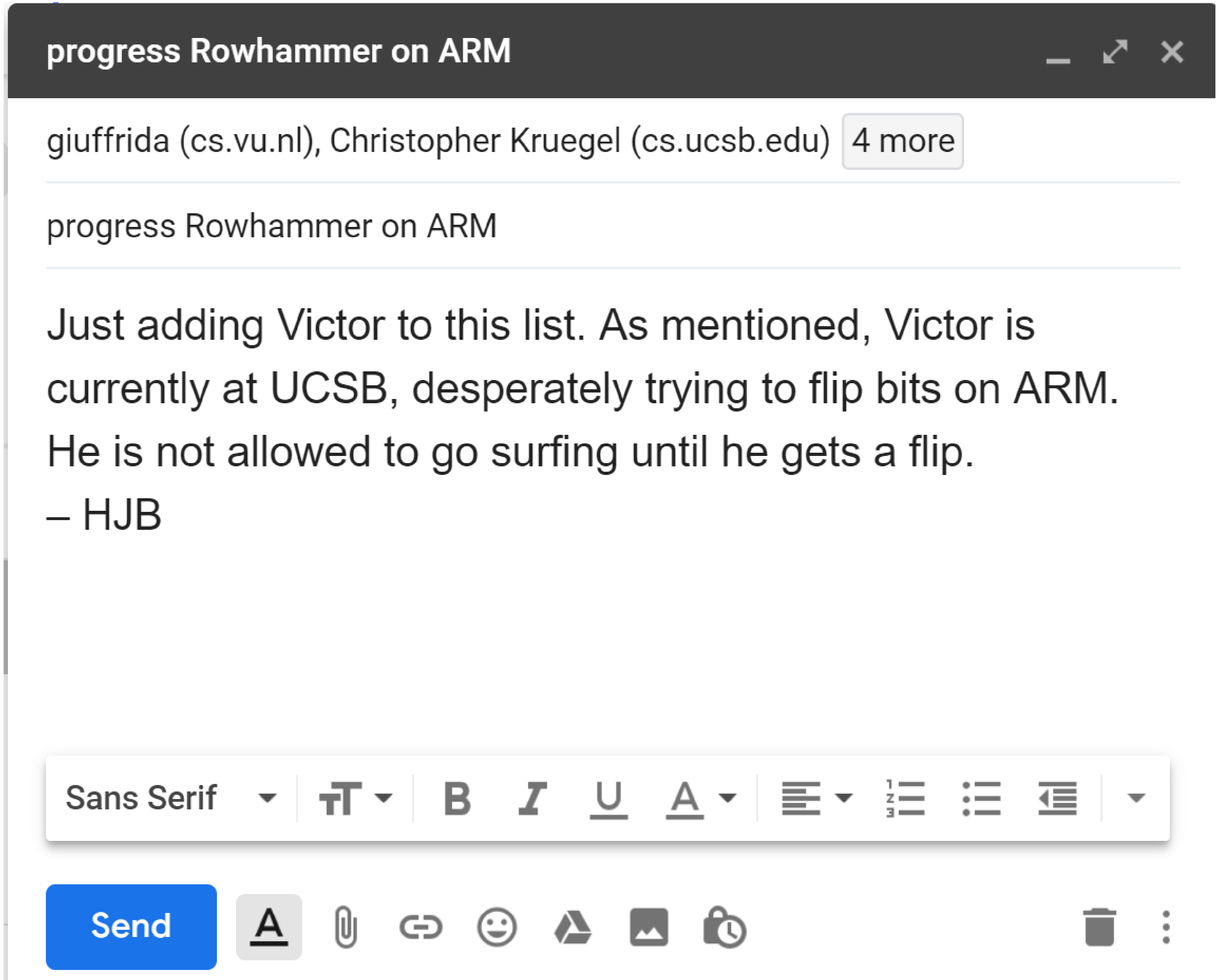
1 week. No results.

3 weeks. No results.

1 month. No result.

So I sent an email.

Email to everyone



Two days later.

Flip.

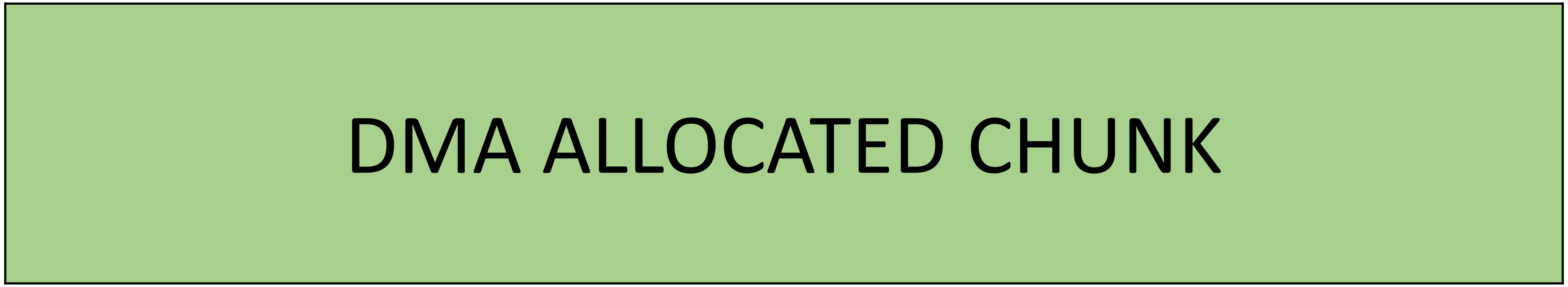
Memory templating on ARM

Direct Memory Access

Android's DMA memory allocator provides everything we need:

- Uncached memory (no **`clflush`** required)
- Physically contiguous memory

Physical memory:



Victor sent me a picture.

Overview

1. **Memory Templating**
   Scan memory for useful bit flips
2. **Land sensitive data**
   Store a crucial data structure on a vulnerable page
3. **Reproduce the bit flip**
   Modify the data structure and get root access

Overview

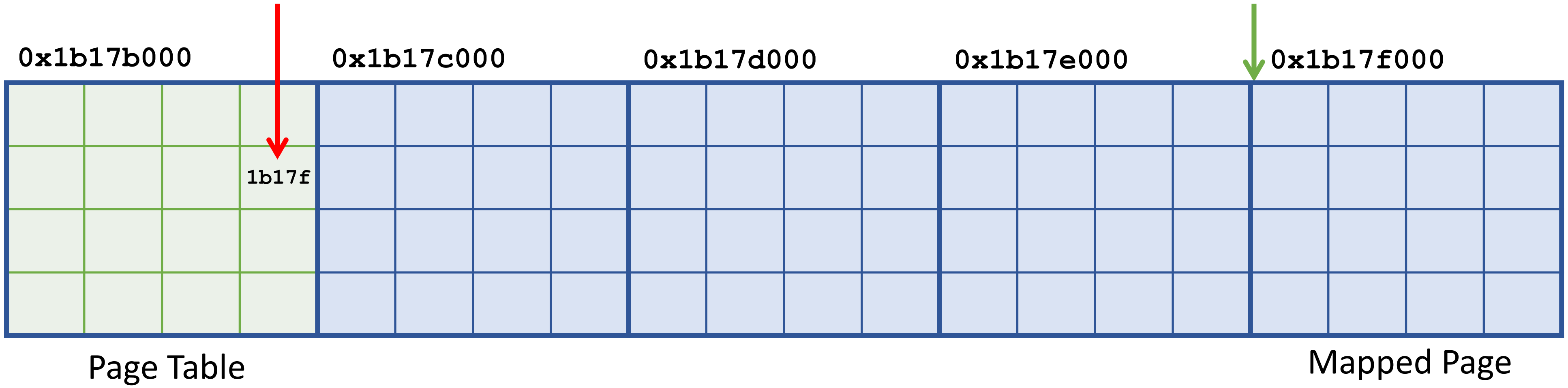
1. Memory Templating
   Scan memory for useful bit flips
2. Land a page table
   Store a page table on a vulnerable page
3. Reproduce the bit flip

But why?

Deterministic Attacks on
Page Table Entries

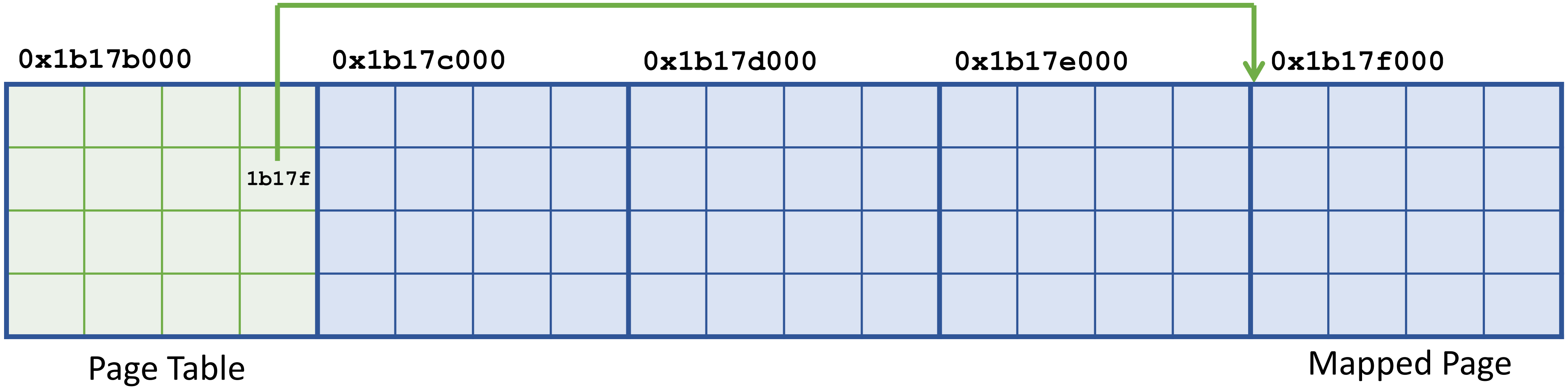
Say we are able to flip bit #14 in a page table entry

PTE: lower 12 bits are properties, so 2nd bit of address



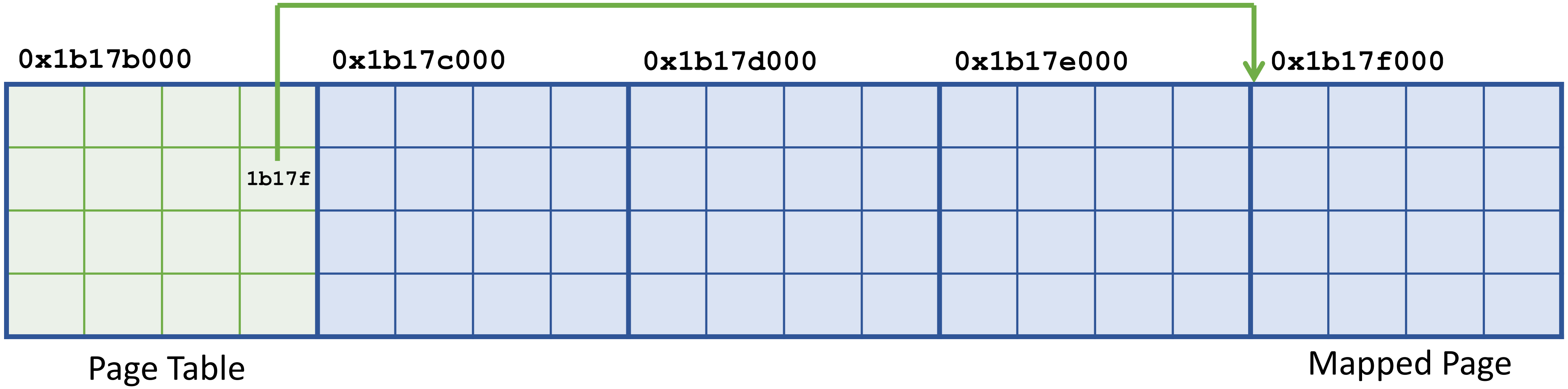
Deterministic Attacks on
Page Table Entries

1. Map a page 4 pages ‘away’ from its page table



Deterministic Attacks on
Page Table Entries

1. Map a page 4 pages 'away' from its page table



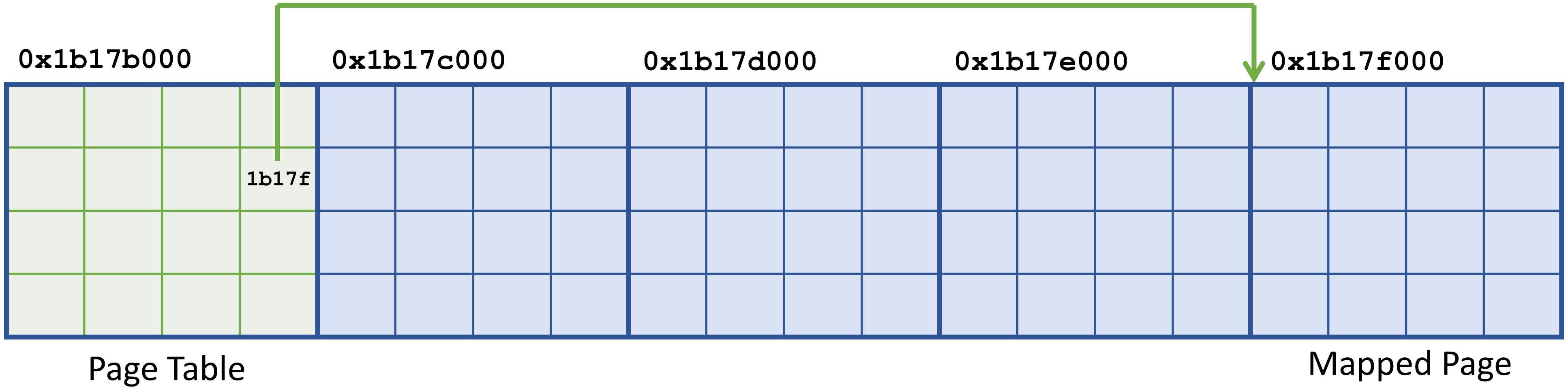
Virtual address `0xb6a57000` maps to Page Table Entry:

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 1 1	x x x x	x x x x	x x x x

which translates to physical page **0x1b17f000**

Deterministic Attacks on
Page Table Entries

1. Map a page 4 pages ‘away’ from its page table
2. Flip bit 2 in the page table entry



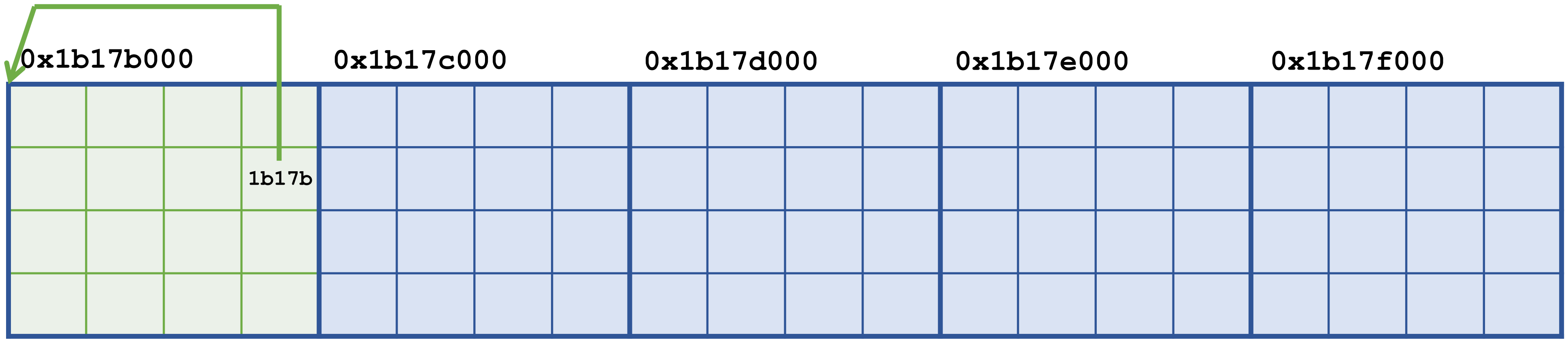
Virtual address `0xb6a57000` maps to Page Table Entry:

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 1 1	x x x x	x x x x	x x x x

which translates to physical page **0x1b17f000**

Deterministic Attacks on
Page Table Entries

1. Map a page 4 pages ‘away’ from its page table
2. Flip bit 2 in the page table entry



Mapped Page Table

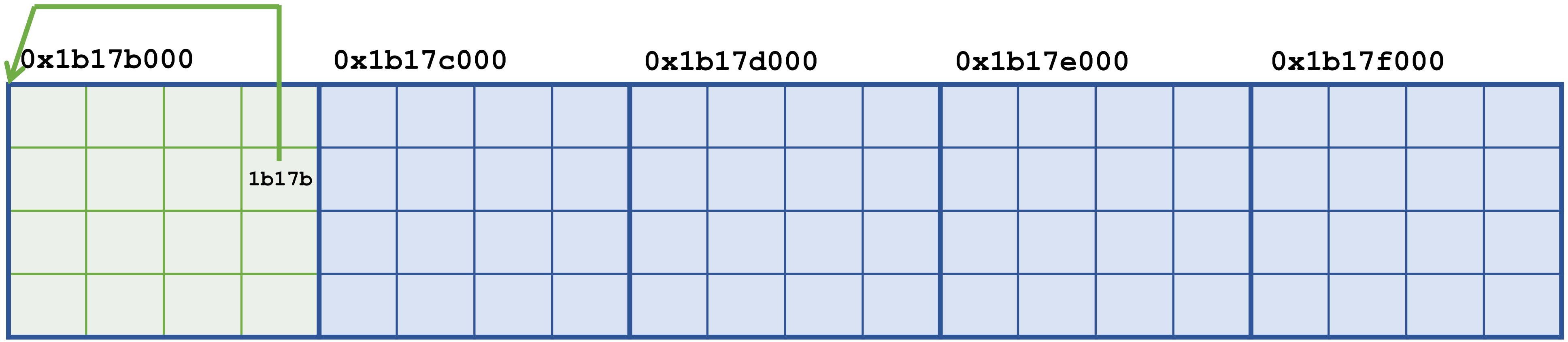
Virtual address `0xb6a57000` maps to Page Table Entry:

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 0 1	x x x x	x x x x	x x x x

which translates to physical page **0x1b17b000**

Deterministic Attacks on
Page Table Entries

1. Map a page 4 pages ‘away’ from its page table
2. Flip bit 2 in the page table entry
3. Write page table entries



Mapped Page Table

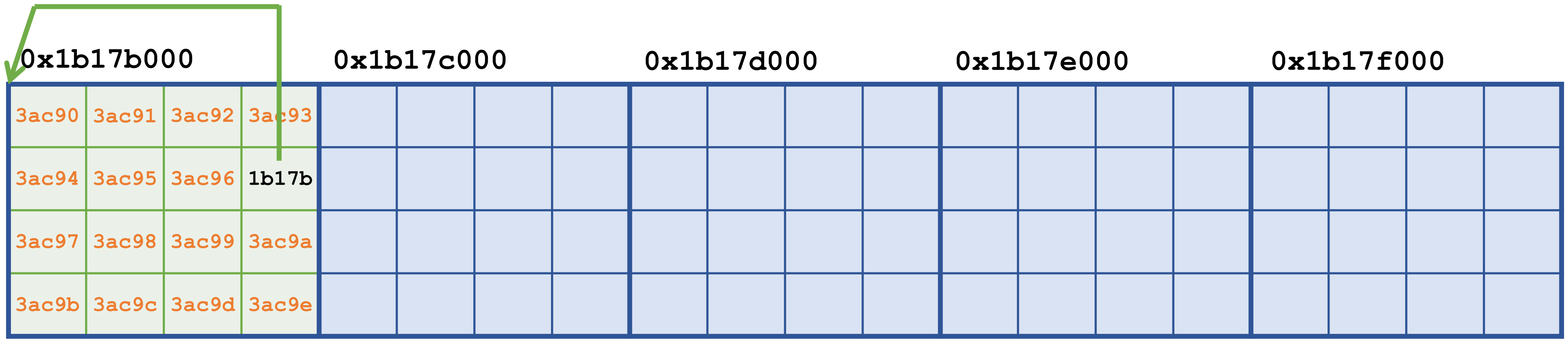
Virtual address `0xb6a57000` maps to Page Table Entry:

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 0 1	x x x x	x x x x	x x x x

which translates to physical page **0x1b17b000**

Deterministic Attacks on
Page Table Entries

1. Map a page 4 pages ‘away’ from its page table
2. Flip bit 2 in the page table entry
3. Write page table entries



Mapped Page Table

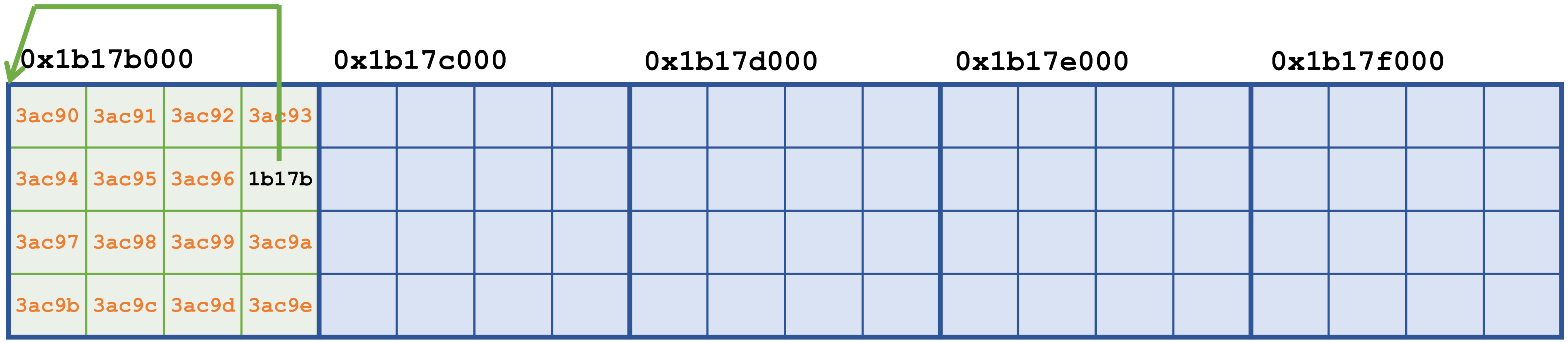
Virtual address `0xb6a57000` maps to Page Table Entry:

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 0 1	x x x x	x x x x	x x x x

which translates to physical page **0x1b17b000**

Deterministic Attacks on
Page Table Entries

1. Map a page 4 pages 'away' from its page table
2. Flip bit 2 in the page table entry
3. Write page table entries
4. Read/write kernel memory



Mapped Page Table

Virtual address `0xb6a57000` maps to Page Table Entry:

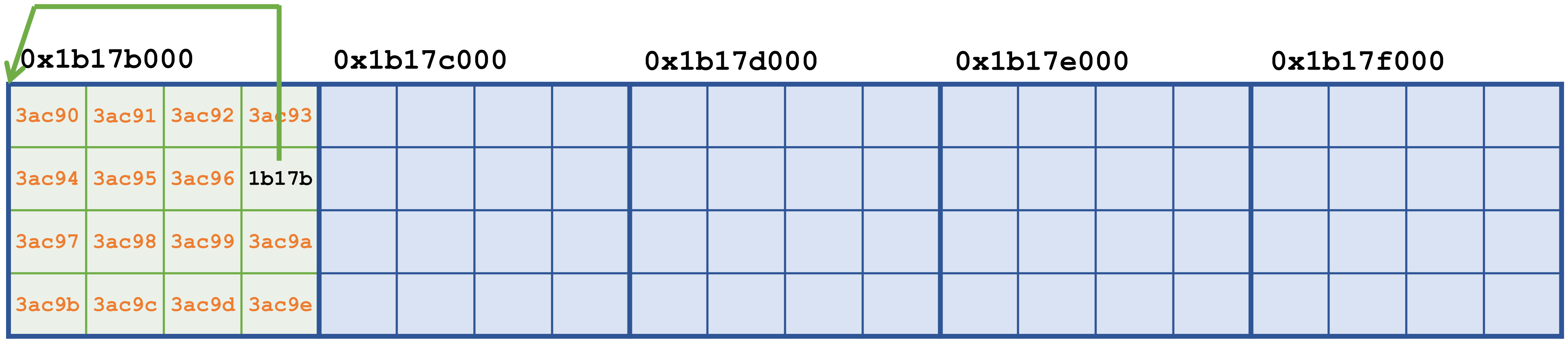
0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 0 1	x x x x	x x x x	x x x x

which translates to physical page **0x1b17b000**

Deterministic Attacks on
Page Table Entries

1. Map a page 4 pages 'away' from its page table
2. Flip bit 2 in the page table entry
3. Write page table entries
4. Read/write kernel memory

Of course, careful Phys Feng Shui needed to ensure PT & Page were mapped at right address, page



Mapped Page Table

Virtual address `0xb6a57000` maps to Page Table Entry:

0 0 0 1	1 0 1 1	0 0 0 1	0 1 1 1	1 1 0 1	x x x x	x x x x	x x x x

which translates to physical page **0x1b17b000**

Overview

1. Memory Templating
   Scan memory for useful bit flips
2. Land sensitive data
   Store a crucial data structure on a vulnerable page
3. Reproduce the bit flip
   Modify the data structure and get root access

Drammer

https://www.vusec.net/projects/drammer/

Published at CCS 2016

CSAW Best Applied Research

Dutch Cyber Security
Research Paper Award, 2017

PWNIE AWARD!

Root causes

Unreliable DRAM

Shared resources

Efficient: give apps direct access to contiguous DMA memory

...

Disclosure

Contacted Google with a list of suggested mitigations on July 25

Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

“Can you publish at another conference, later this year?”

Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

“Can you publish at another conference, later this year?”

“What if we support you financially?”

Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

“Ok, could you then perhaps obfuscate some parts of the paper?”

Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

“Ok, could you then perhaps obfuscate some parts of the paper?”

Rewarded $4000 for a *critical* issue

Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

"Ok, could you then perhaps obfuscate some parts of the paper?"

Rewarded $4000 for a *critical* issue

(because *"it doesn't work on the devices in our Reward Program"*)

Disclosure

Contected Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

“Ok, could you then perhaps obfuscate some parts of the paper?”

Rewarded $4000 for a *critical* issue

(because *“it doesn’t work on the devices in our Reward Program”*)

Now it does

Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

“Ok, could you then perhaps obfuscate some parts of the paper?”

Rewarded $4000 for a *critical* issue

Partial hardening in November’s updates

“We will continue to work on a longer term solution”

Disclosure

Contacted Google with a list of suggested mitigations on July 25

(91 days before #CCS16)

“Ok, could you then perhaps obfuscate some parts of the paper”

Rewarded $4000 for a *critical* issue

Partial hardening in November’s updates

“We will continue to work on a longer term solution”

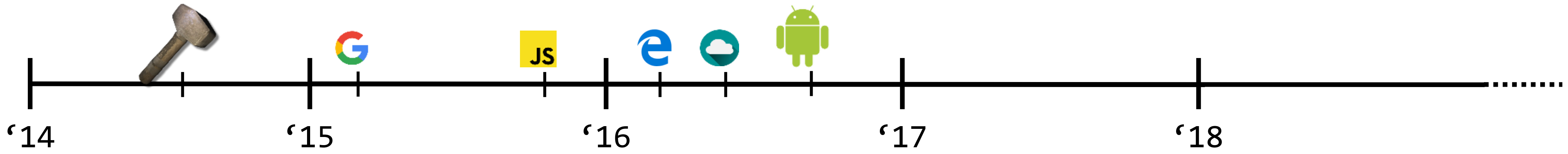
$4000,-

4 months of work

9 people

4 months of work
9 people

Rowhammer Evolution



[1] CMU finds first bit flip (2014)
[2] Google Project Zero: 1st Rowhammer root Exploit (flipping PTEs)
[3] Rowhammer.js: 1st RH bit flip in JavaScript
[4] Dedup est Machina: Breaking Microsoft Edge’s sandbox
[5] Flip Feng Shui: Breaking the cloud
[6] Drammer: rooting android

But not from Javascript...

the grugq @thegrugq · 24 okt. 2016

Cool work, clever hack -- LPE that require installing malicious apps don't put "millions of devices at risk"

Tweet vertalen

2 24 22

Victor van der Veen @vvdveen · 25 ott 2016

I wouldn't be surprised if we could pull this one from a browser actually...

Traduci dalla lingua originale: inglese

1 4 5

the grugq
@thegrugq

Following

In risposta a @vvdveen e @vu5ec

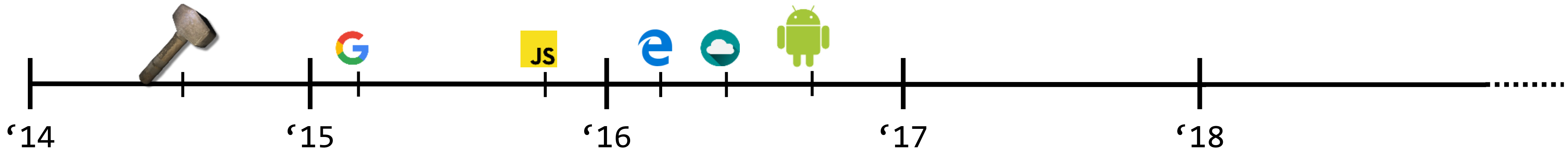
love to see it happen. :)

Goal 4

Bug-free exploitation on Phones

from Javascript

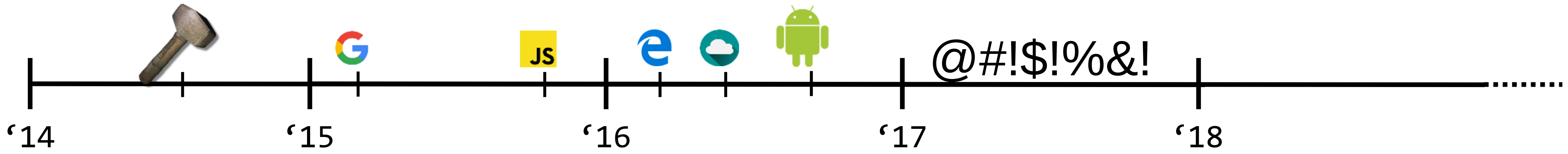
Rowhammer Evolution



[1] CMU finds first bit flip (2014)
[2] Goo
[3] Row
[4] Ded
[5] Fli
[6] Dra

Can we do this from Javascript?

Rowhammer Evolution



[1] CMU finds first bit flip (2014)
[2] Goo
[3] Row
[4] Ded
[5] Fli
[6] Dra

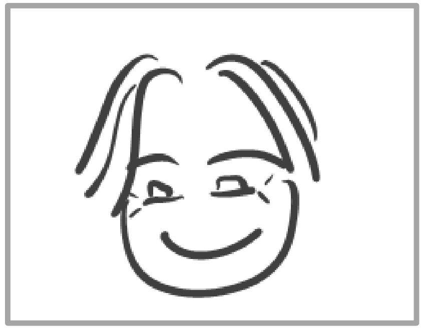
Can we do this from Javascript?

Including the GPU

Pietro Frigo

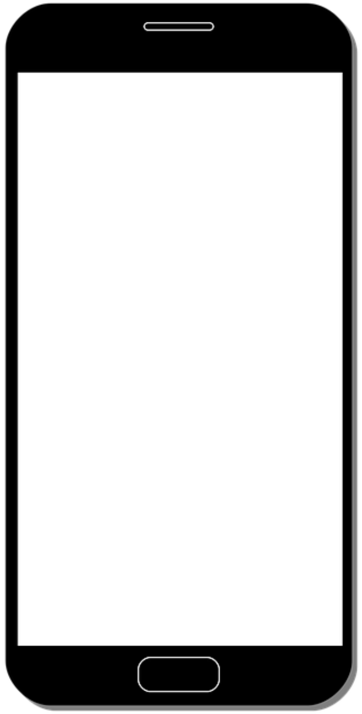
Kaveh Razavi

Herbert Bos

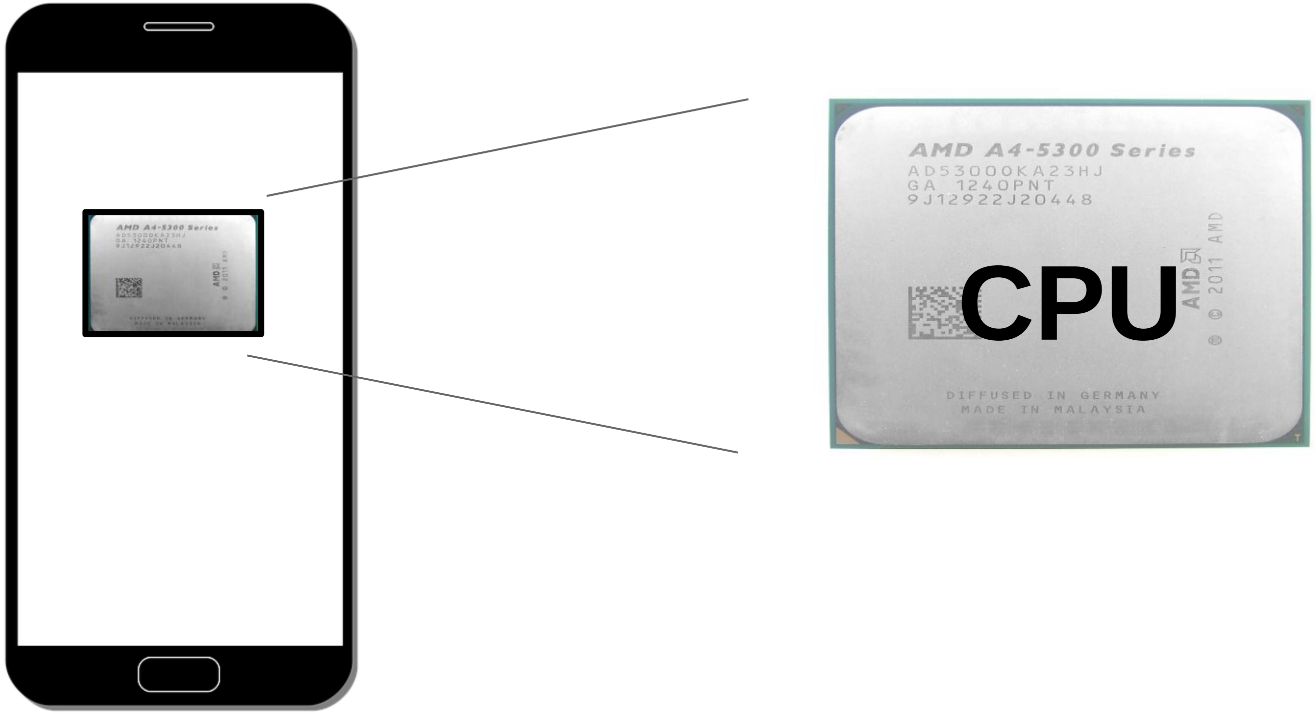


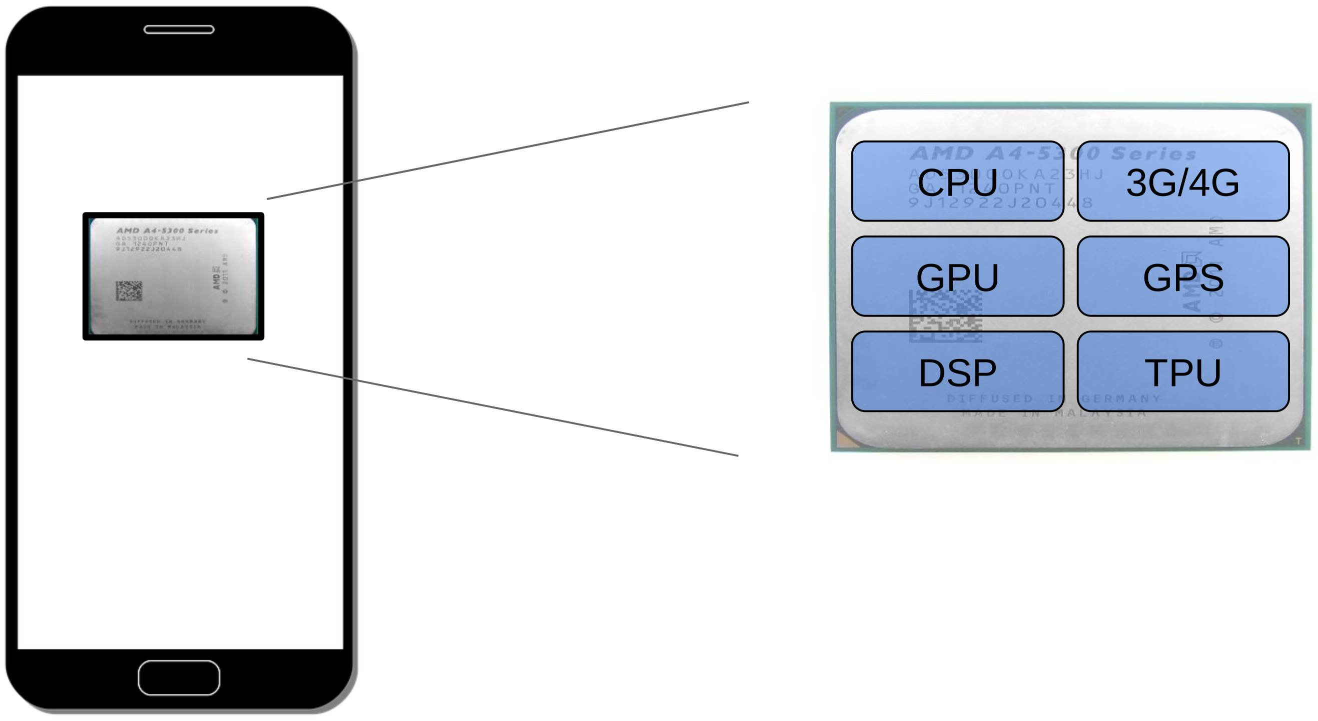
Cristiano Giuffrida

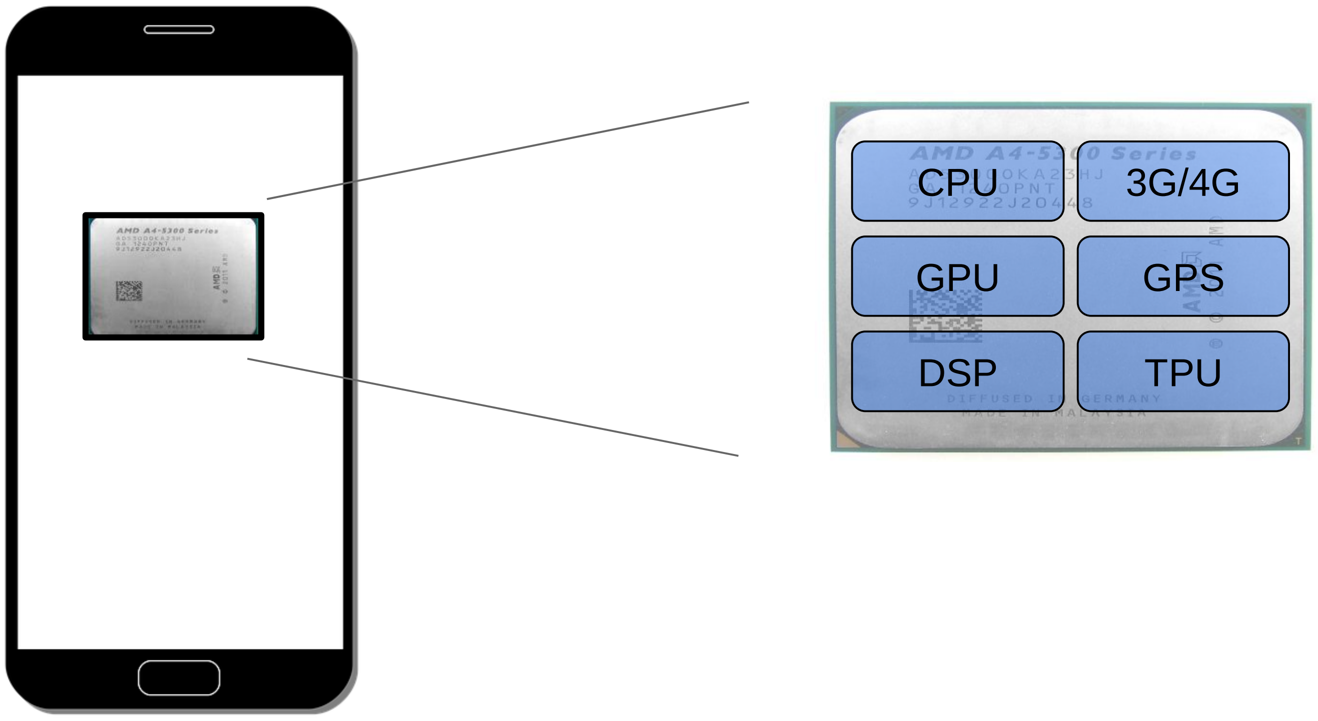
Security & Privacy 2018




AMD A4-5300 Series
MADE IN MALAYSIA

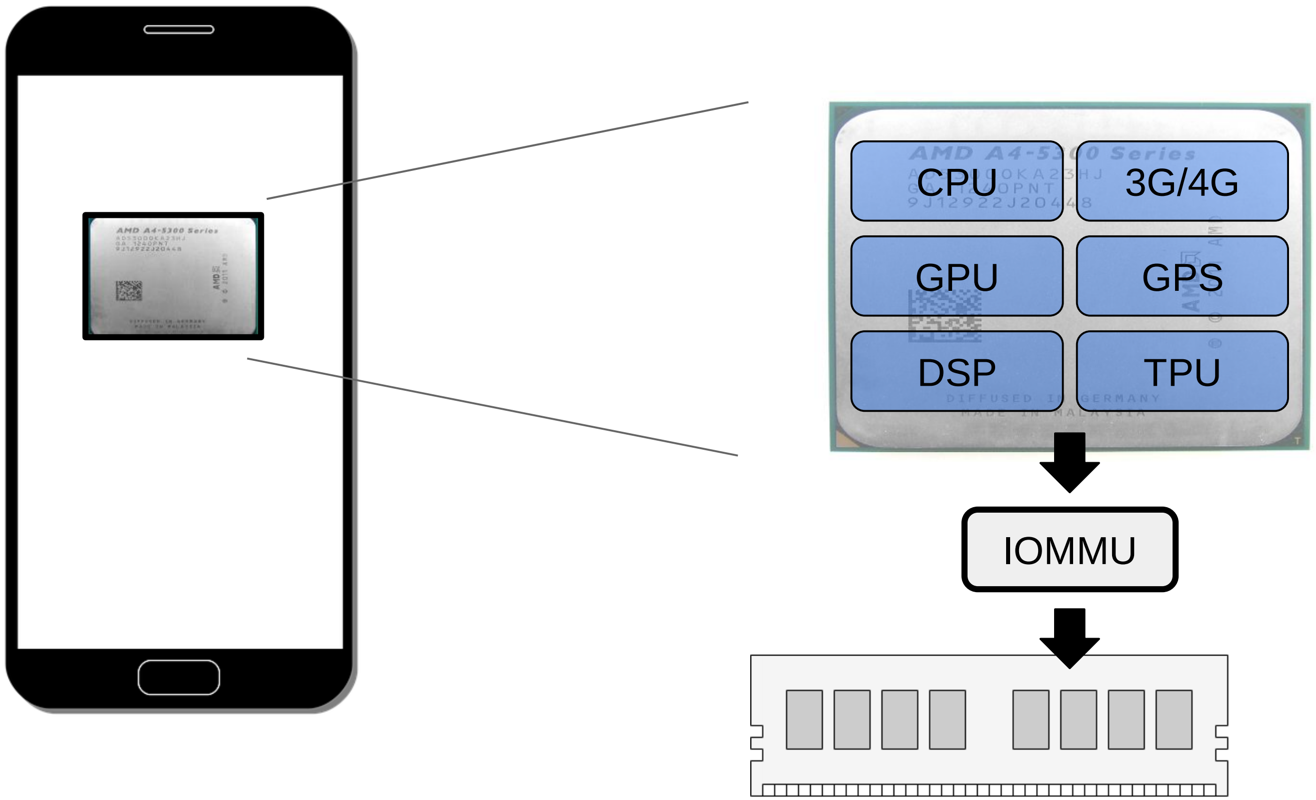

AMD A4-5300 Series
AD53000KA23HJ
GA 1240PNT
9J12922J20448
CPU
AMD
© 2011 AMD
DIFFUSED IN GERMANY
MADE IN MALAYSIA


CPU
3G/4G
GPU
GPS
DSP
TPU



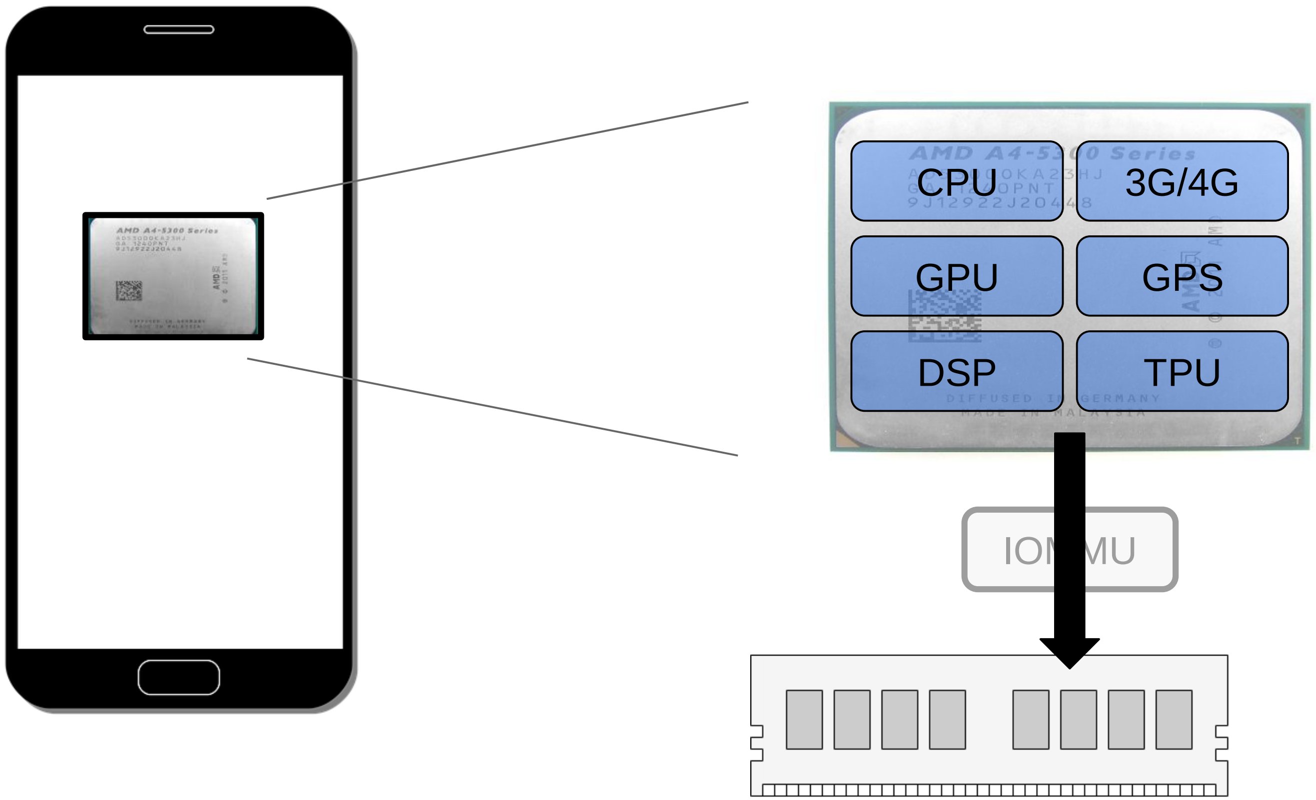
More co-processors

Greater attack surface



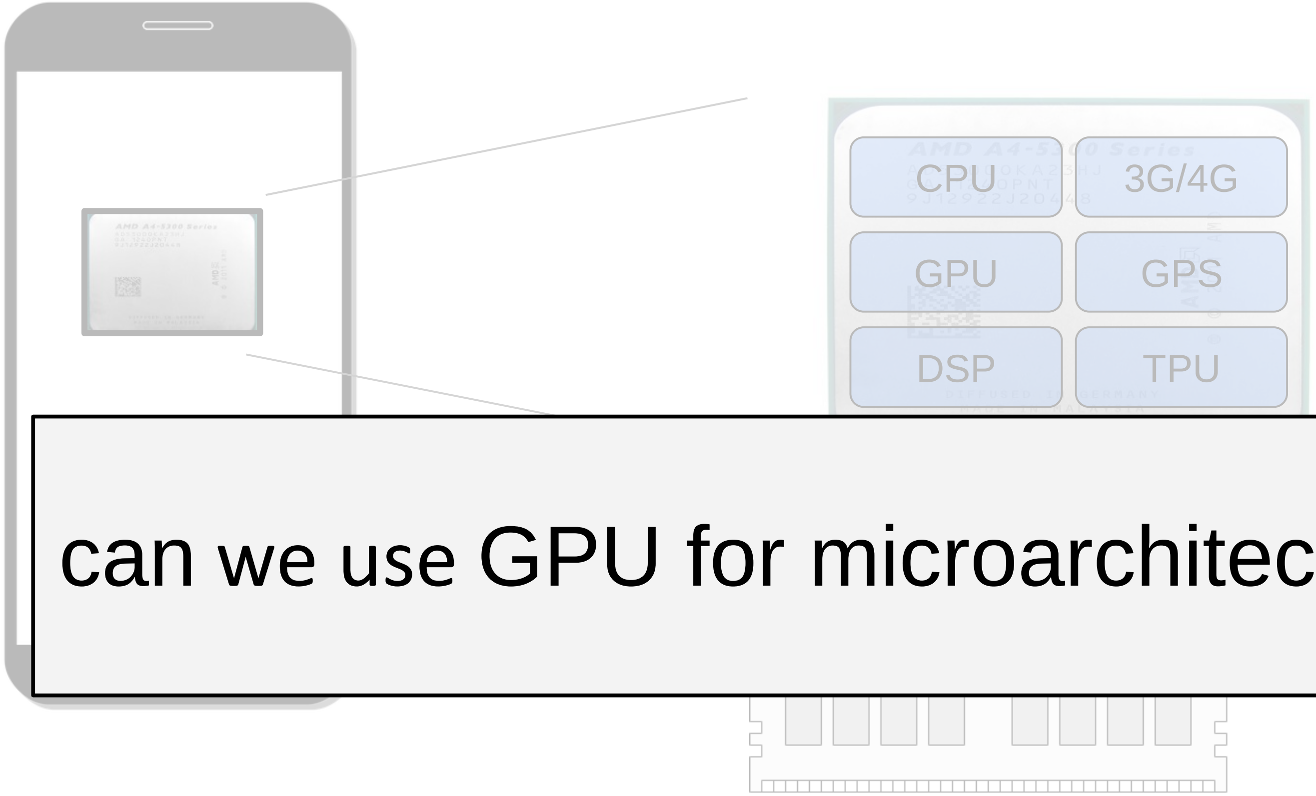
Access control

- Effective against standard exploitation vector



Access control

- Effective against standard exploitation vector
- **Fail to address microarchitectural attacks**

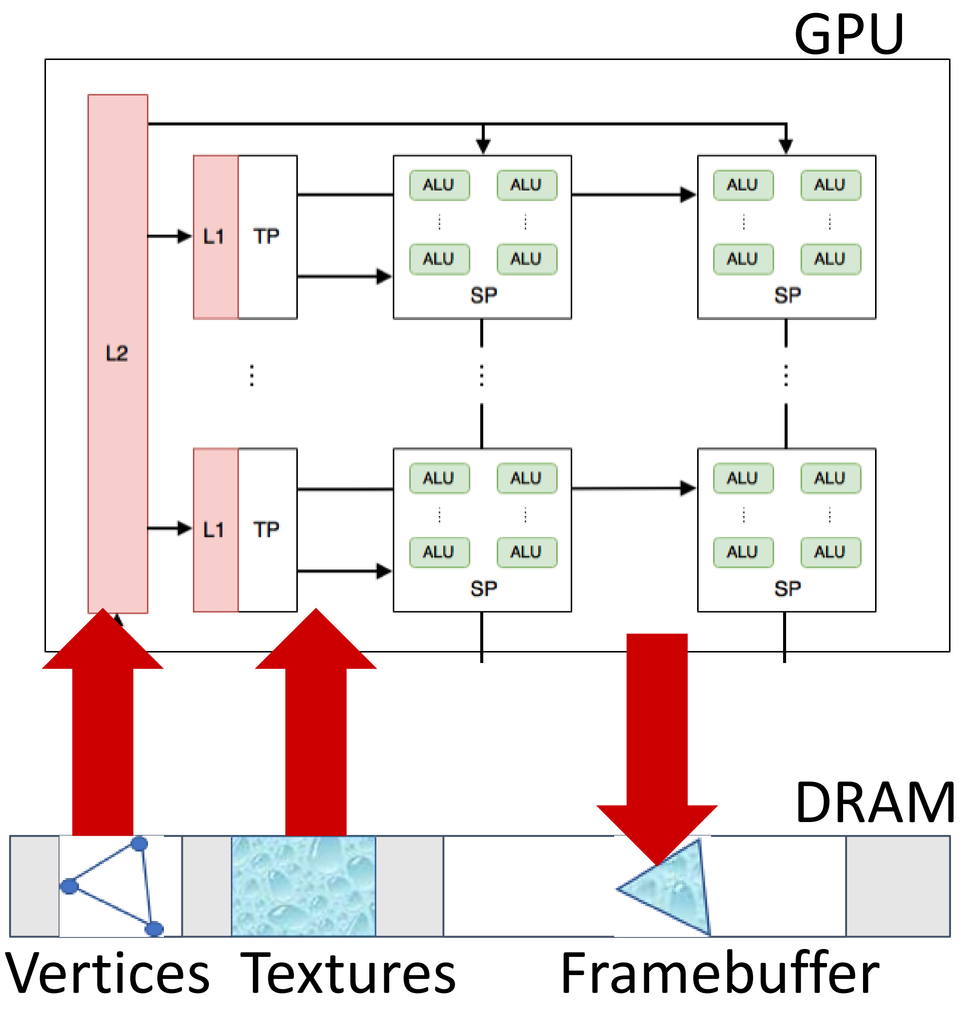


Access control

- Effective against standard exploitation vector
- **Fail to address**
ttacks

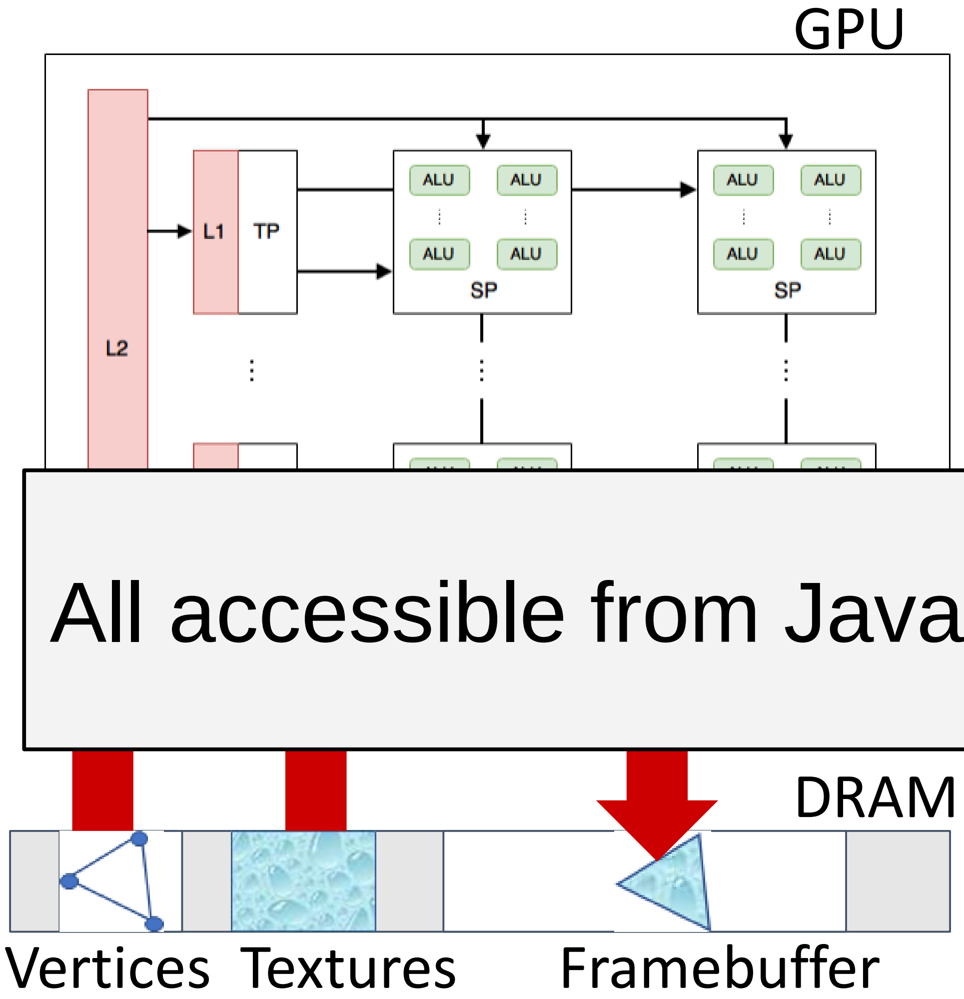
can we use GPU for microarchitectural attacks (RH)?

GPU architecture



1. Read Vertices
2. Read Textures
3. Write to Framebuffer

GPU architecture



1. Read Vertices
2. Read Textures
3. uffer

All accessible from JavaScript, thanks to WebGL

Research

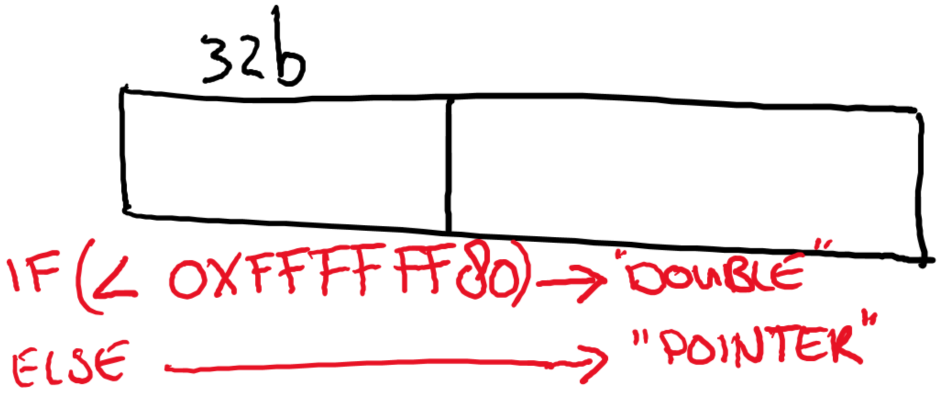
1. Reverse engineered architecture (caches!)
   ➔ to bypass them

2. Build\t highly accurate timers
   ➔ needed for side channel

3. Figured out how to get large contiguous memory areas
   ➔ needed for Rowhammer

End-to-end exploit

A bit like the one in Dedup Est Machina
“Type flipping”



Flip bit in pointer ➔ double ➔read value
Flip bit in double ➔ forge pointer

End-to-end exploit

on phones!

from JavaScript!

End-to-end exploit

on phones!

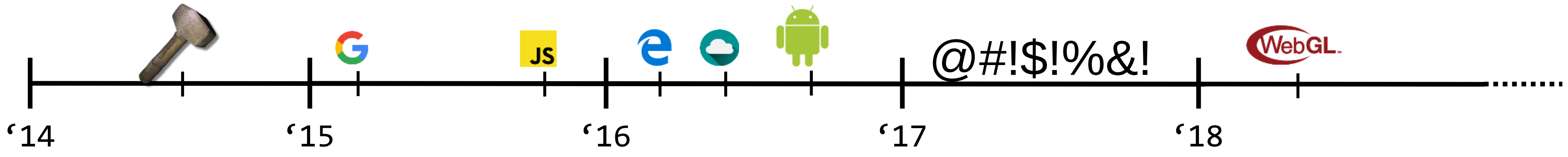
from JavaScript!

$0,-

Goal 5

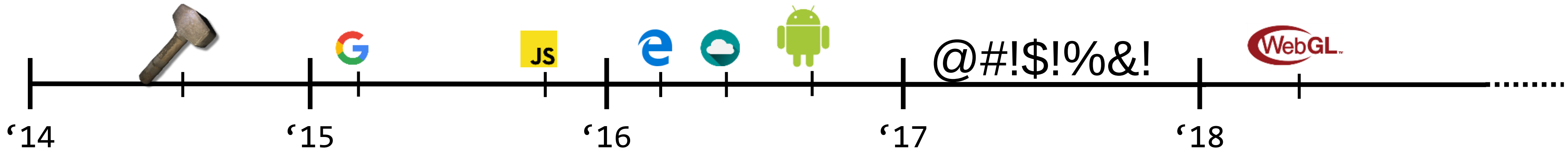
What about servers?

Rowhammer Evolution



[1] CMU finds first bit flip (2014)
[2] Google Project Zero: 1st Rowhammer root Exploit (flipping PTEs)
[3] Rowhammer.js: 1st RH bit flip in JavaScript
[4] Dedup est Machina: Breaking Microsoft Edge’s sandbox
[5] Flip Feng Shui: Breaking the cloud
[6] Drammer: rooting android
[7] Grand Pwning Unit: attack from the GPU (faster!)

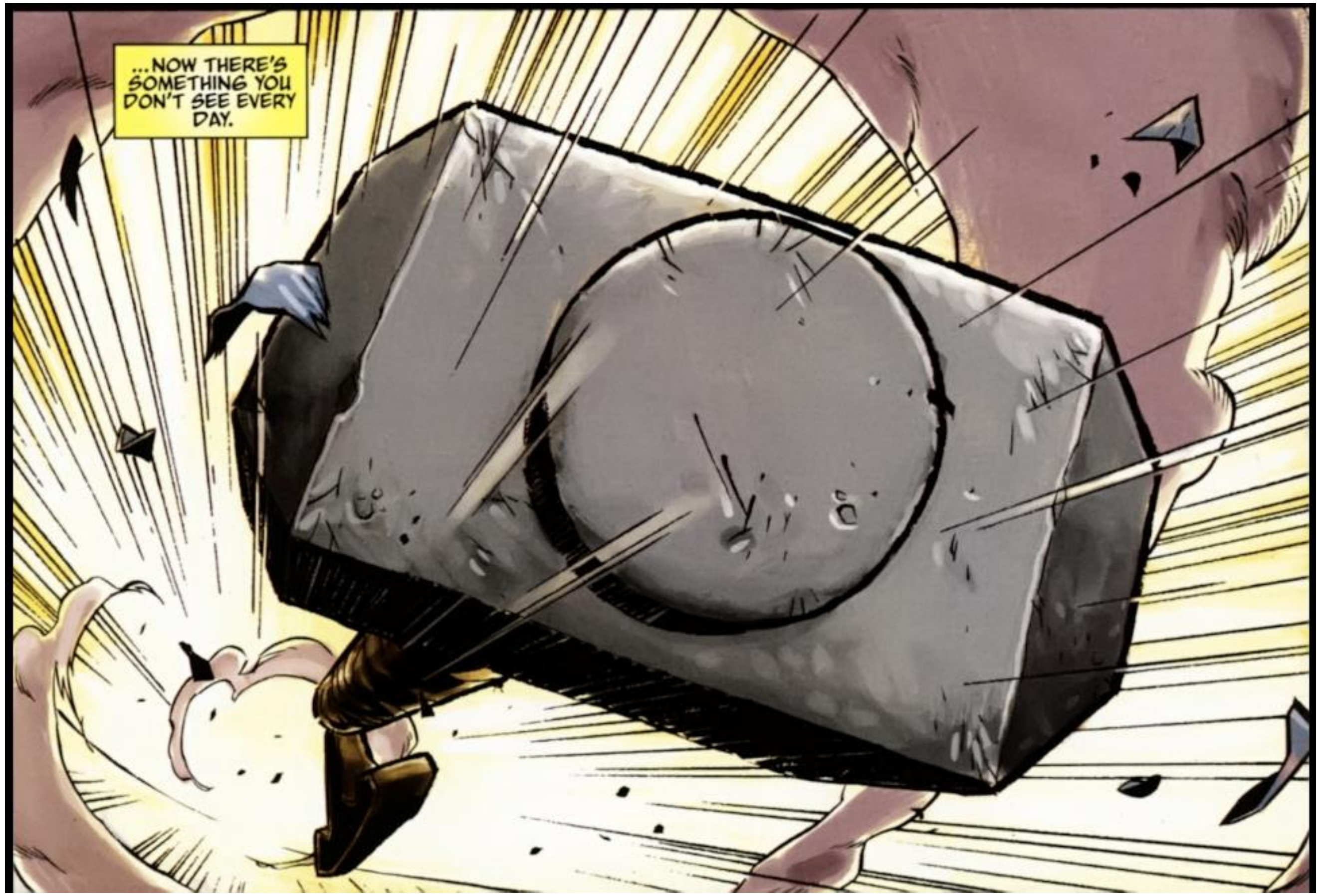
Rowhammer Evolution



[1] CMU finds first bit flip (2014)
[2] Goo
[3] Row
[4] Ded
[5] Fli
[6] Dra
[7] Gra

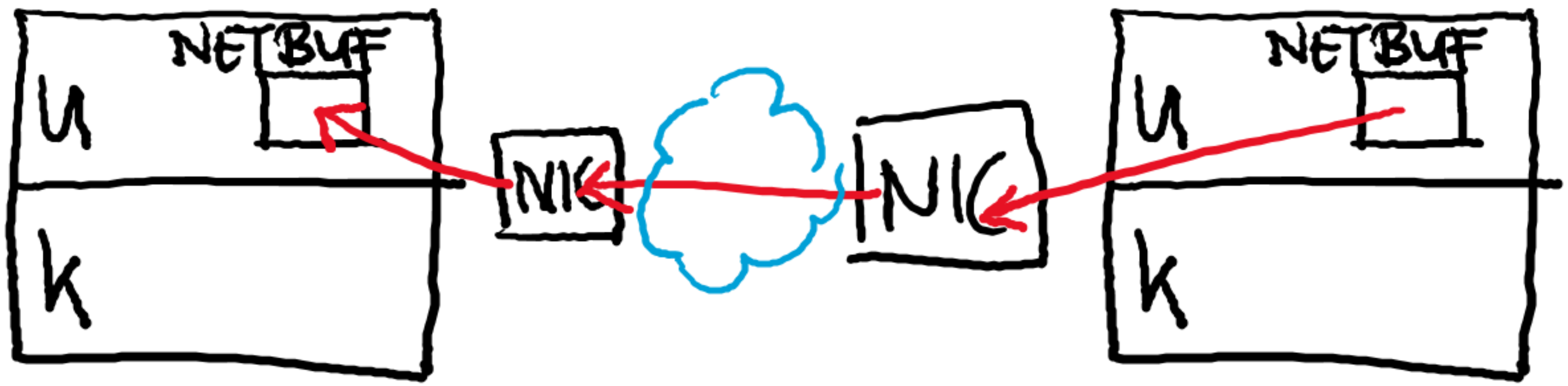
So far, Rowhammer requires local code execution. Can we attack servers over the network?

Throwhammer



Fast networks

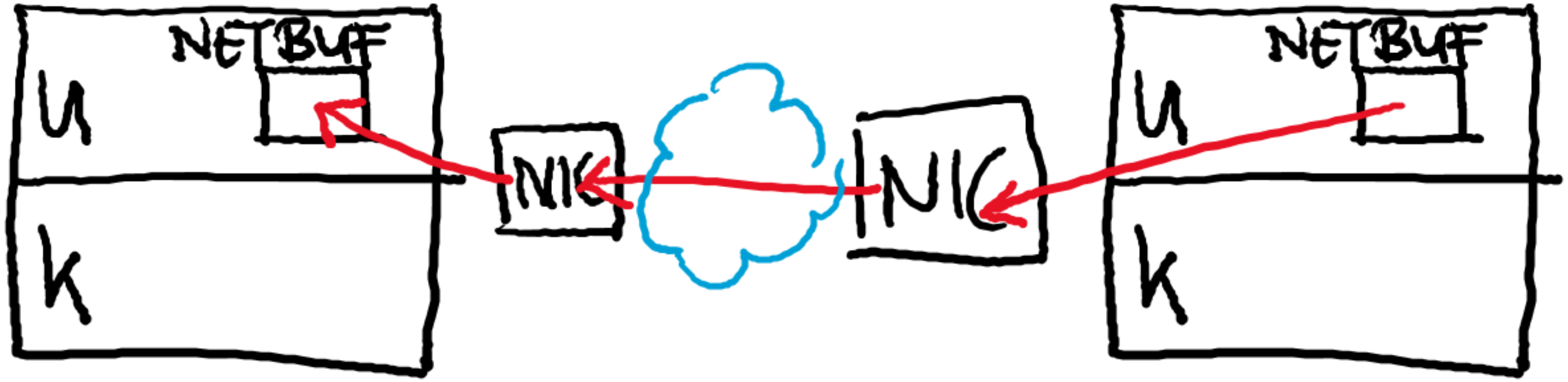
RDMA



Fast networks

RDMA

We can flip bits over the network

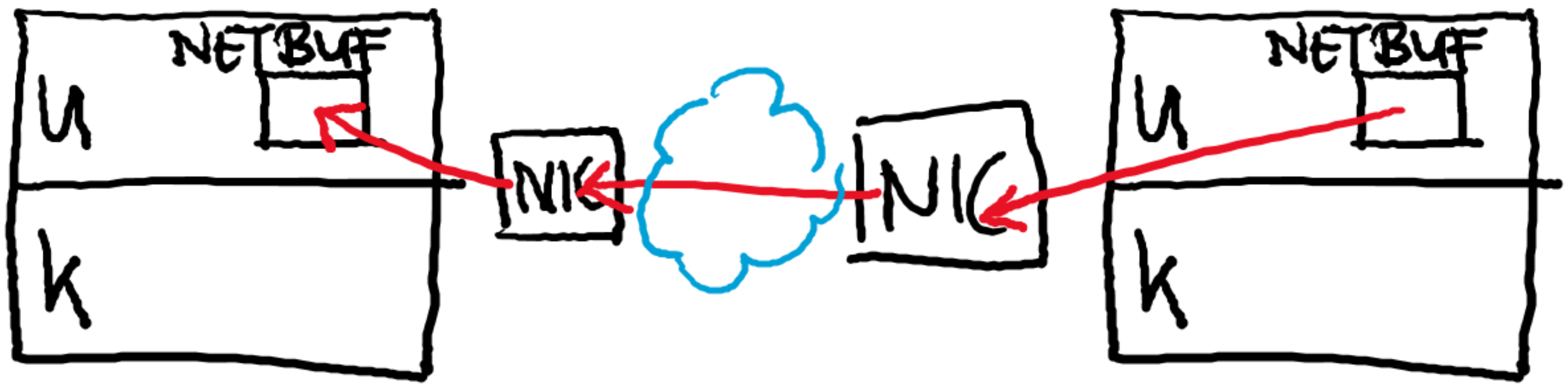


Fast networks

RDMA

We can flip bits over the network

Moreover, we can exploit server software

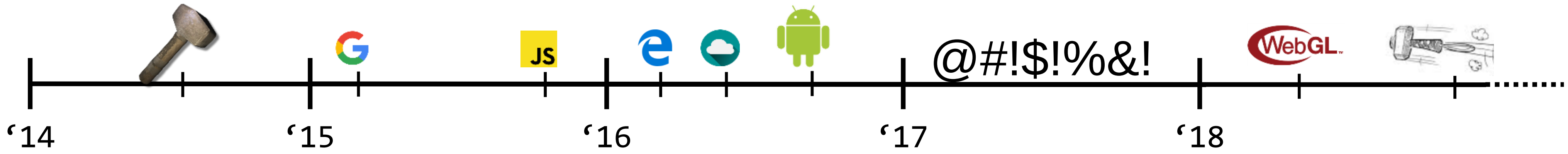


Rowhammer Evolution

$0,-

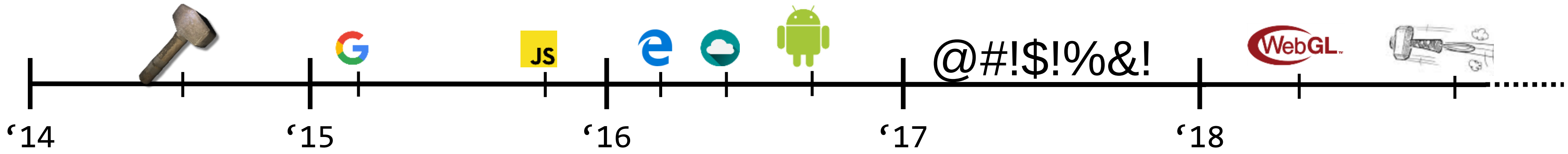
[1] CMU finds first bit flip (2014)
[2] Google Project Zero: 1st Rowhammer root Exploit (flipping PTEs)
[3] Rowhammer.js: 1st RH bit flip in JavaScript
[4] Dedup est Machina: Breaking Microsoft Edge's sandbox
[5] Flip Feng Shui: Breaking the cloud
[6] Drammer: rooting android
[7] Grand Pwning Unit: attack from the GPU (faster!)
[8] Throwhammer: attack servers over the network

What is missing?



[1] CMU finds first bit flip (2014)
[2] Google Project Zero: 1st Rowhammer root Exploit (flipping PTEs)
[3] Rowhammer.js: 1st RH bit flip in JavaScript
[4] Dedup est Machina: Breaking Microsoft Edge’s sandbox
[5] Flip Feng Shui: Breaking the cloud
[6] Drammer: rooting android
[7] Grand Pwning Unit: attack from the GPU (faster!)
[8] Throwhammer: attack servers over the network

What is missing?



[1] CMU finds first bit flip (2014)
[2] Google Project Zero: 1st Rowhammer root Exploit (flipping PTEs)
[3] Rowhammer.js: 1st RH bit flip in JavaScript
[4] Dedup est Machina: Breaking Microsoft Edge’s sandbox
[5] Fl
[6] Dr
[7] Gr
[8] Th

Can we do this on ECC memory?

Goal 6

Flipping bits on ECC memory

Flipping bits on ECC memory

Lucian Cojocar

Kaveh Razavi

Herbert Bos

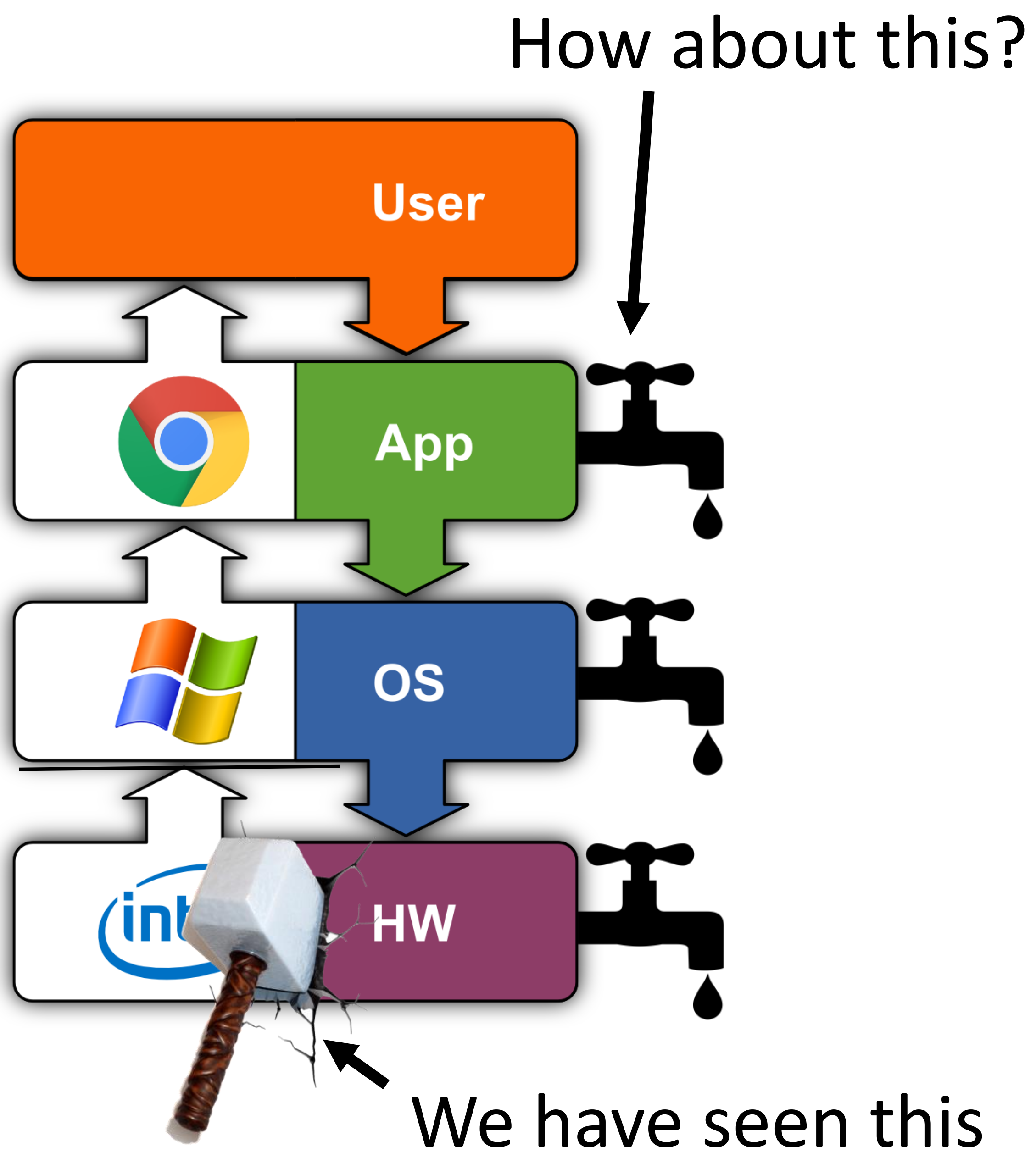
Cristiano Giuffrida

Part II
Side Channels

Software Exploitation:

2018

Goal: Controllable from Software



Side channels – what do we want to leak?

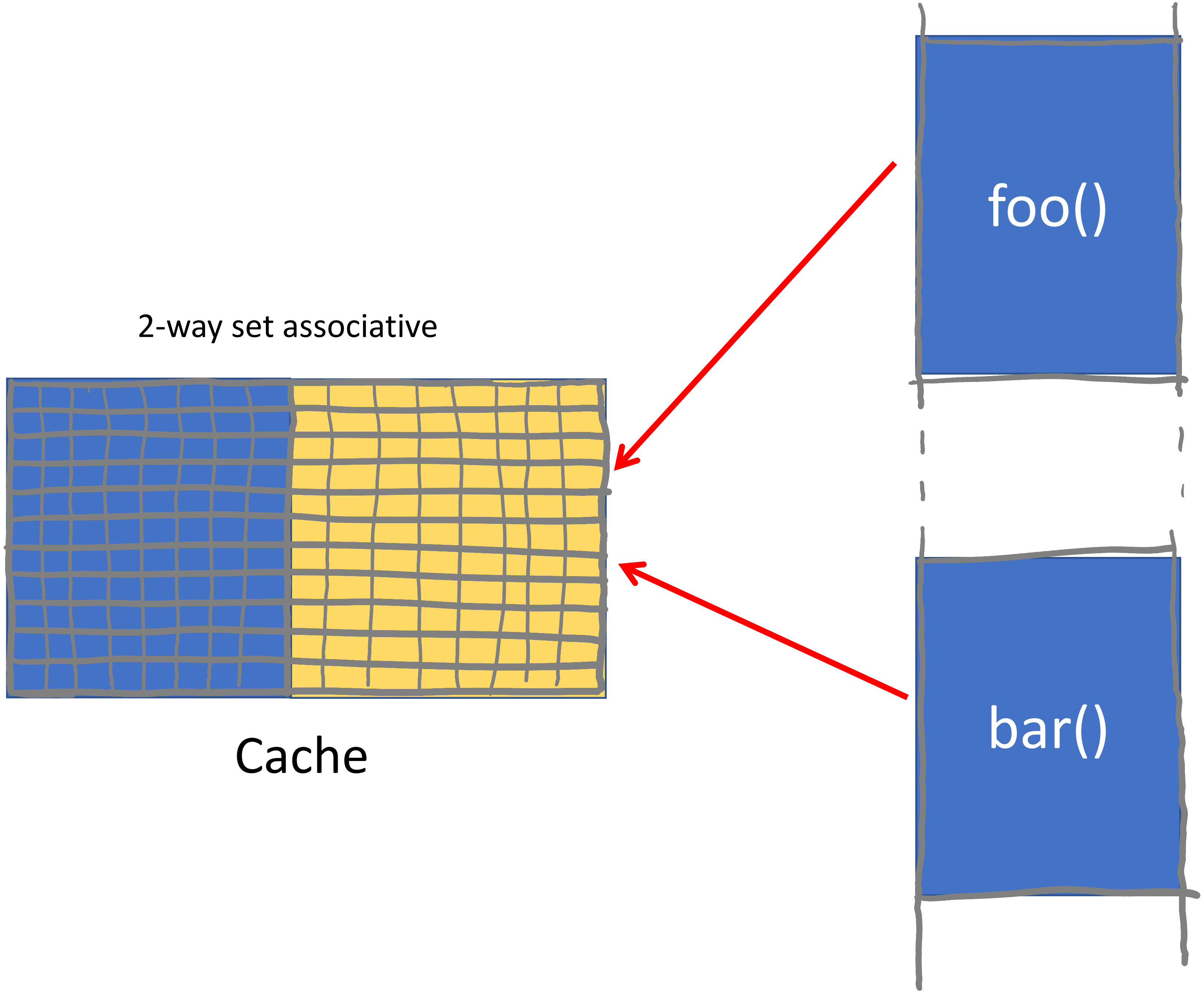
Passwords, keys, and other secret user data

Addresses (breaking ASLR)

Cache Side Channels

Intuition only

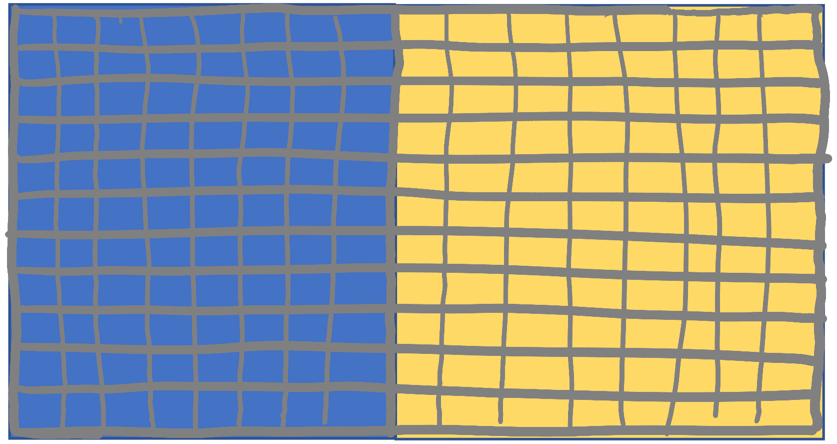
Prime and Probe



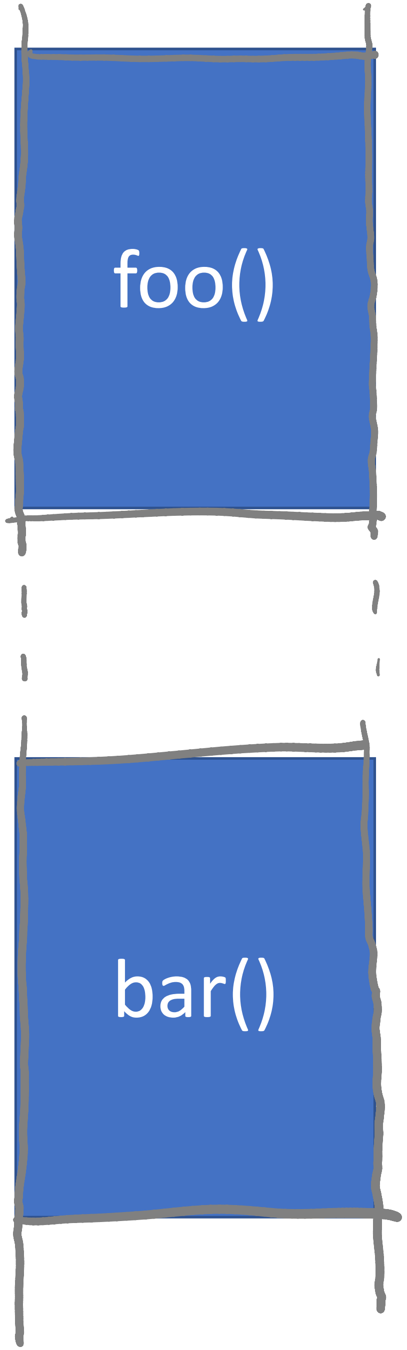
Prime and Probe

```
for i in key_length:
    if (keybit(i) == 1)
        foo();
    else
        bar ();
```

2-way set associative

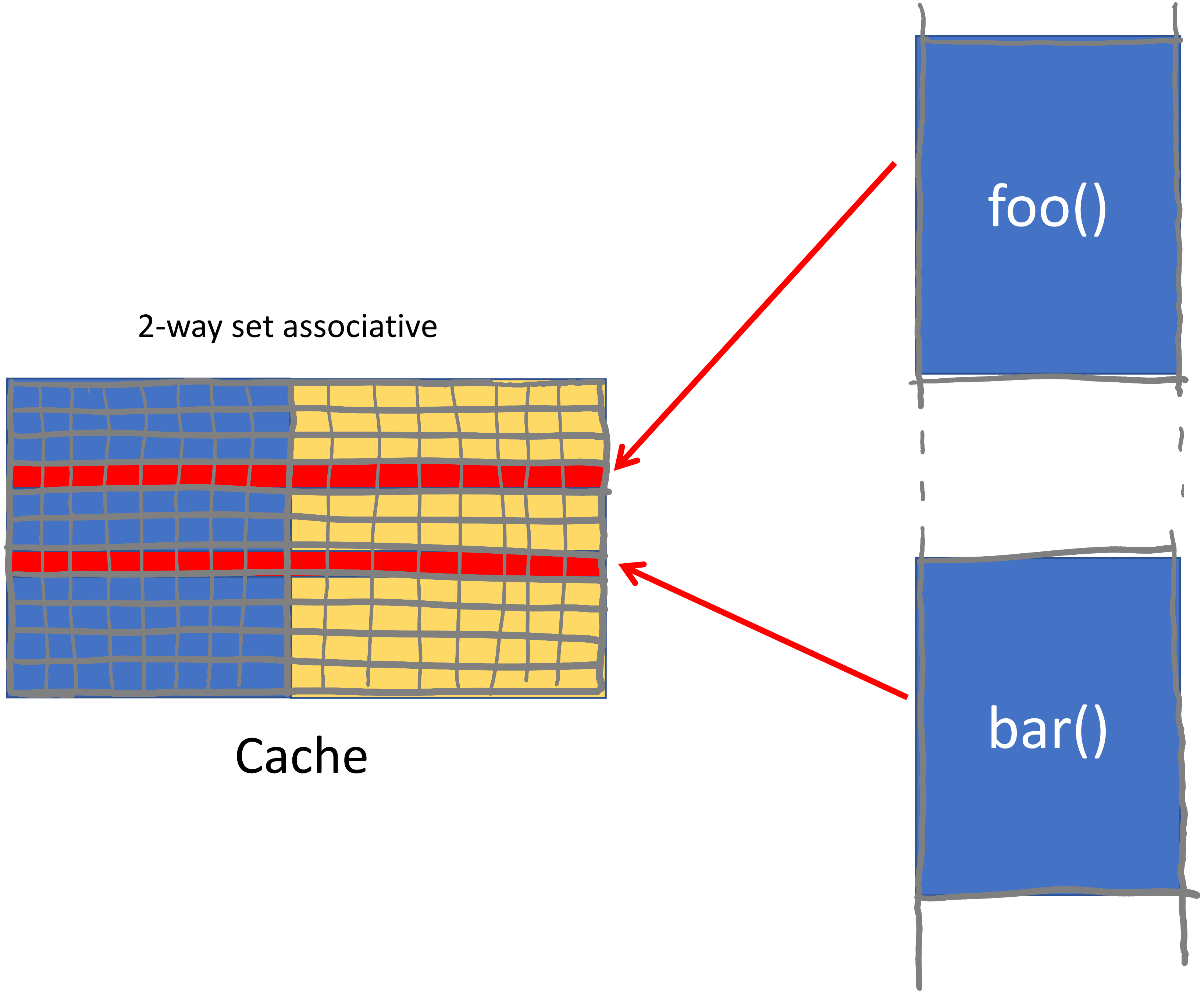


Cache



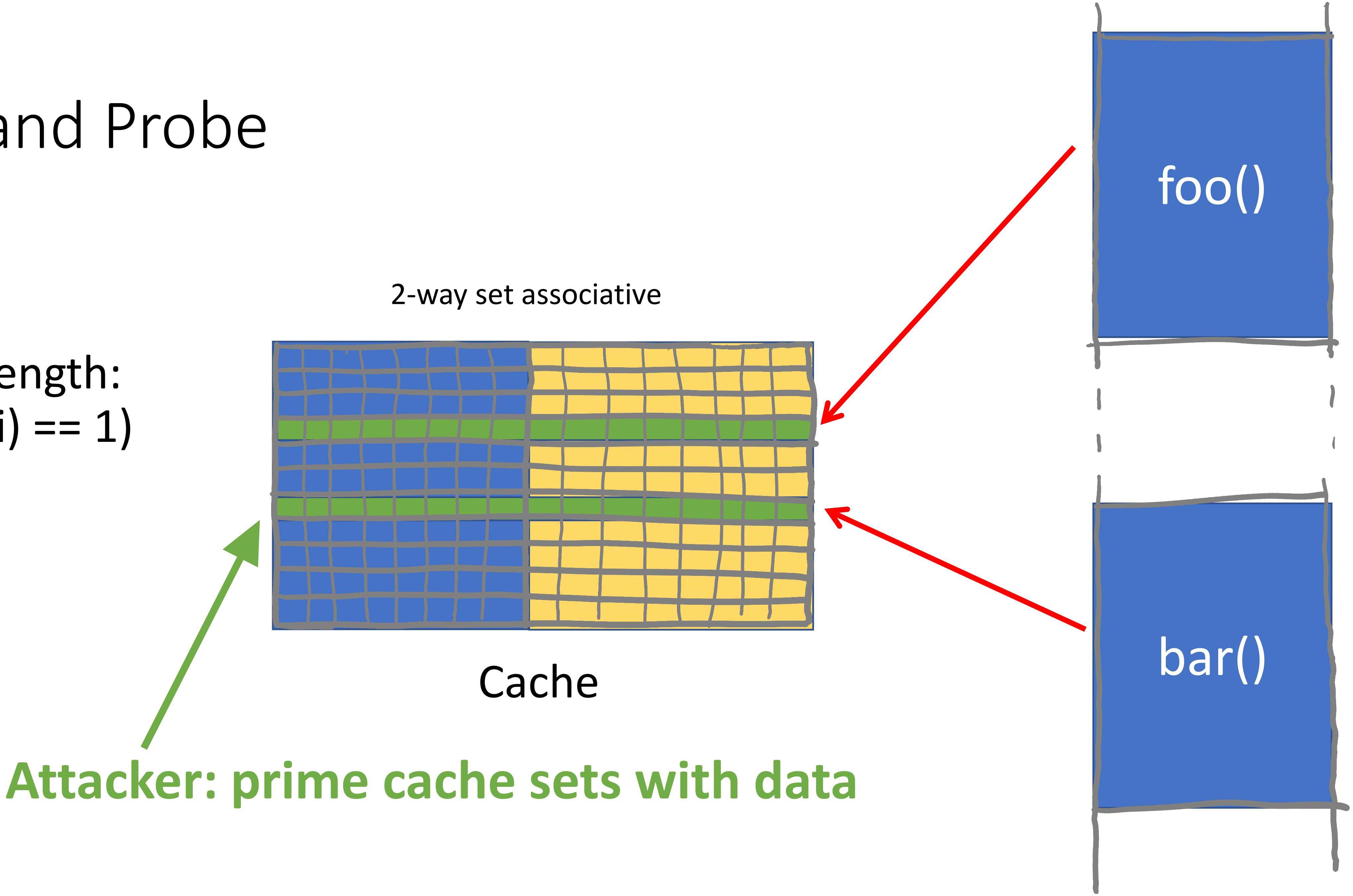
Prime and Probe

```
for i in key_length:
    if (keybit(i) == 1)
        foo();
    else
        bar ();
```



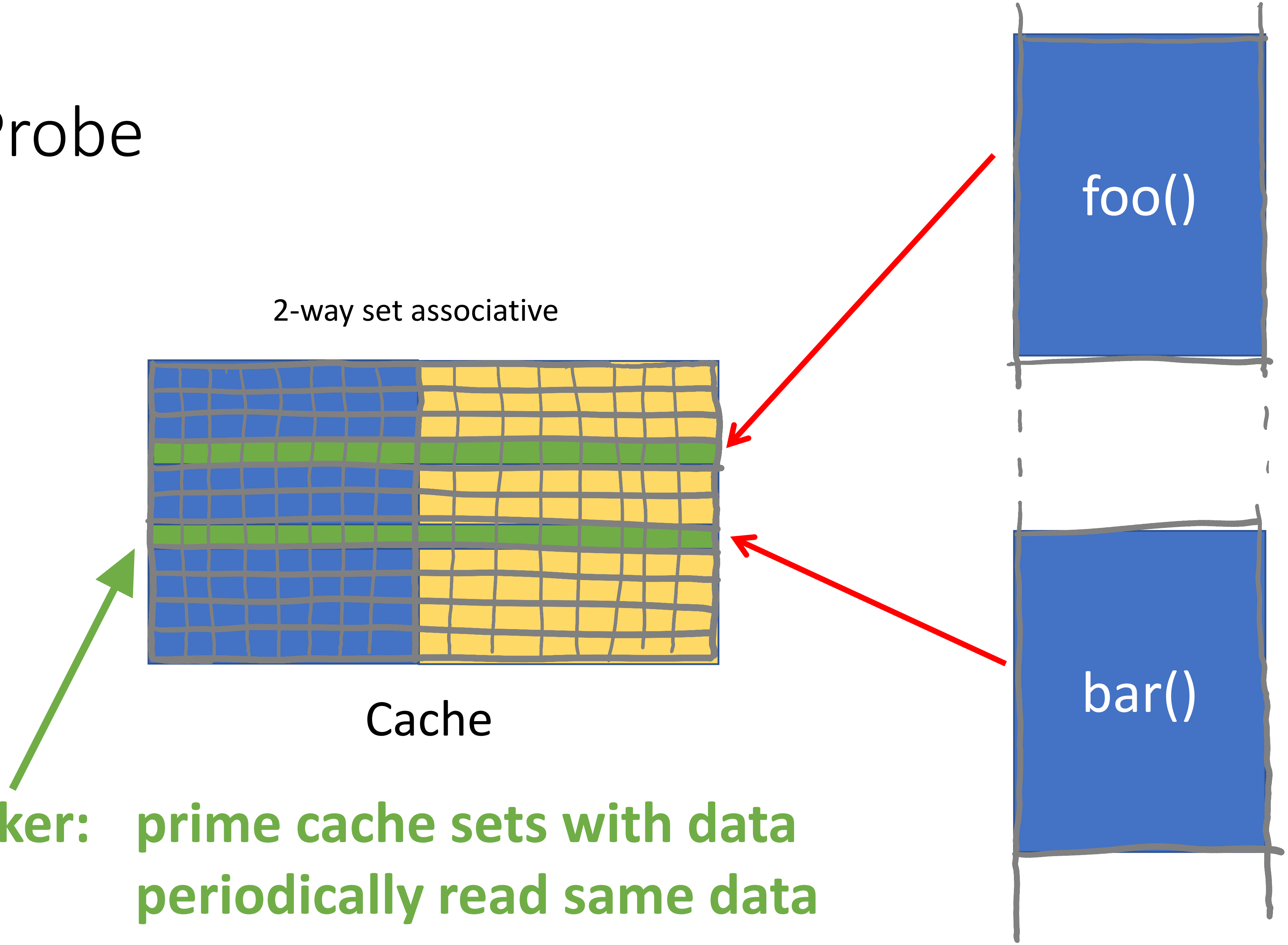
Prime and Probe

```
for i in key_length:
    if (keybit(i) == 1)
        foo();
    else
        bar ();
```



Prime and Probe

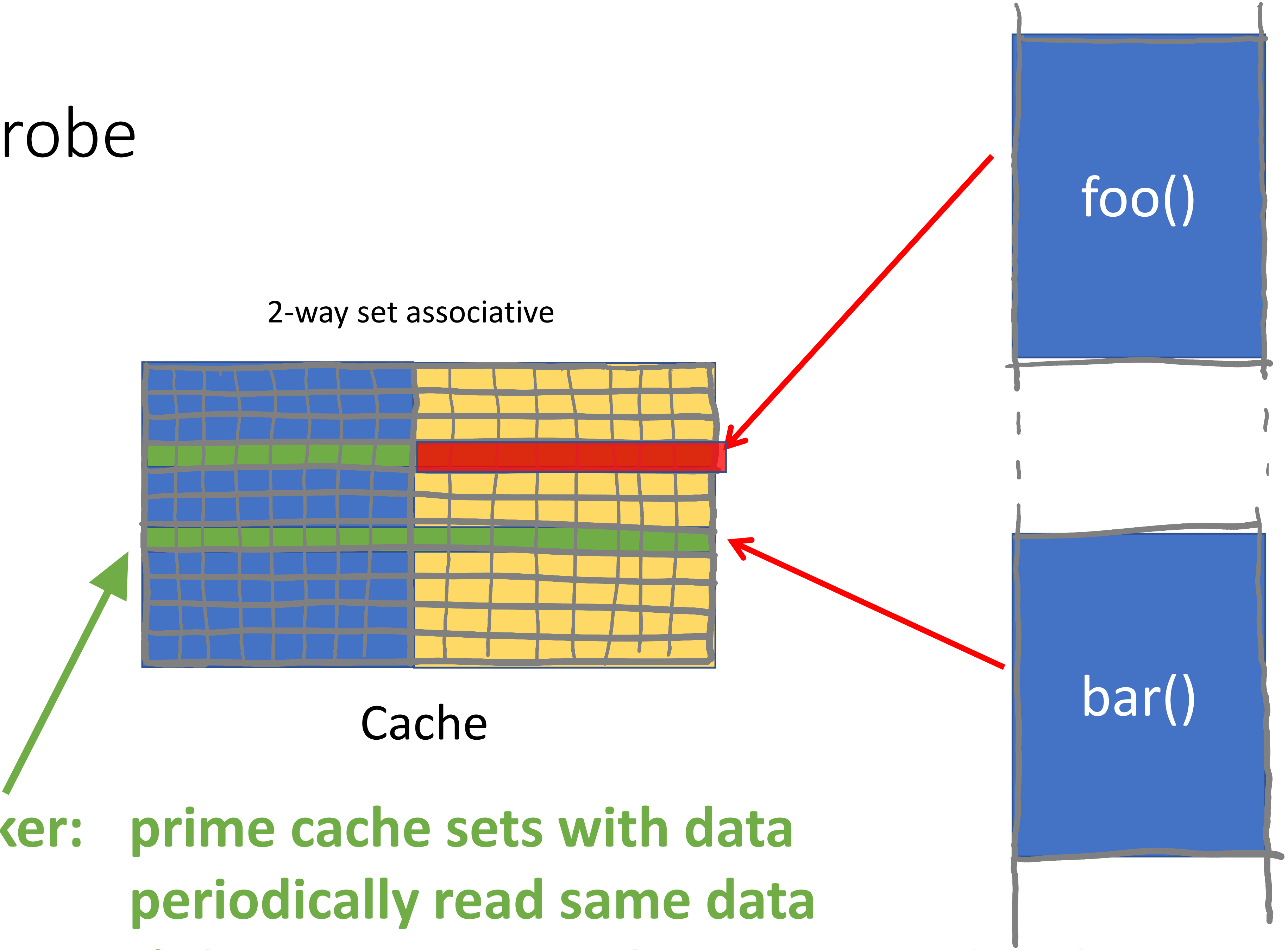
```
for i in key_length:
    if (keybit(i) == 1)
        foo();
    else
        bar ();
```



Attacker: prime cache sets with data
periodically read same data

Prime and Probe

```
for i in key_length:
    if (keybit(i) == 1)
        foo();
    else
        bar ();
```



Attacker: prime cache sets with data
periodically read same data
if slow: victim must have accessed cache set

But maybe we do not have a key to leak…

We want to leak addresses, to break ASLR

AnC

ASLR ^ Cache

AKA "Side channeling the MMU"

Ben Gras


The Volunteer State
JUL
Tennessee
18
PWNAGE
PWNAGE
WHEEL
TAX
DECAL
KN
Class Code
2000
Issue Year
2007
20007 PWNAGE

Code Reuse:

crucial requirement

Need to find address of code (and data)

Goal: break ASLR (from Javascript)

Say we have a JS object

- “What are addresses of heap and code?”

Result:

- ASLR is fundamentally insecure
- Broken without relying on special features/settings
    - - Dedup
    - - Overcommit
    - - Threadspraying

Goal: break ASLR (from Javascript)

Fundamental

The way modern processors translate VA → PA

- MMU
- PT walks

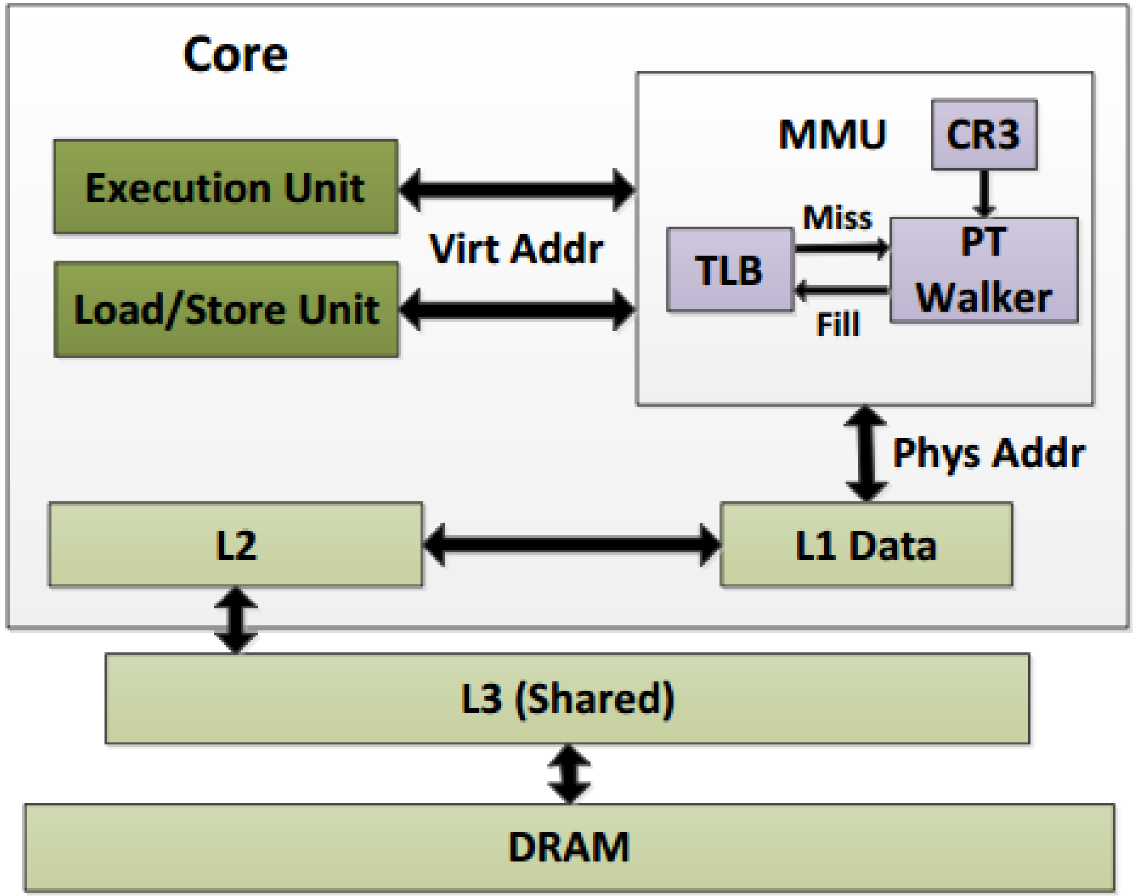
The way modern systems use caches

- PTs also cached

Conclusion

Secure ASLR and caching are mutually exclusive

Memory organization in Intel

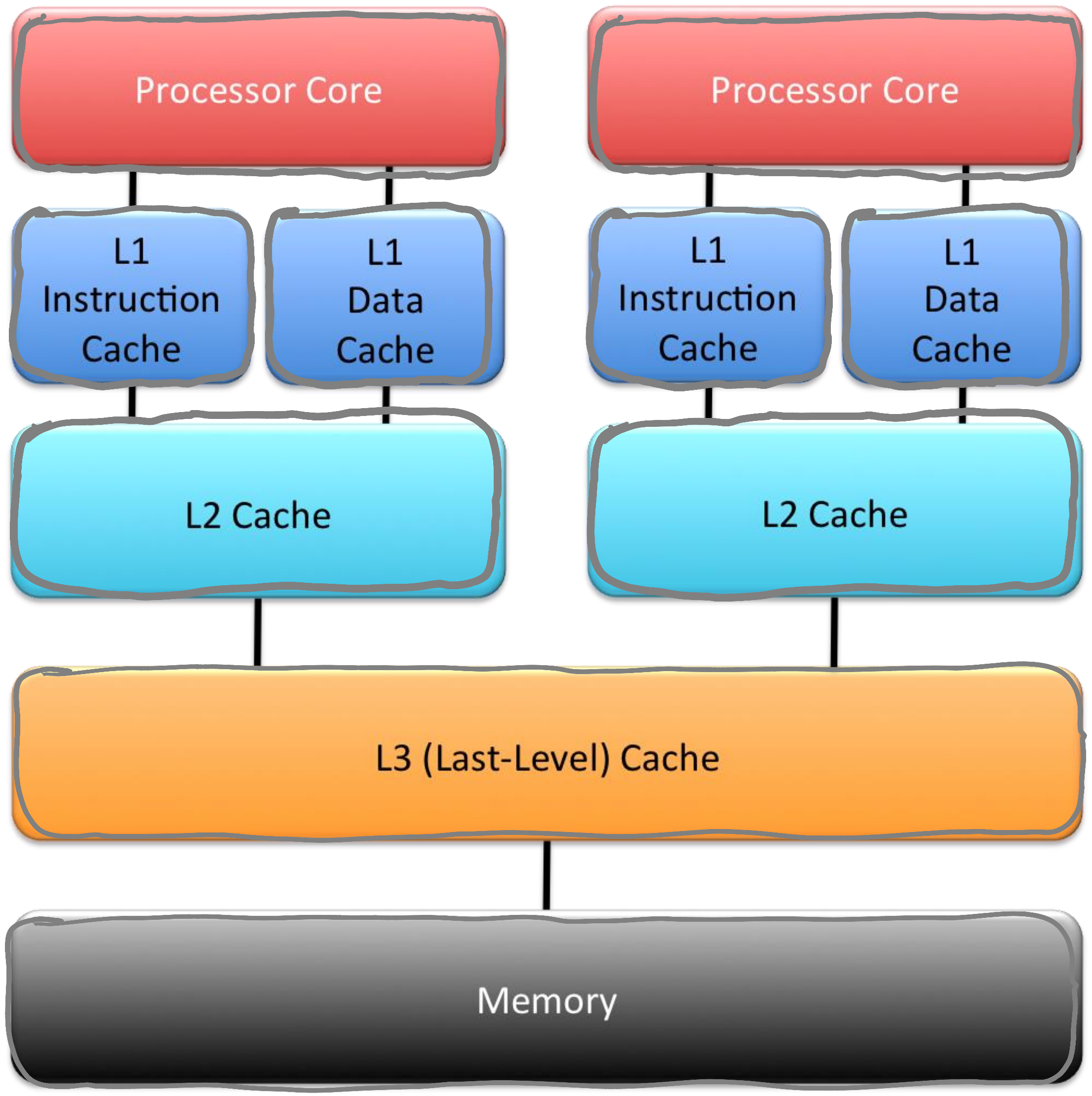


Caches

- Physically tagged
- N-way set associative (e.g., 16)
- 64B cache lines
- LLC is inclusive

Caches

- Physically tagged
- N-way set associative (e.g., 16)
- 64B cache lines
- LLC is inclusive



MMU

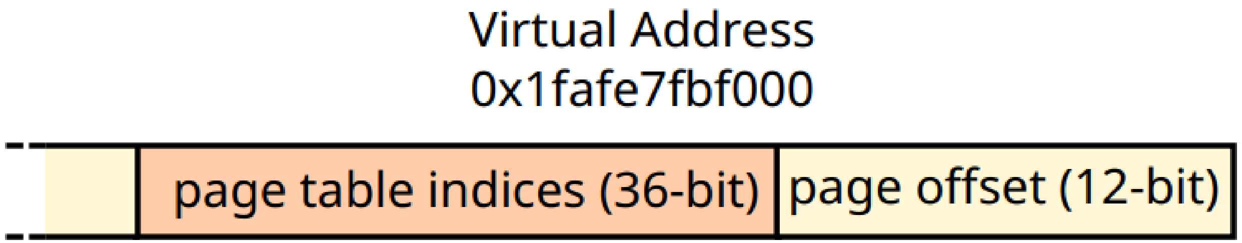
TLB translates VA → PA

- Before accessing data or instruction (cache phys. tagged)

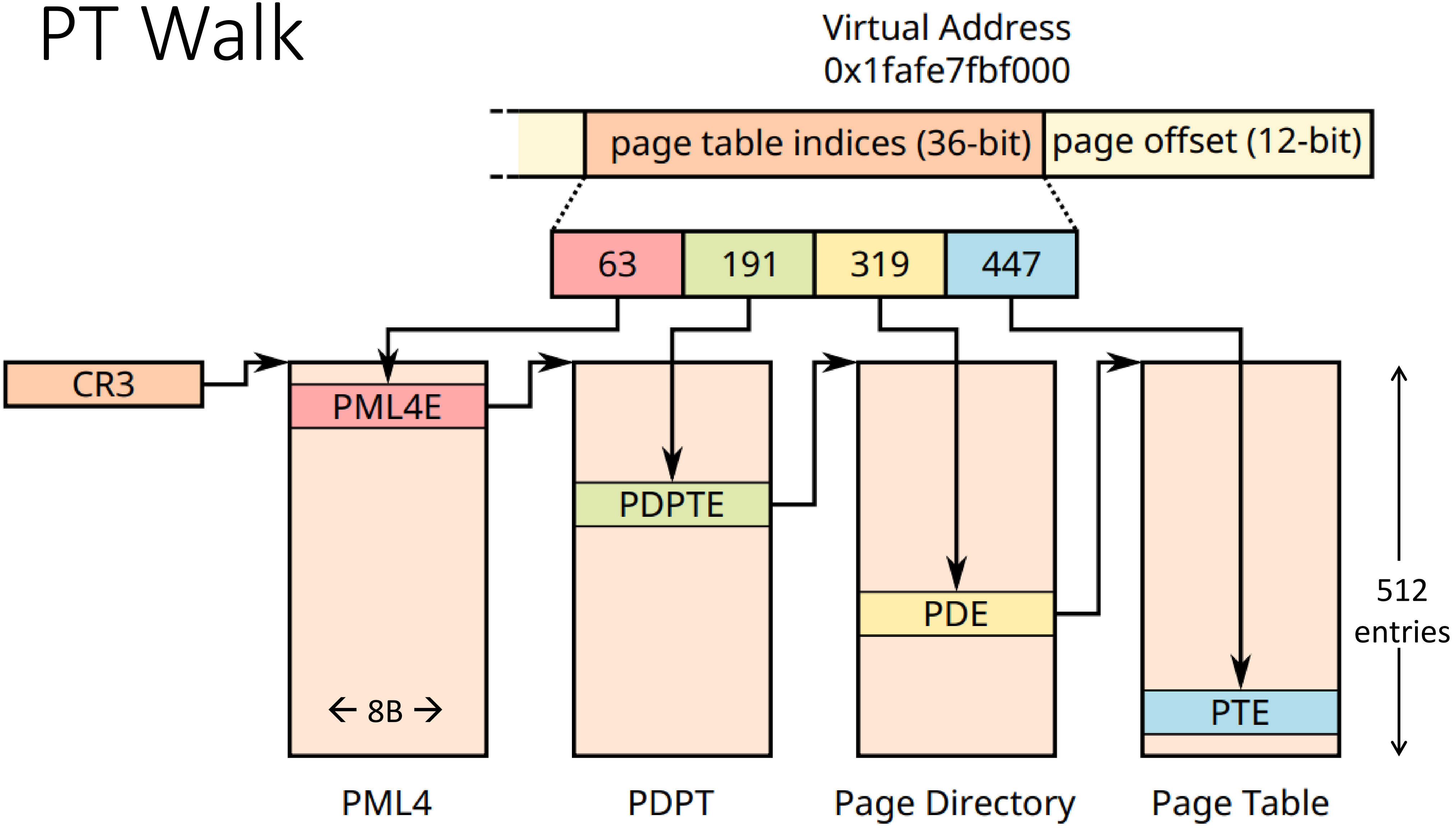
On miss: PT walk

- For attack, we will clear the TLB to force PT walk

PT Walk


Virtual Address
0x1fafe7fbf000
page table indices (36-bit)
page offset (12-bit)

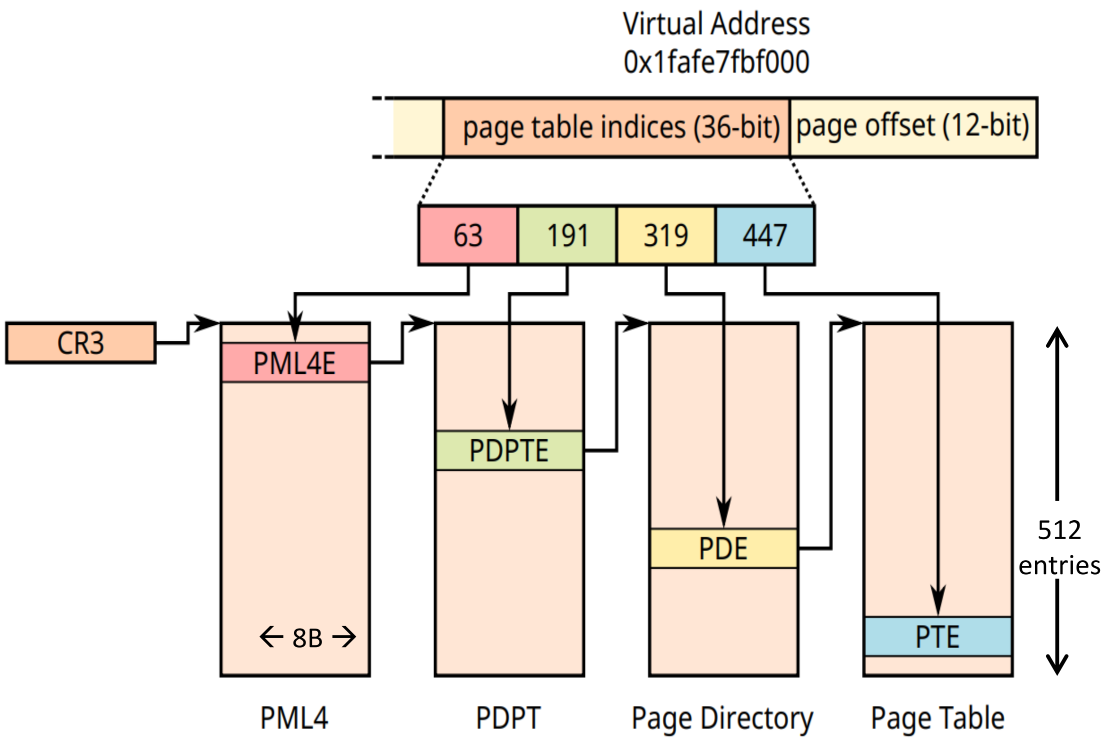
PT Walk


Virtual Address
0x1fafe7fbf000
page table indices (36-bit)
page offset (12-bit)
63
191
319
447
CR3
PML4E
PDPTE
PDE
PTE
8B
512
entries
PML4
PDPT
Page Directory
Page Table

Important Observation (1)

PT Walk

ALSR Linux heap: 28 bits



If we know each entry in the PT used in the walk → we know the VA

Each PT level contains 9 bits of entropy

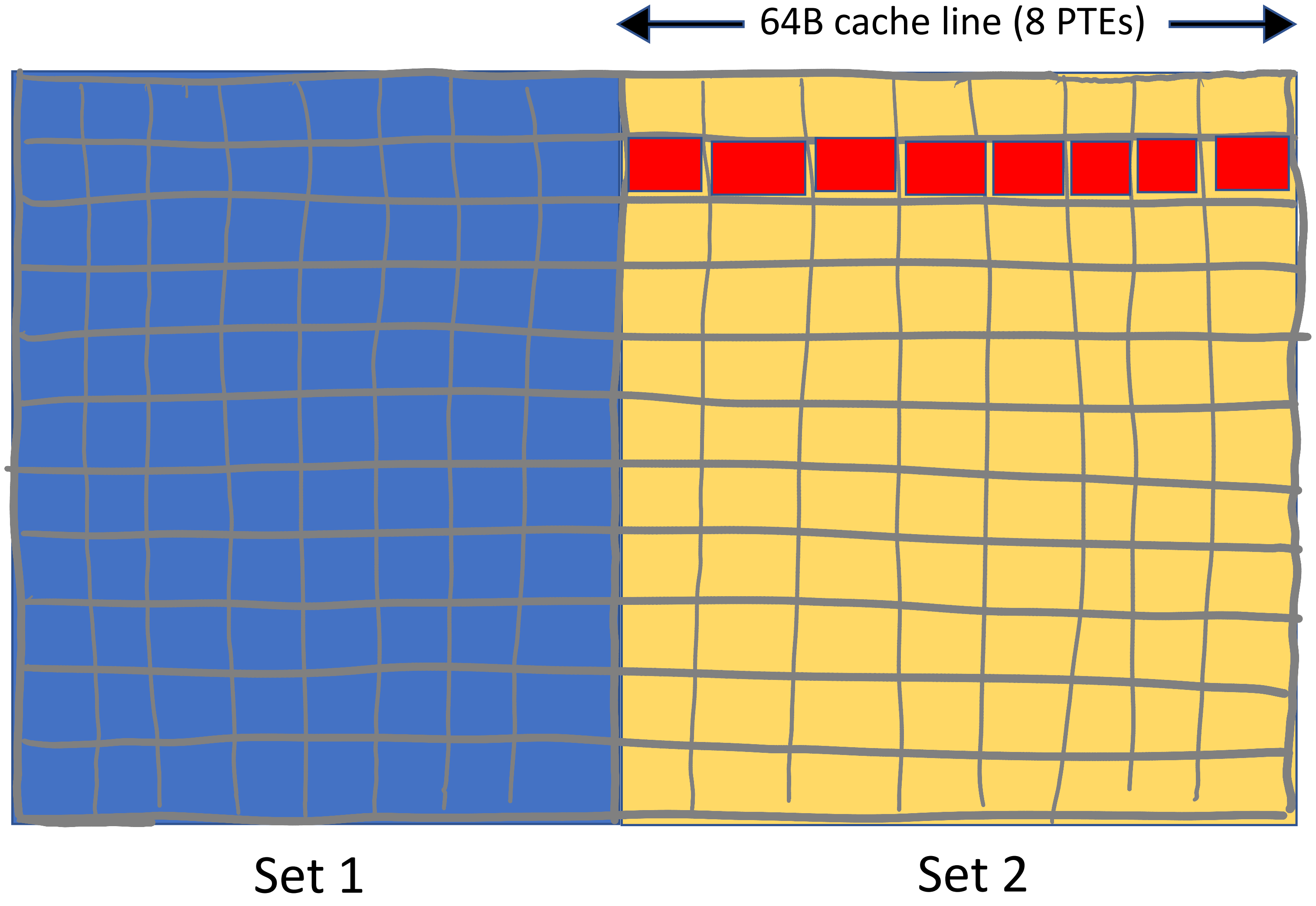
(last level only 1 bit)

PTs are cached too

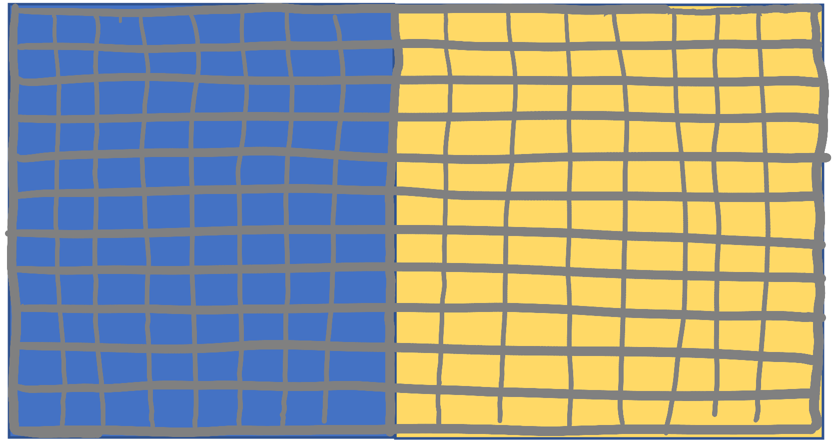
Each PT contains

$2^{12}/2^{3} = 2^{9}$ PTEs

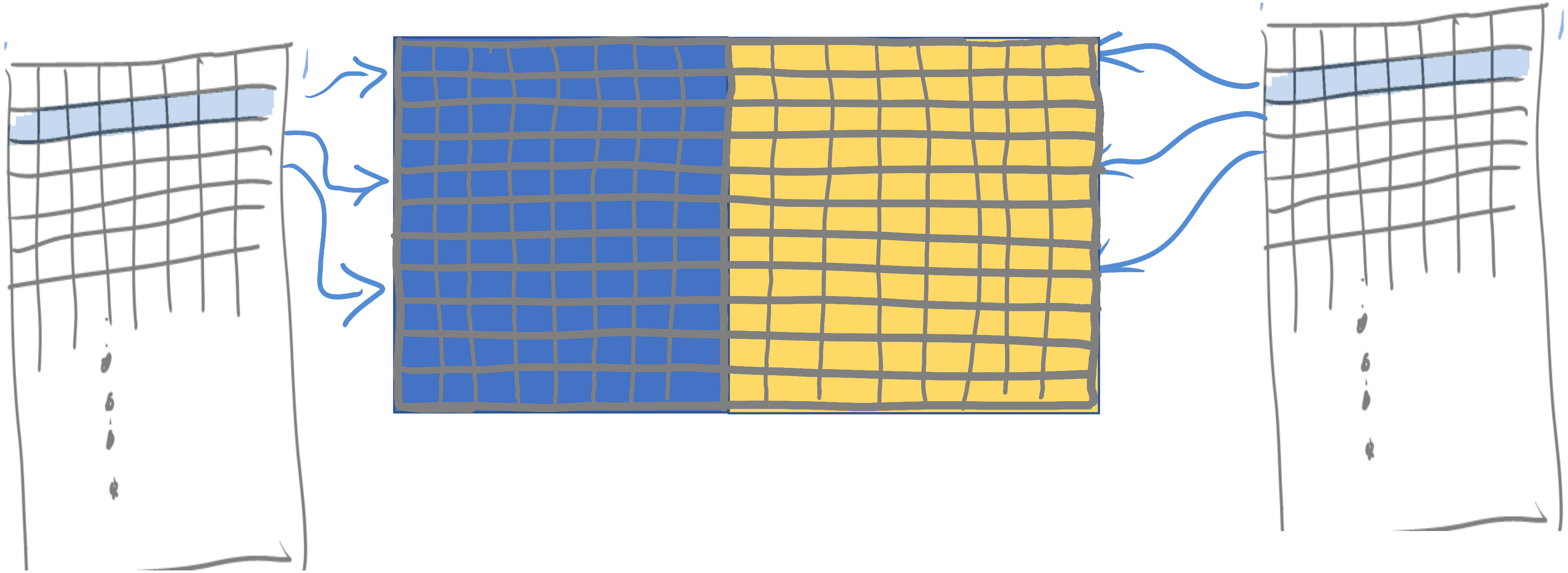
or $2^{9}/2^{3}$ = 64 cache lines



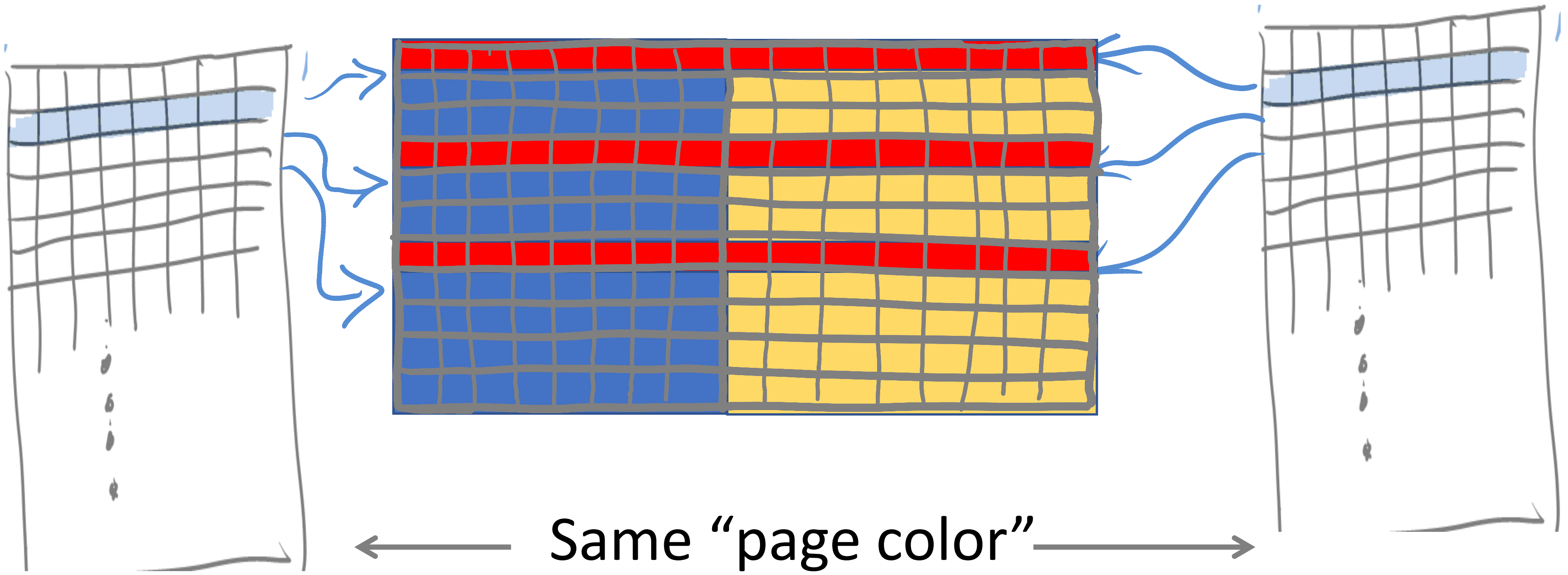
Cache sets



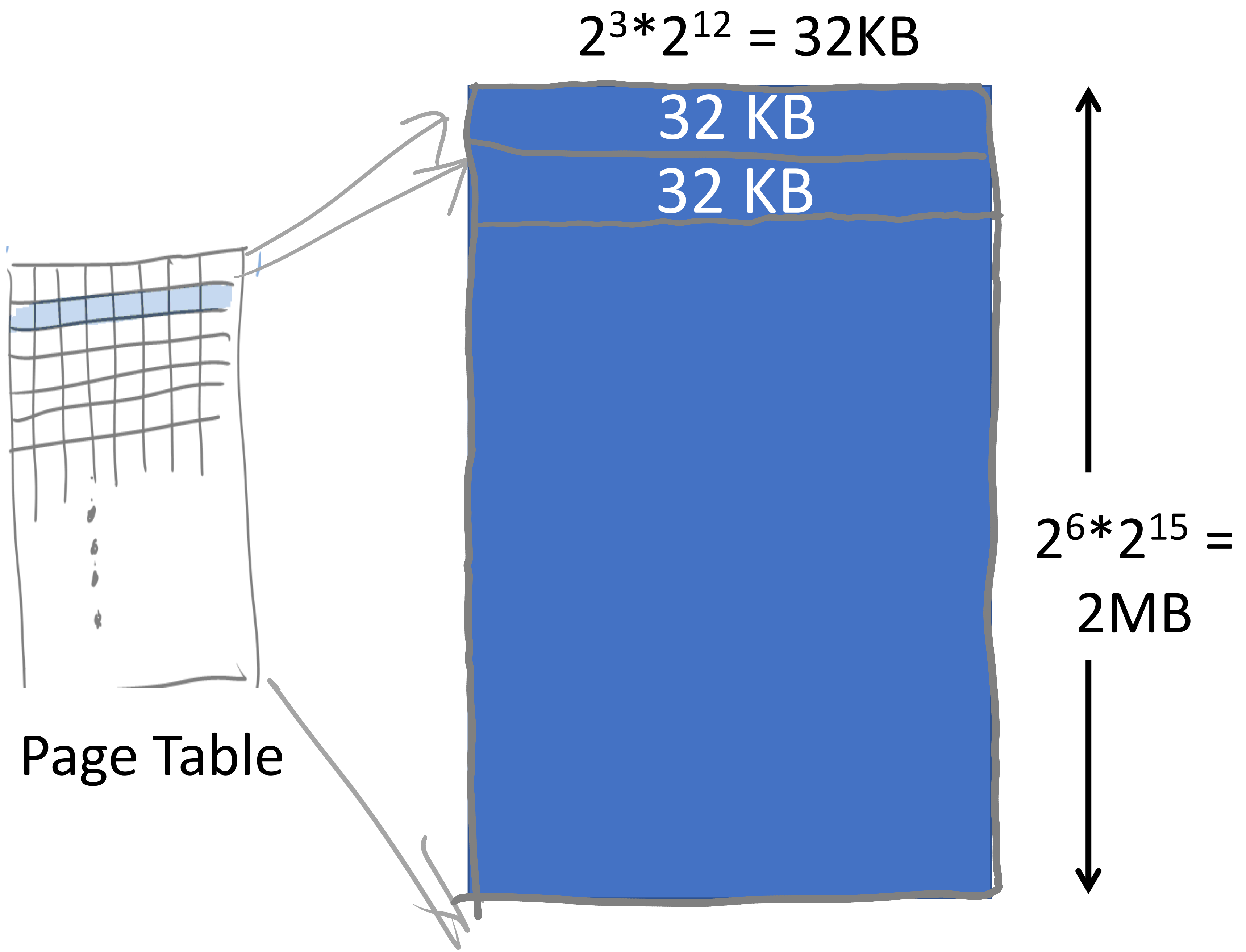
Cache sets



Cache sets



If first cacheline of 2 pages in same cache set
→ All cache lines in the 2 pages share (different) cache set



2^3*2^{12} = 32KB
32 KB
32 KB
2^6*2^{15} =
2MB
Page Table

Important observation (2)

If we know which cache line of a PT was accessed during PT walk

Gives us 6 of the 9 bits of entropy

“It can be any of these 8 PTEs out of 512 PTEs on page”

What we need

1. *Identifying the cache lines that host the PTEs*
2. *Identifying page offsets of the cache lines*
3. *Identifying cache line offsets of the PT entries*

Intuitively

Say there is only 1 PT and we want the 9 bits for address A

Allocate large number of pages

Evict a target cache line at offset *t*

- Access all pages at that cacheline offset (also flushes TLBs)
- Time the access to A (+ some offset, to make sure we hit other cache line)
- PT walk begins
- If access takes longer → this line at offset *t* *must* have contained PTE

In reality: more PTs

Two more problems:

- We know the cache line that contains PTE, but of which level?
- We now know cache line: 6 bits. How about remaining 3?

Both problems have same solution: sliding

<u>Say PTL1</u>

- Probe address + 4KB, +8 KB, …, +32KB
- At some point will be on new cacheline in PT (slower access for our data)
- If this happens at +4KB, we know we were the last entry in the line. If it happens at +8KB, we were the one before that, etc

<u>If it does not happen at +32KB → higher level</u>

For PTL2, the stride is 2MB
(Note that a cache line switch for PTL2 always also incurs one in PTL1)

As we move up, doing so requires access to memory that is increasingly far apart to do the final trick → we must force a cache line switch

How about PTL3 and PTL4?

PTL3 : need 8GB crossing in AS

Problem: we can allocate only 2GB

PTL4 : need 4TB crossing in AS

For these levels we use knowledge about the memory allocators in FF and Chrome

See paper for details.

Concl AnC

BTW: we assume we have a timer

So we can measure diff between cached and (non cached) memory access from JS

- Not trivial (but solved problem): see paper

So…

ASLR fundamentally insecure

Very hard to fix

- Page coloring (keep browser memory separate) → hard
- Detection (performance counters) → hard
- Secure timers → hard
- Separate caches → expensive

So...

How much

do *you* think?

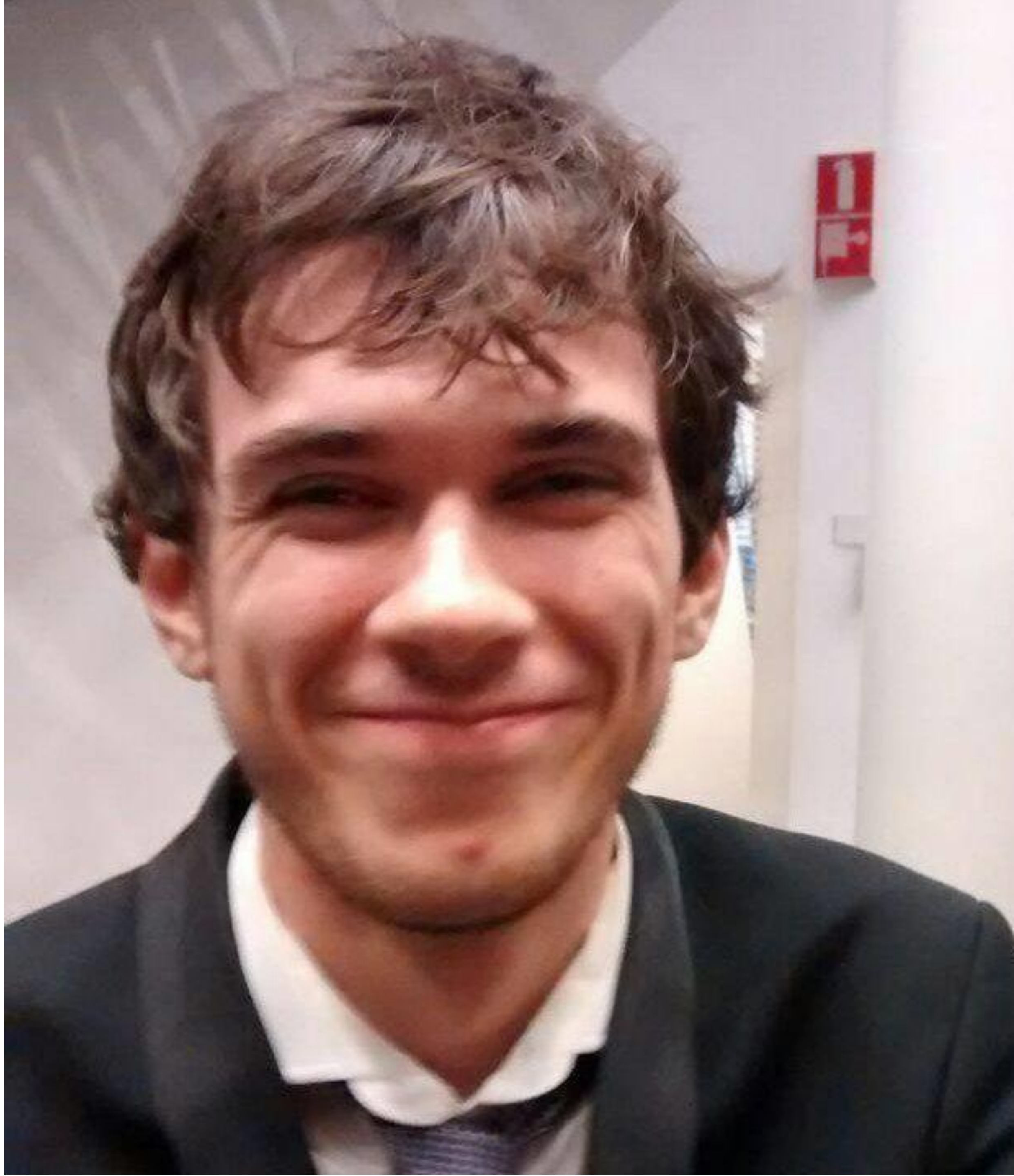
Page coloring (keep browser memory separate) → hard
Detection (performance counters) → hard
Secure timers → hard
Separate caches → expensive

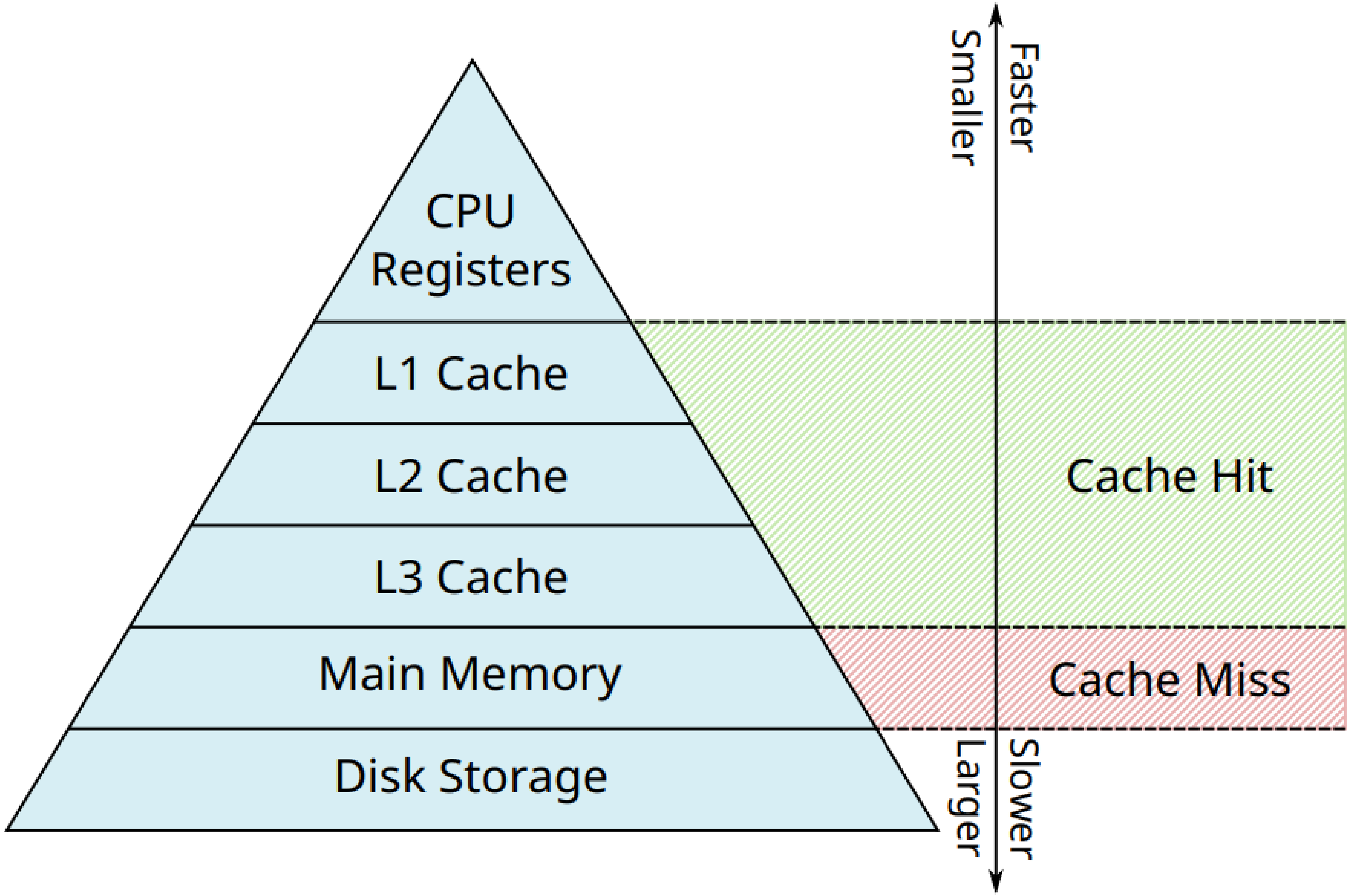
Malicious Management Unit

Why Stopping Cache Attacks in Software is Harder Than You Think

USENIX Security'18

Stephan van Schaik

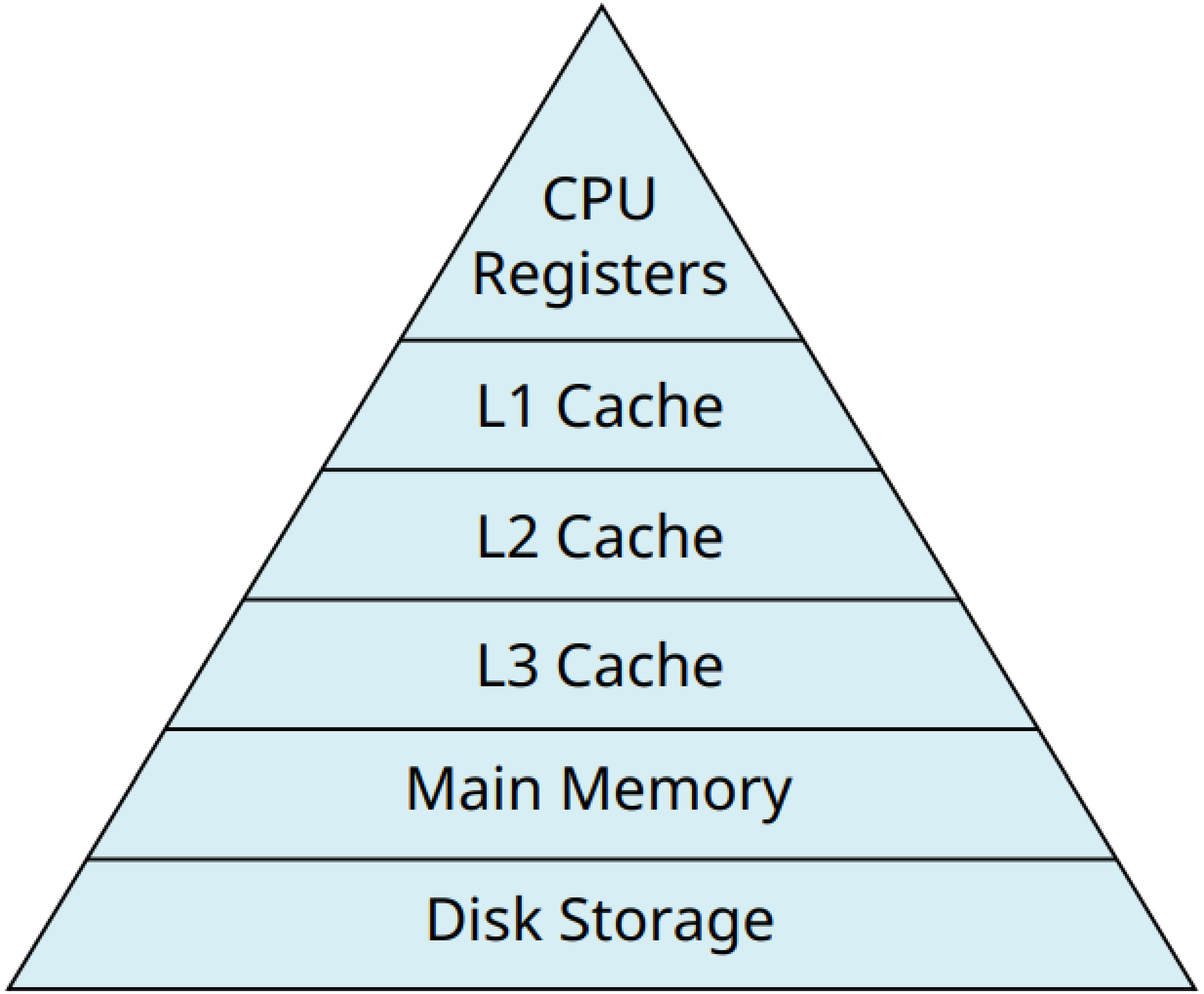


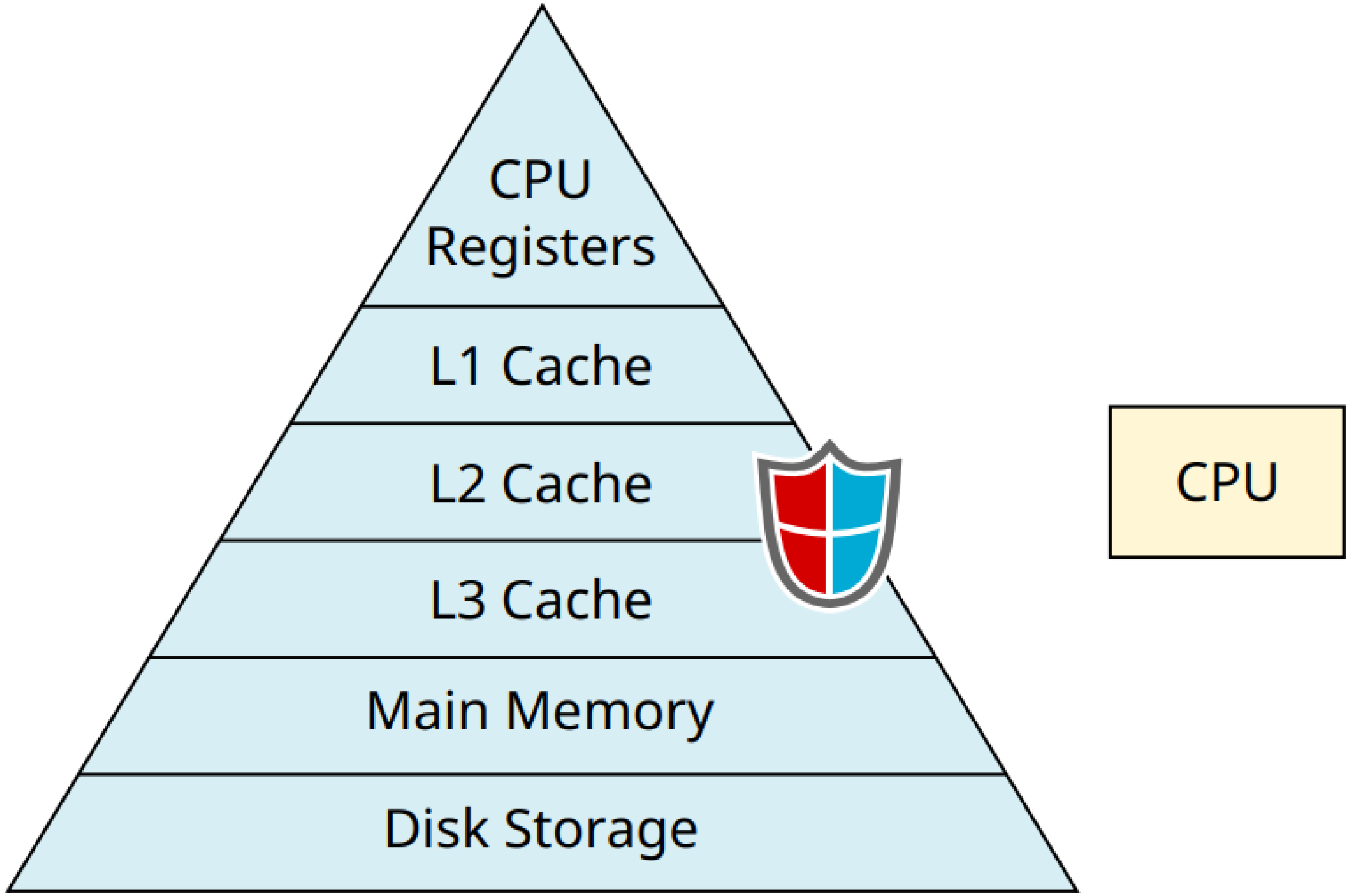


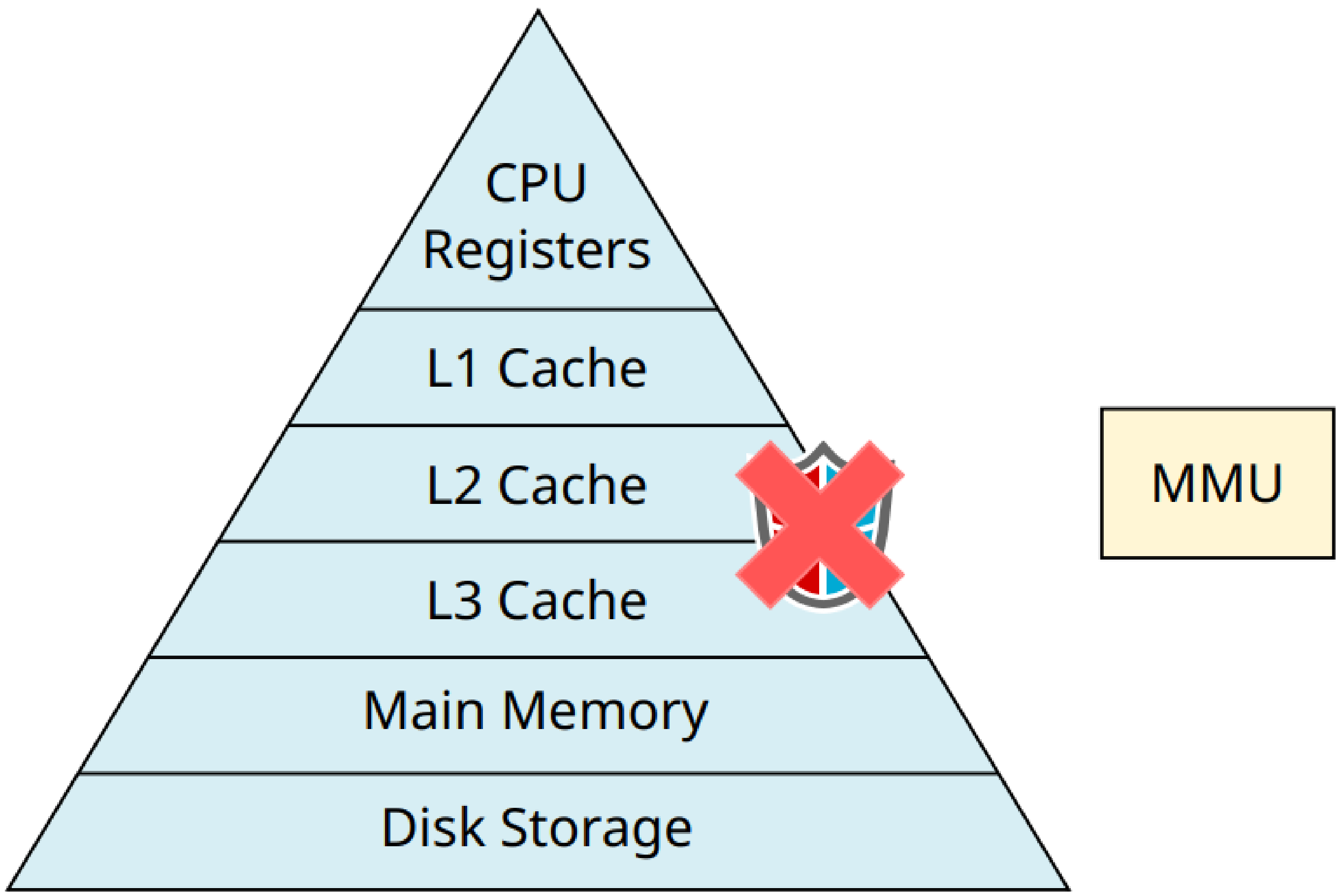
Memory accesses are not performed in constant time

Caches matter

- Caches are shared resources
- Caches can be manipulated
- Spy on other processes
- Input events
- Leak sensitive data




CPU
Registers
L1 Cache
L2 Cache
L3 Cache
Main Memory
Disk Storage
CPU

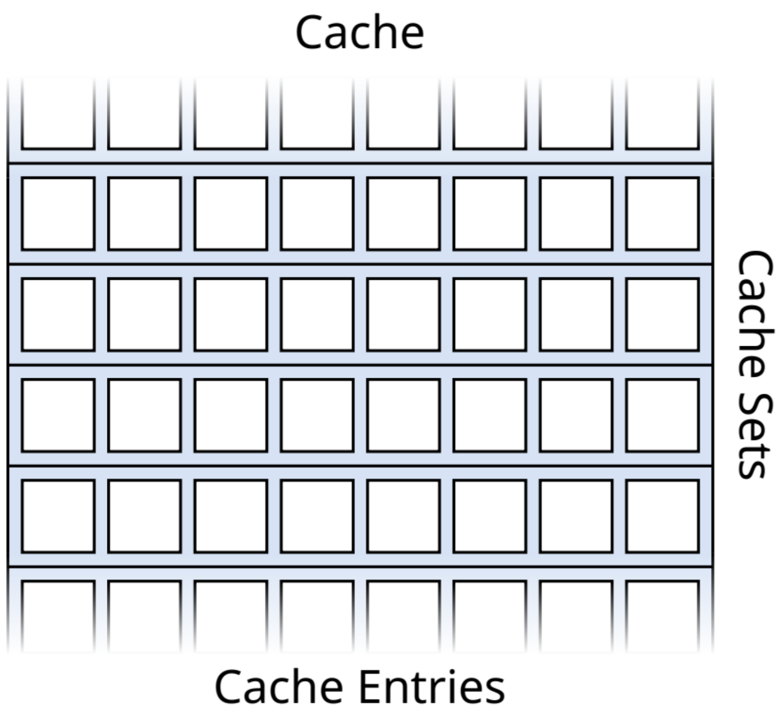

CPU
Registers
L1 Cache
L2 Cache
L3 Cache
Main Memory
Disk Storage
MMU

AES

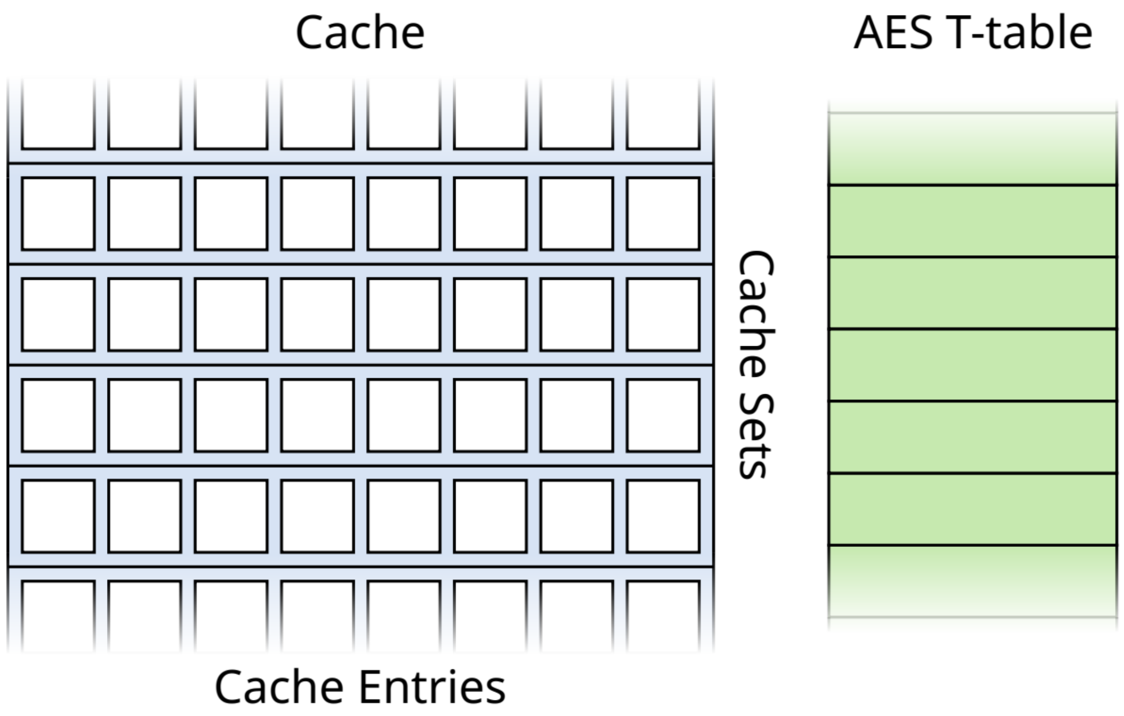
- Advanced Encryption Standard
- Software implementations use T-tables
- $T[p_i \oplus k_i]$
- Indices are key-dependent
- Elements may be in main memory or the cache

An example of PRIME + PROBE against AES

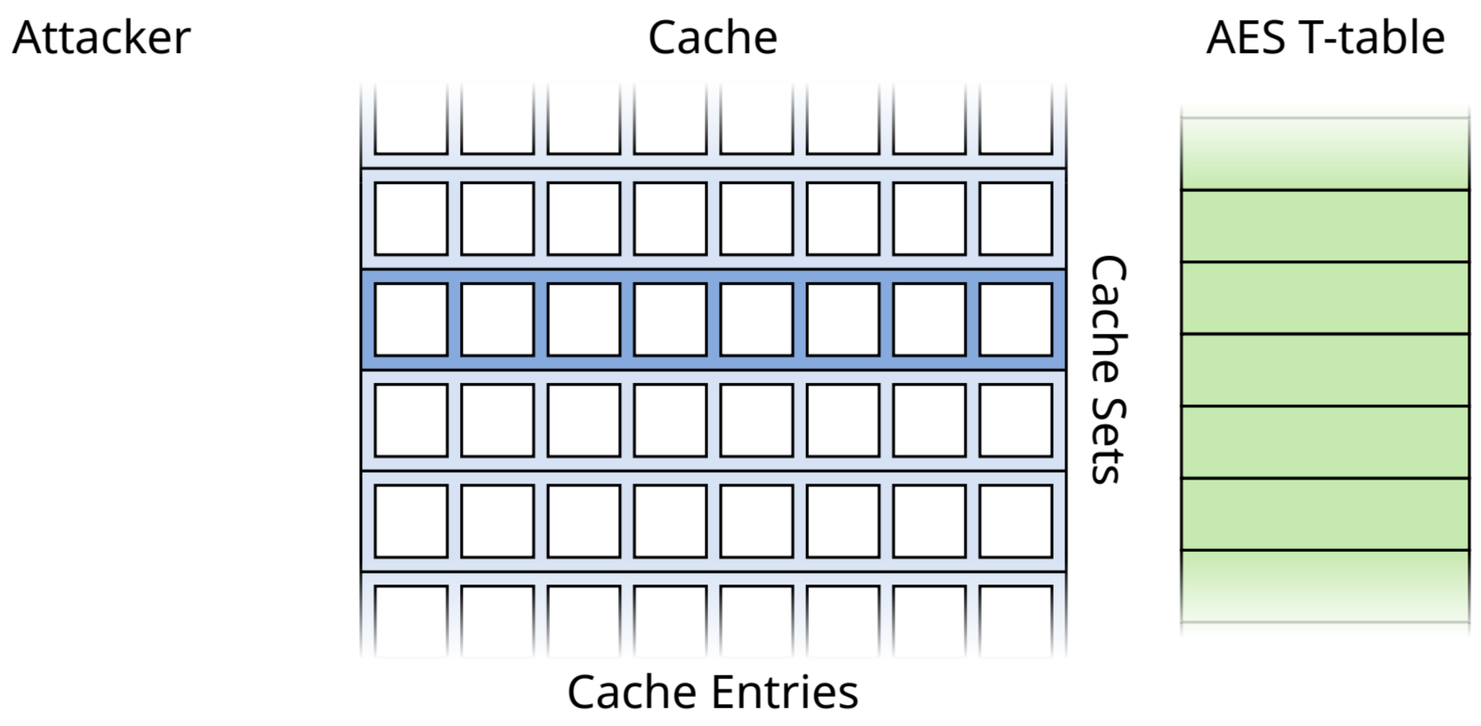
PRIME + PROBE



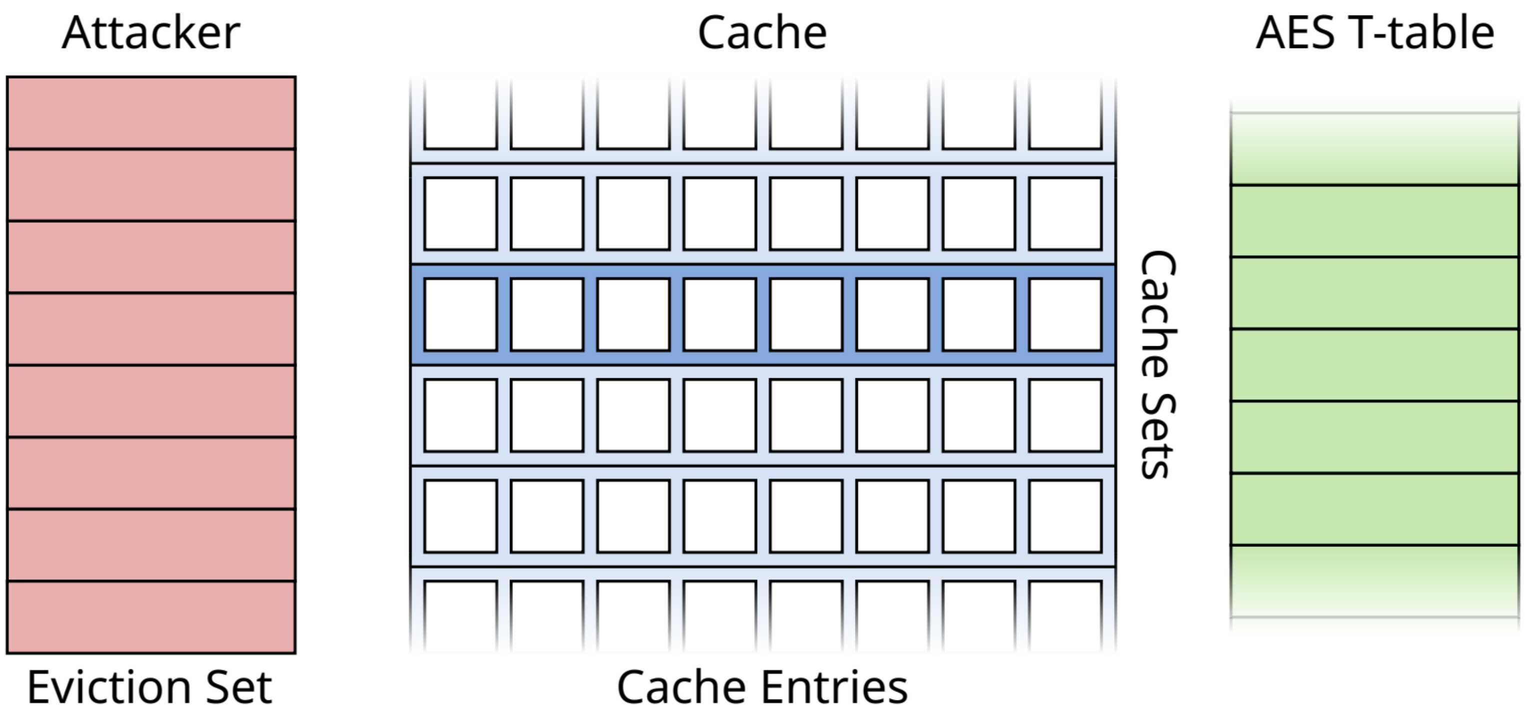
PRIME + PROBE



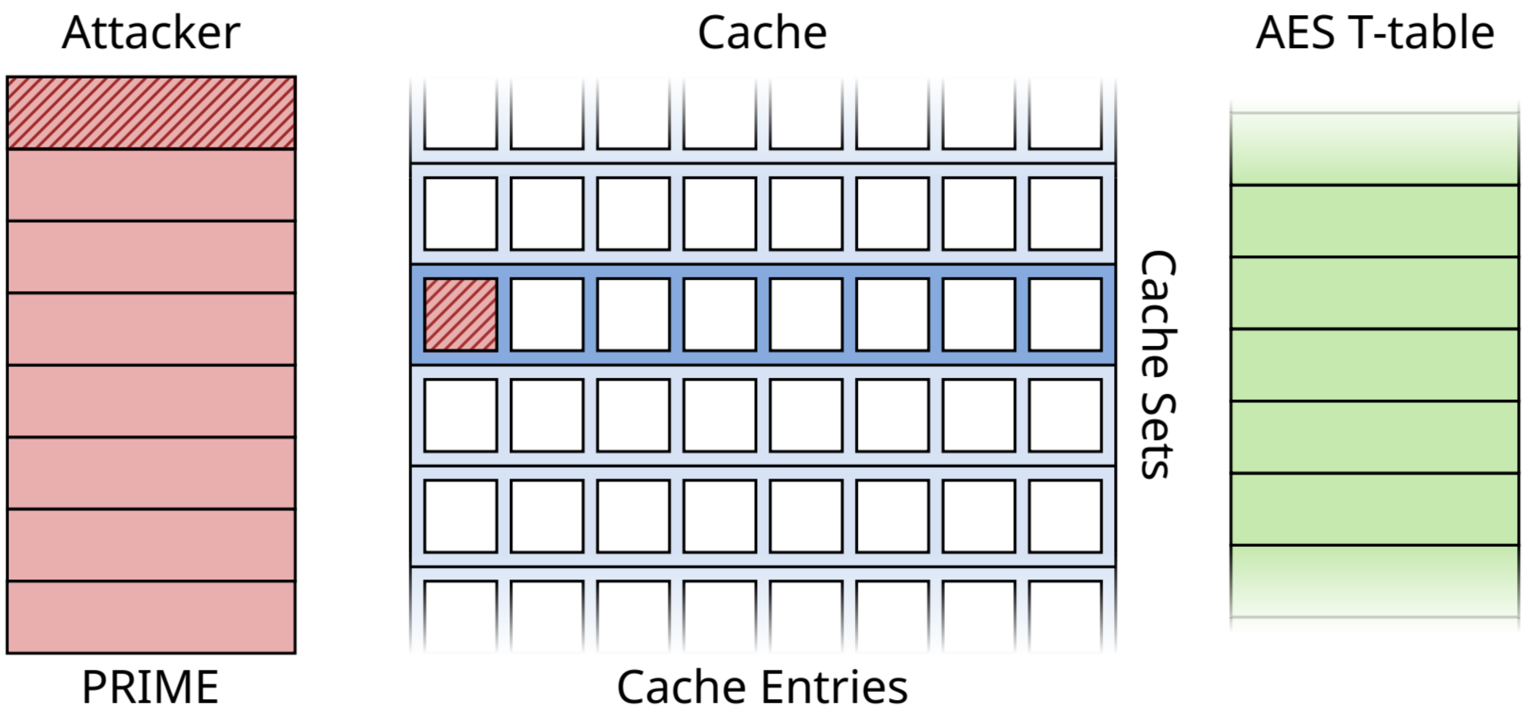
PRIME + PROBE



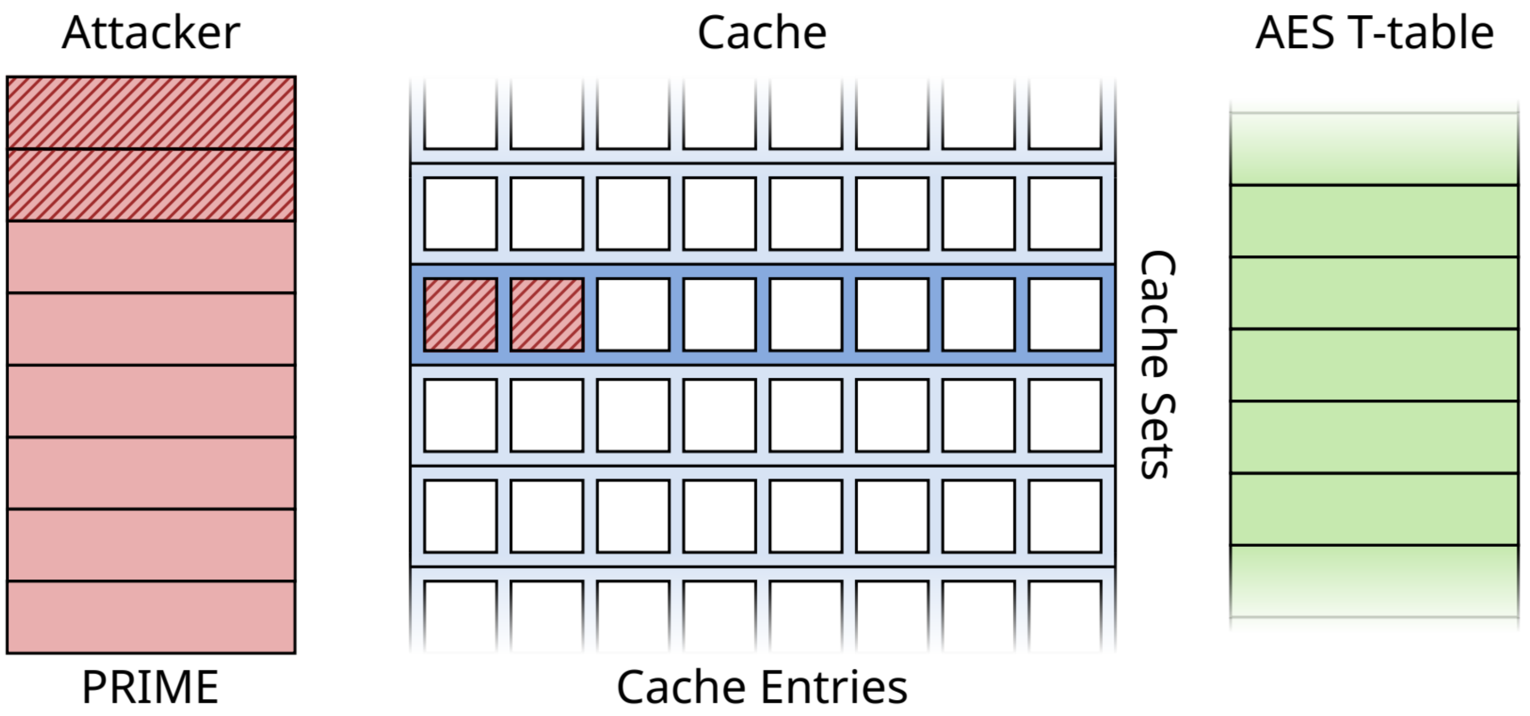
PRIME + PROBE


Attacker
Cache
AES T-table
Cache Sets
Eviction Set
Cache Entries

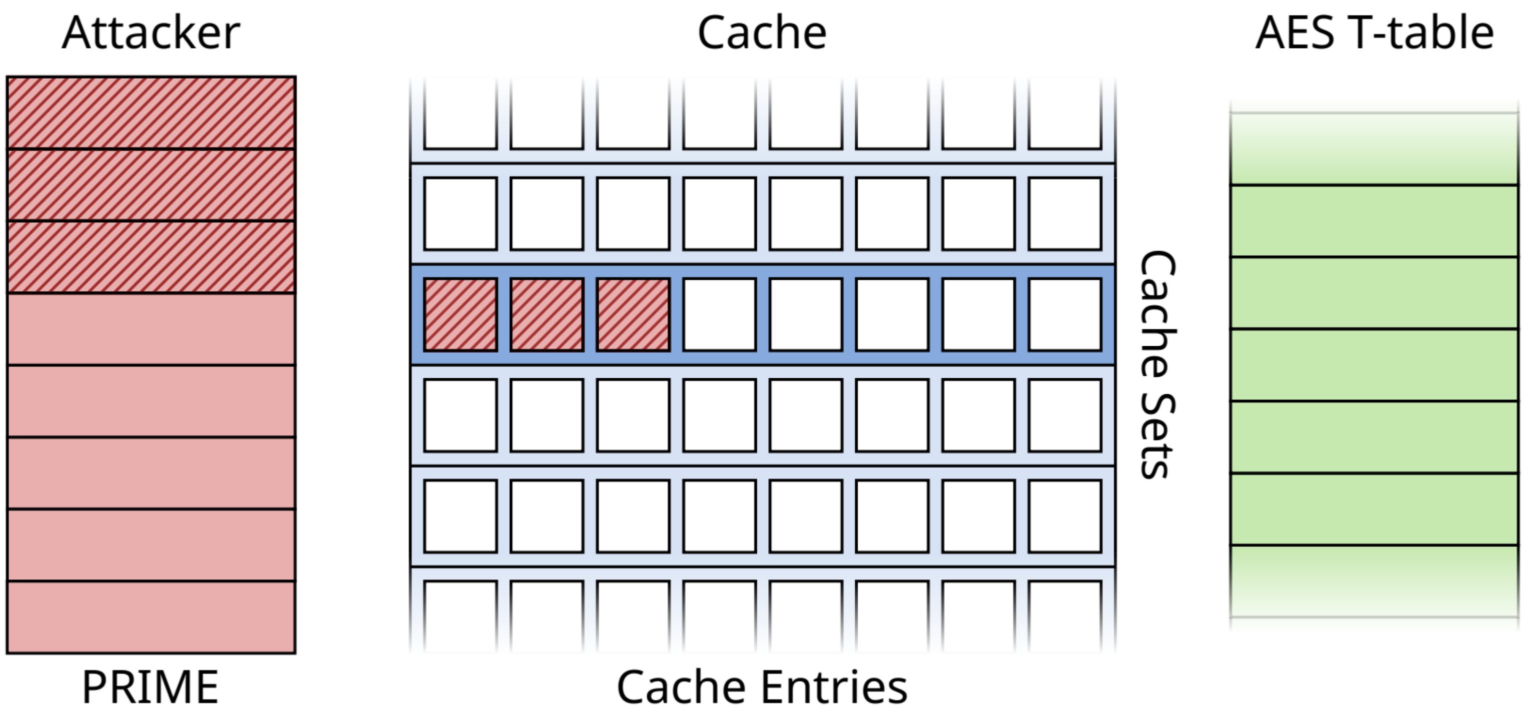
PRIME + PROBE



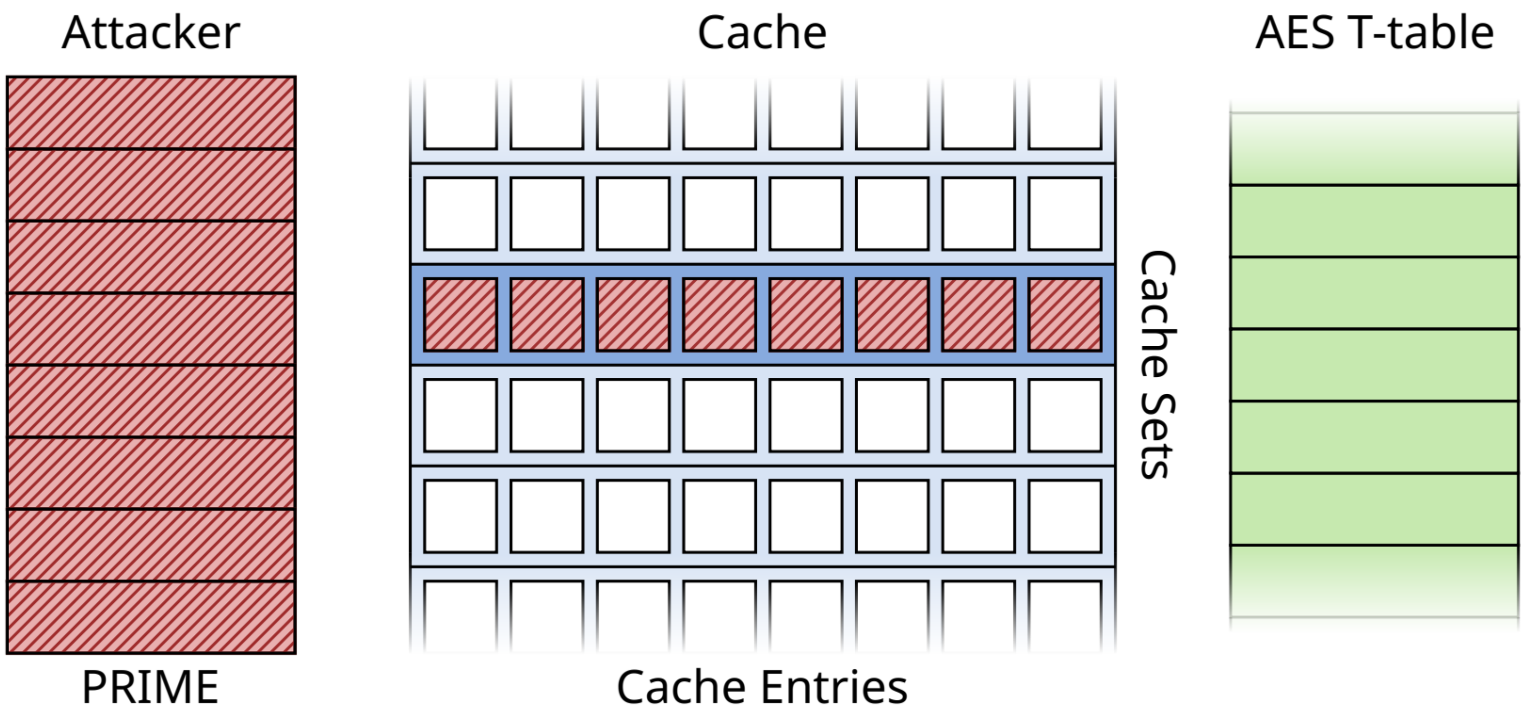
PRIME + PROBE



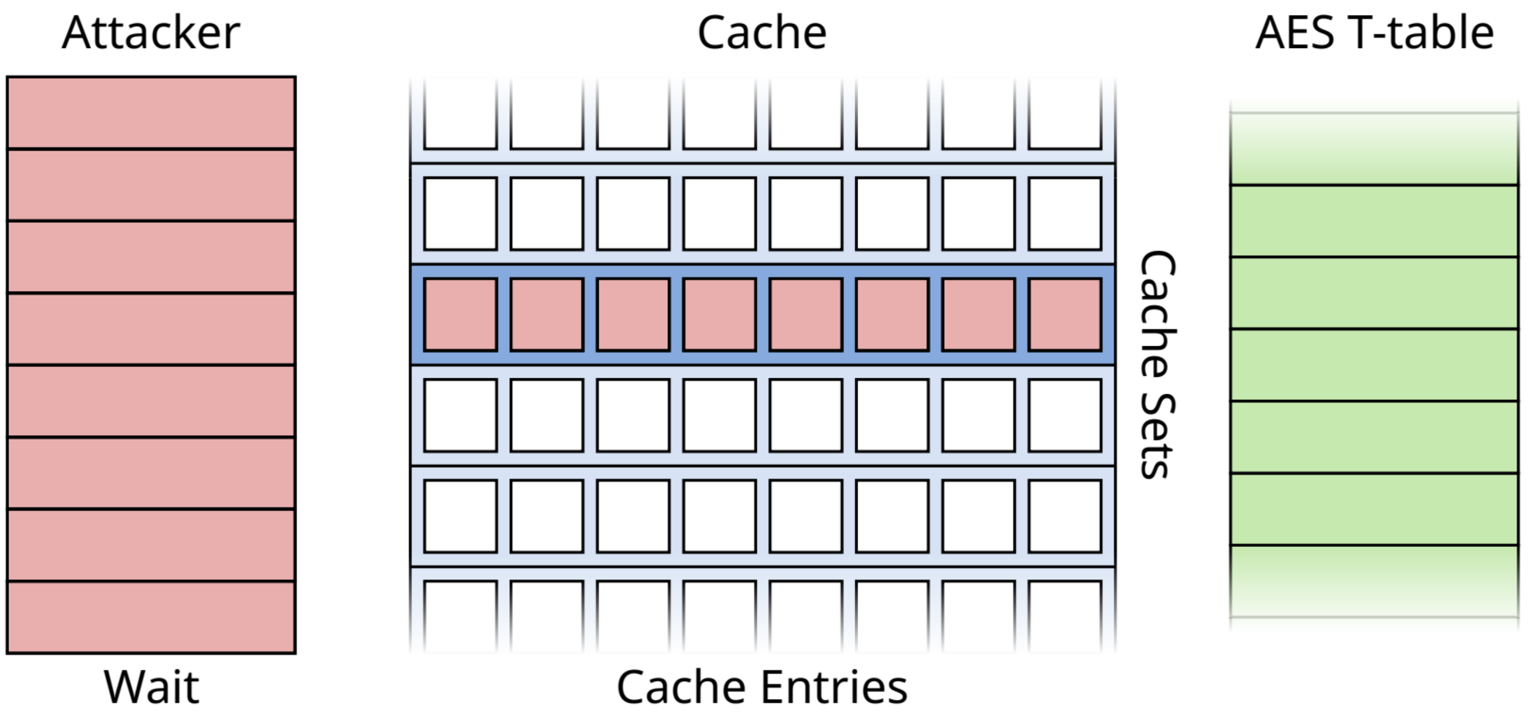
PRIME + PROBE



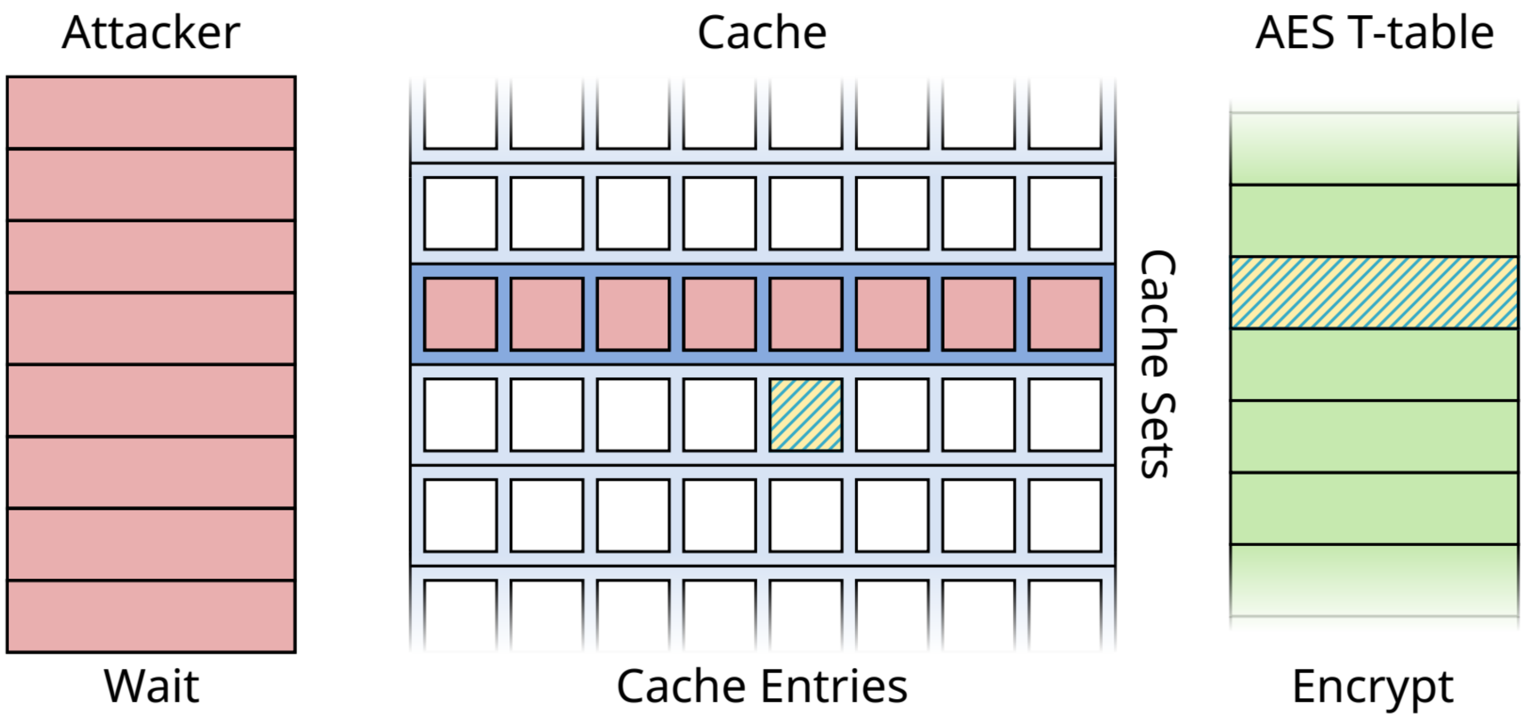
PRIME + PROBE



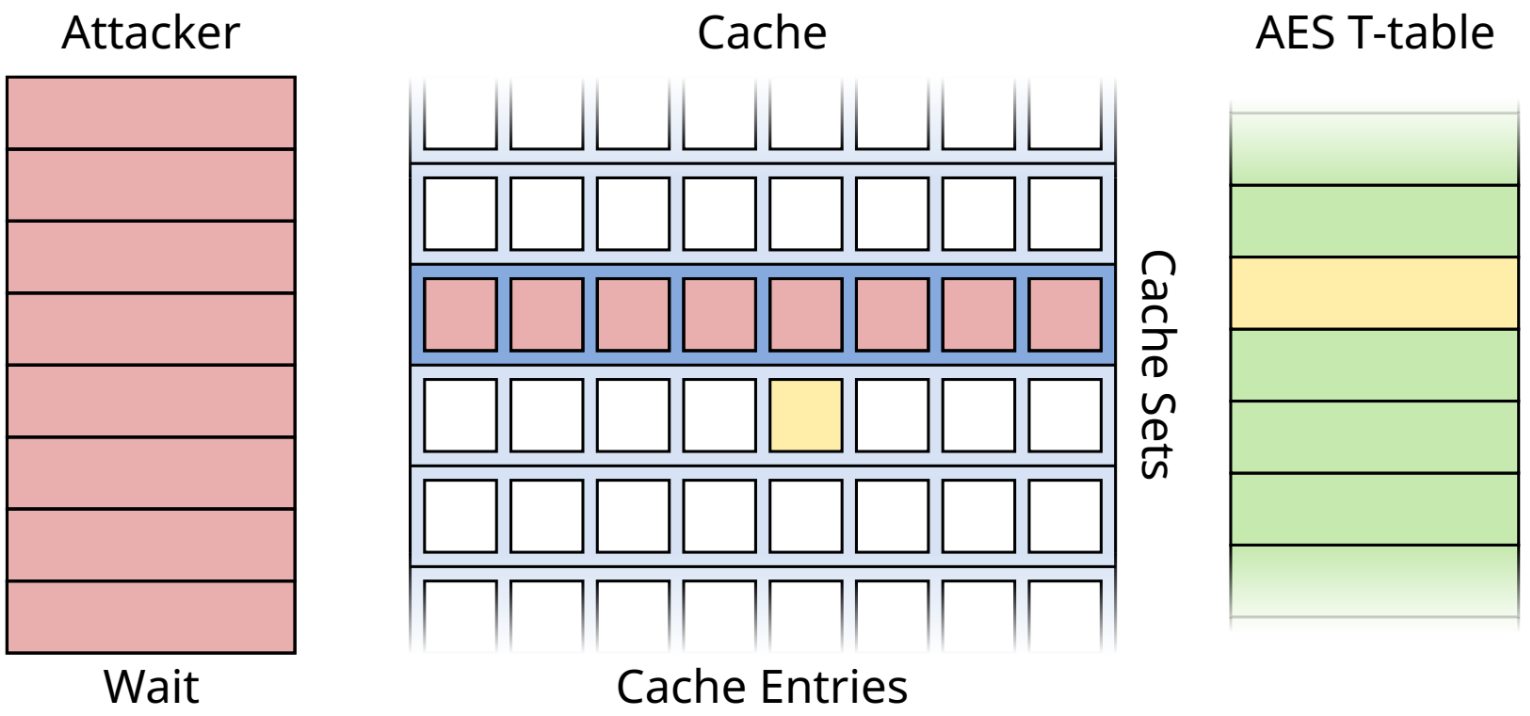
PRIME + PROBE



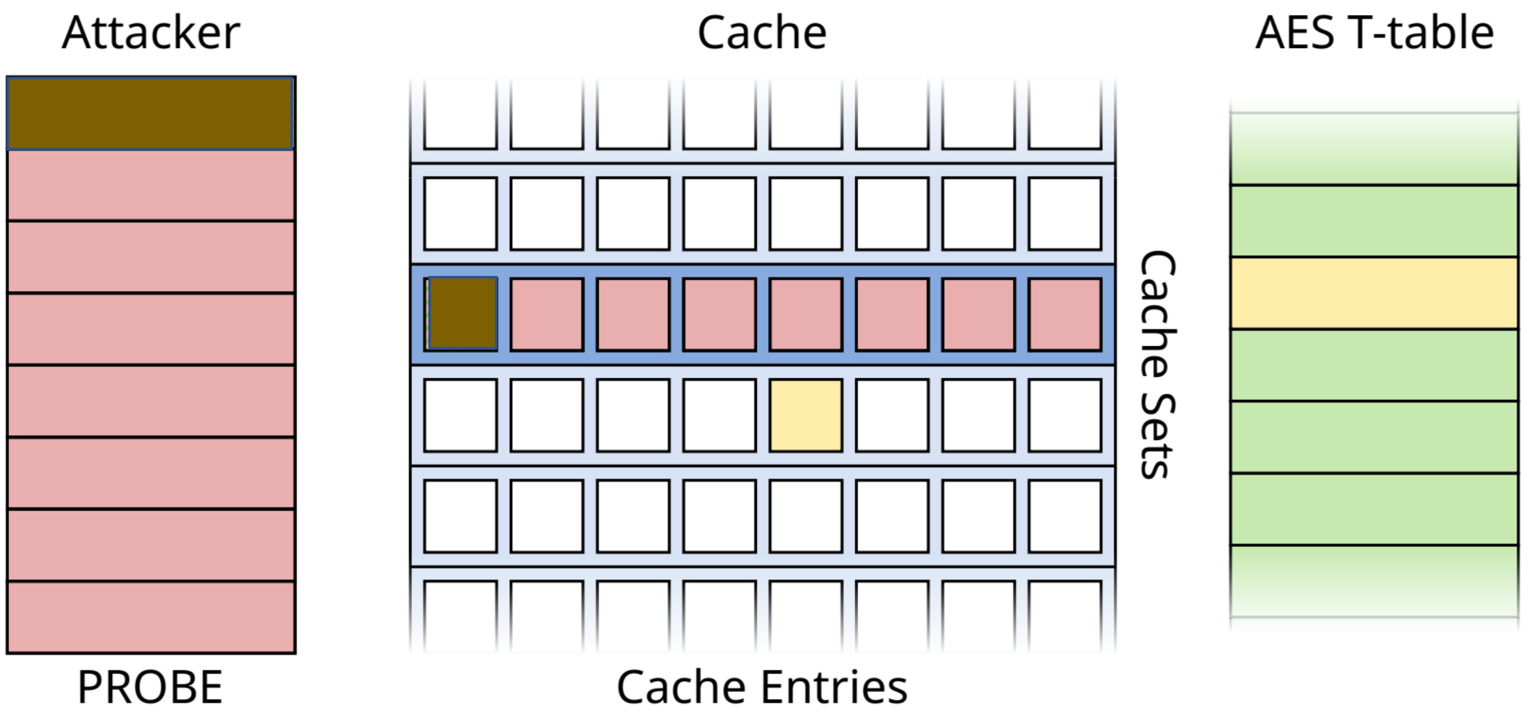
PRIME + PROBE


Attacker
Cache
AES T-table
Cache Sets
Wait
Cache Entries
Encrypt

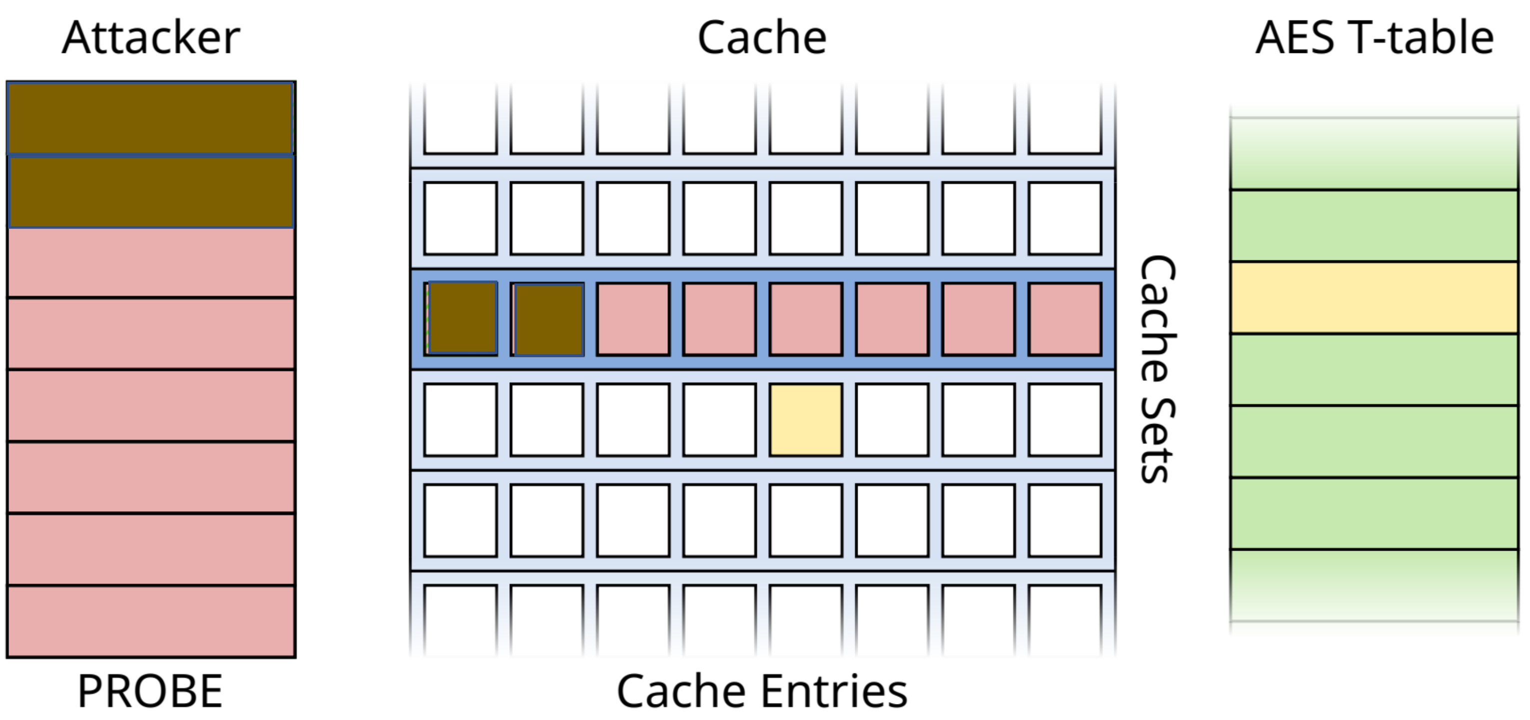
PRIME + PROBE



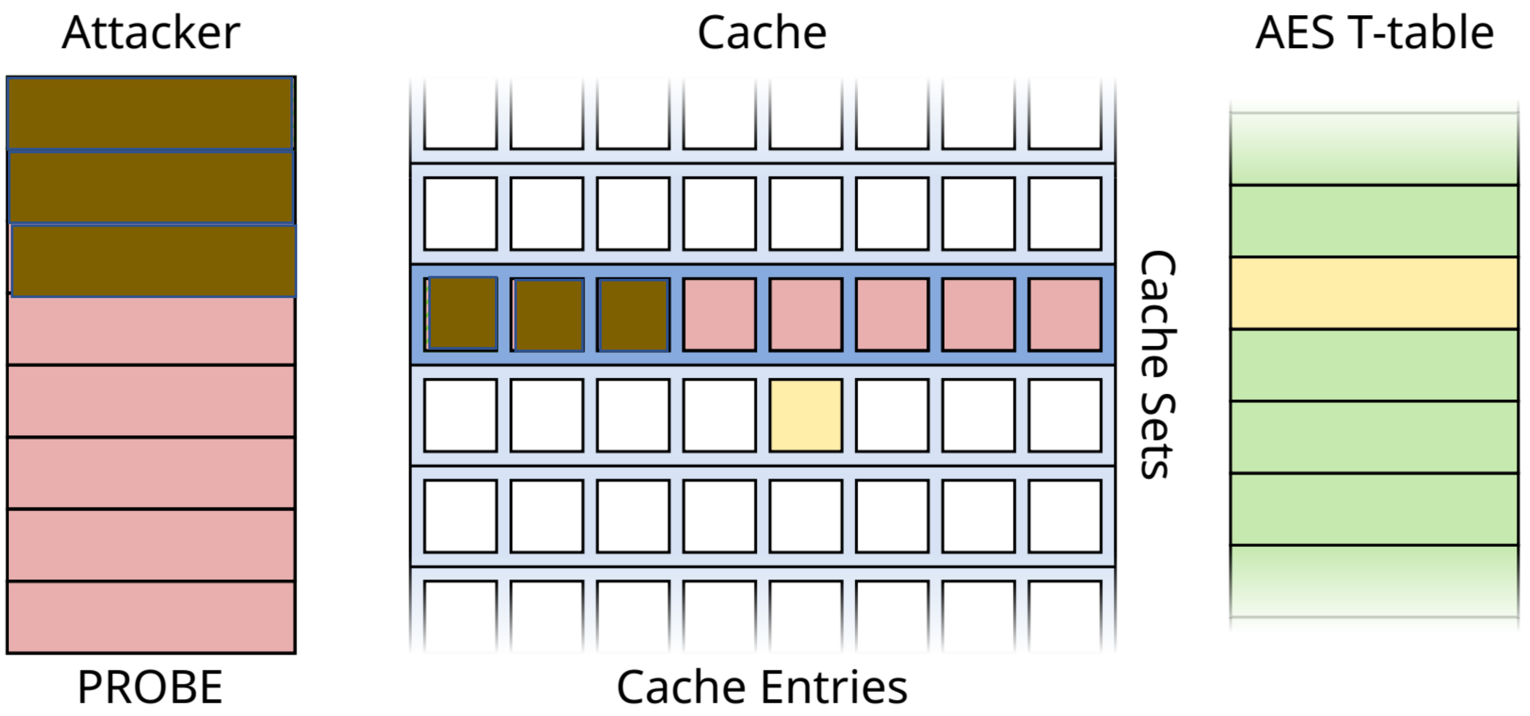
PRIME + PROBE



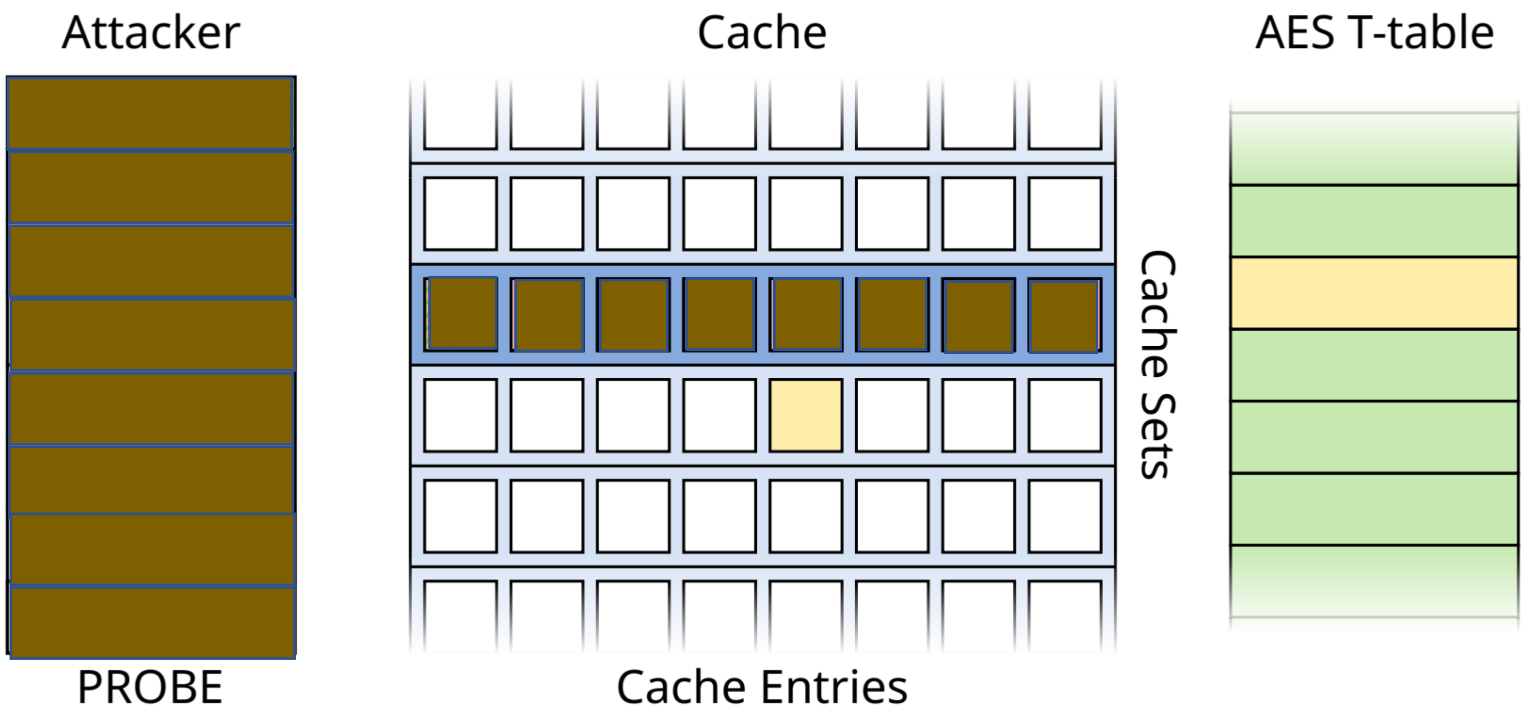
PRIME + PROBE



PRIME + PROBE

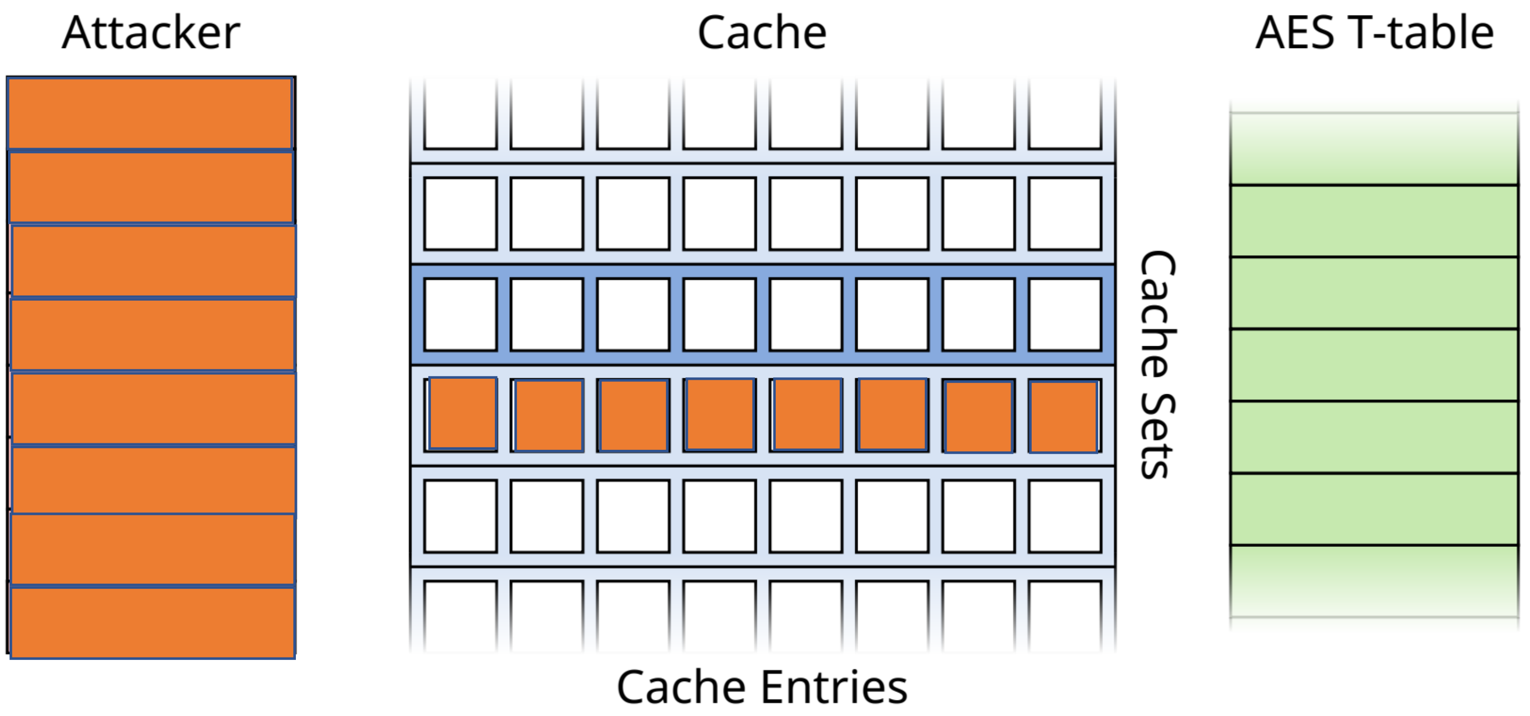


PRIME + PROBE

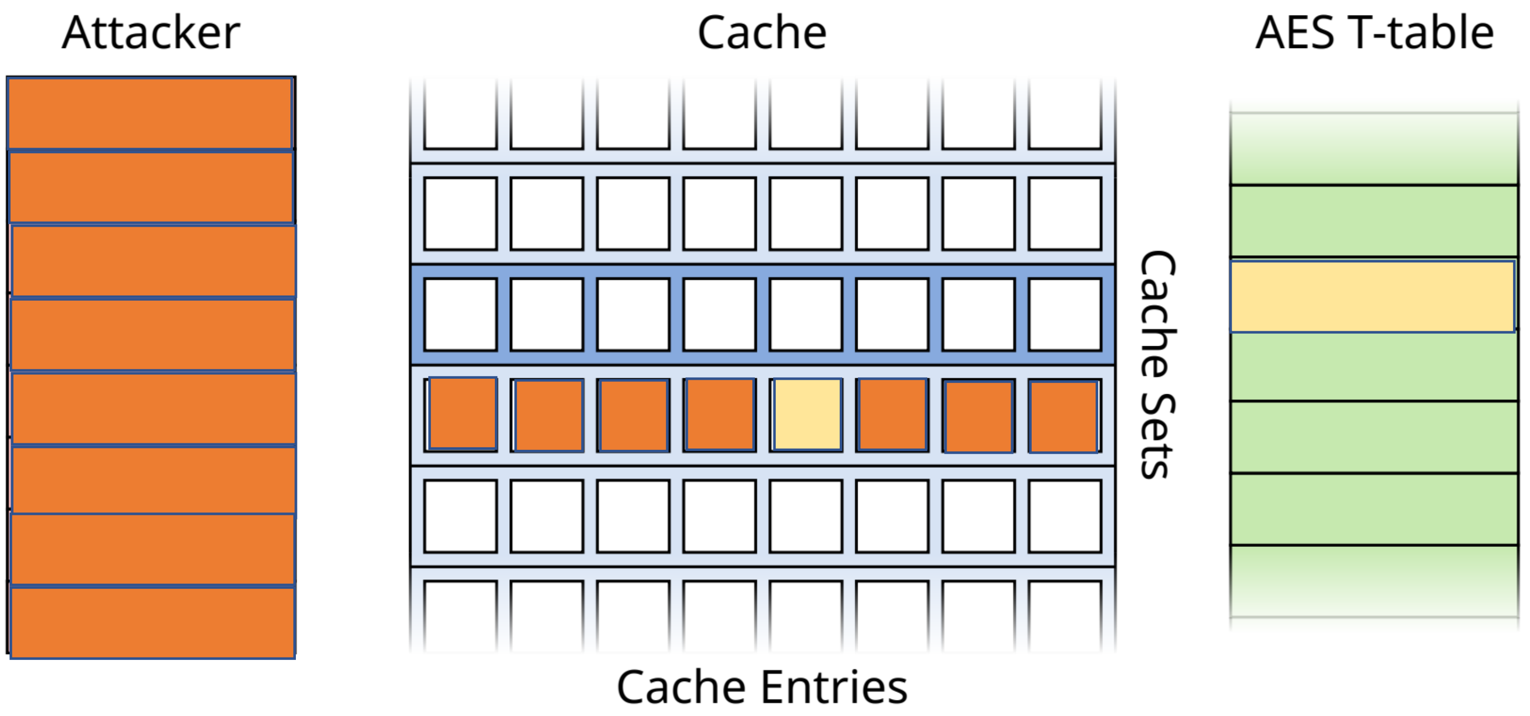


AES encrypt used another cache set

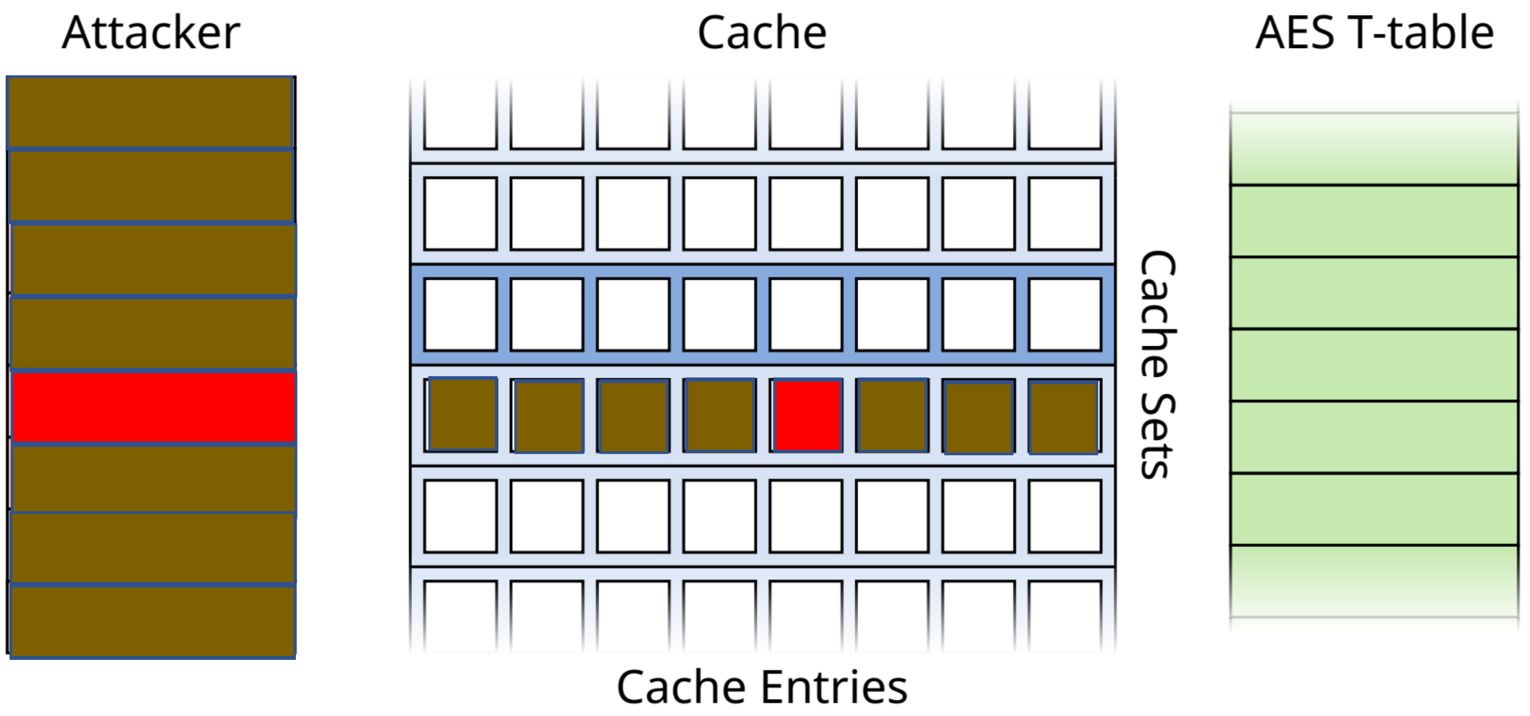
PRIME + PROBE



PRIME + PROBE

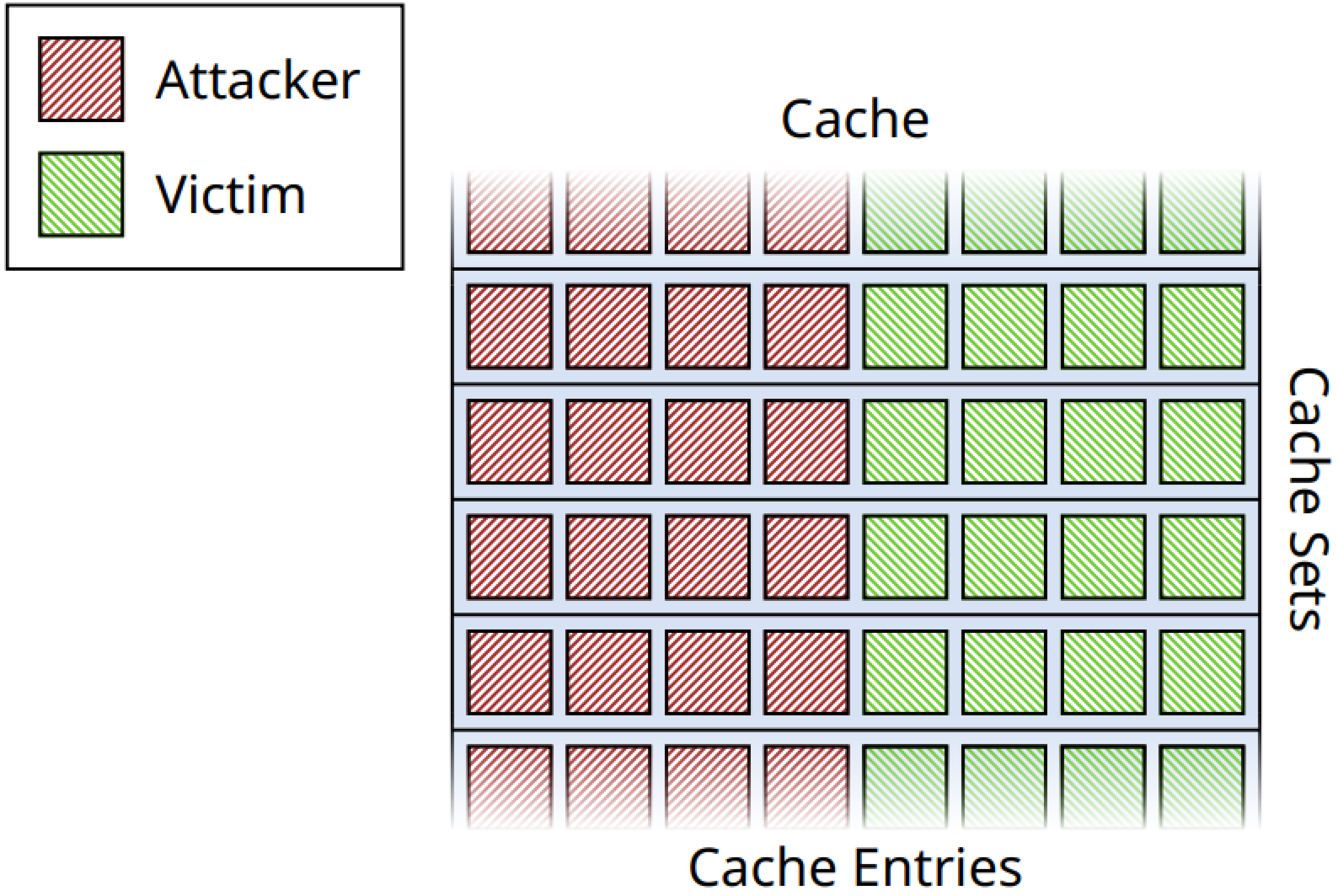


PRIME + PROBE

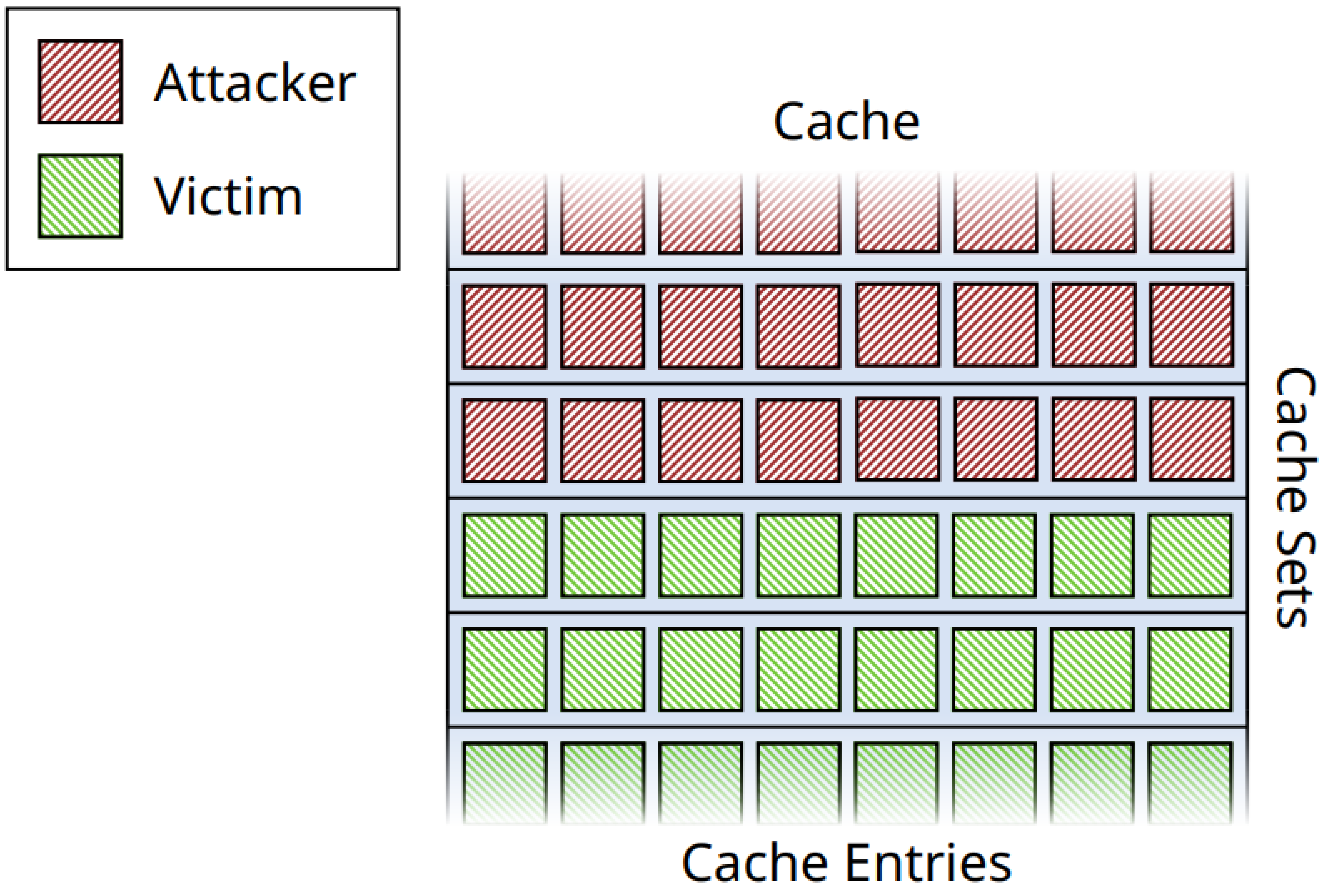


AES encrypt used the same cache set

What about defenses?



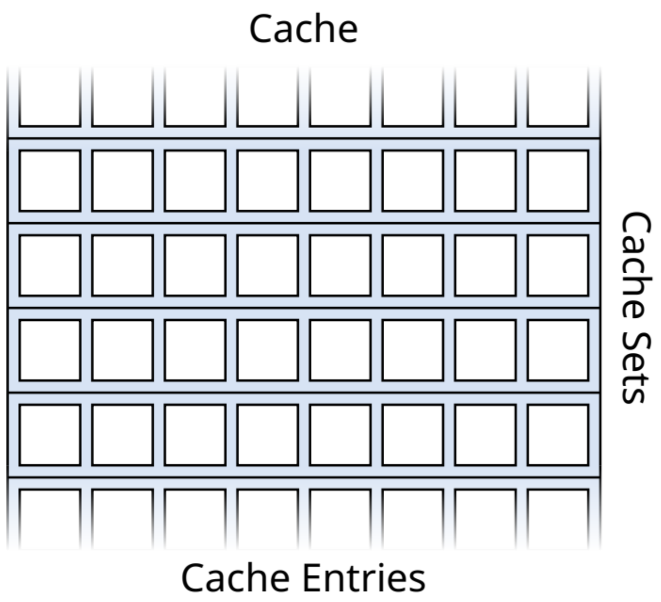
Way Partitioning



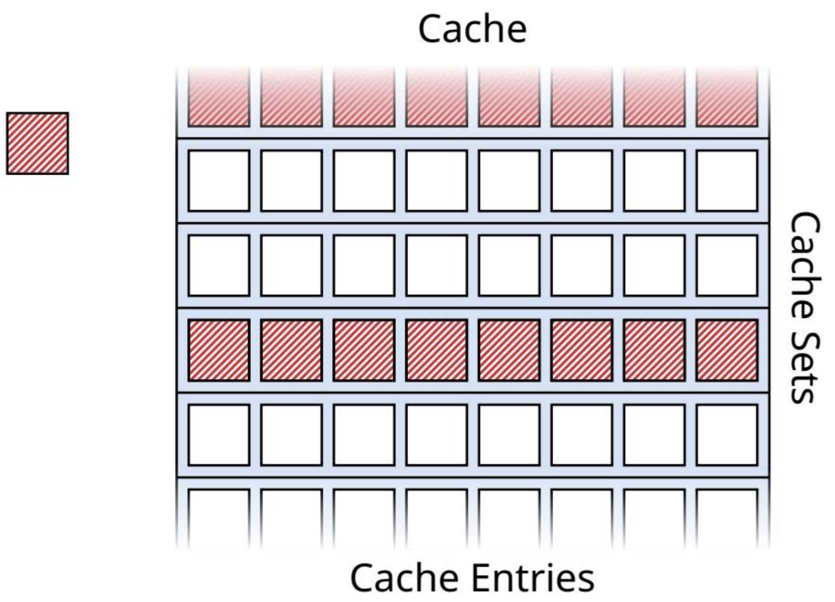
Set Partitioning

The magic of page coloring

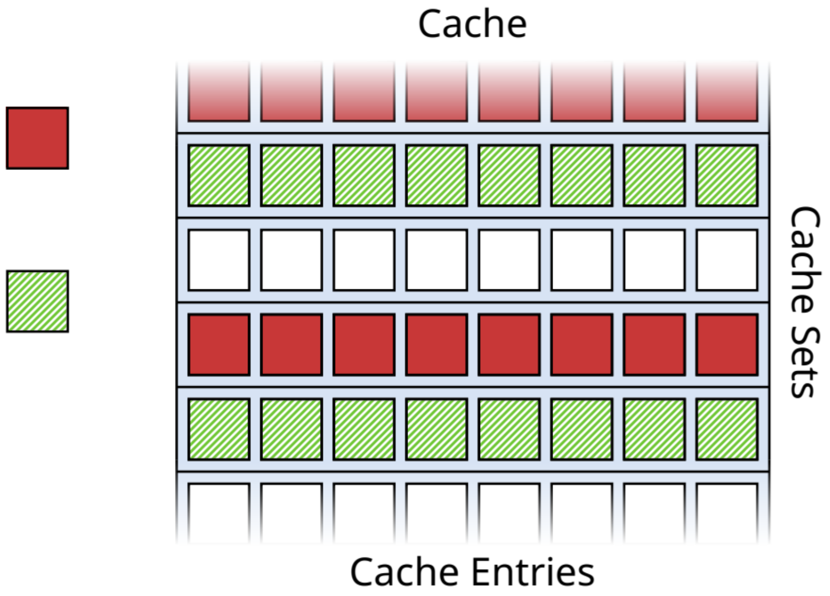
Page coloring



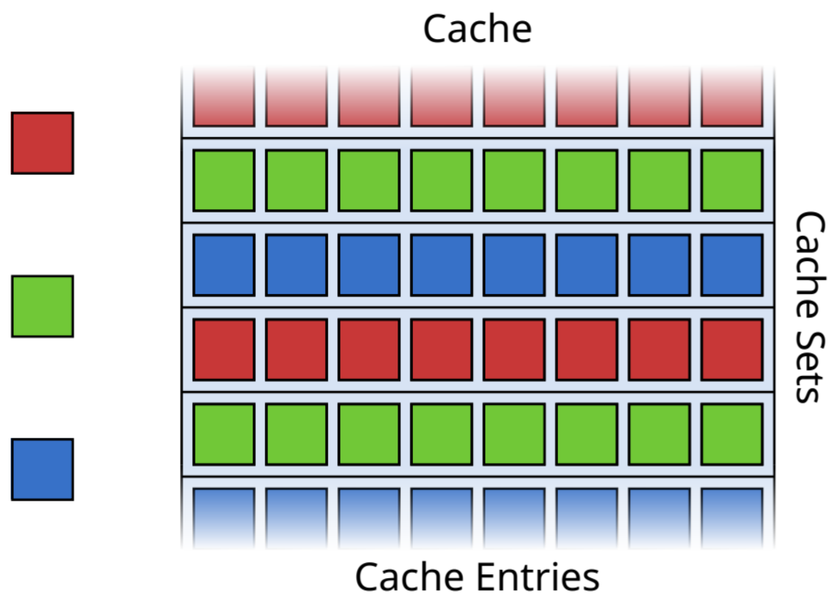
Page coloring



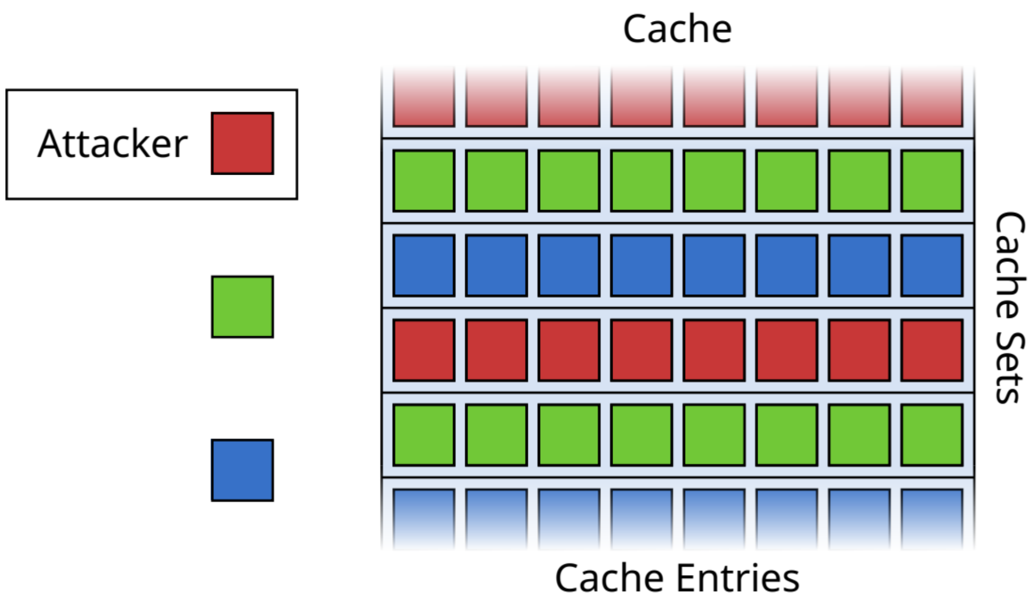
Page coloring



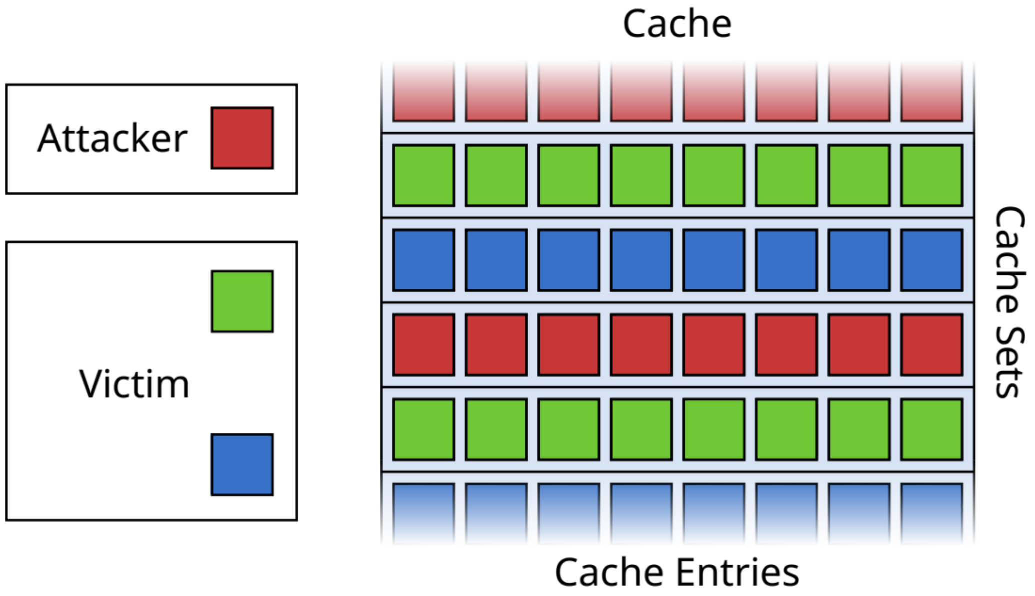
Page coloring



Page coloring



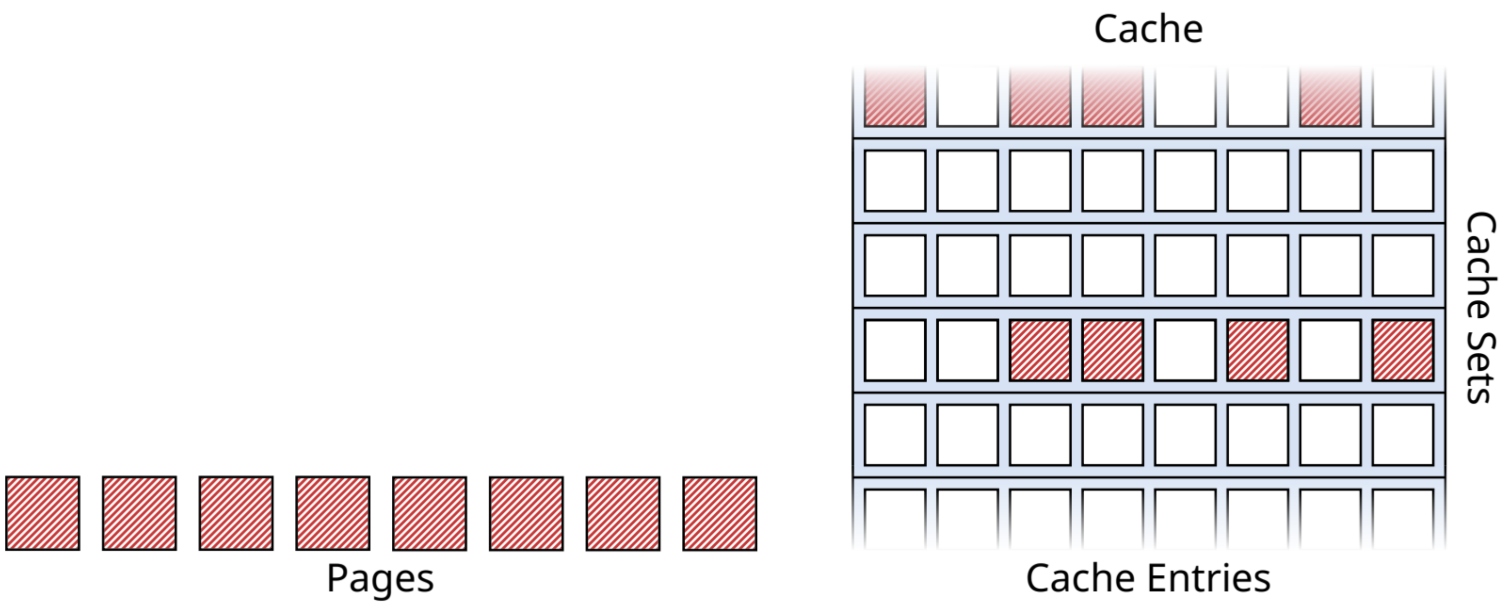
Page coloring



Victim and attacker are nicely isolated

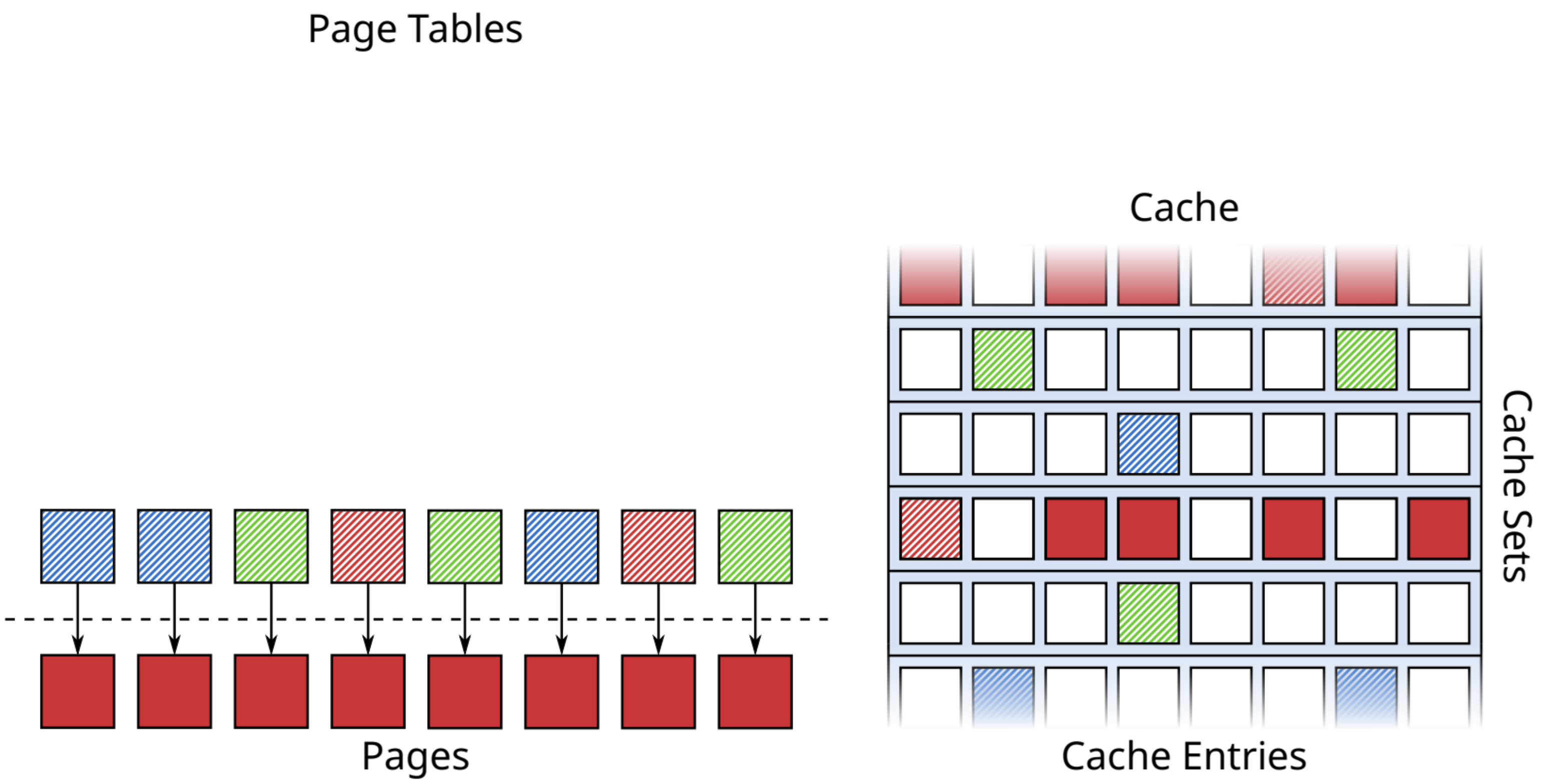
Or are they?

Page coloring



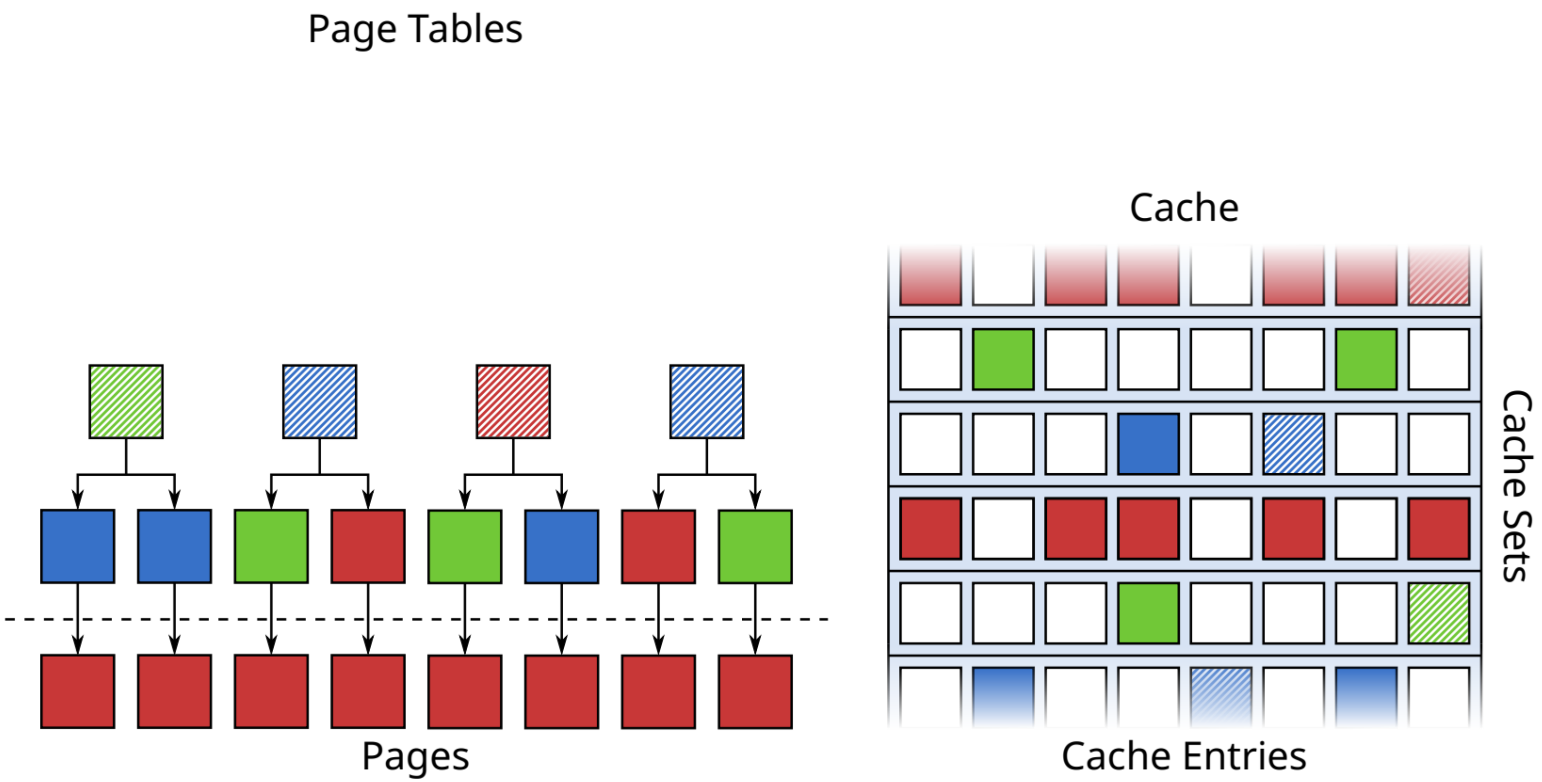
The attacker can only allocate red pages

Page coloring



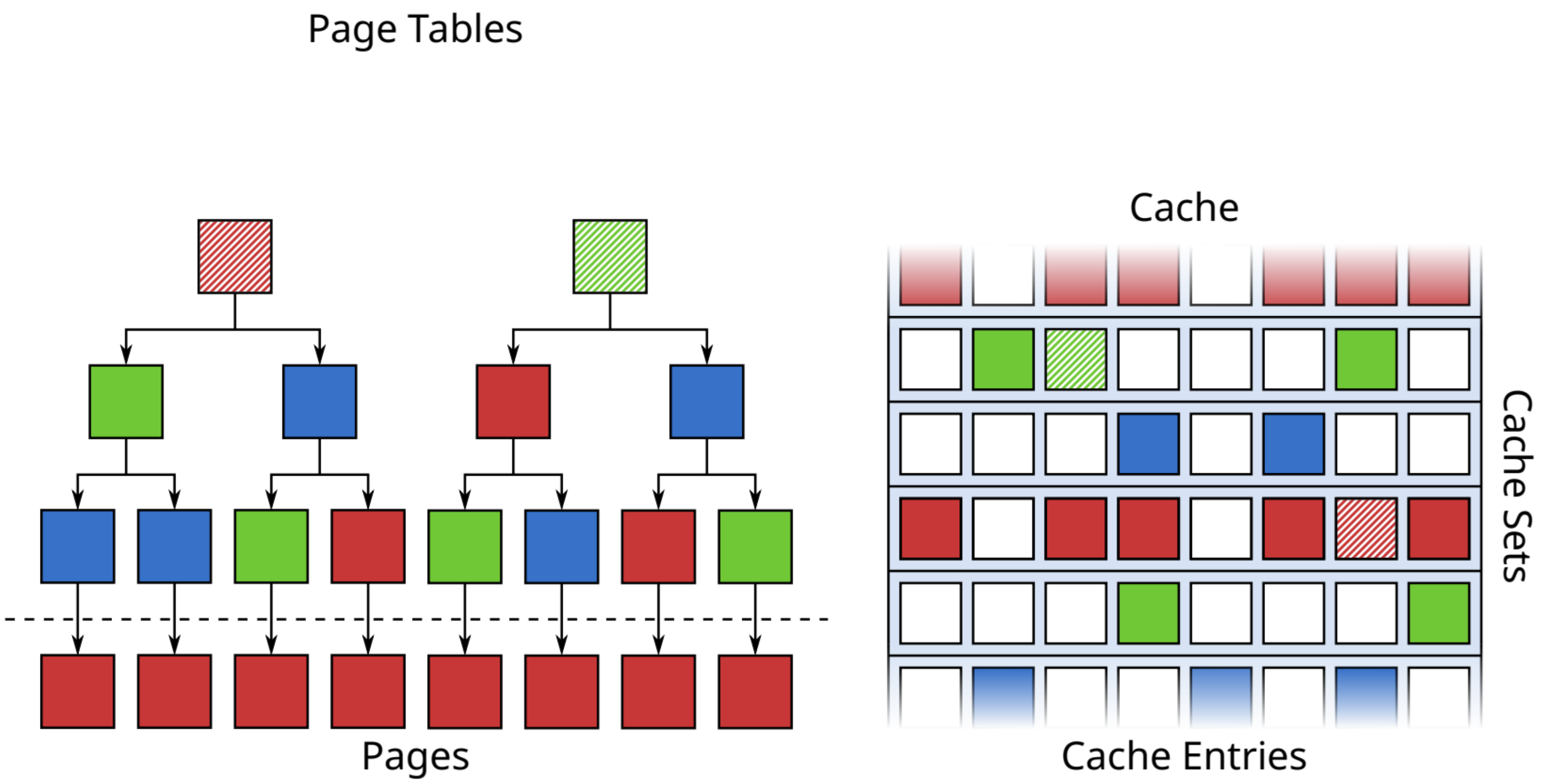
However, the page tables aren't colored

Page coloring



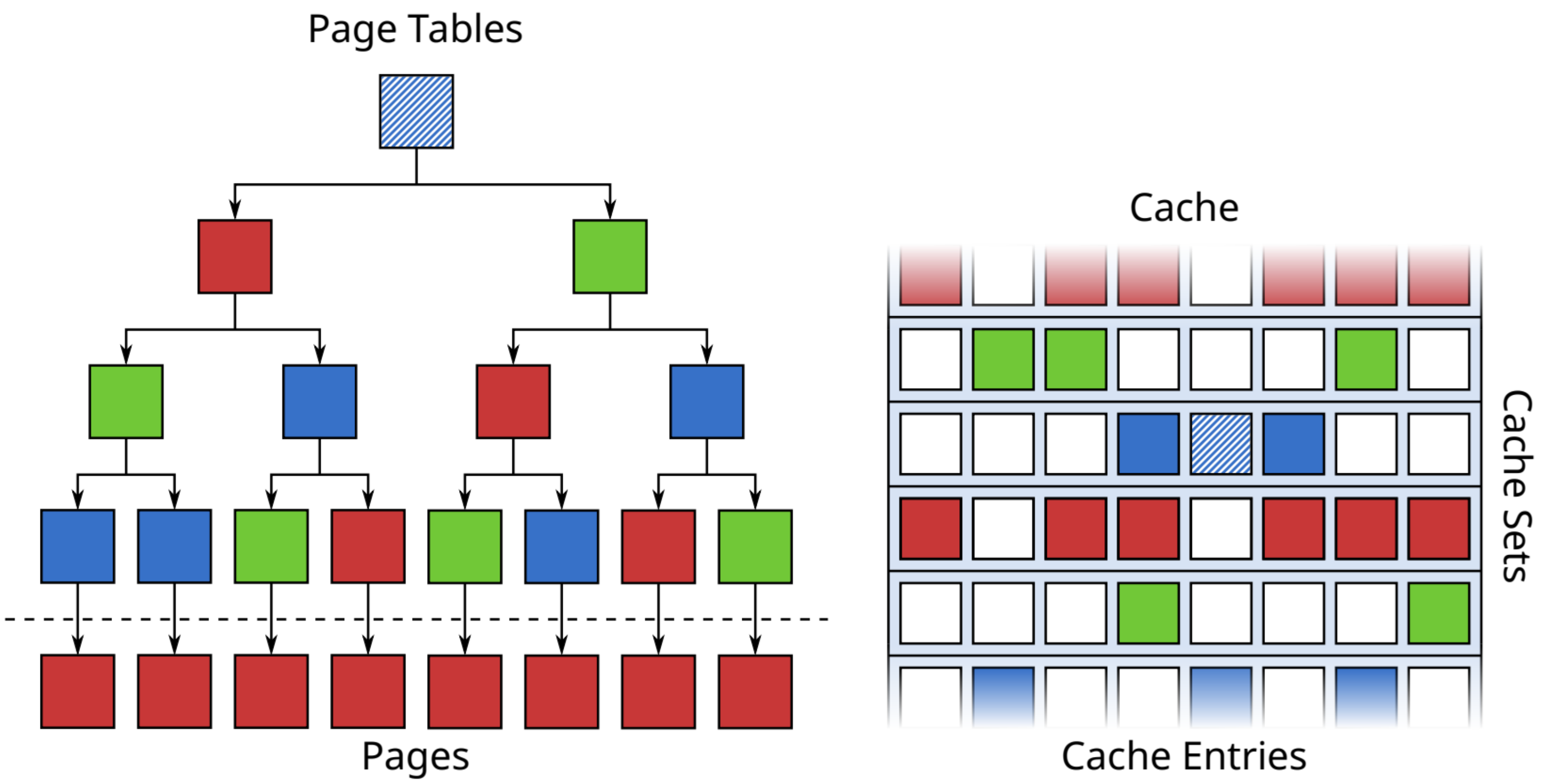
However, the page tables aren't colored

Page coloring



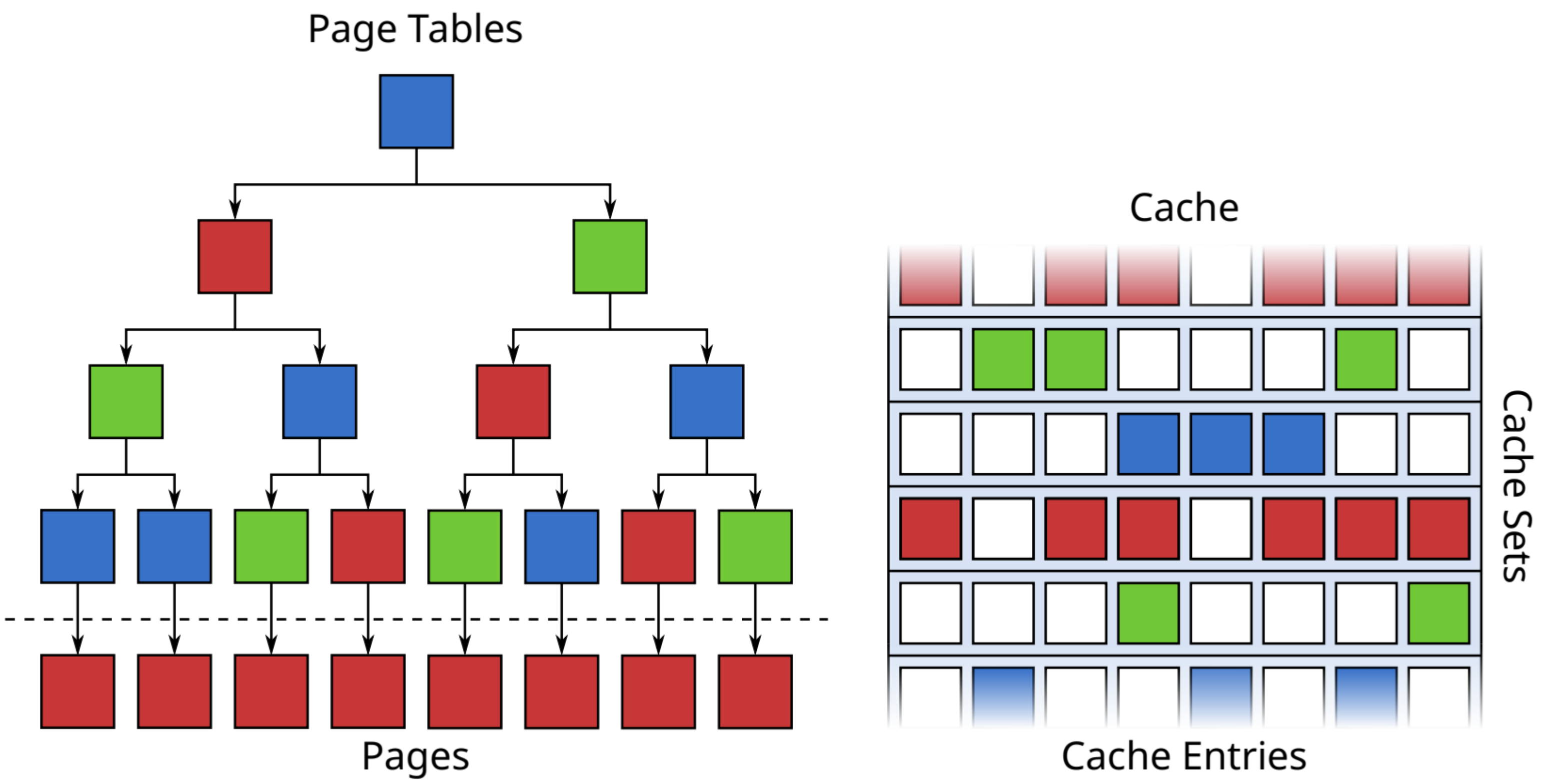
However, the page tables aren't colored

Page coloring



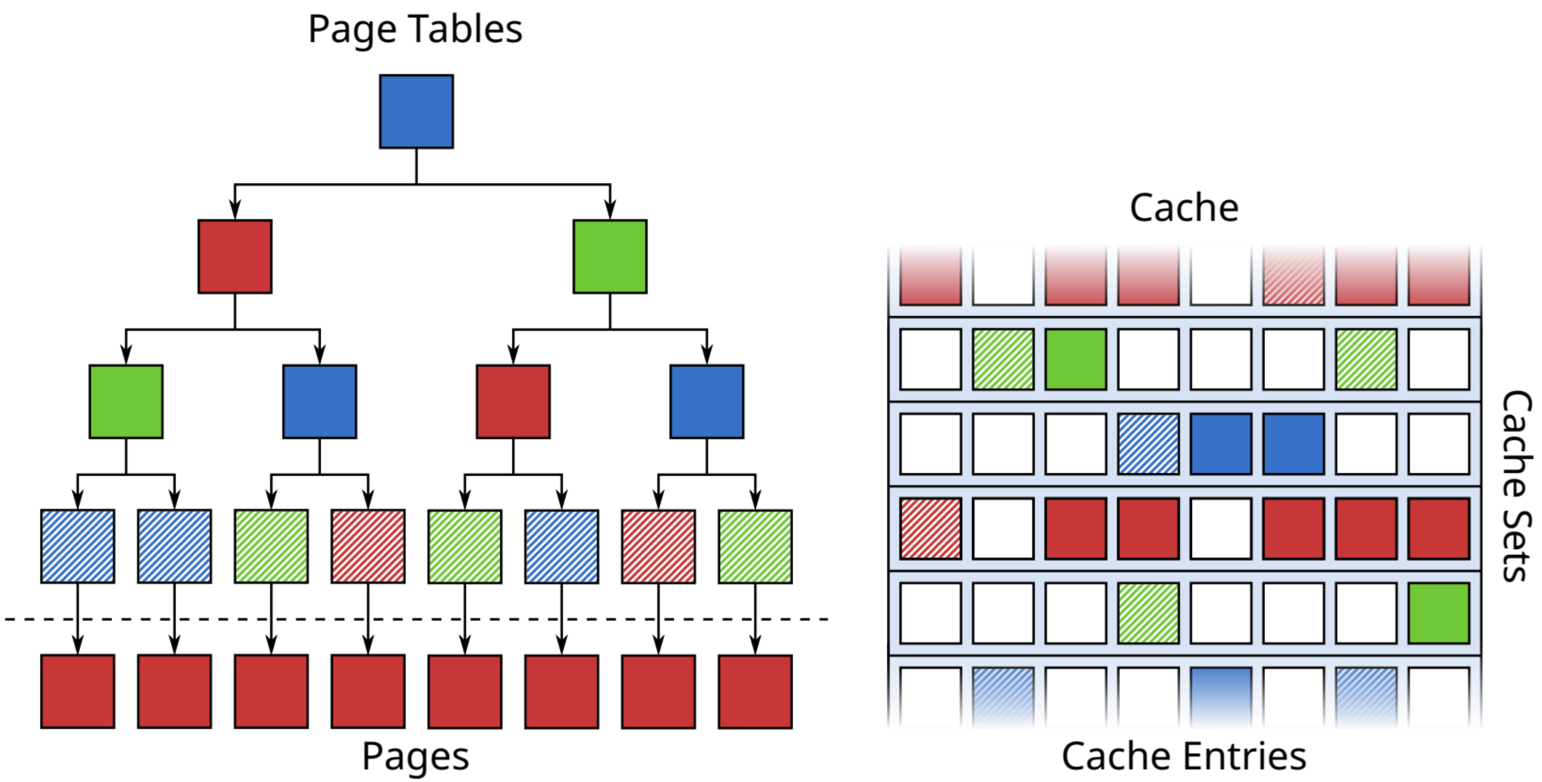
However, the page tables aren't colored

Page coloring



However, the page tables aren't colored

Page coloring



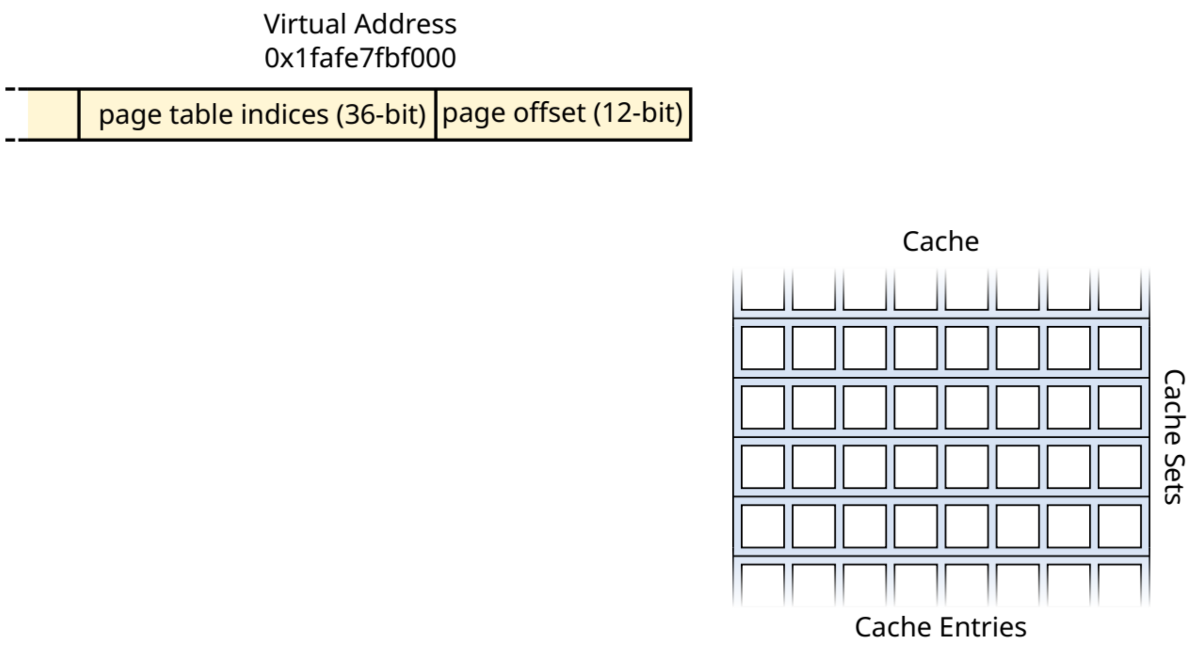
Can we control the page tables for cache attacks?

XLATE attacks

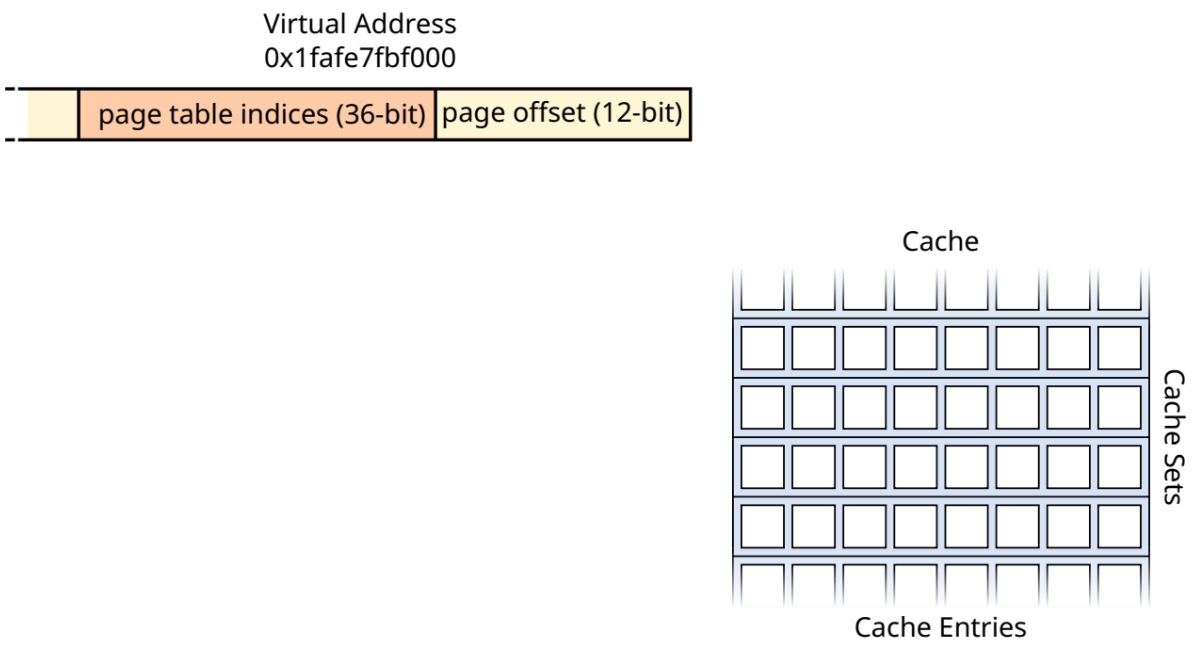
- Memory Management Unit (MMU)
- Translates virtual addresses into their physical counterparts
- Hence translate or XLATE attacks
- XLATE + PROBE caches page tables instead of pages

How does the MMU perform page walks?

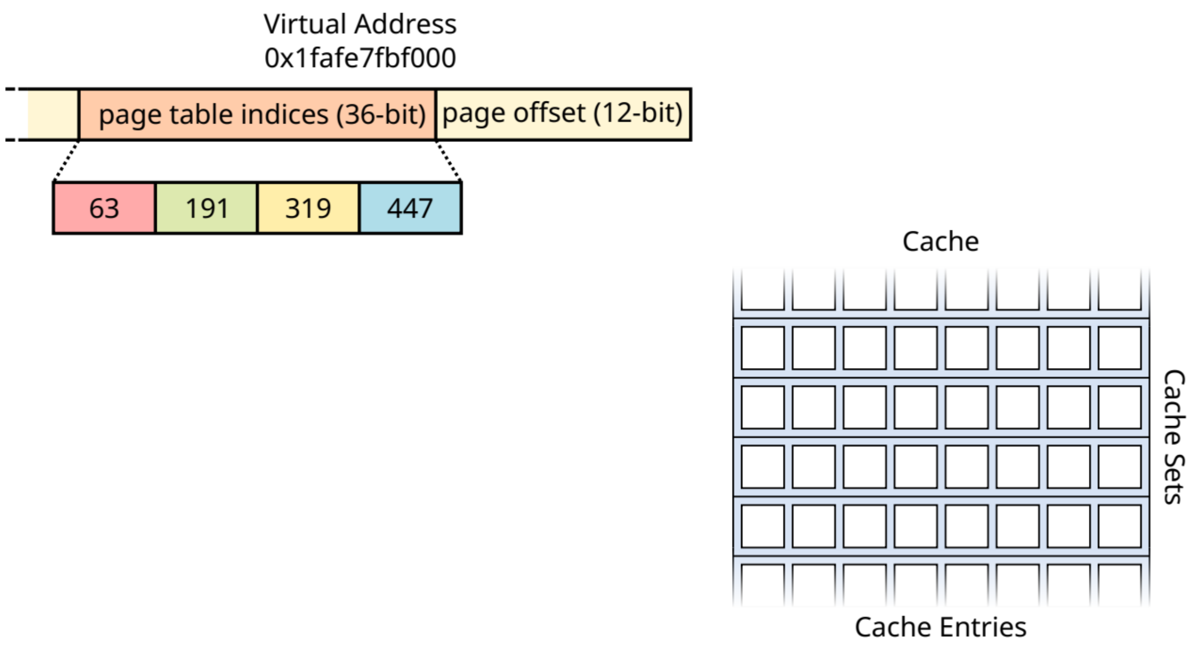
Page table walks



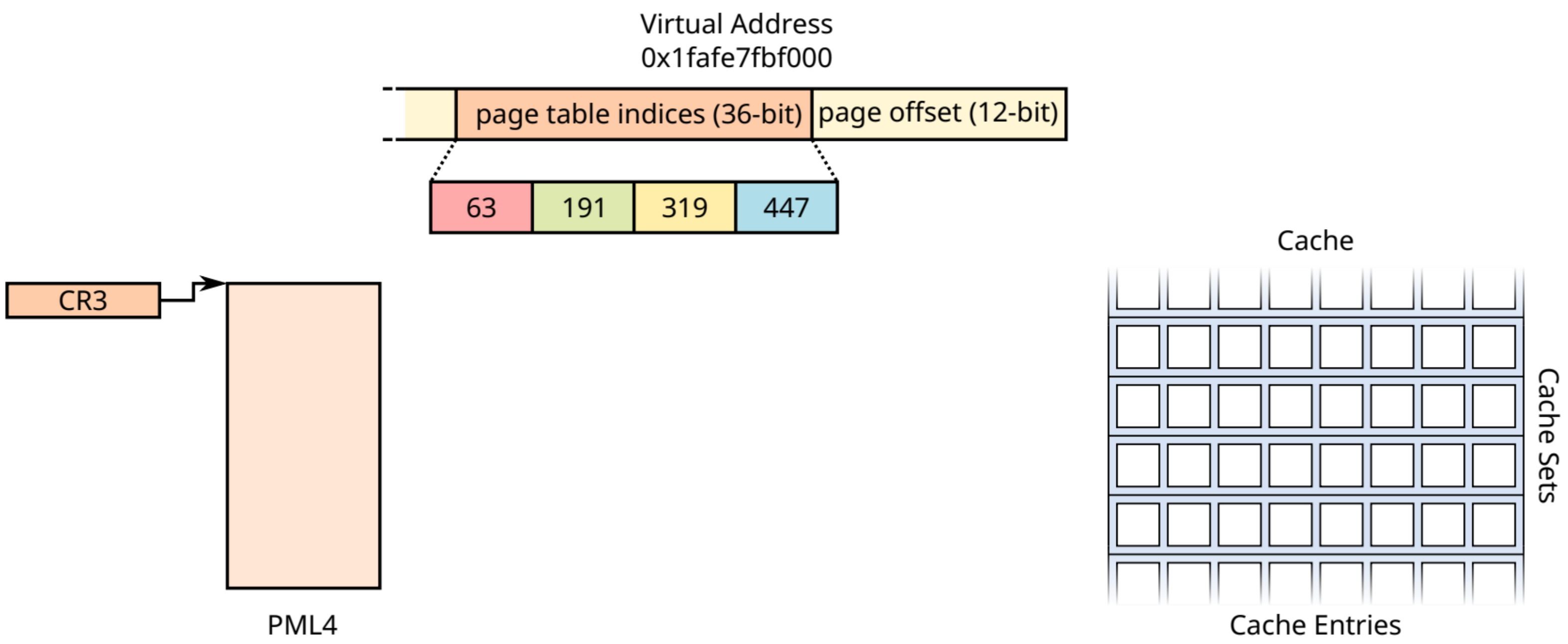
Page table walks



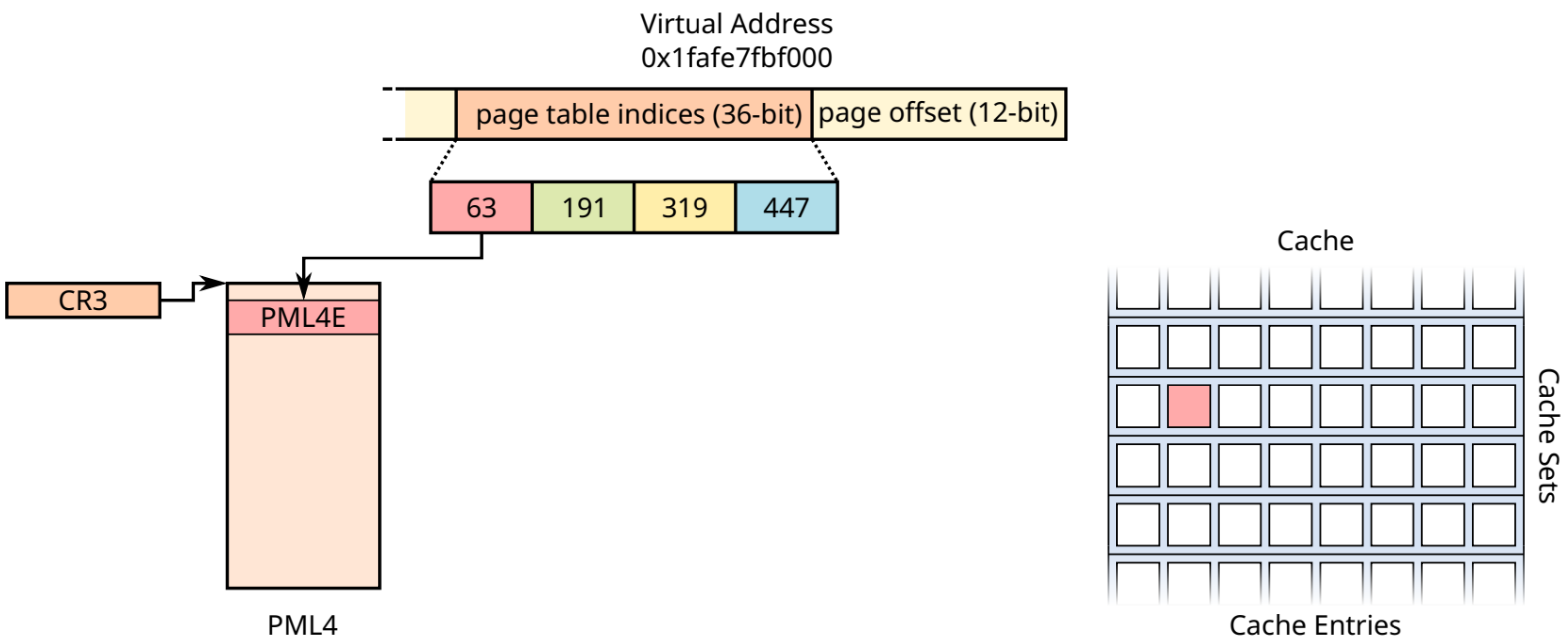
Page table walks



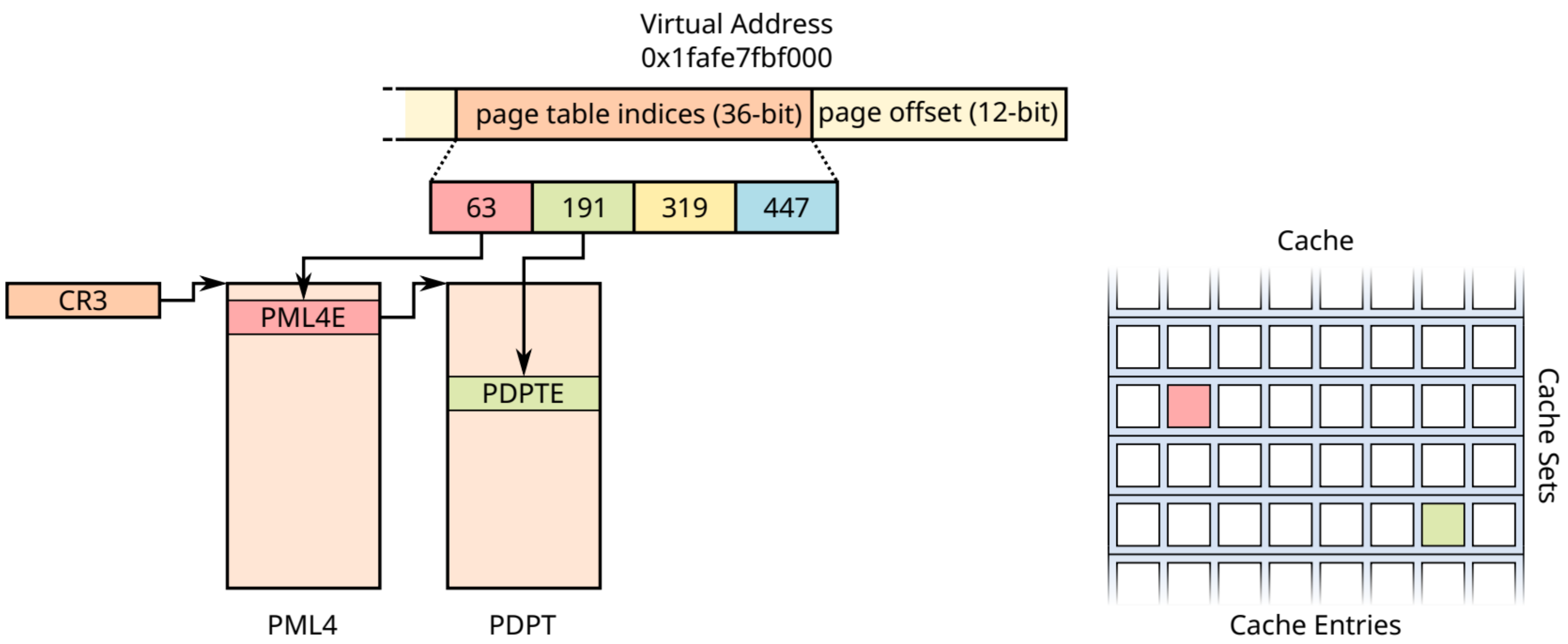
Page table walks



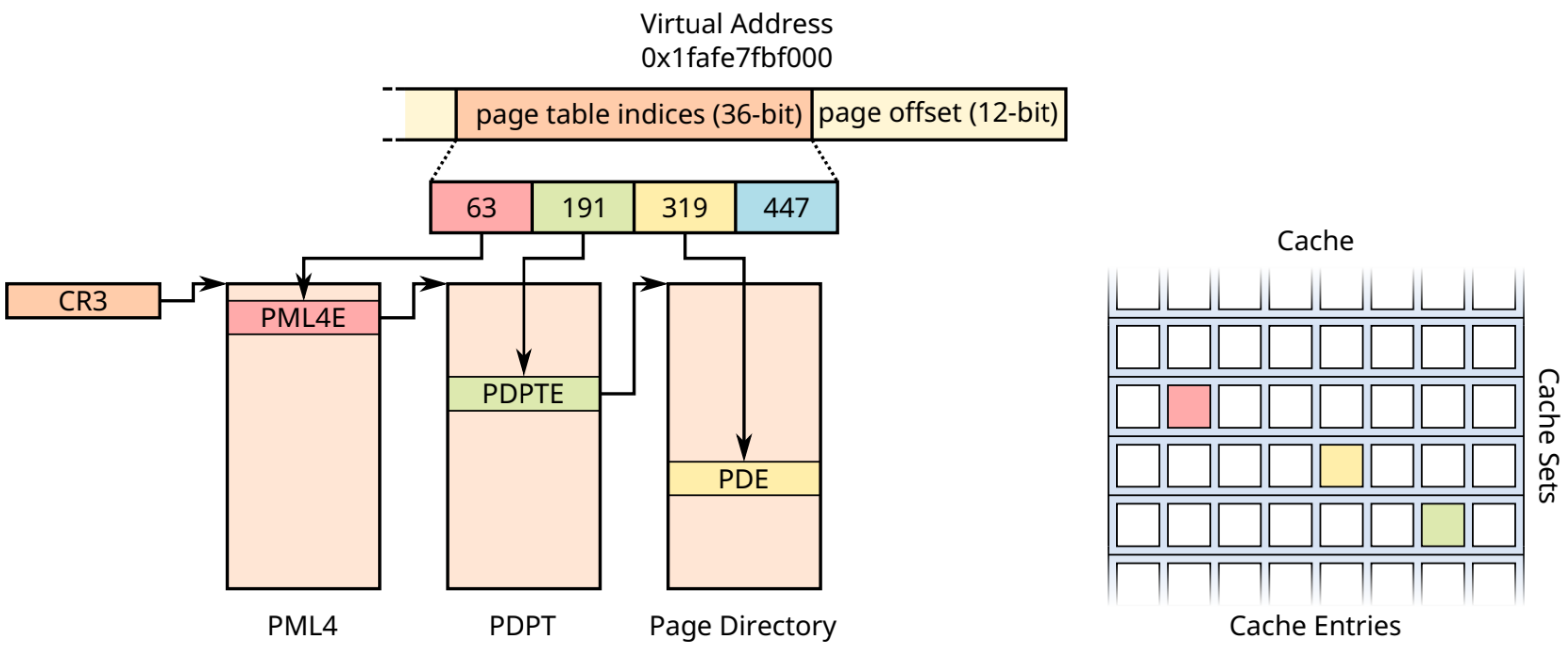
Page table walks



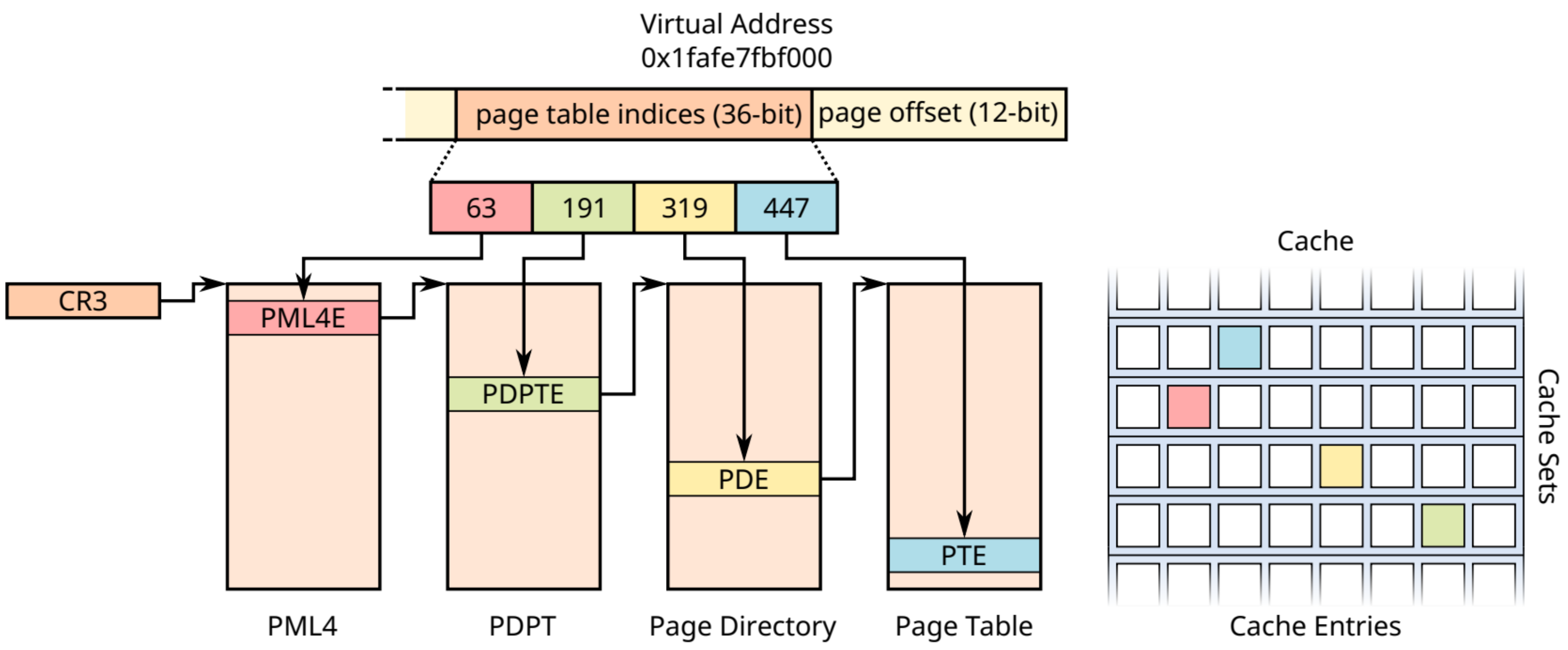
Page table walks



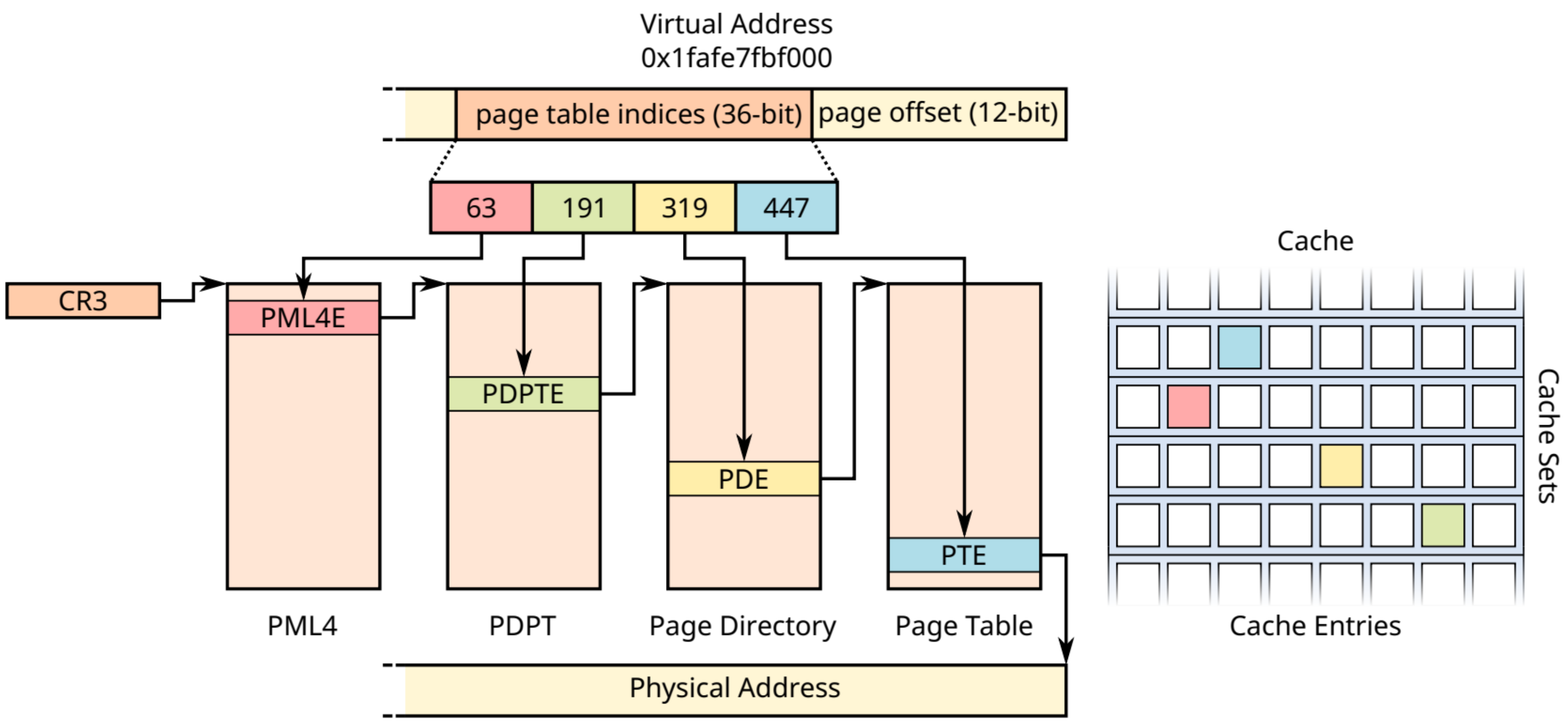
Page table walks



Page table walks



Page table walks

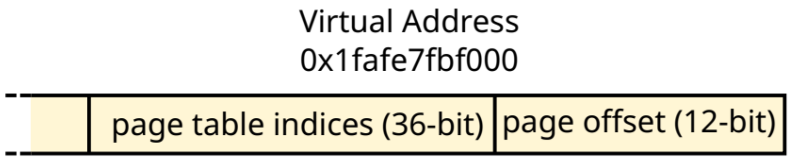


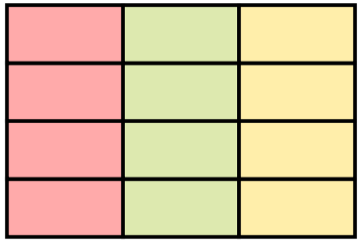
Can we do a XLATE+ PROBE?

Challenges

- Avoid noise from high-level page tables
- Avoid noise from pages
- Build eviction sets

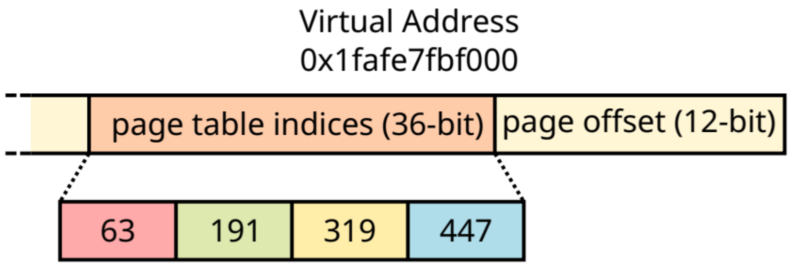
Translation Caches

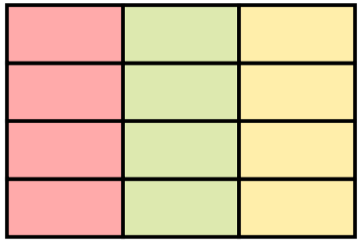




Translation Cache

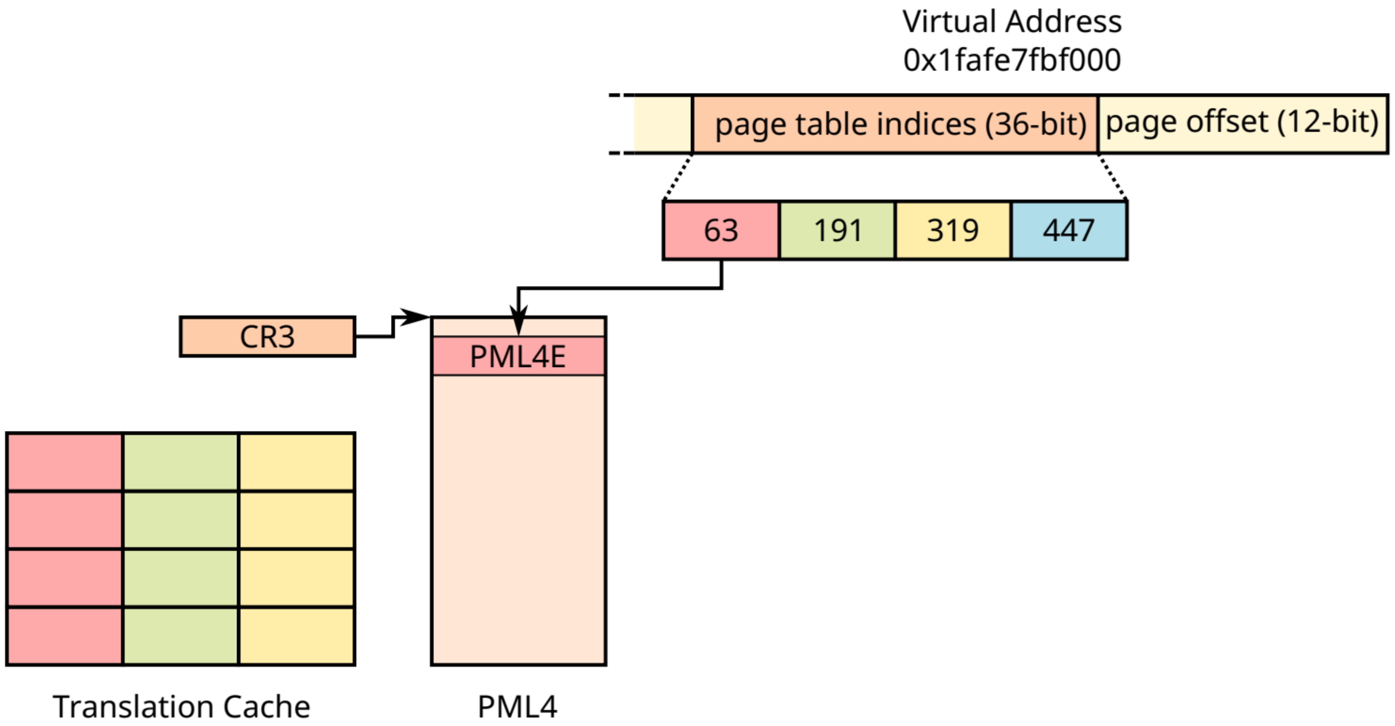
Translation Caches



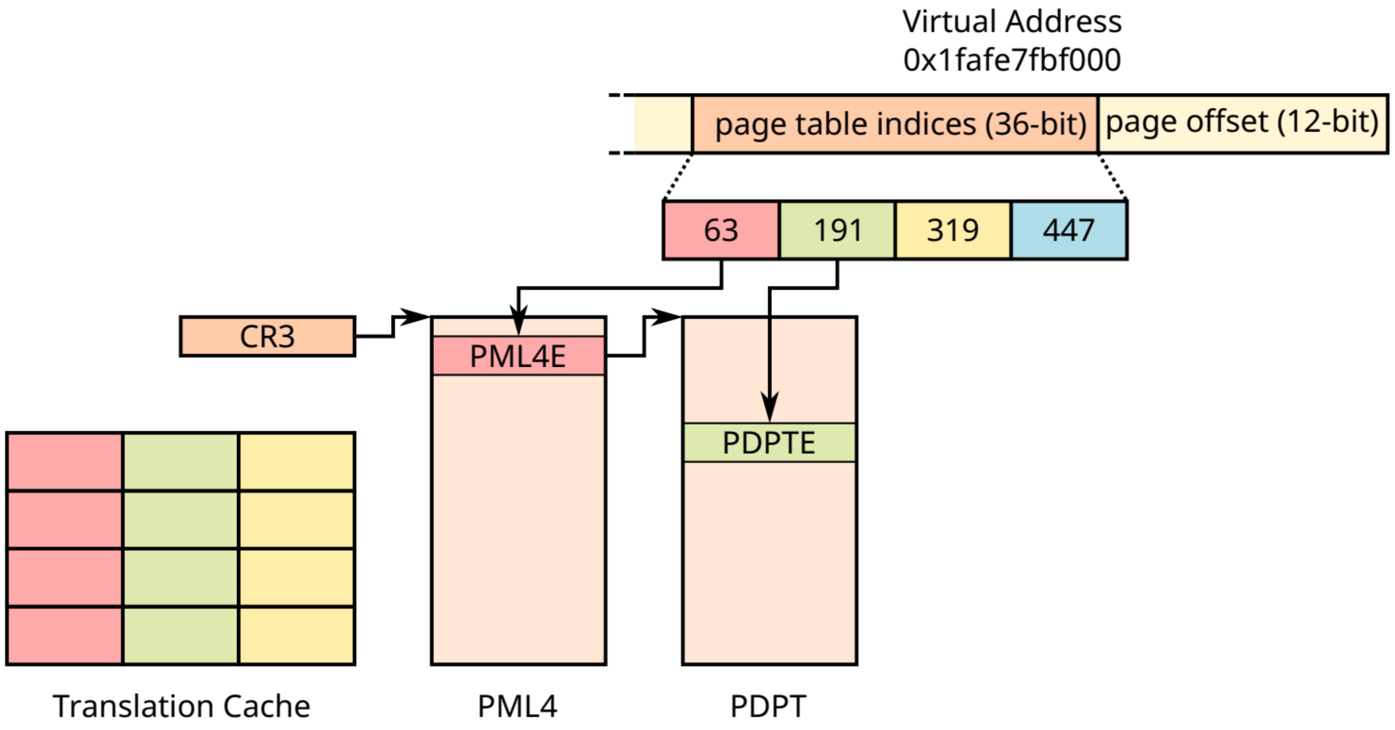


Translation Cache

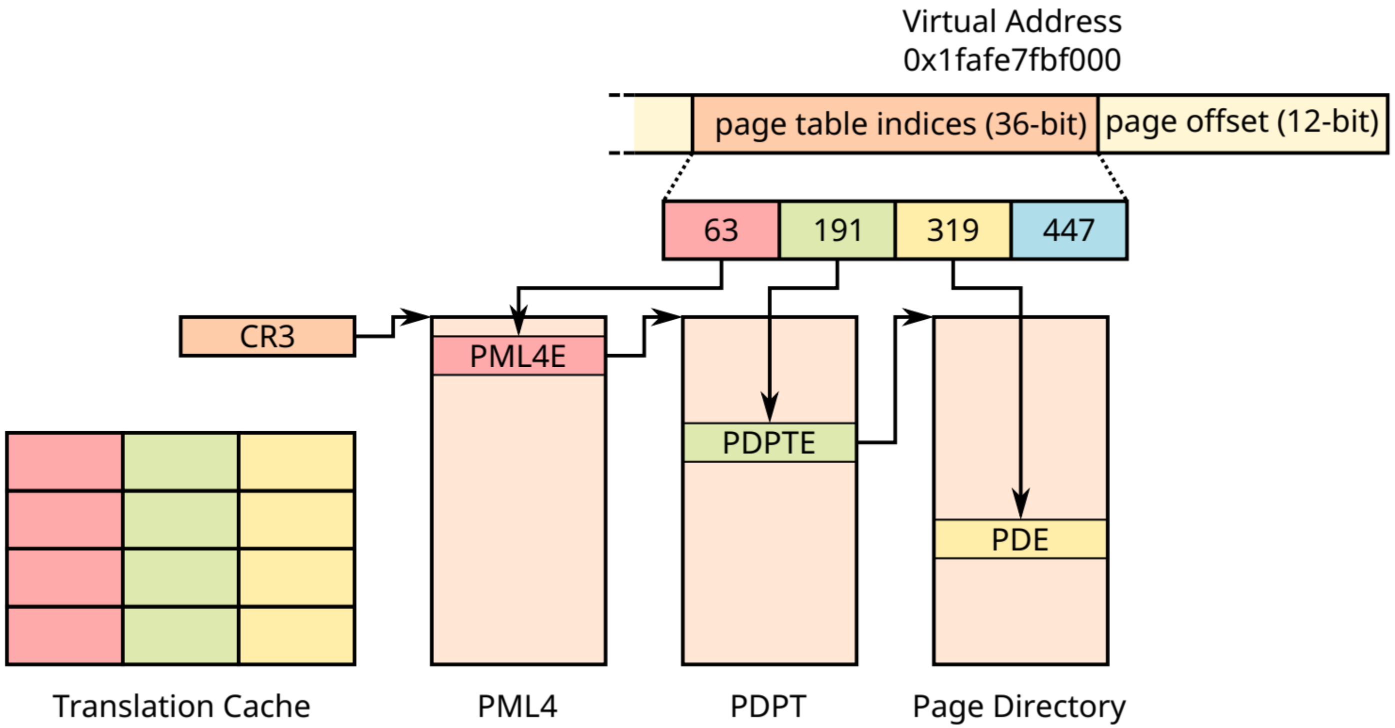
Translation Caches



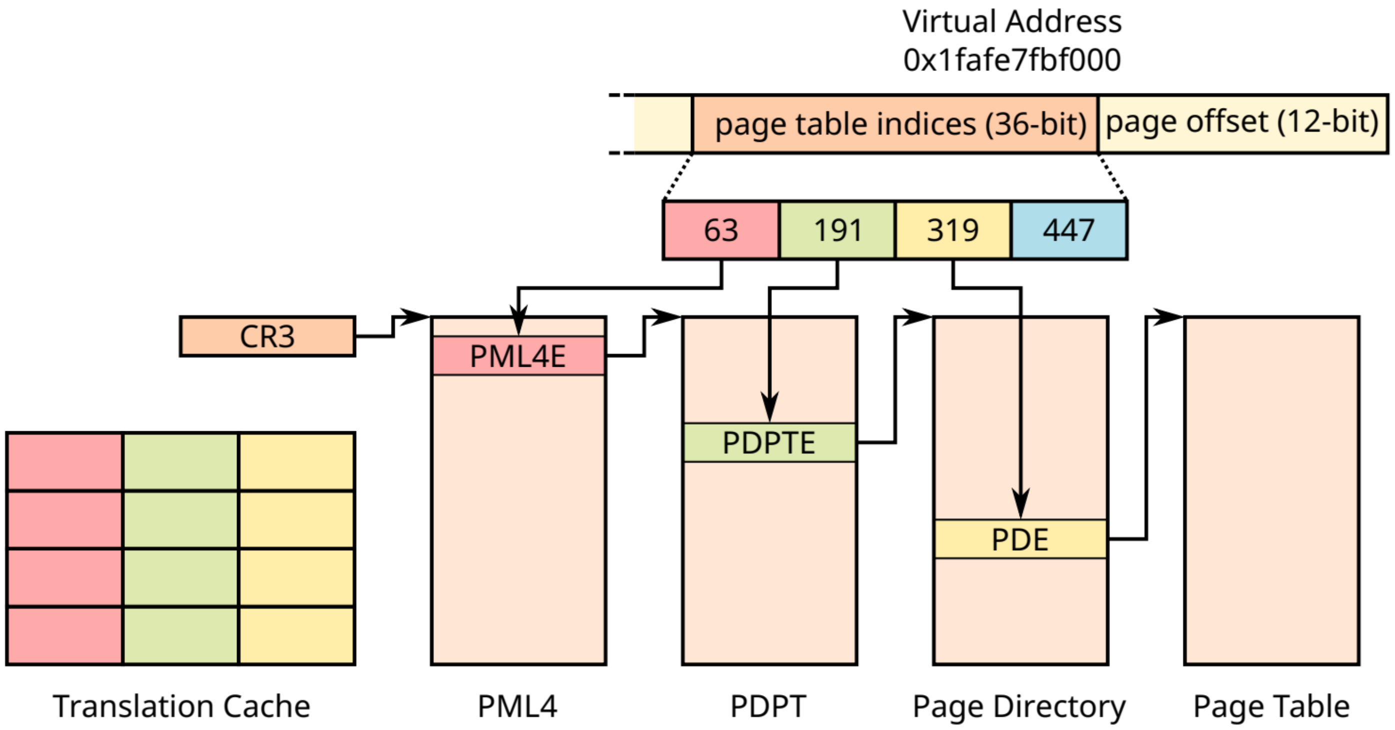
Translation Caches



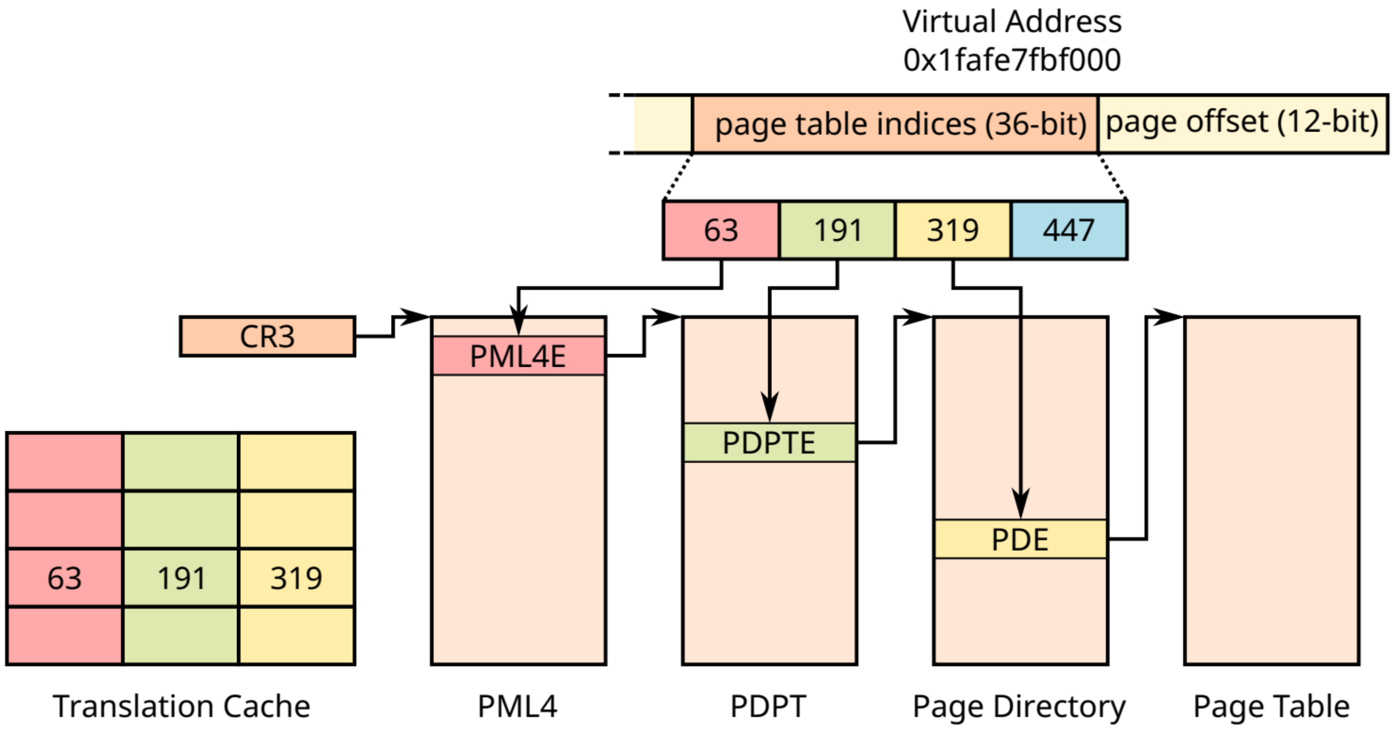
Translation Caches



Translation Caches

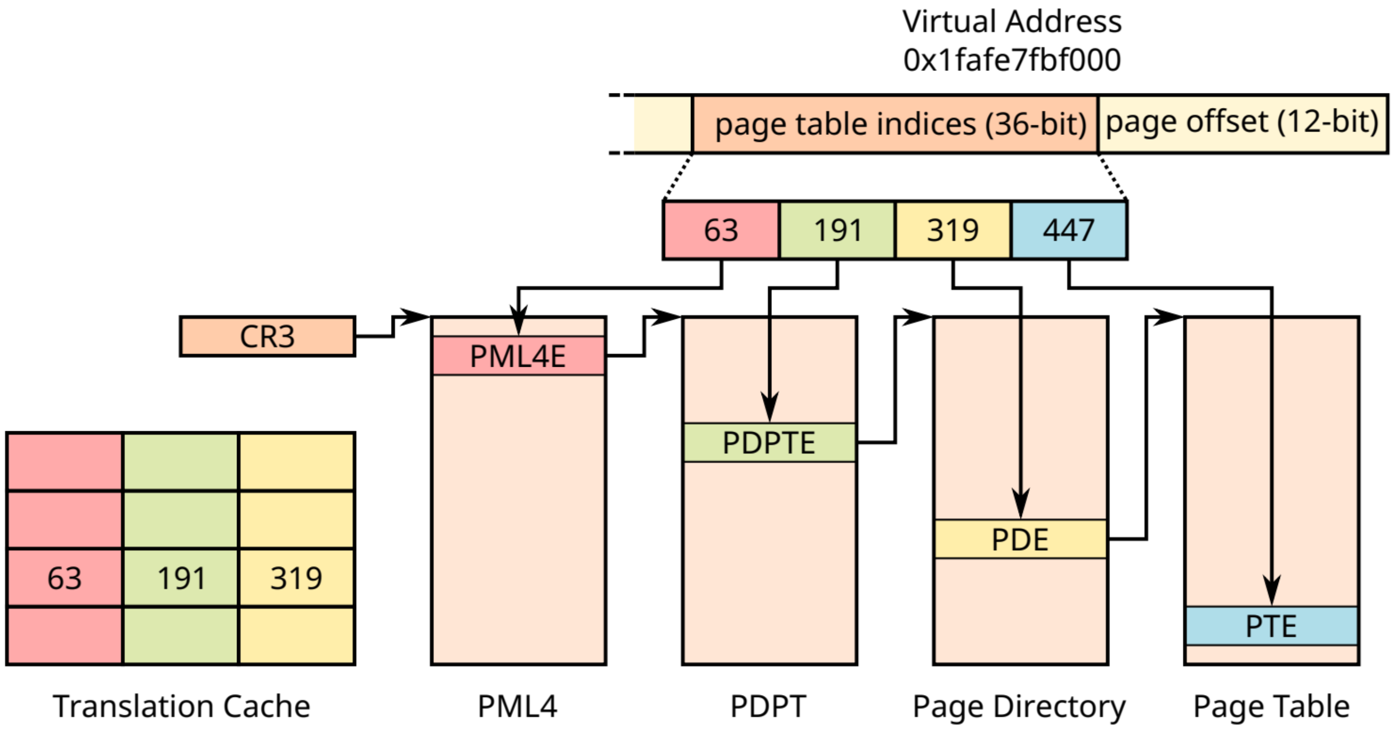


Translation Caches

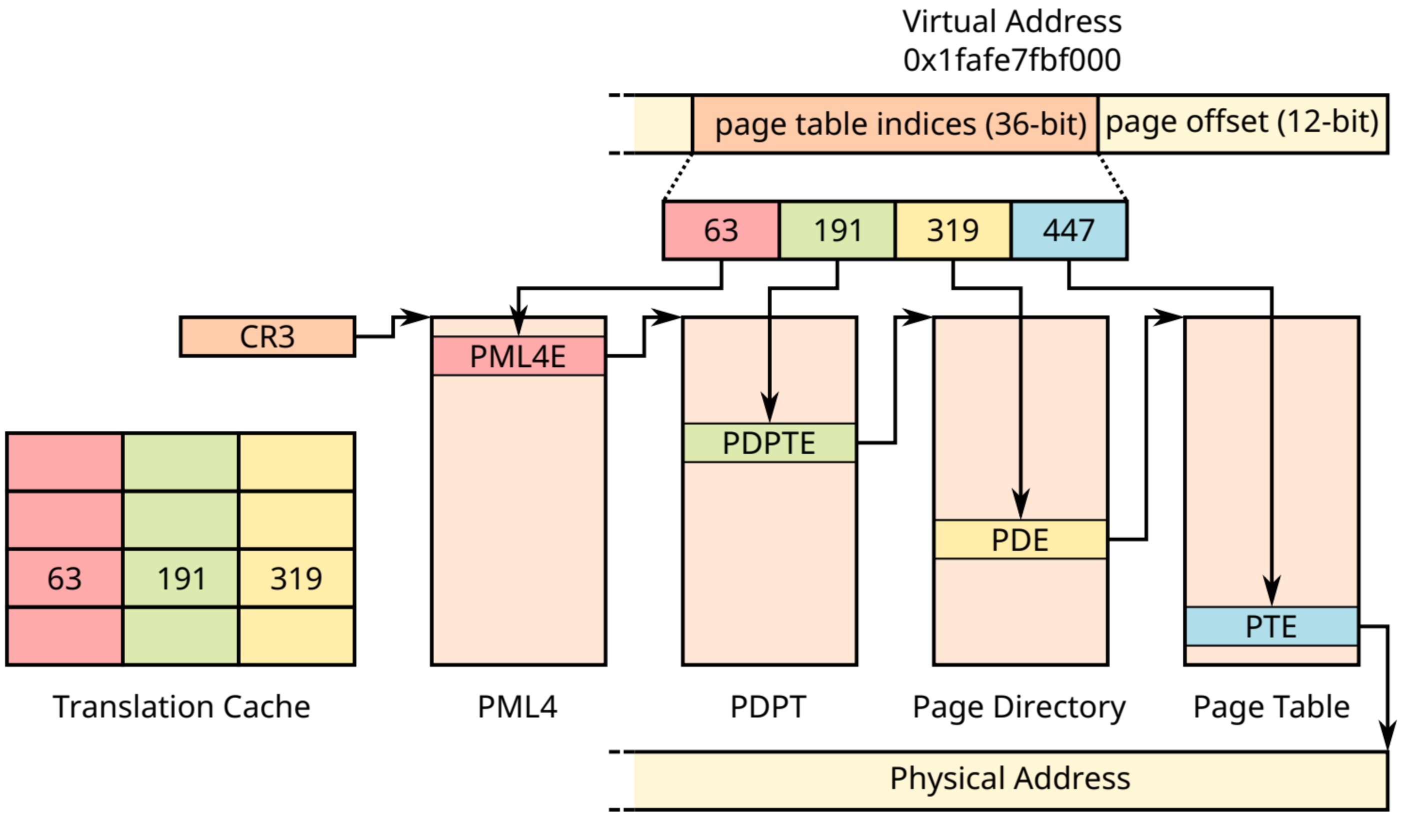


Translation caches cache intermediate page tables

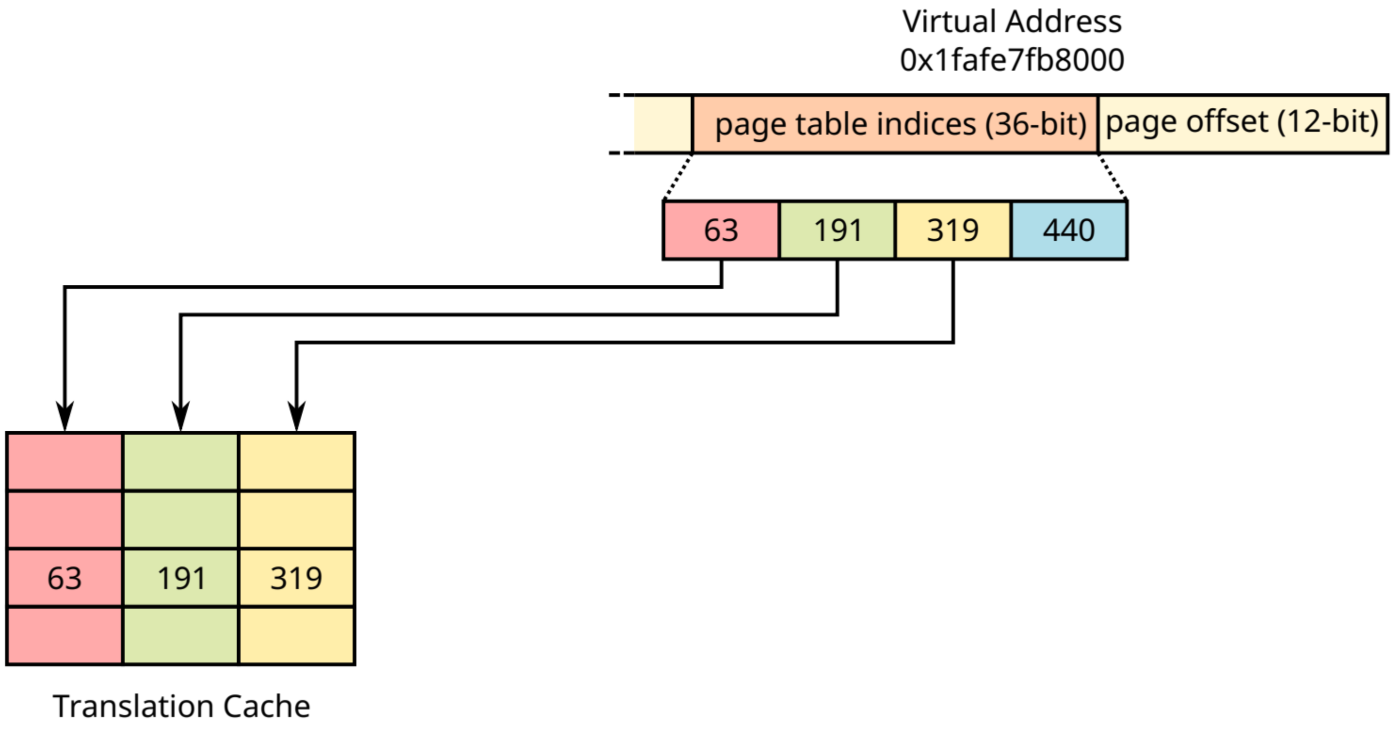
Translation Caches



Translation Caches

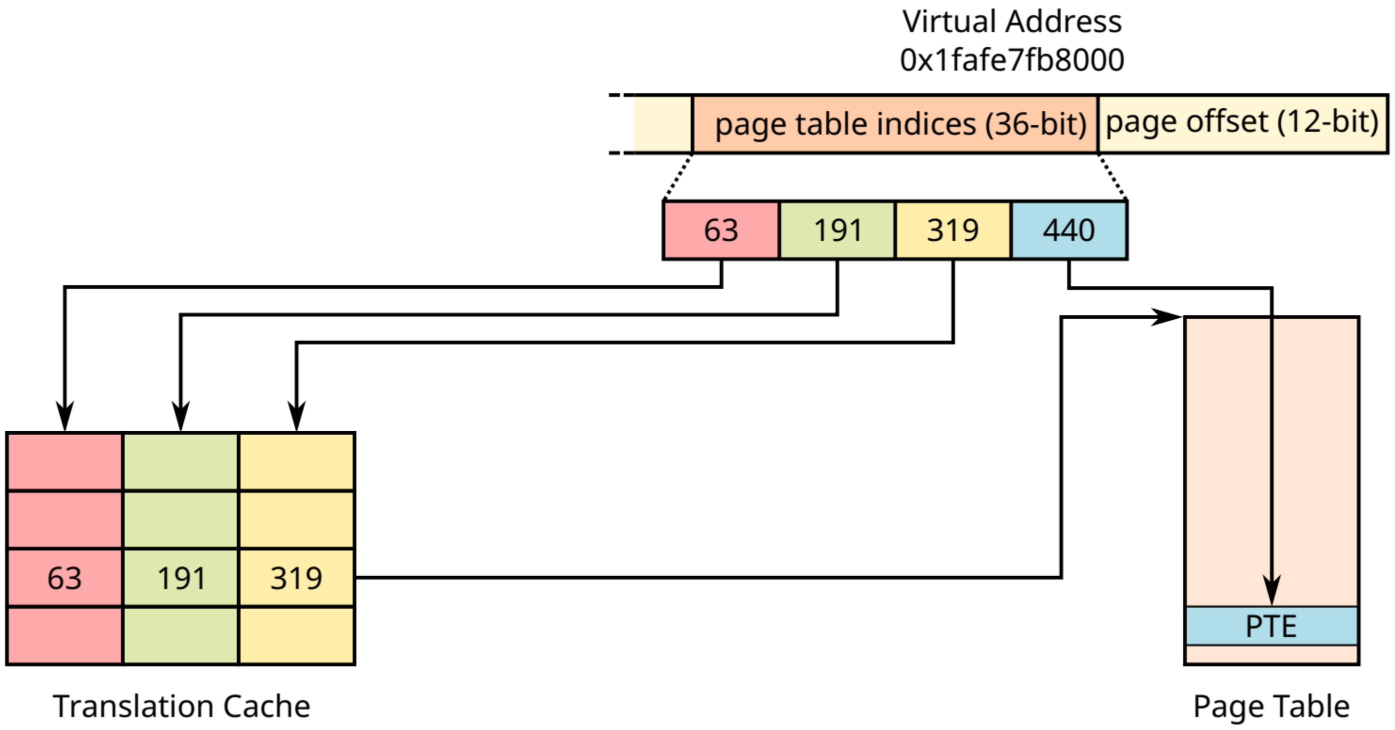


Translation Caches

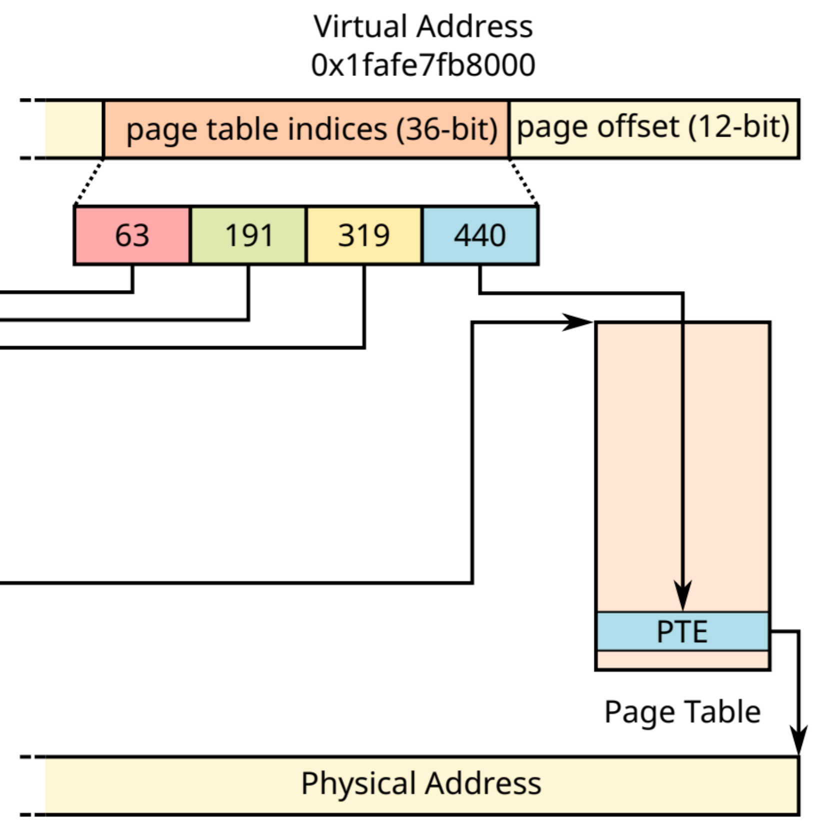


Translation caches cache intermediate page tables

Translation Caches



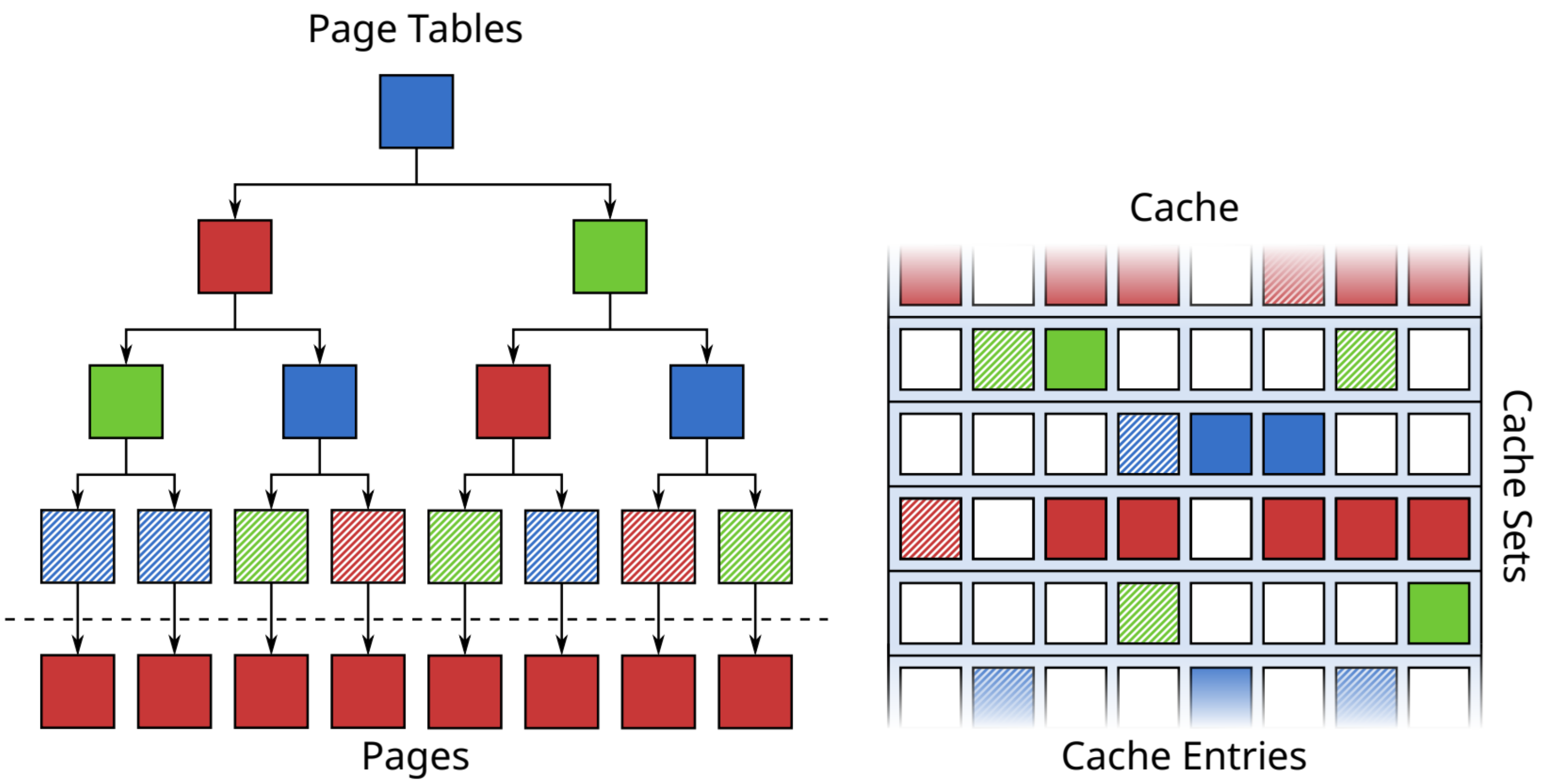
Translation Caches



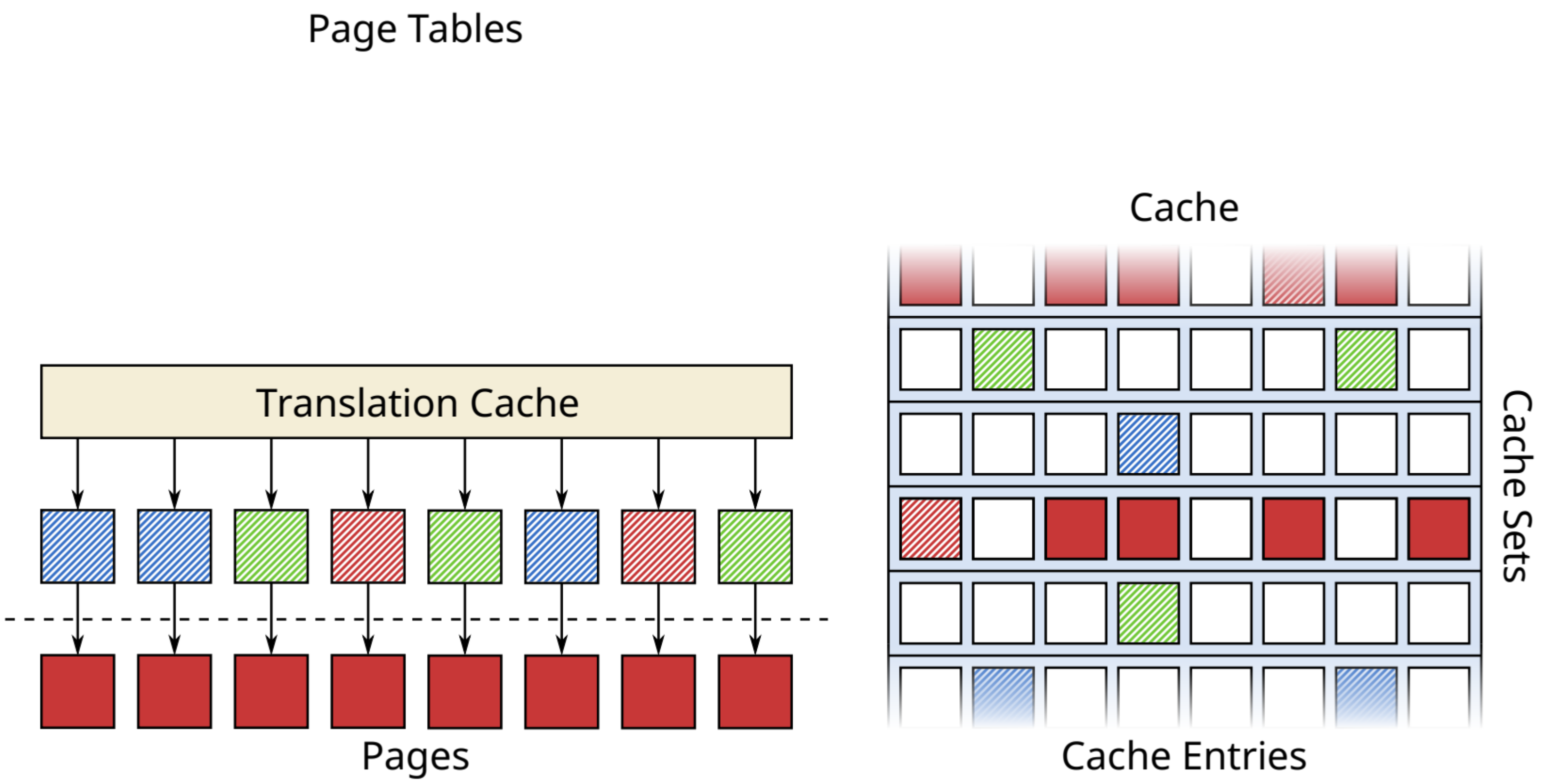
We reverse engineered size of these tables

Ideal for reducing noise of PT walk

Translation Caches



Translation Caches

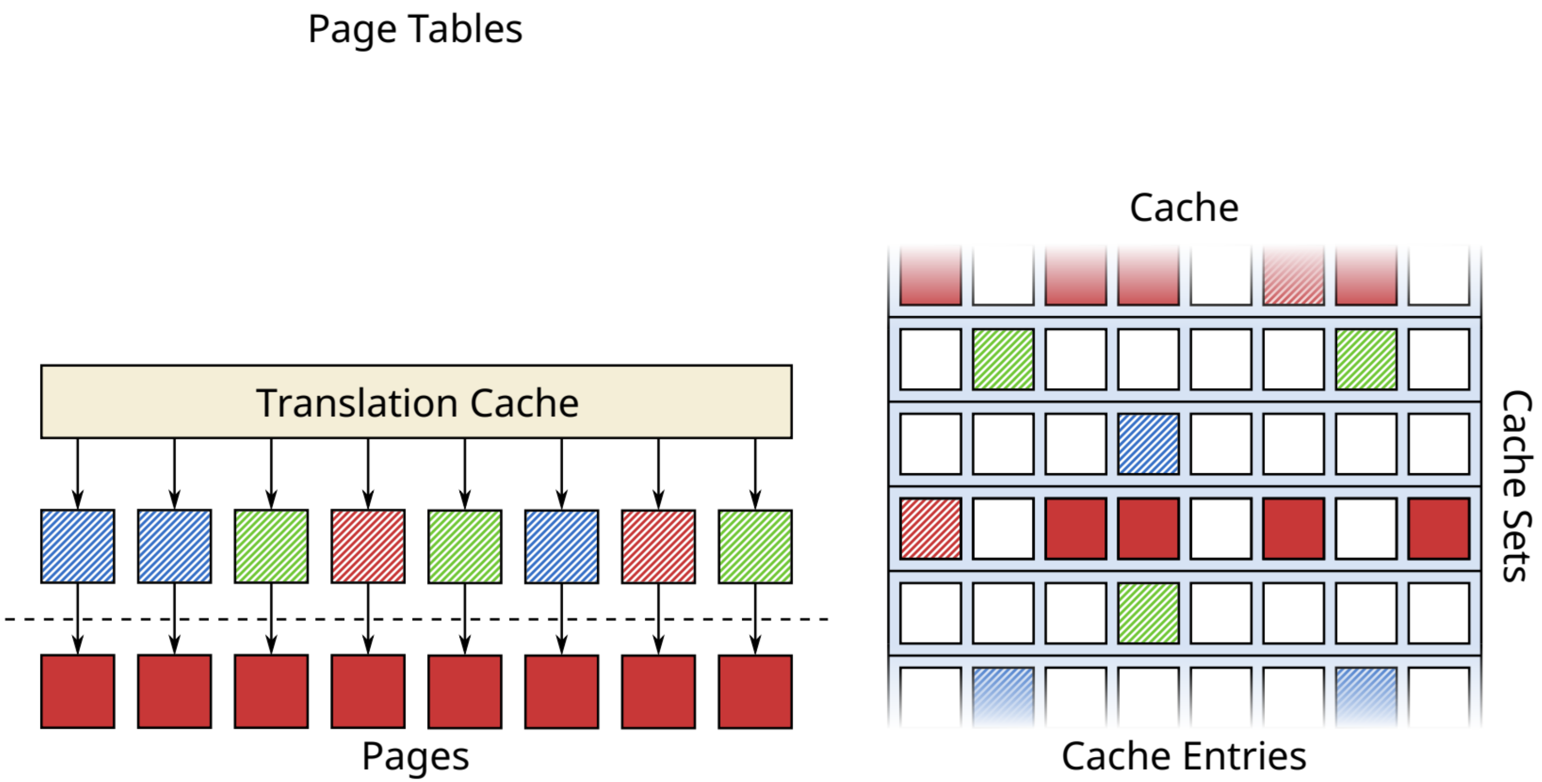


Translation caches skip page table walks

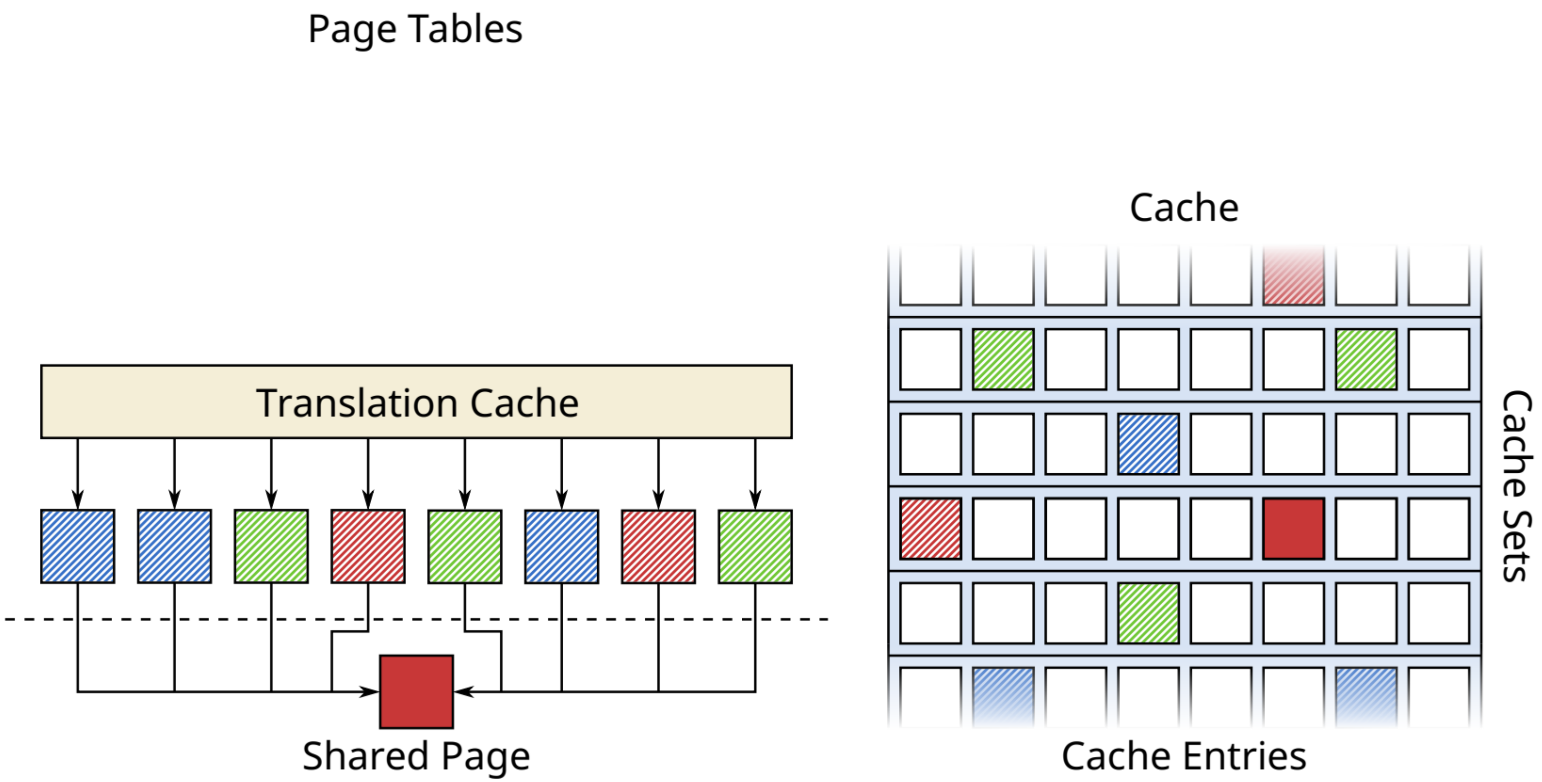
Challenges

- ✓ Avoid noise from high-level page tables
- Avoid noise from pages
- Build eviction sets

Shared Memory



Shared Memory

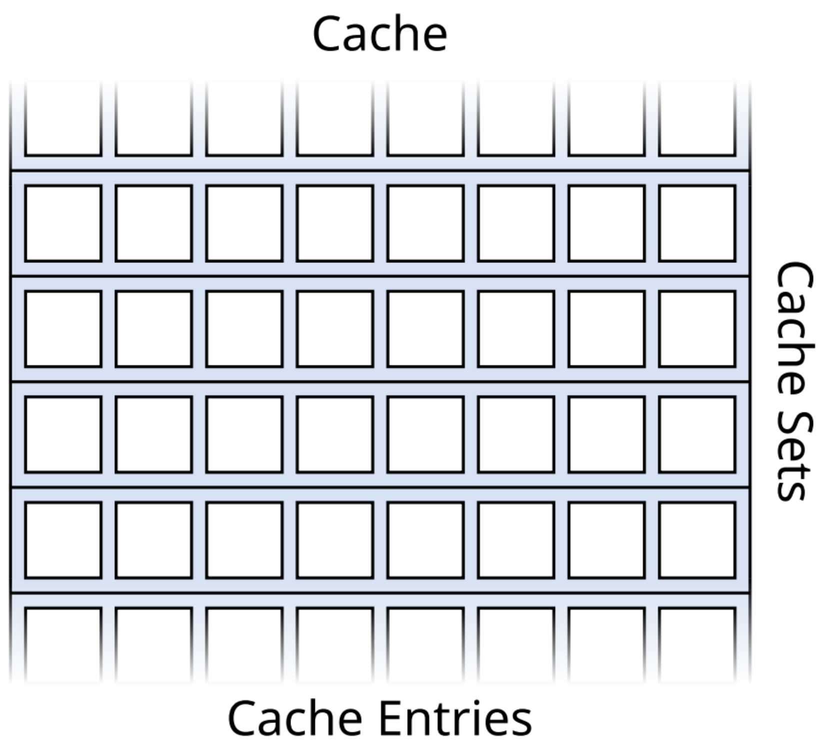


Use shared memory to reduce noise

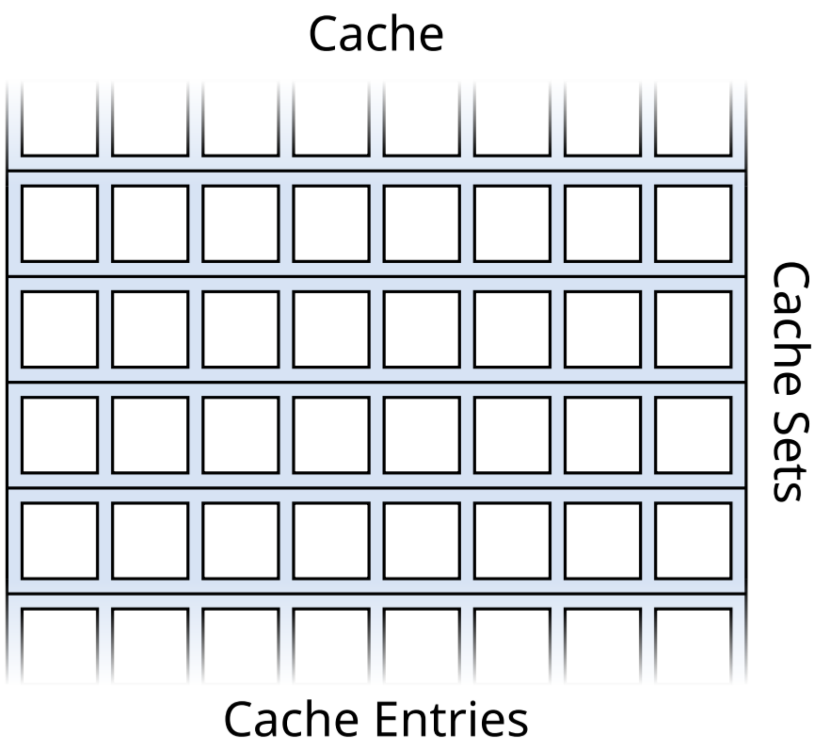
Challenges

- ✓ Avoid noise from high-level page tables
- ✓ Avoid noise from pages
- Build eviction sets

Building Eviction Sets

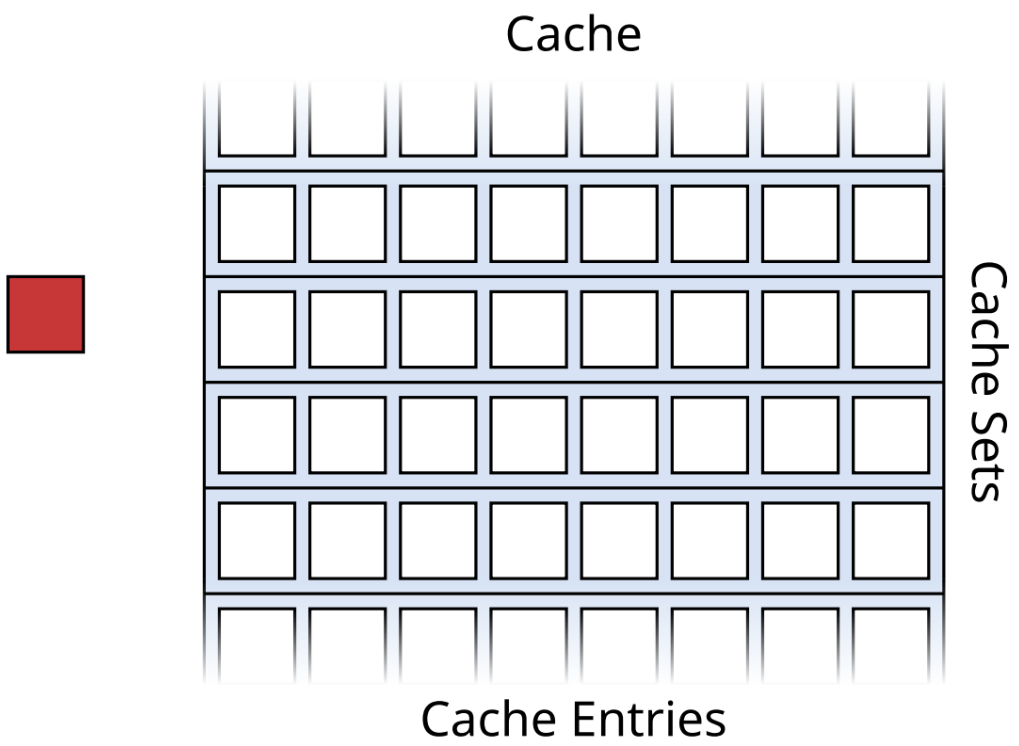


Building Eviction Sets



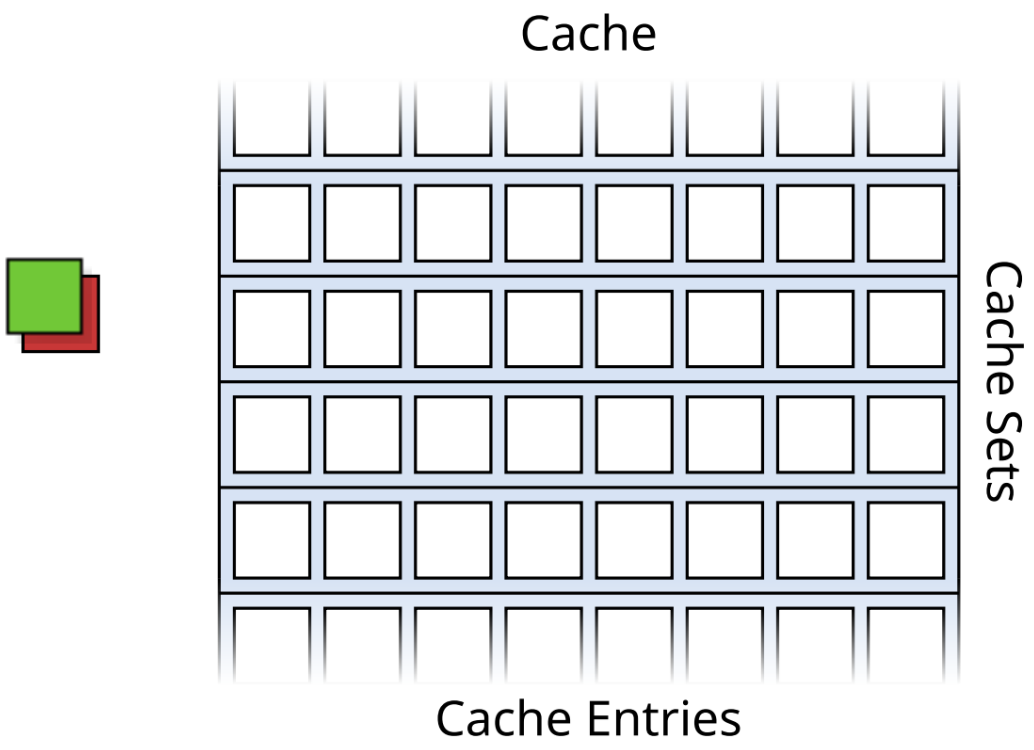
Allocate pages

Building Eviction Sets



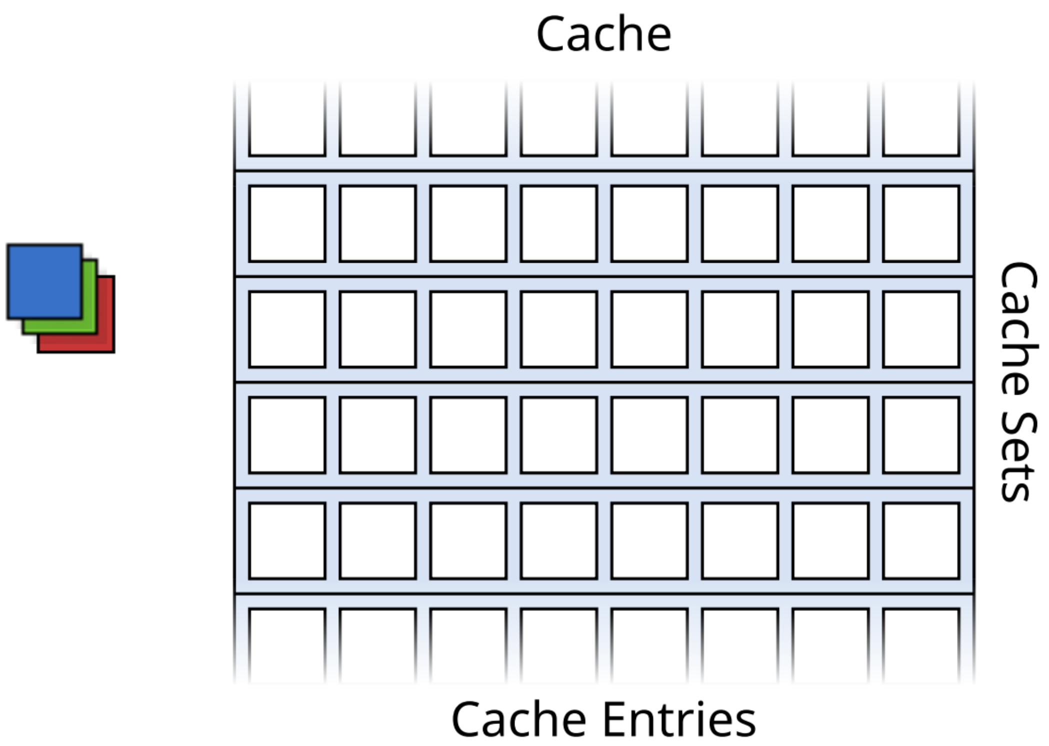
Allocate pages

Building Eviction Sets



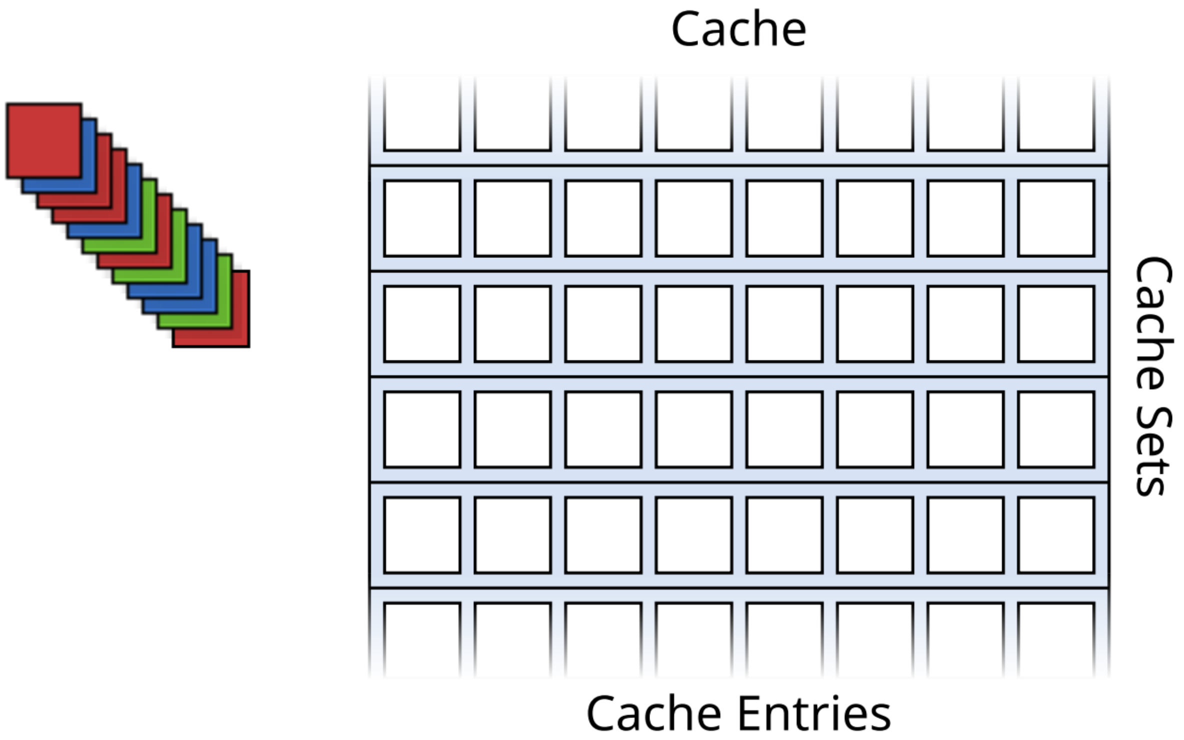
Allocate pages

Building Eviction Sets



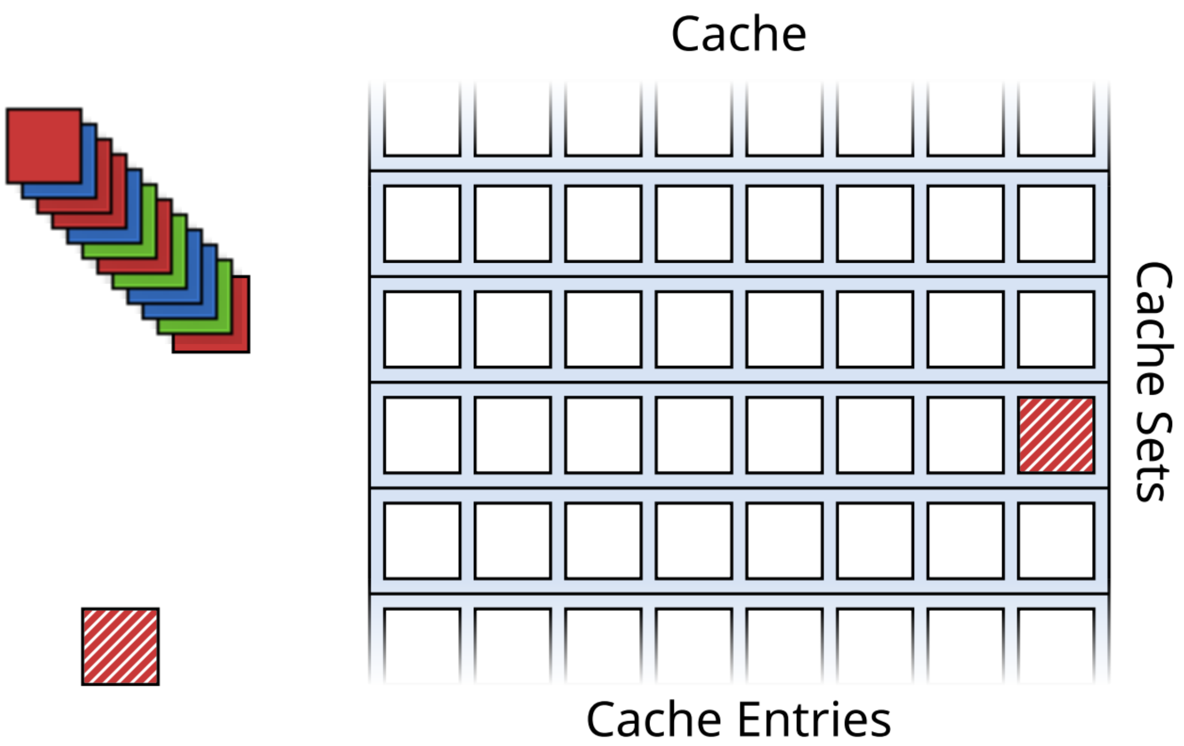
Allocate pages

Building Eviction Sets



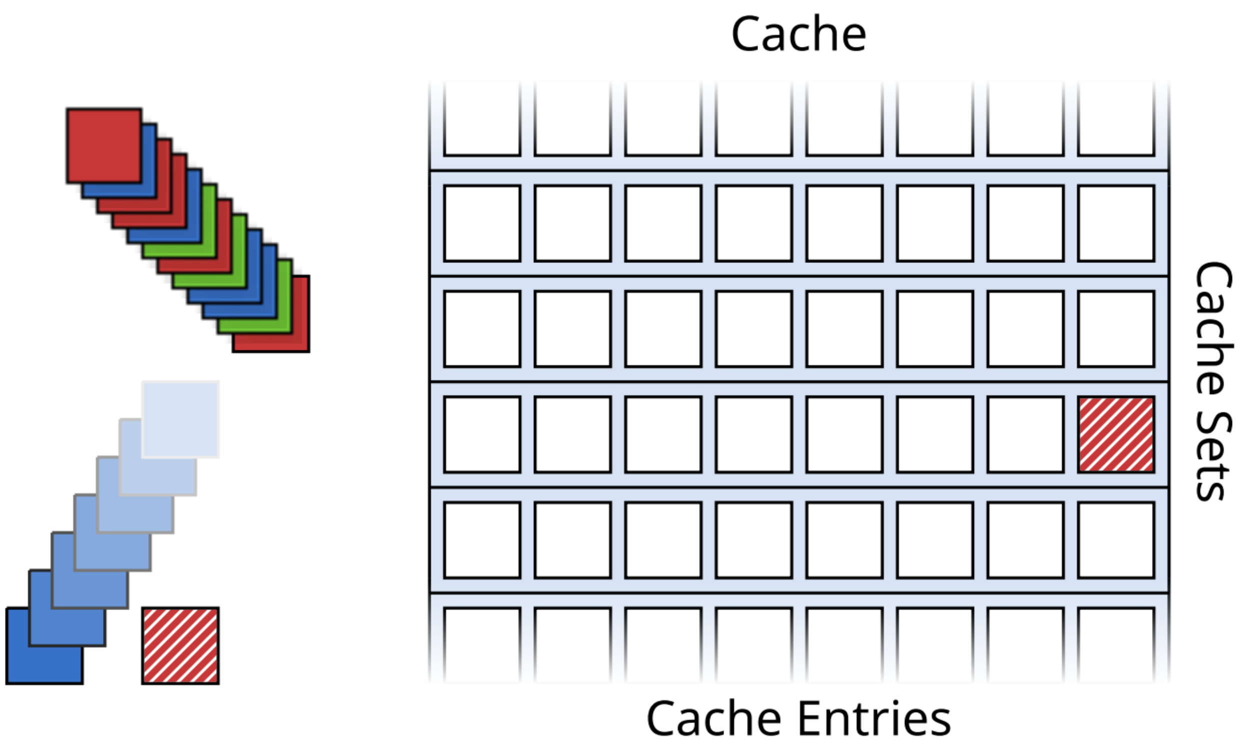
Allocate pages

Building Eviction Sets



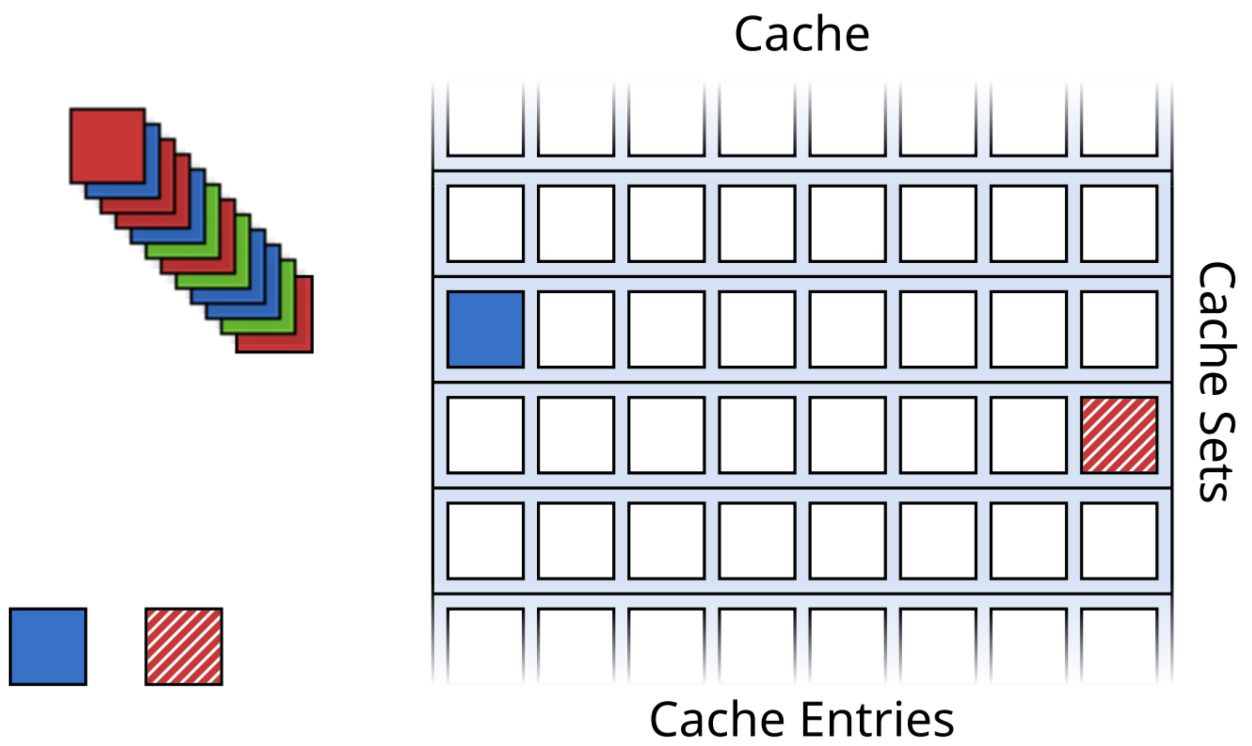
Load target into cache

Building Eviction Sets



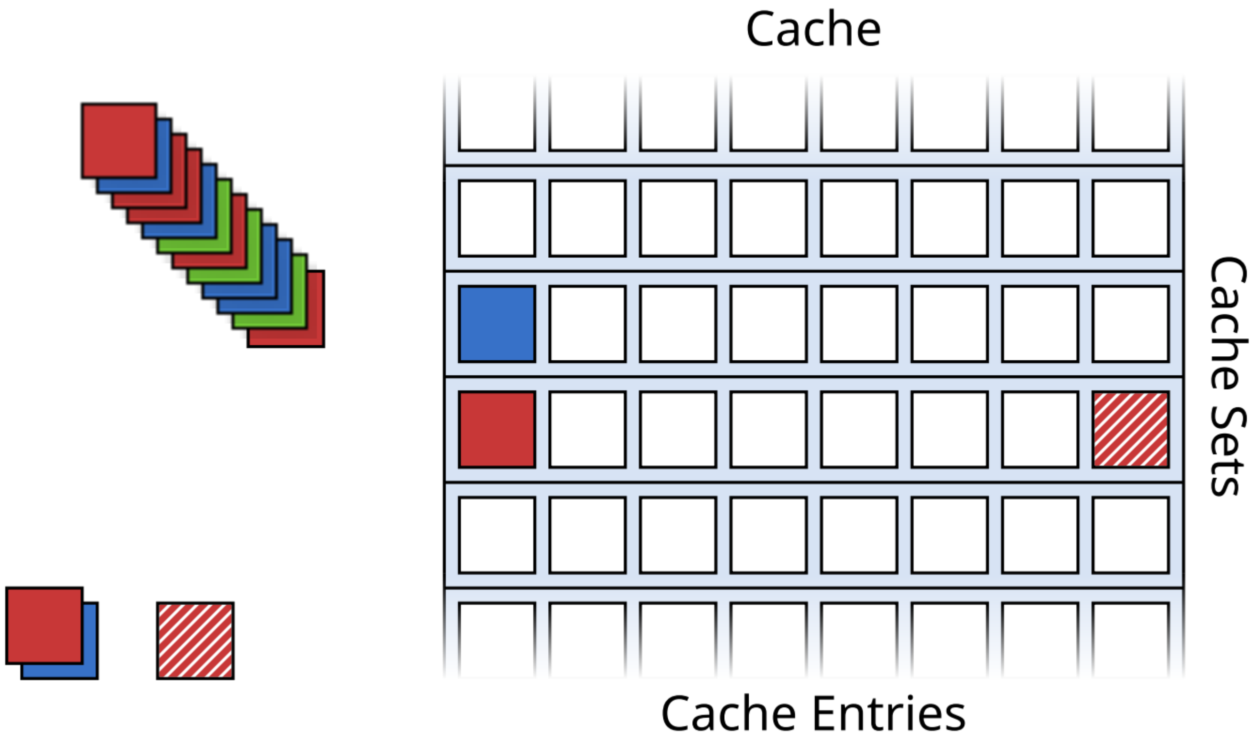
Draw pages and try to evict the target

Building Eviction Sets



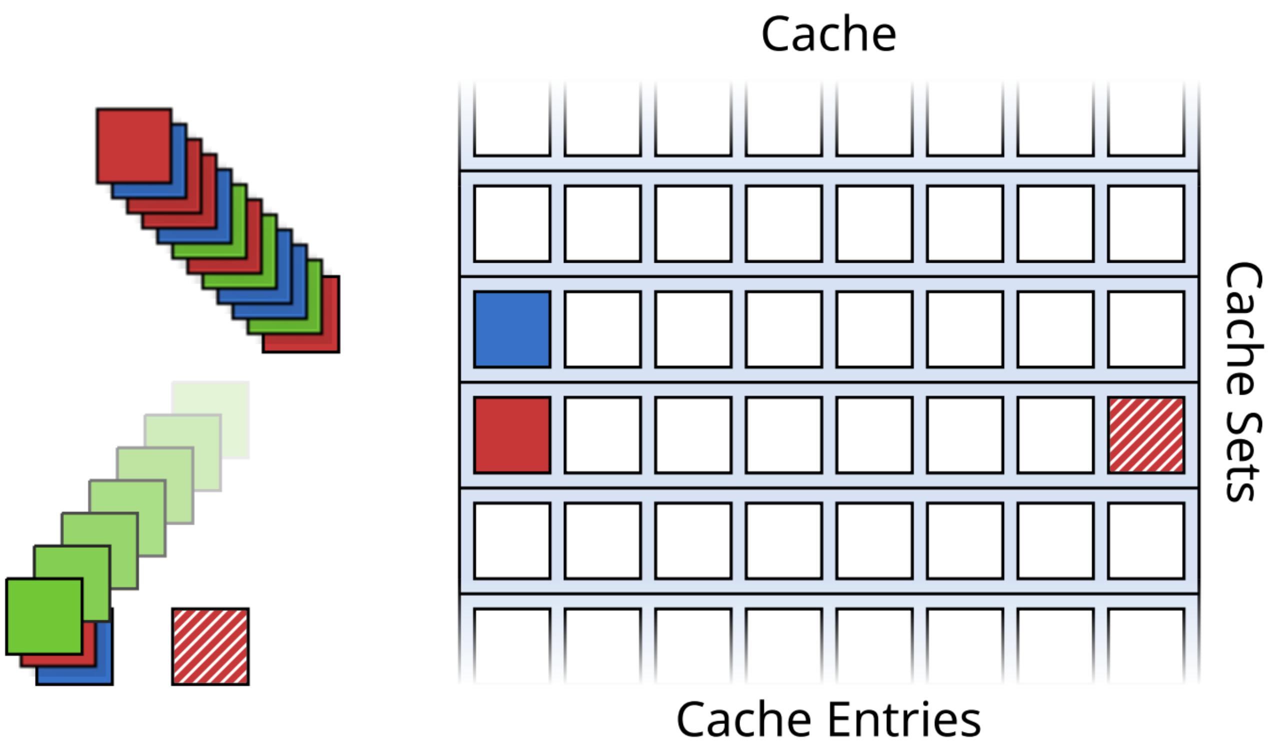
Draw pages and try to evict the target

Building Eviction Sets



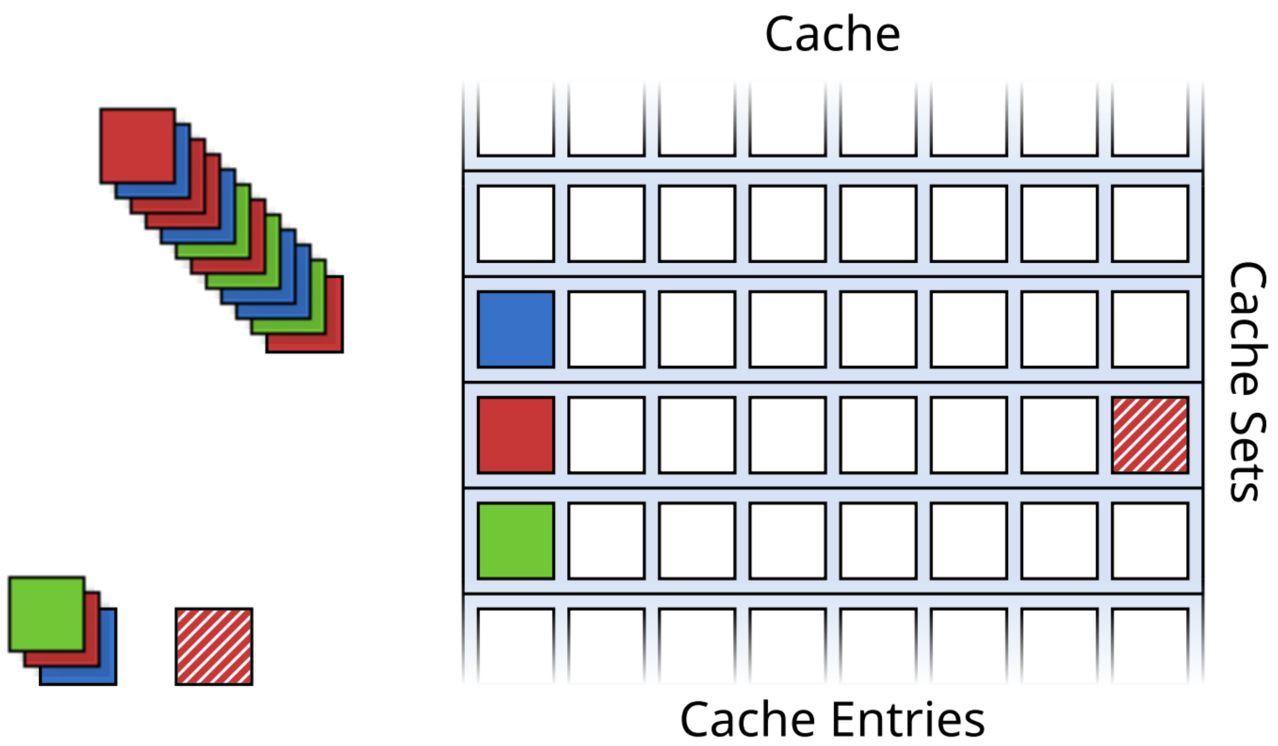
Draw pages and try to evict the target

Building Eviction Sets



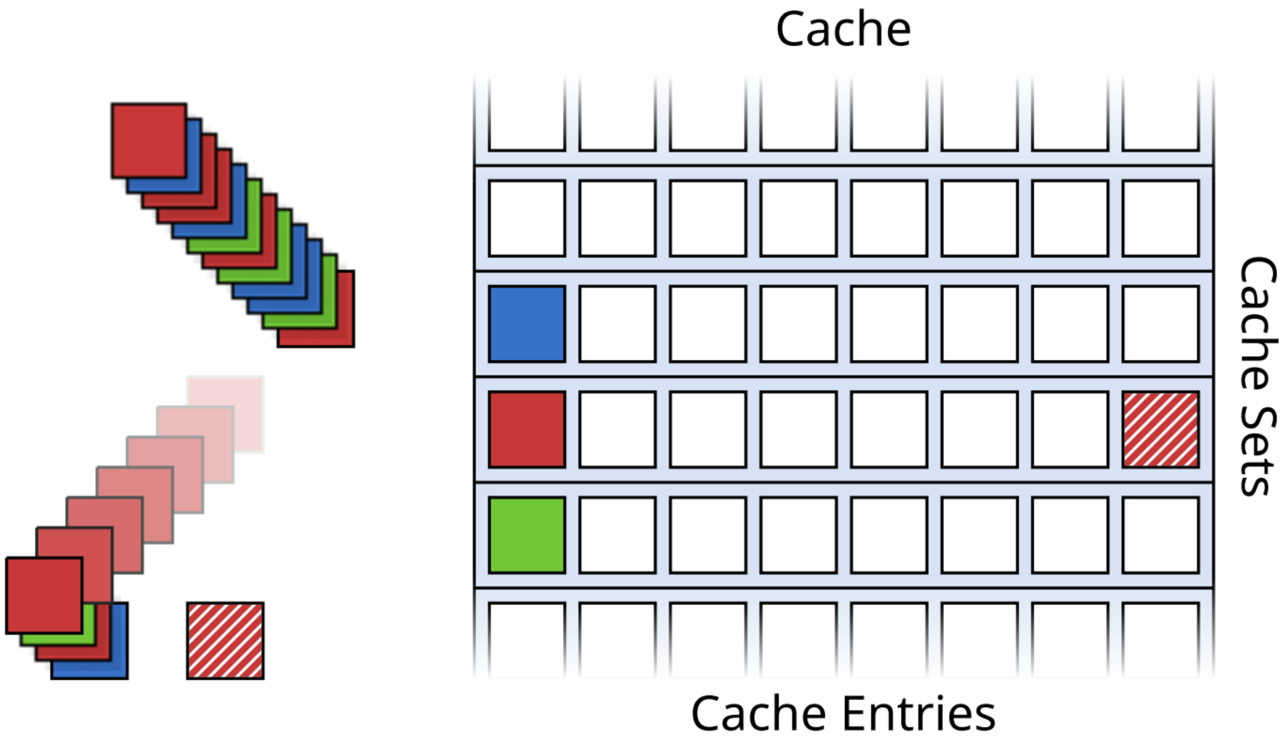
Draw pages and try to evict the target

Building Eviction Sets



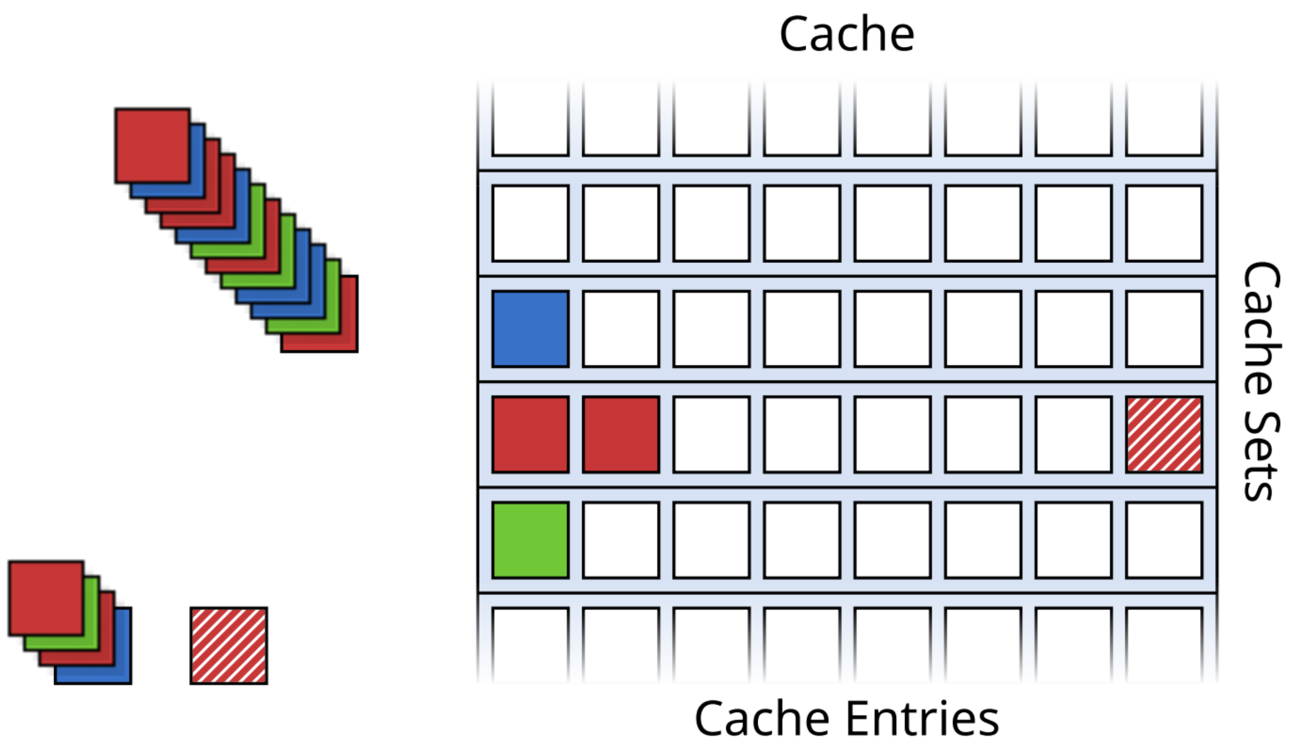
Draw pages and try to evict the target

Building Eviction Sets



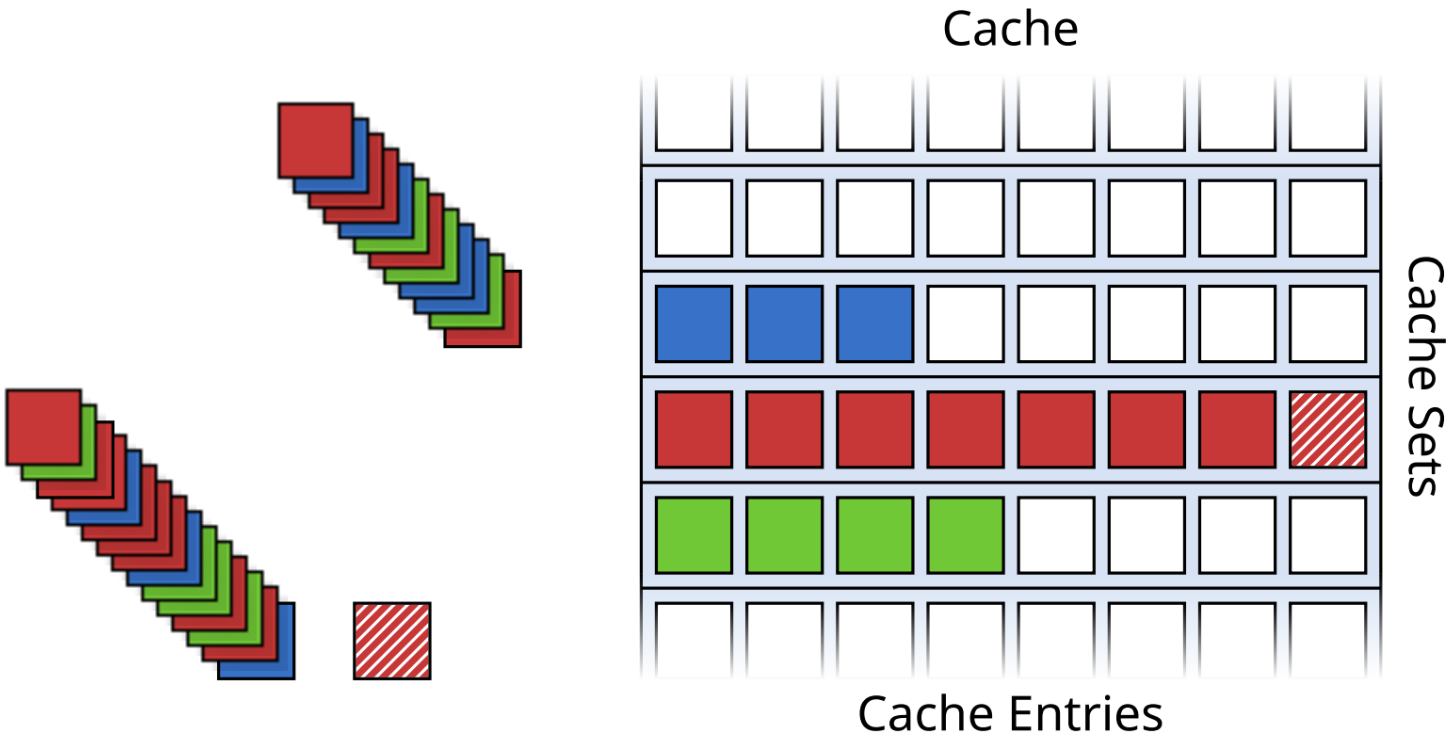
Draw pages and try to evict the target

Building Eviction Sets



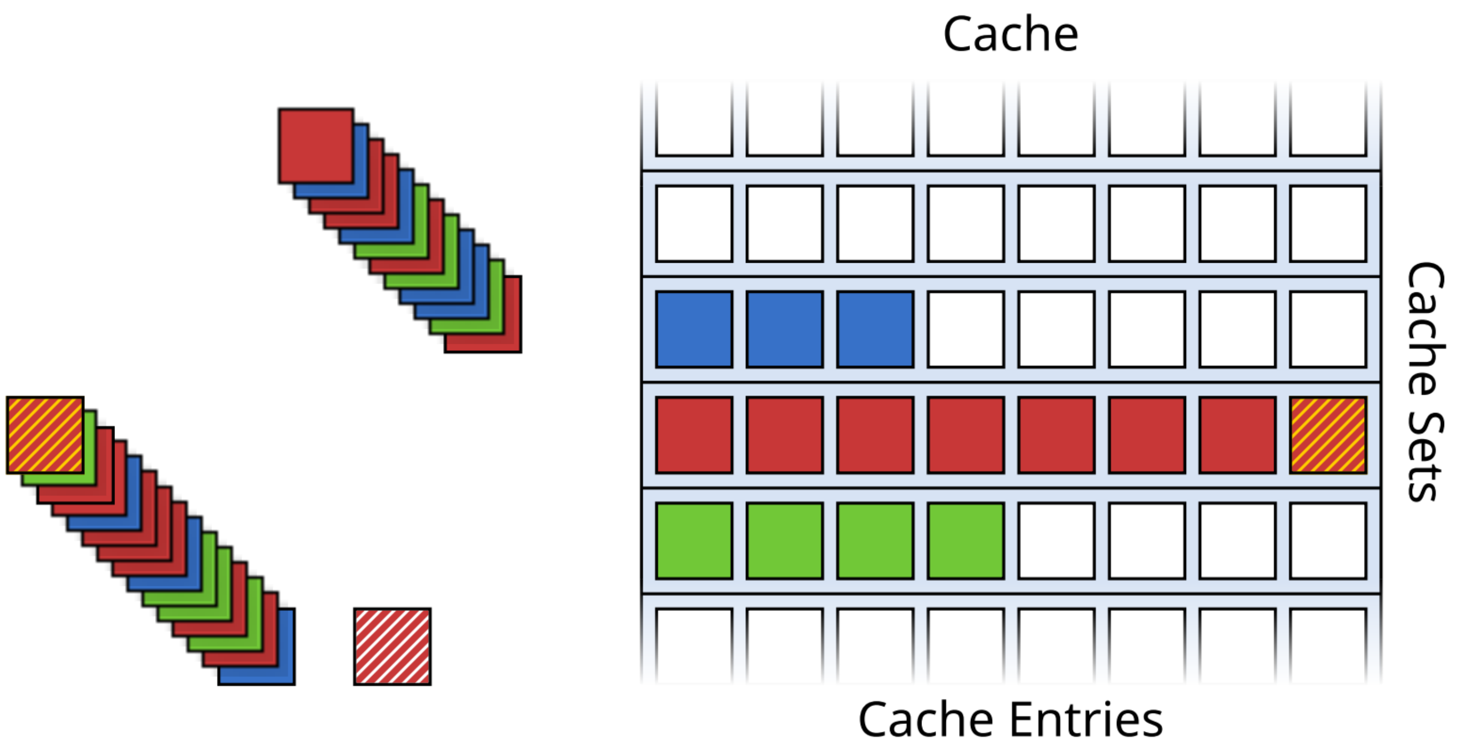
Draw pages and try to evict the target

Building Eviction Sets



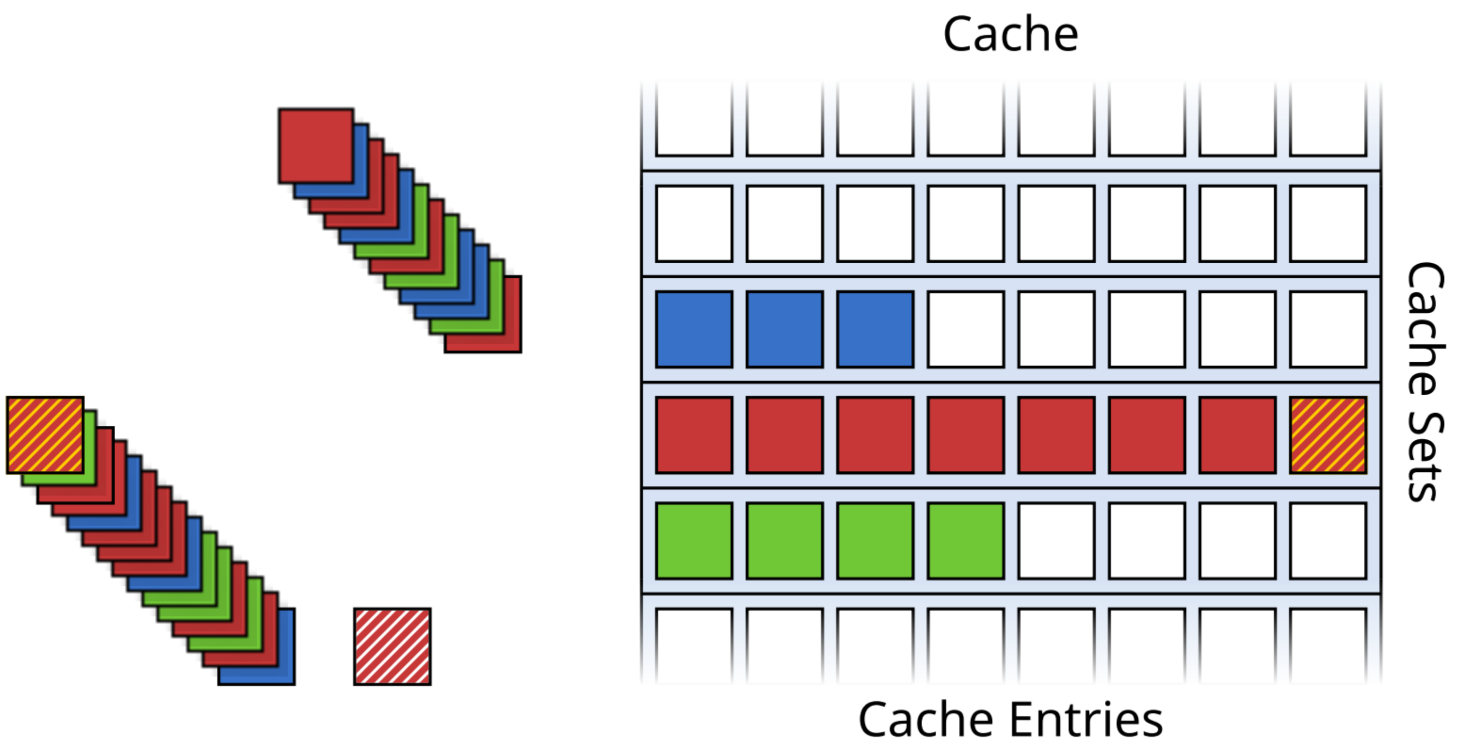
Draw pages and try to evict the target

Building Eviction Sets



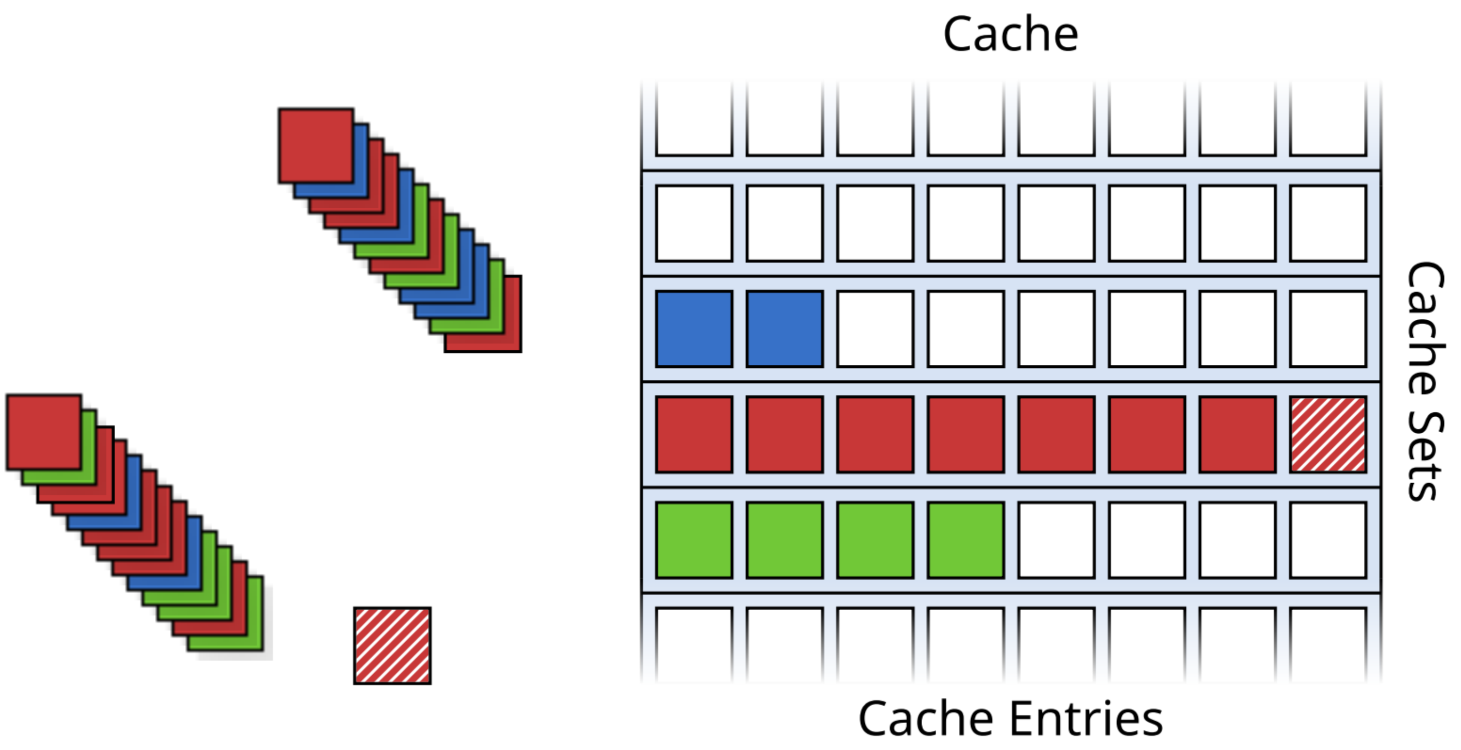
Found an eviction set

Building Eviction Sets



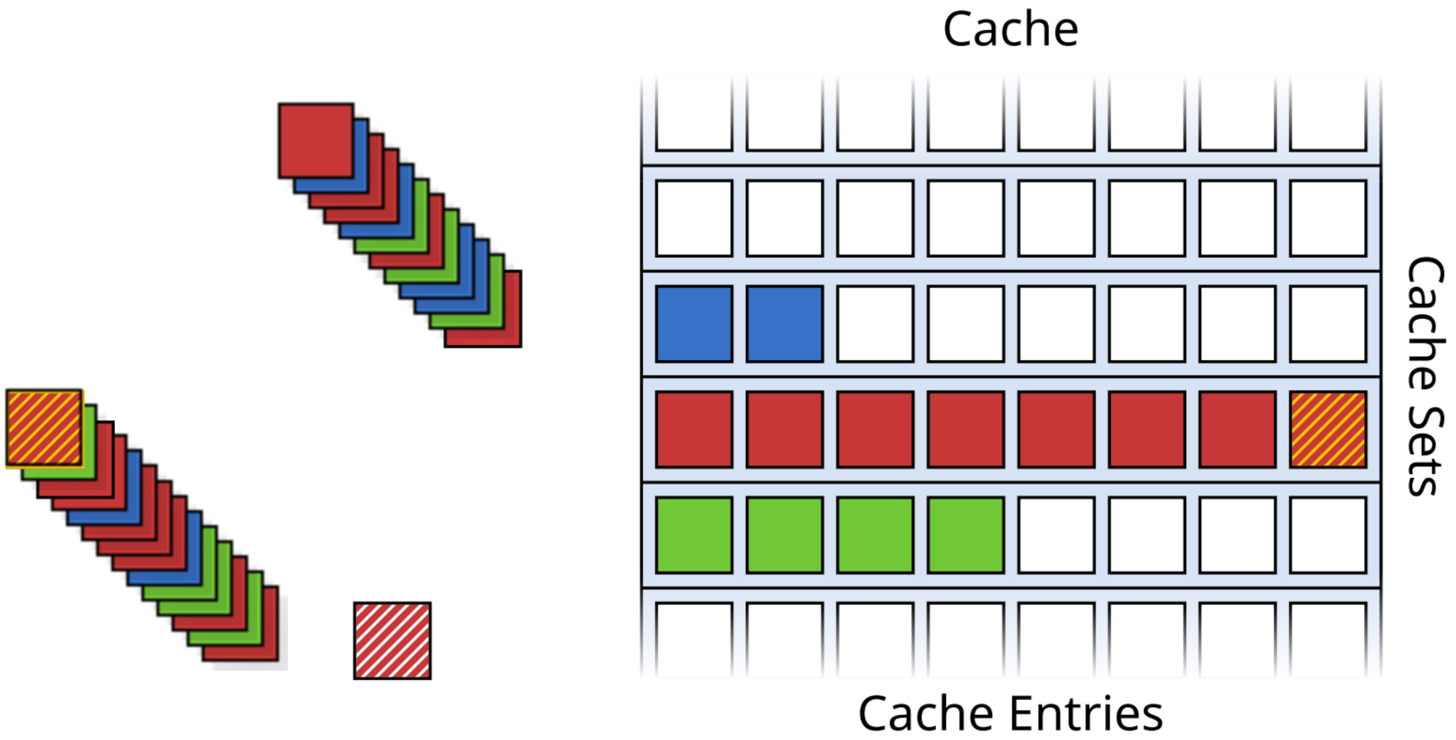
Optimize the eviction set

Building Eviction Sets



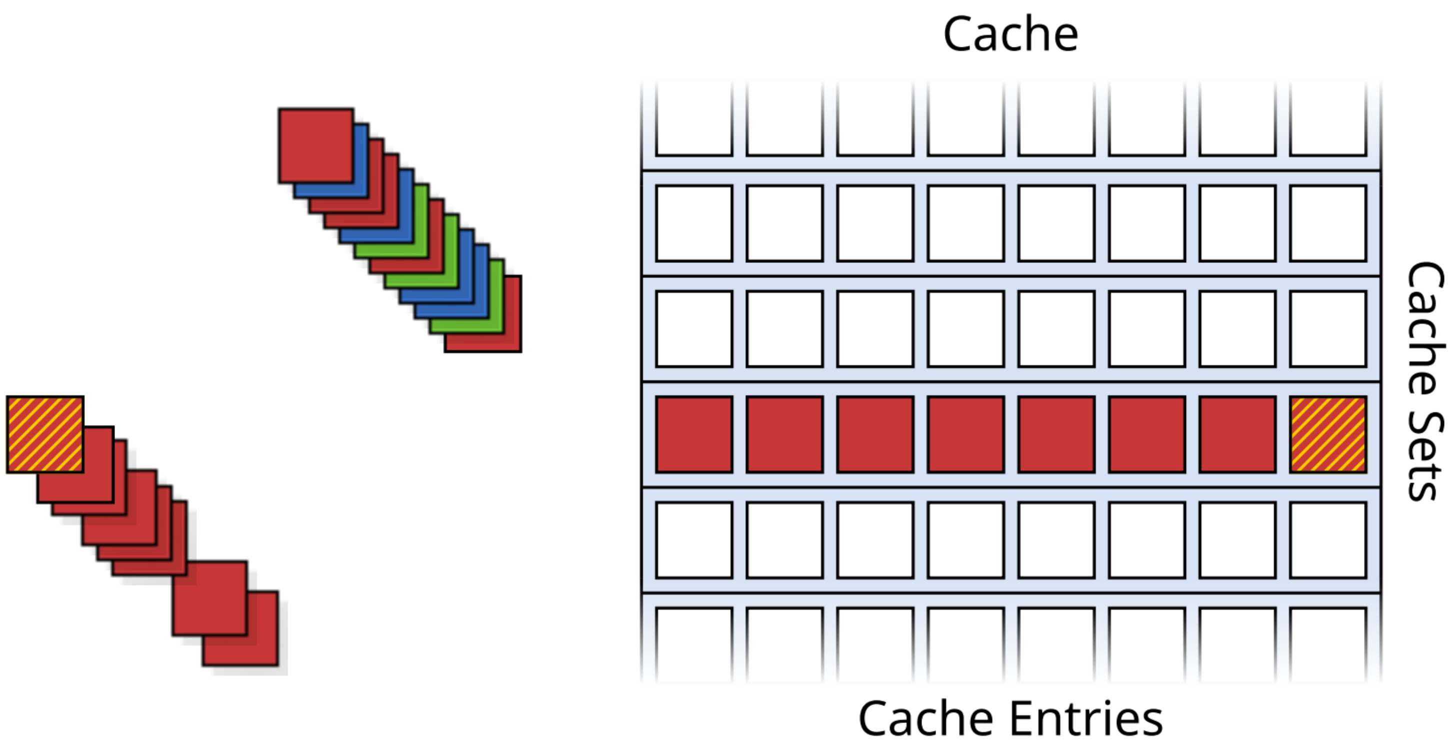
Optimize the eviction set

Building Eviction Sets



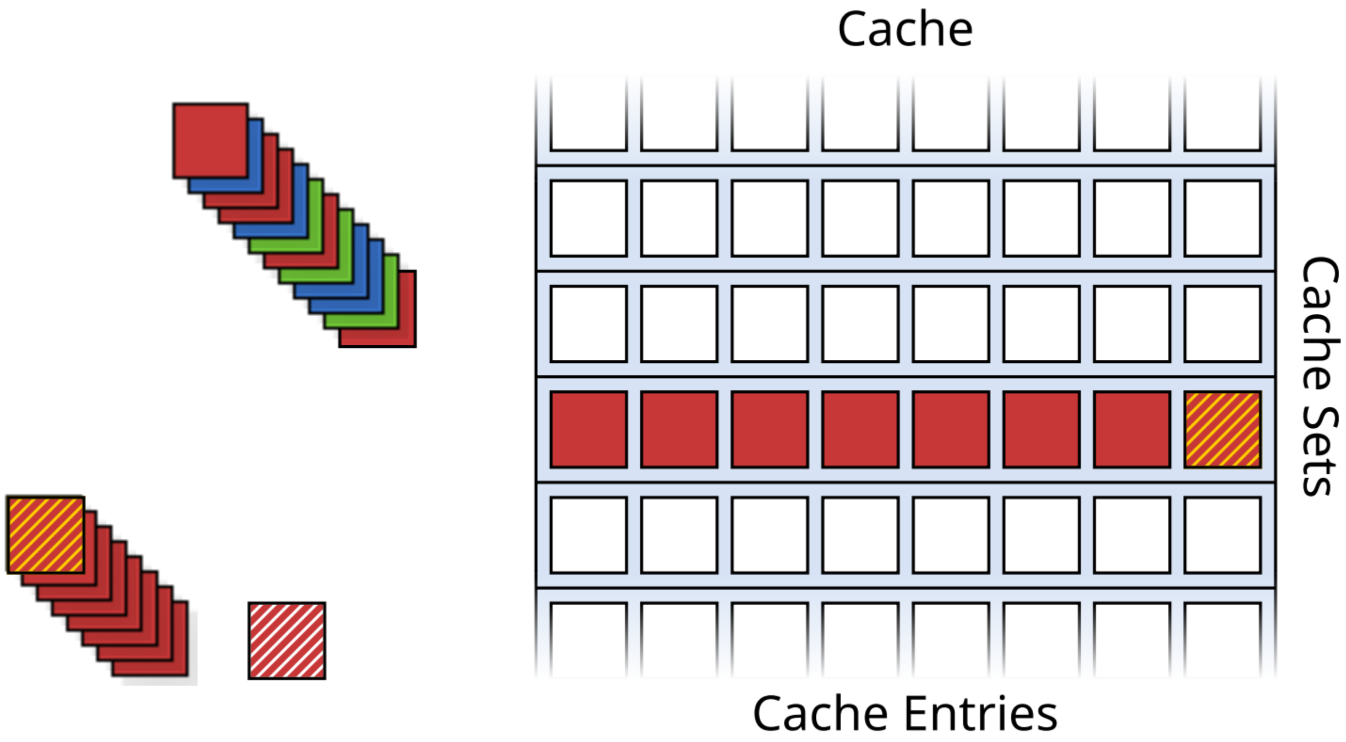
Optimize the eviction set

Building Eviction Sets



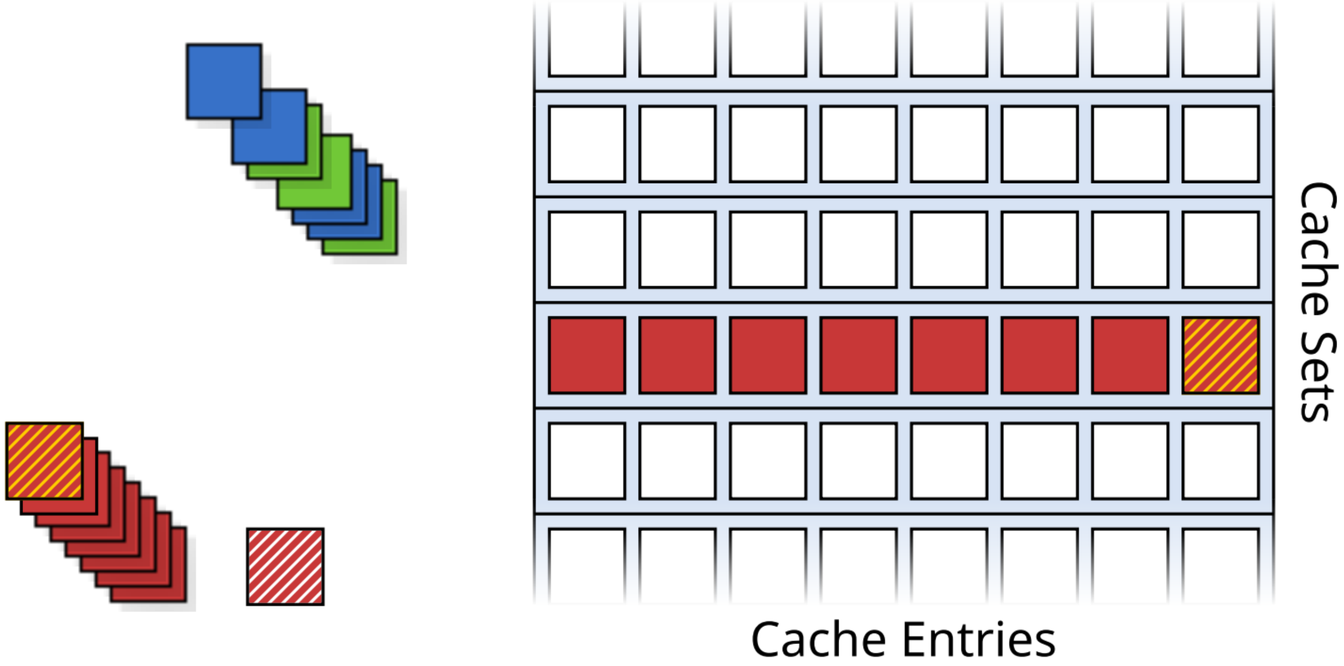
Optimize the eviction set

Building Eviction Sets



Optimal eviction set found

Building Eviction Sets



Filter red pages

Keep going until you have all eviction sets

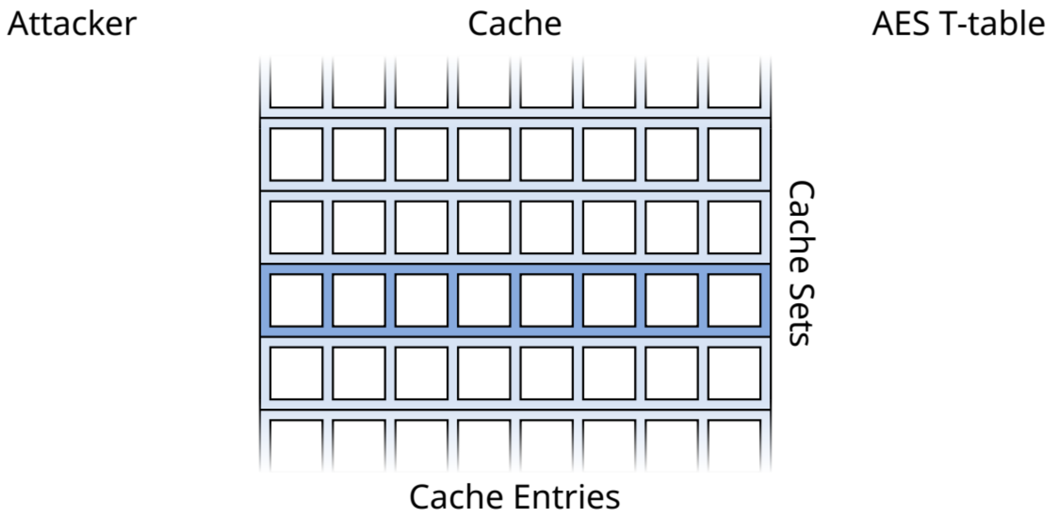
Also works for page tables

Challenges

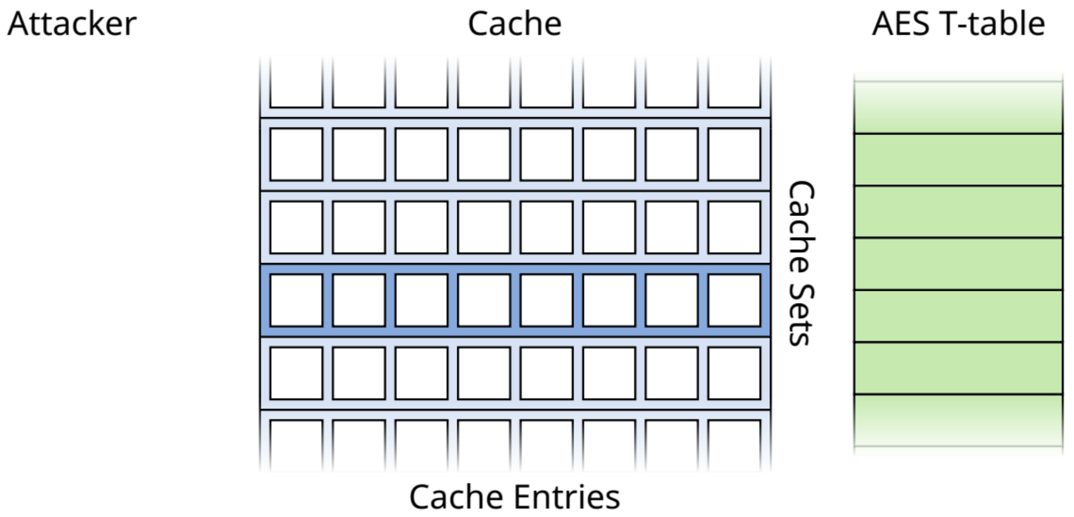
- Avoid noise from high-level page tables
- Avoid noise from pages
- Build eviction sets

Xlate & Probe: the Big Picture

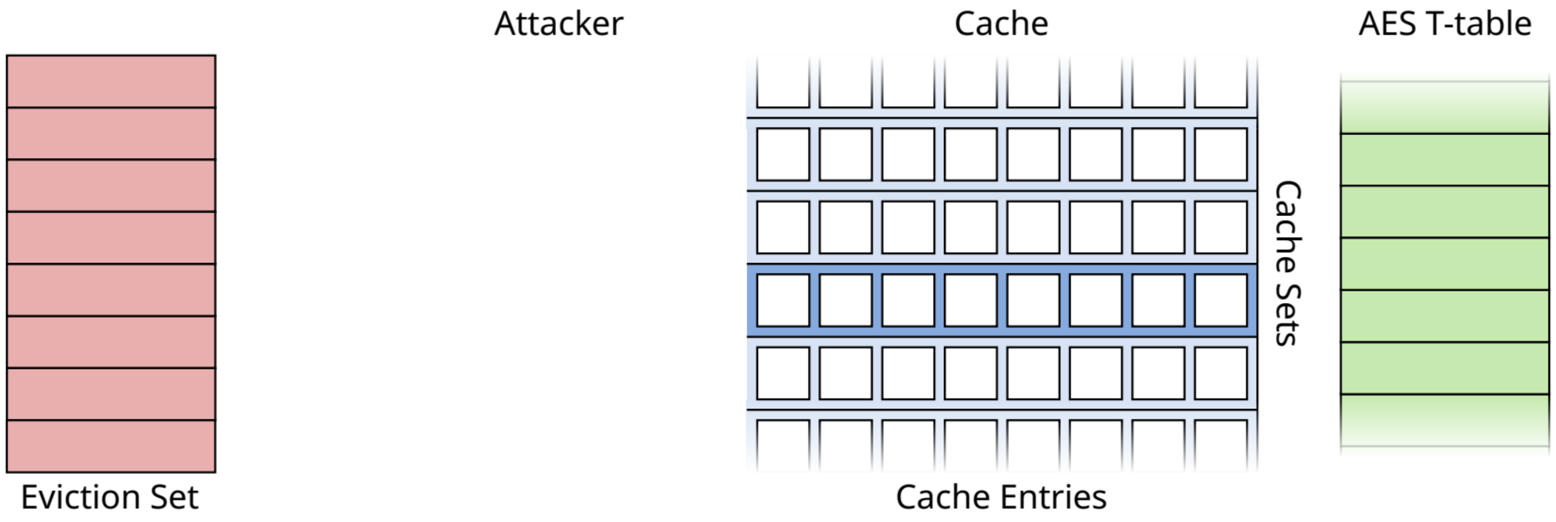
XLATE + PROBE



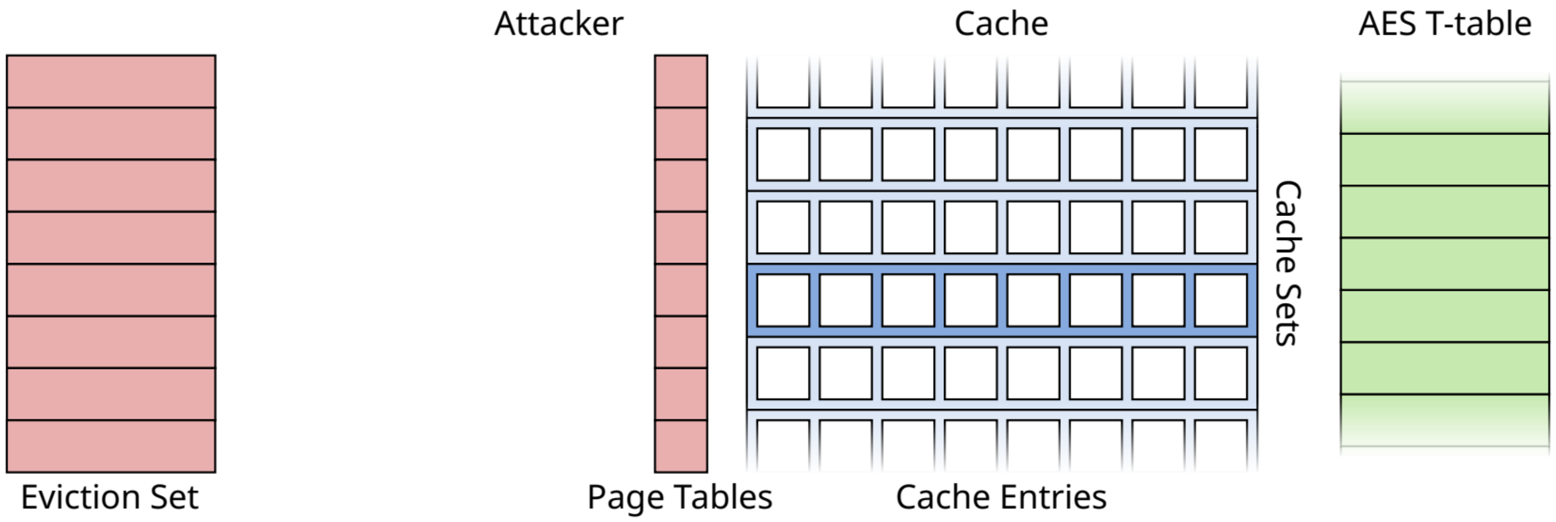
XLATE + PROBE



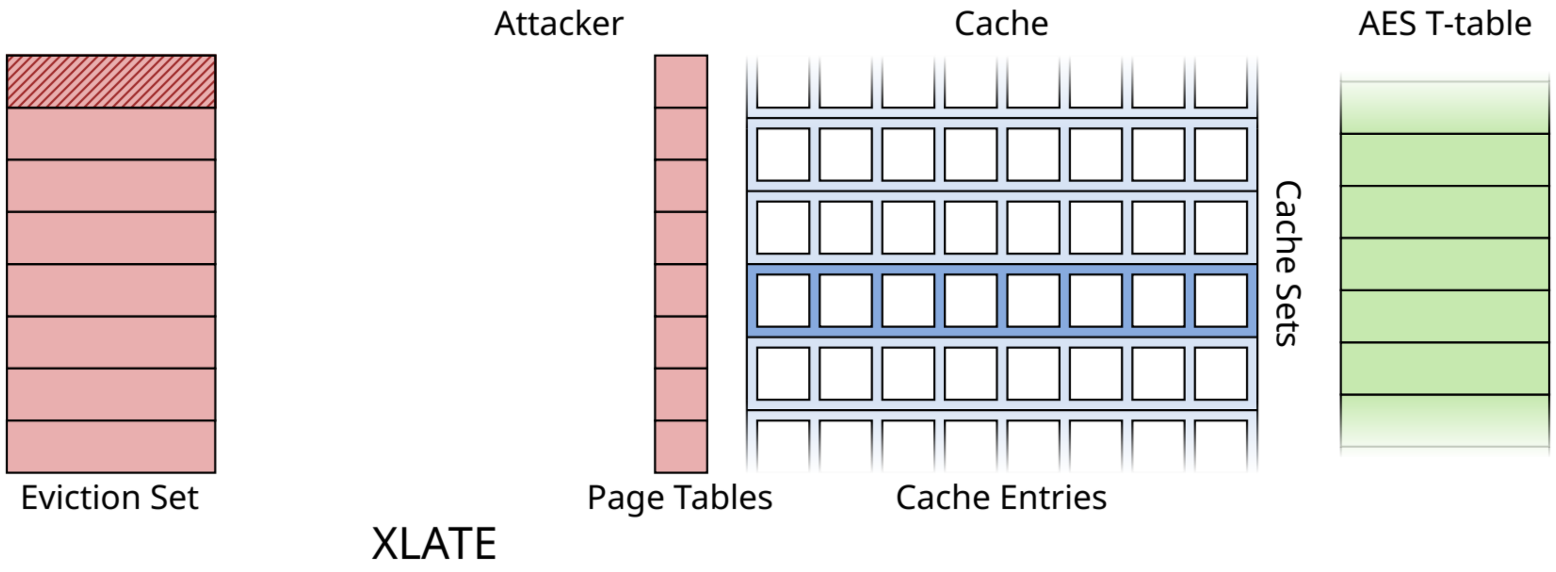
XLATE + PROBE


Attacker
Cache
AES T-table
Cache Sets
Eviction Set
Cache Entries

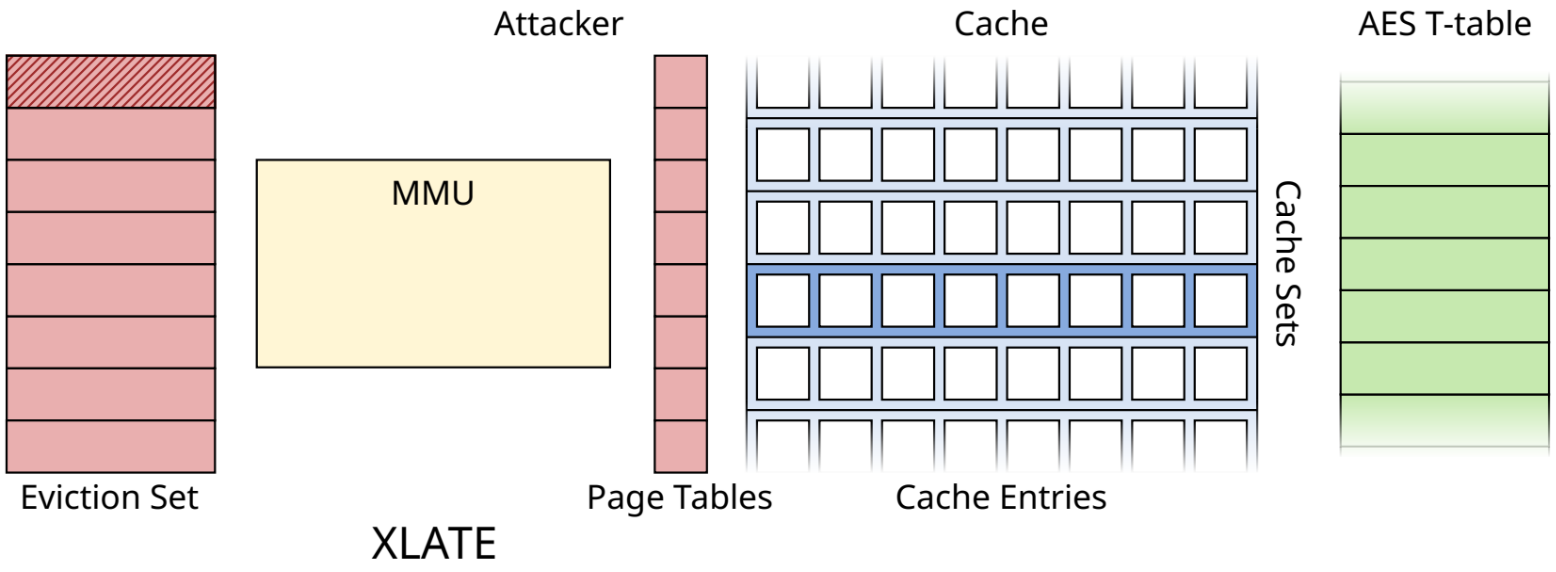
XLATE + PROBE


Attacker
Cache
AES T-table
Cache Sets
Eviction Set
Page Tables
Cache Entries

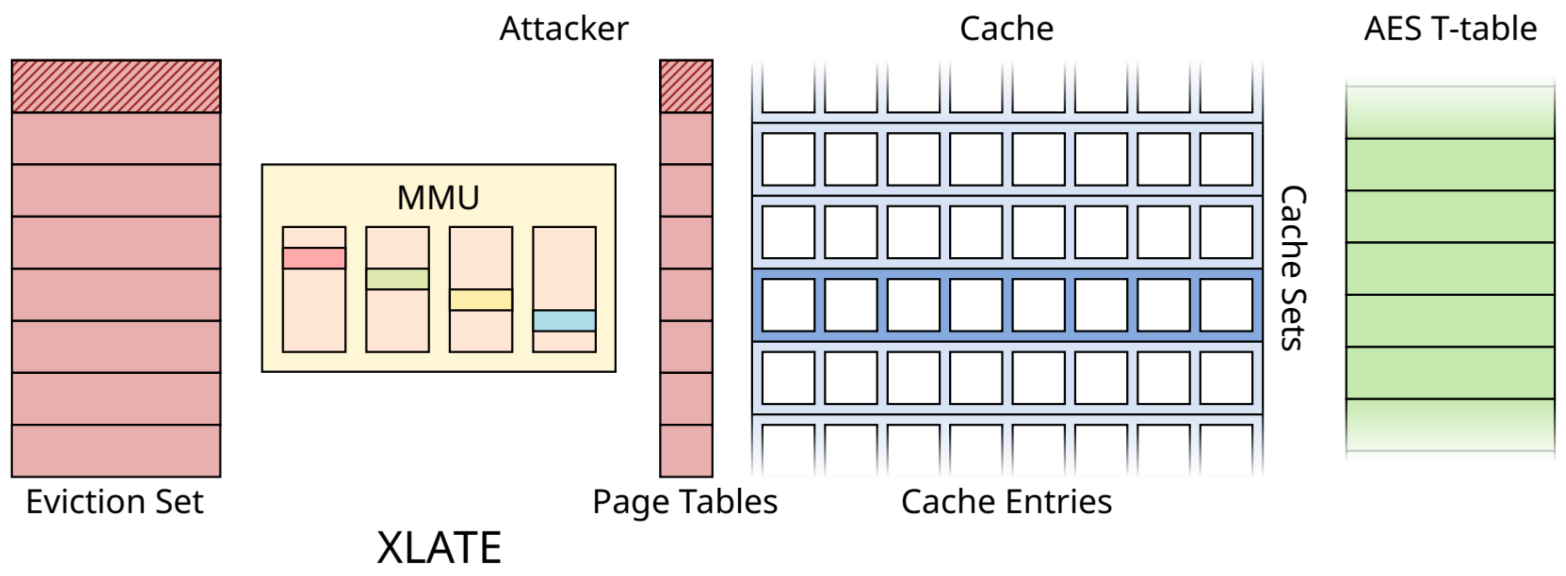
XLATE + PROBE


Attacker
Cache
AES T-table
Cache Sets
Eviction Set
Page Tables
Cache Entries
XLATE

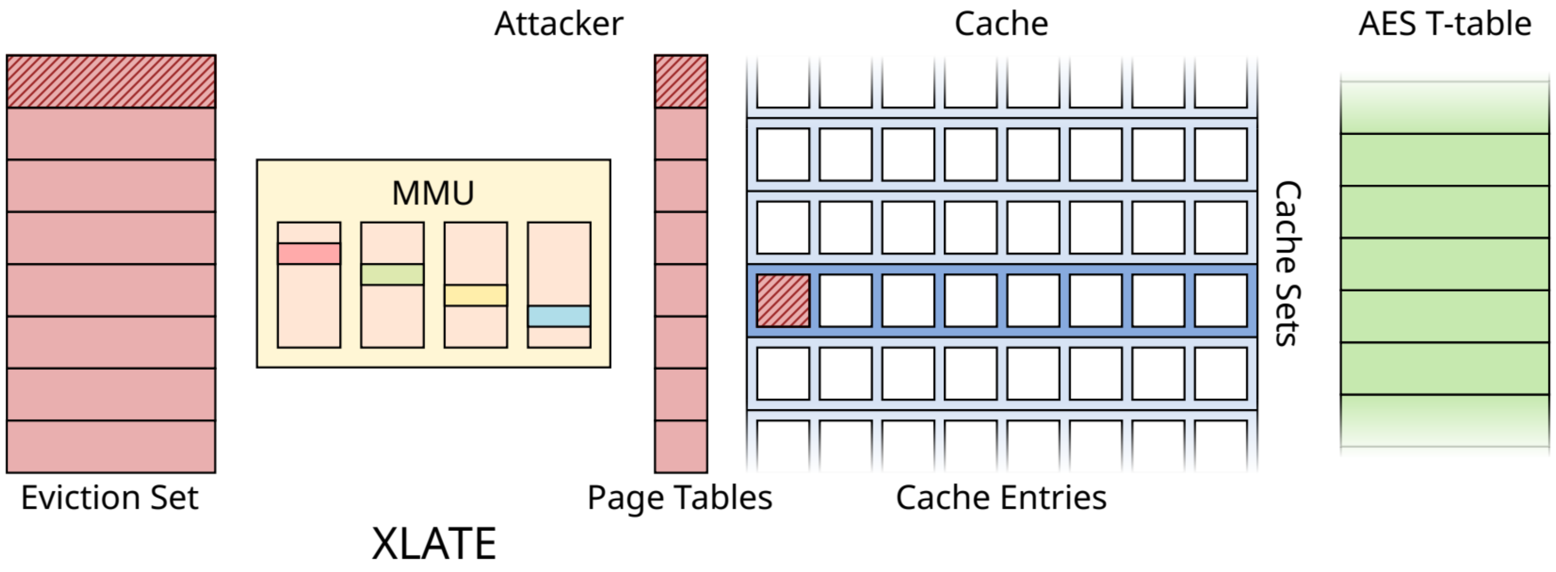
XLATE + PROBE


Attacker
Cache
AES T-table
MMU
Cache Sets
Eviction Set
Page Tables
Cache Entries
XLATE

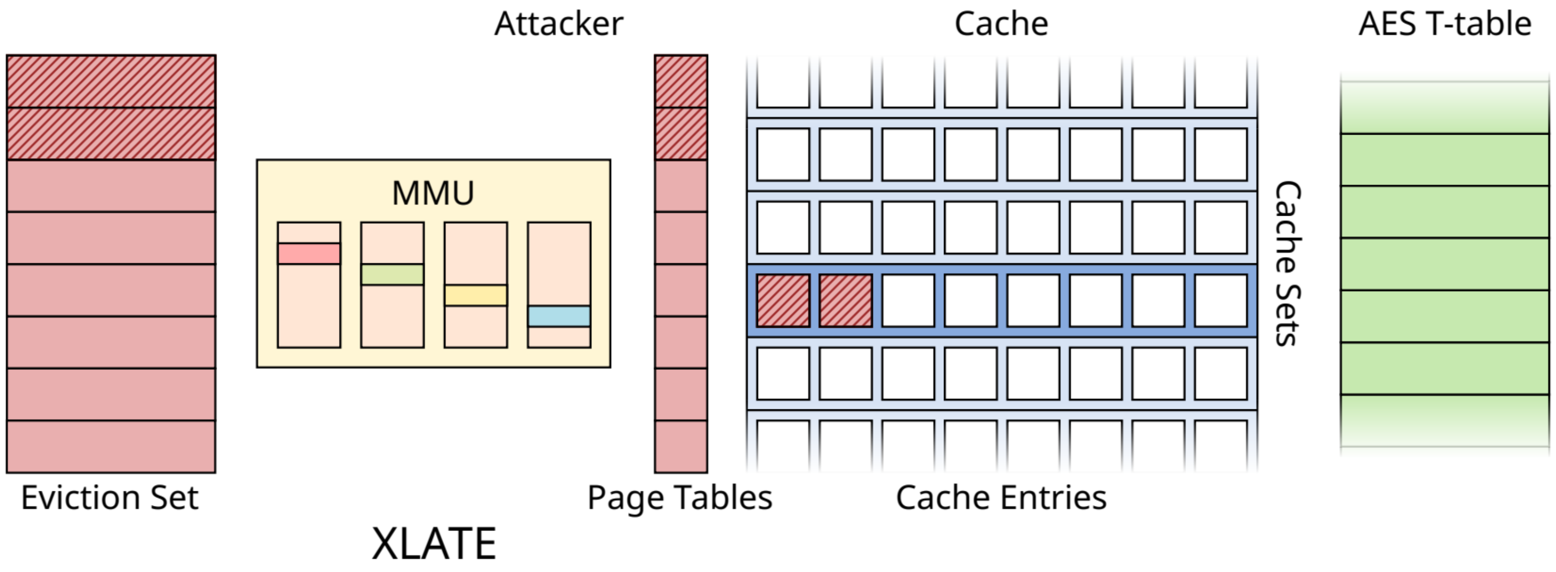
XLATE + PROBE


Attacker
Cache
AES T-table
MMU
Cache Sets
Eviction Set
Page Tables
Cache Entries
XLATE

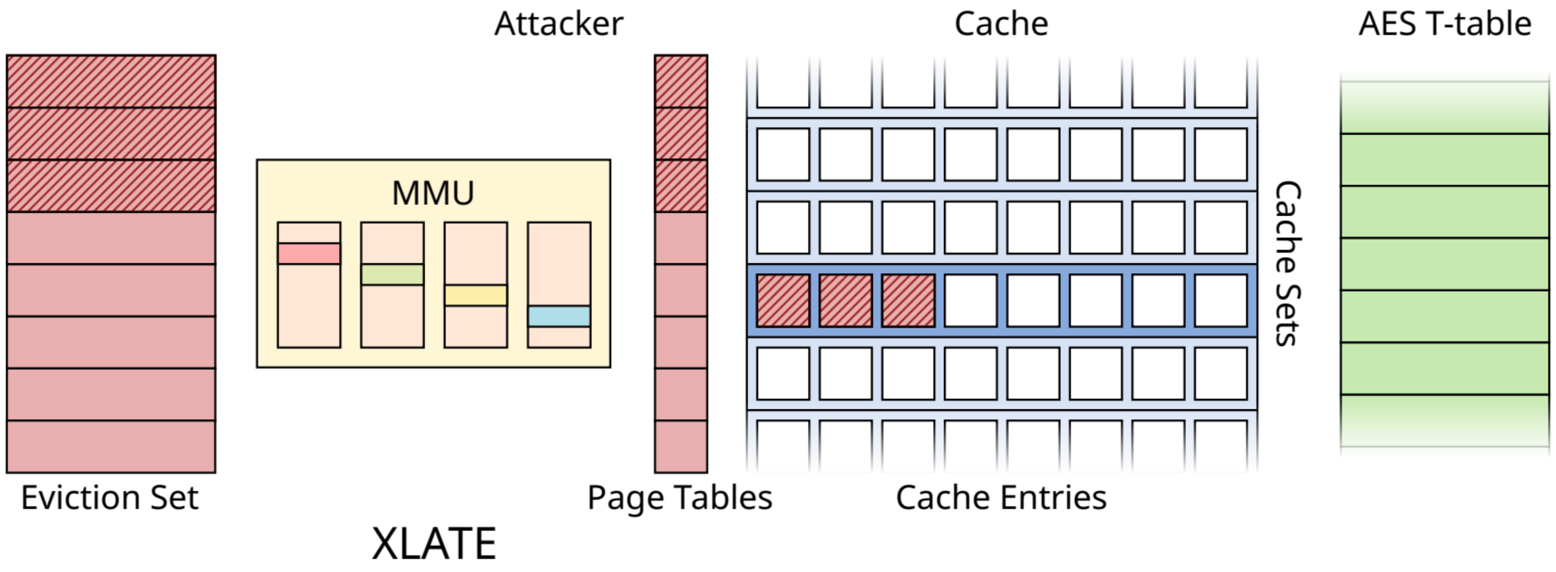
XLATE + PROBE


Attacker
Cache
AES T-table
MMU
Cache Sets
Eviction Set
Page Tables
Cache Entries
XLATE

XLATE + PROBE

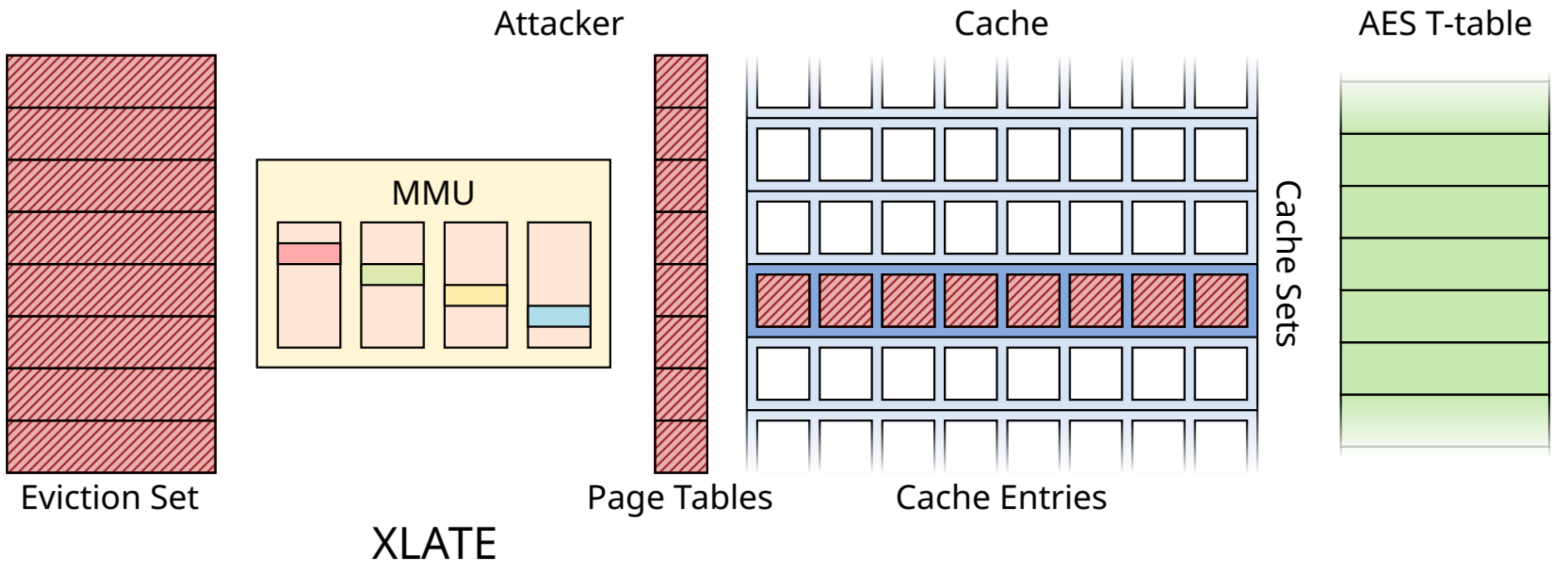

Attacker
Cache
AES T-table
MMU
Cache Sets
Eviction Set
Page Tables
Cache Entries
XLATE

XLATE + PROBE

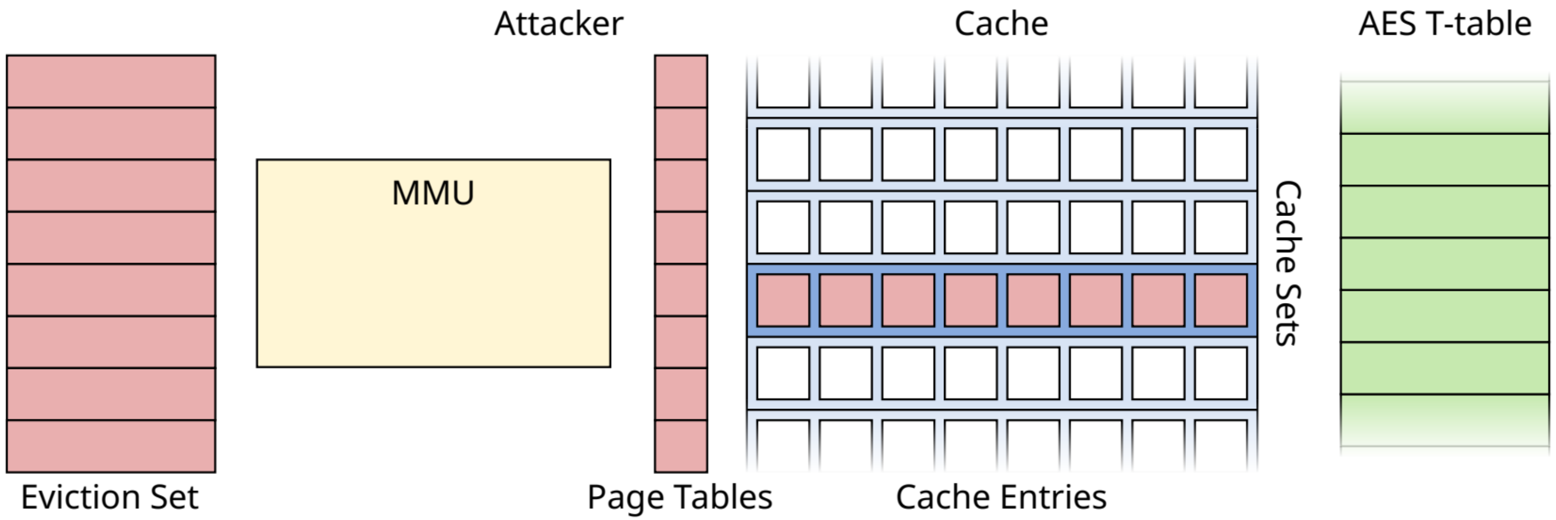


Attacker
Cache
AES T-table
MMU
Cache Sets
Eviction Set
Page Tables
Cache Entries
XLATE

XLATE + PROBE

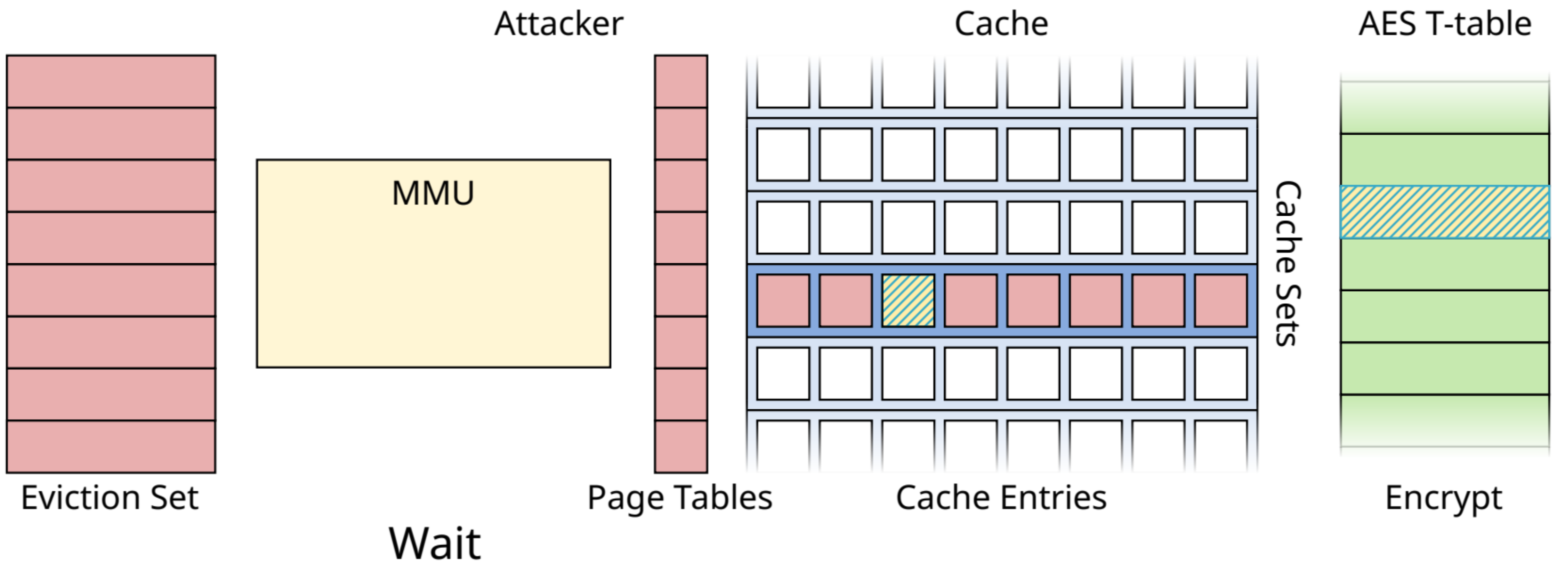

Attacker
Cache
AES T-table
MMU
Cache Sets
Eviction Set
Page Tables
Cache Entries
XLATE

XLATE + PROBE

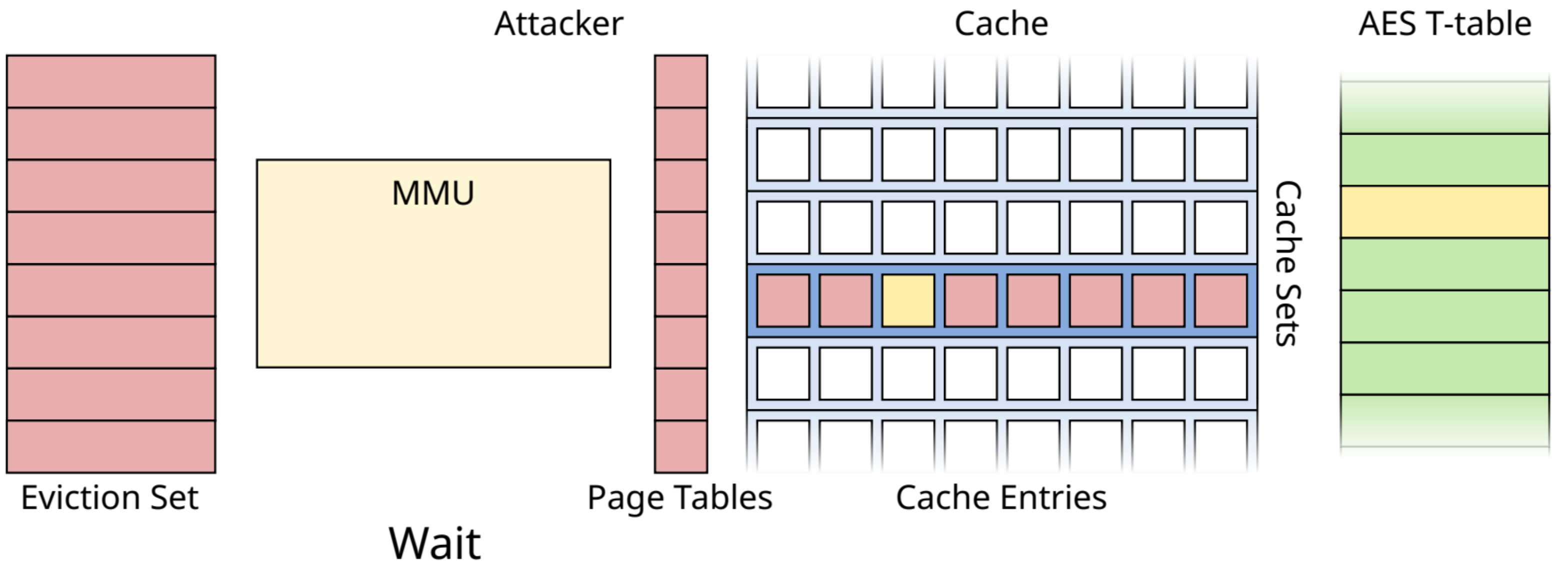


Wait

XLATE + PROBE

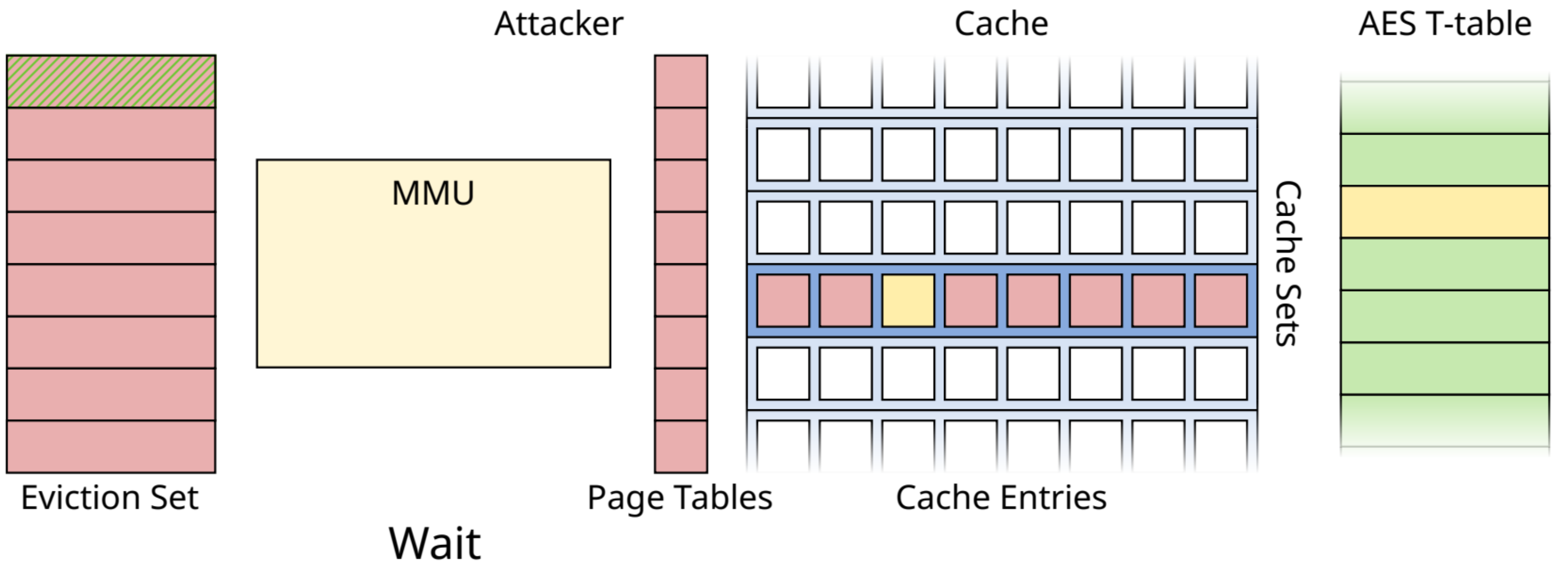

Attacker
Cache
AES T-table
MMU
Cache Sets
Eviction Set
Page Tables
Cache Entries
Encrypt
Wait

XLATE + PROBE



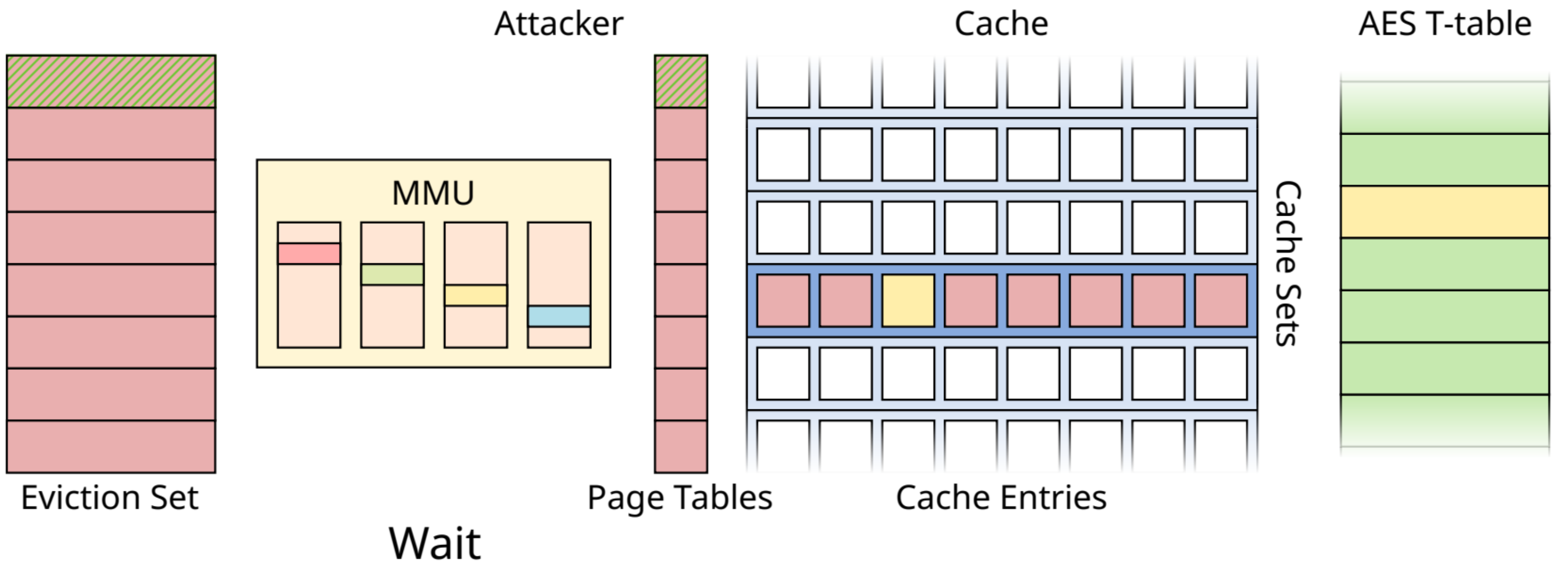
Attacker
Cache
AES T-table
MMU
Cache Sets
Eviction Set
Page Tables
Cache Entries
Wait

XLATE + PROBE

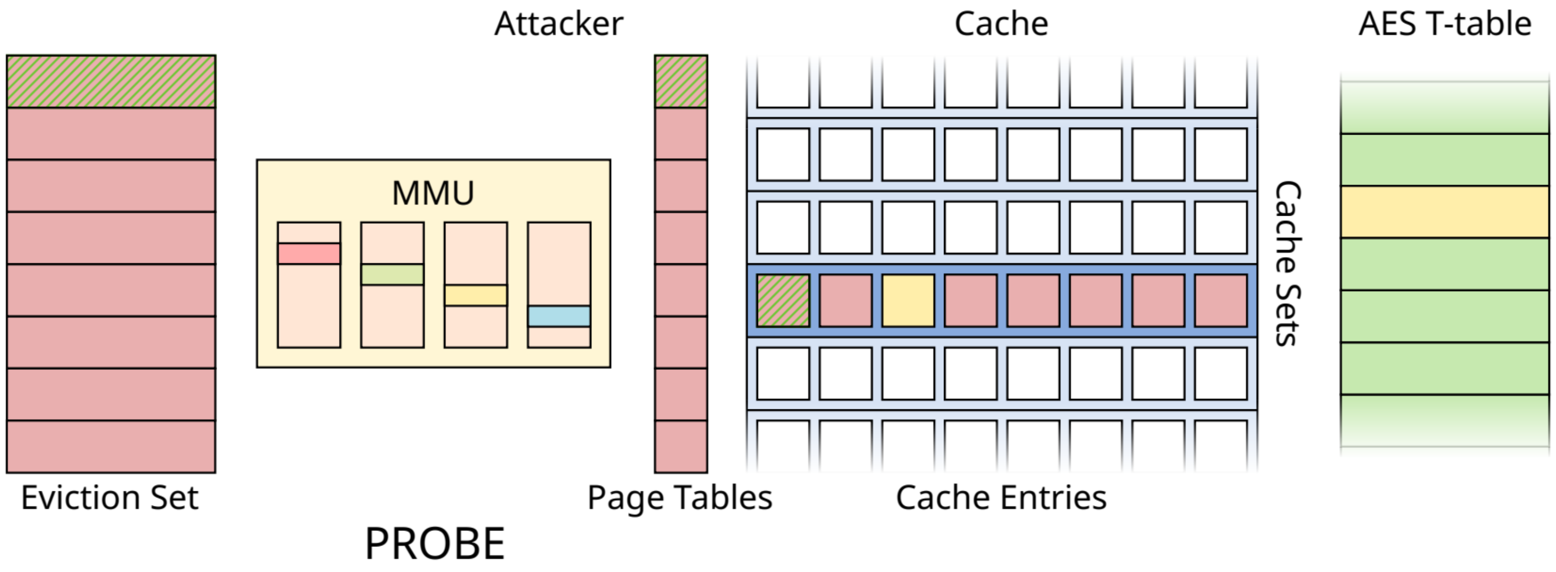


Attacker
Cache
AES T-table
MMU
Cache Sets
Eviction Set
Page Tables
Cache Entries
Wait

XLATE + PROBE

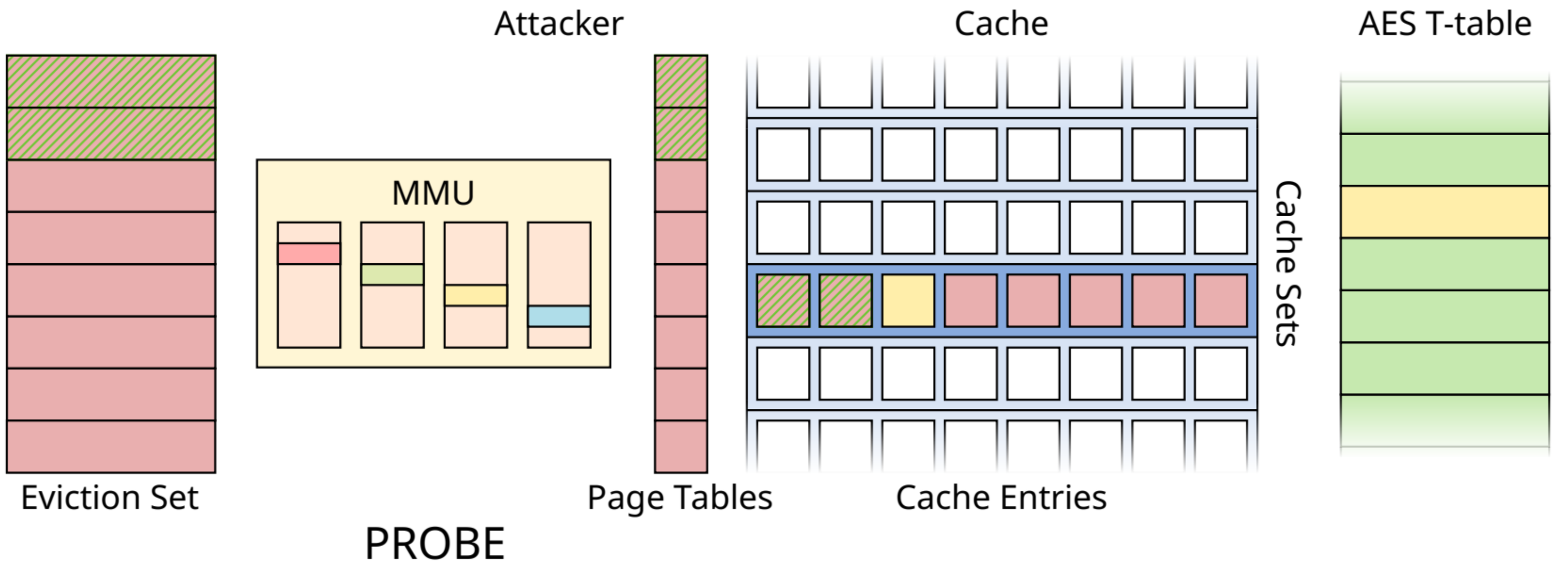

Attacker
Cache
AES T-table
MMU
Cache Sets
Eviction Set
Page Tables
Cache Entries
Wait

XLATE + PROBE

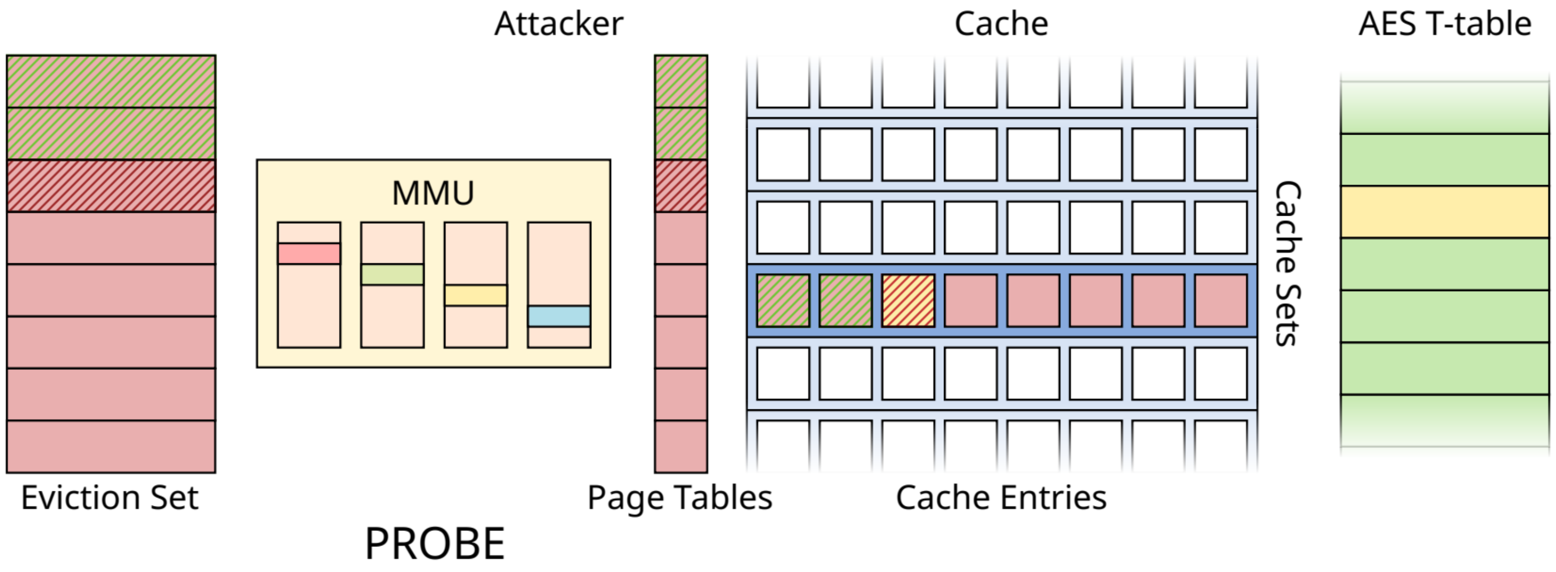


Attacker
Cache
AES T-table
MMU
Cache Sets
Eviction Set
Page Tables
Cache Entries
PROBE

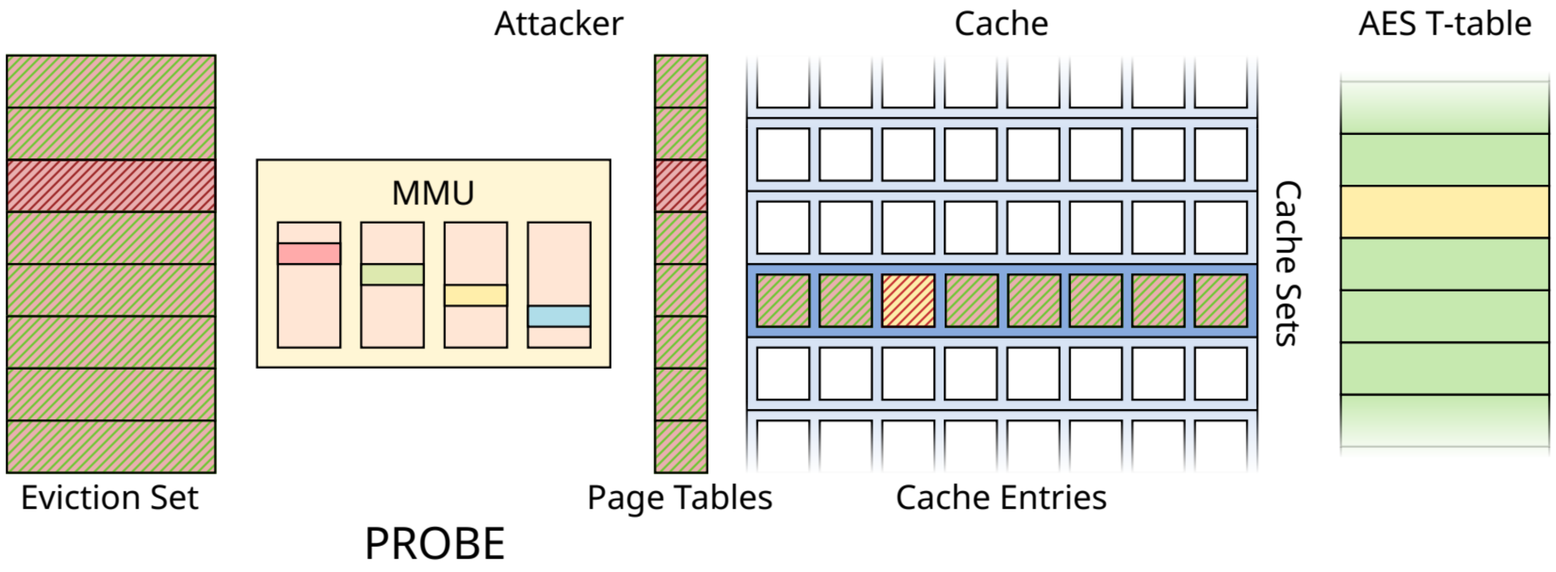
XLATE + PROBE


Attacker
Cache
AES T-table
MMU
Cache Sets
Eviction Set
Page Tables
Cache Entries
PROBE

XLATE + PROBE

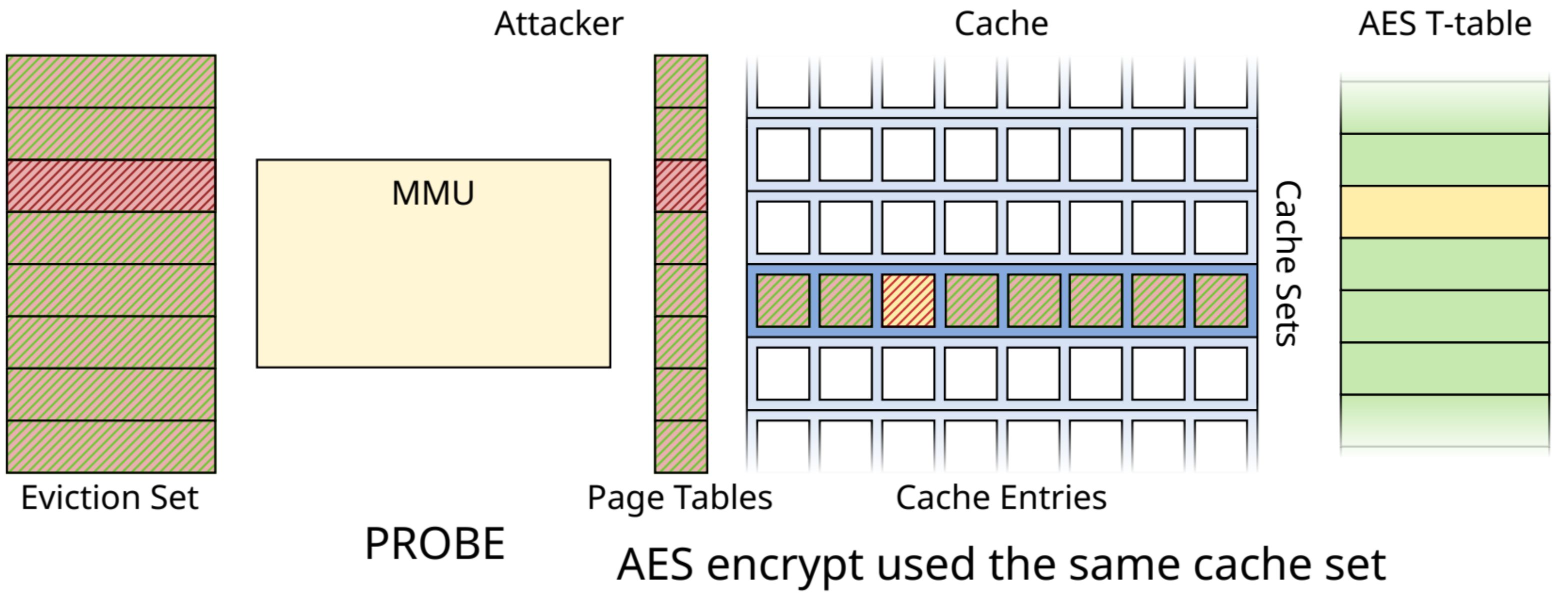

Attacker
Cache
AES T-table
MMU
Cache Sets
Eviction Set
Page Tables
Cache Entries
PROBE

XLATE + PROBE

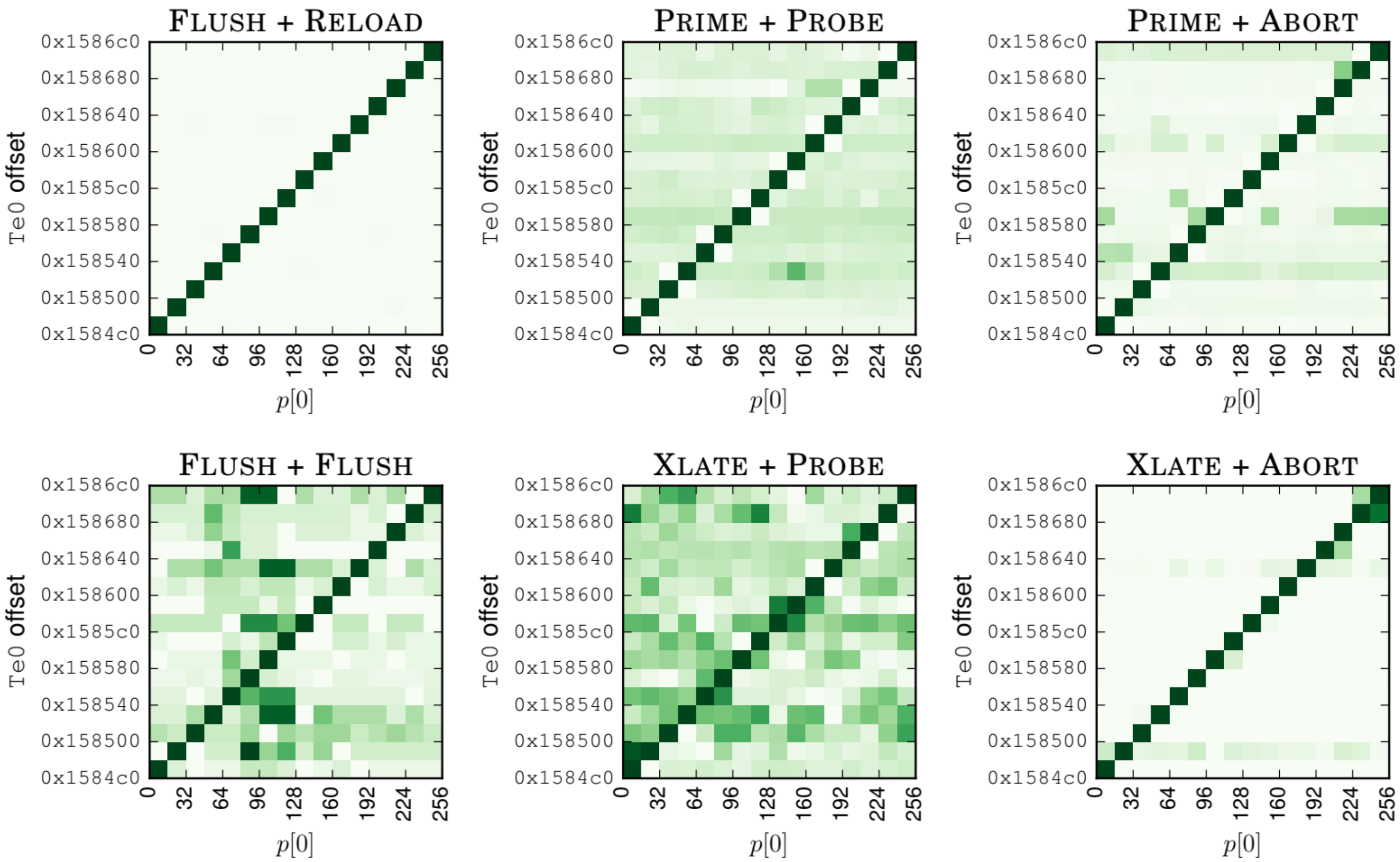


Attacker
Cache
AES T-table
MMU
Cache Sets
Eviction Set
Page Tables
Cache Entries
PROBE

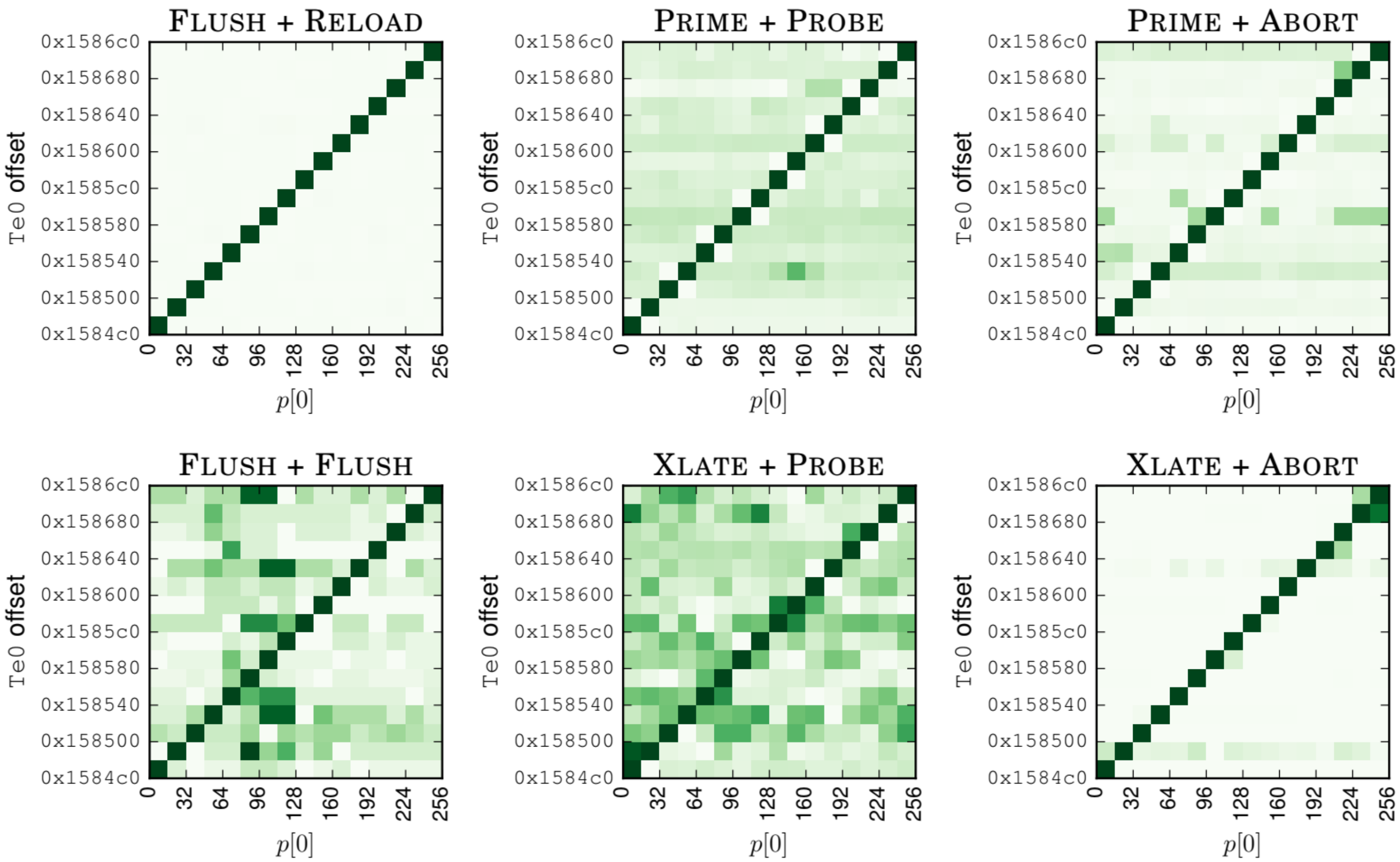
XLATE + PROBE


Attacker
Cache
AES T-table
MMU
Cache Sets
Eviction Set
Page Tables
Cache Entries
PROBE
AES encrypt used the same cache set

Effectiveness

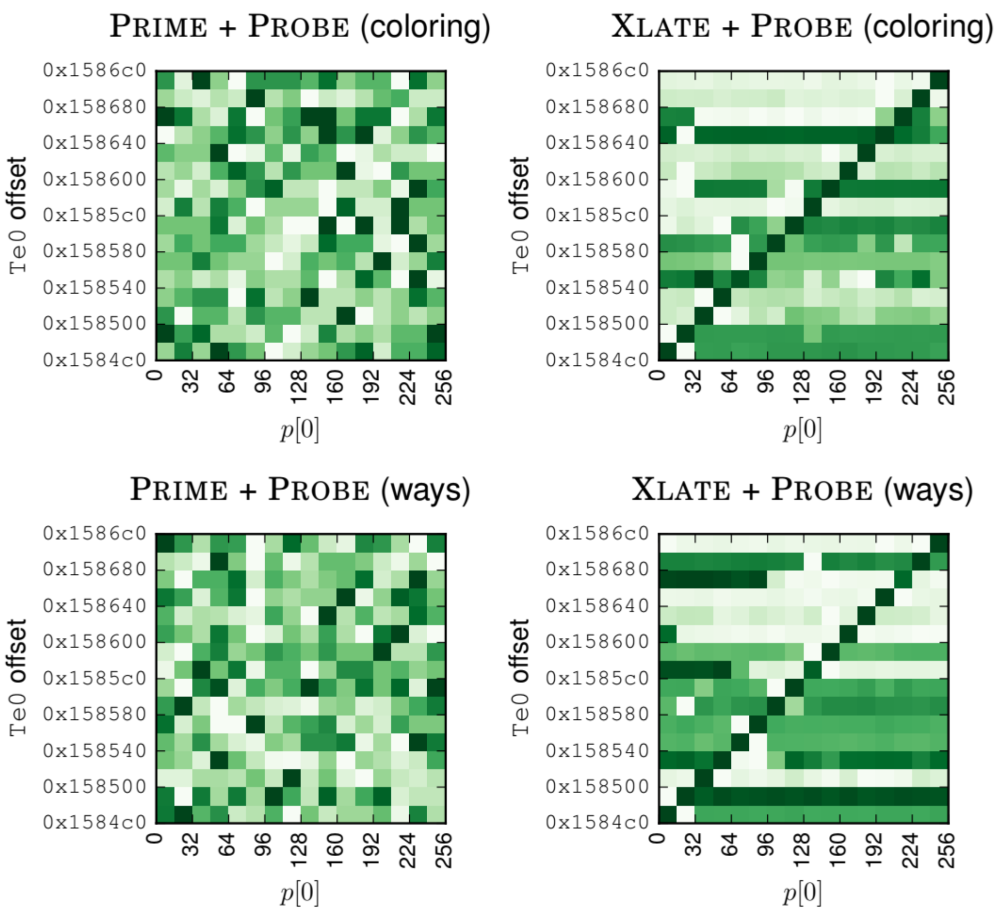


Effectiveness



XLATE + PROBE is effective against AES T-tables

Cache Defenses



XLATE + PROBE bypasses set and way partitioning

Conclusions so far

- Indirect cache attacks are practical
- Must reconsider cache defenses

`https://vusec.net/projects/xlate`

Conclusions so far

- Indirect cache attacks are practical
- Must reconsider cache defenses

https://vusec.net/projects/xlate

Yes.

$0,-

TLBleed

AKA "Side channeling the TLB"

Ben Gras

The Volunteer State
Tennessee
JUL
18
PWNAGE
PWNAGE
KN
WHEEL
TAX
DECAL
Class Code
2000
Issue Year
2007

TLBleeders


Ben Gras

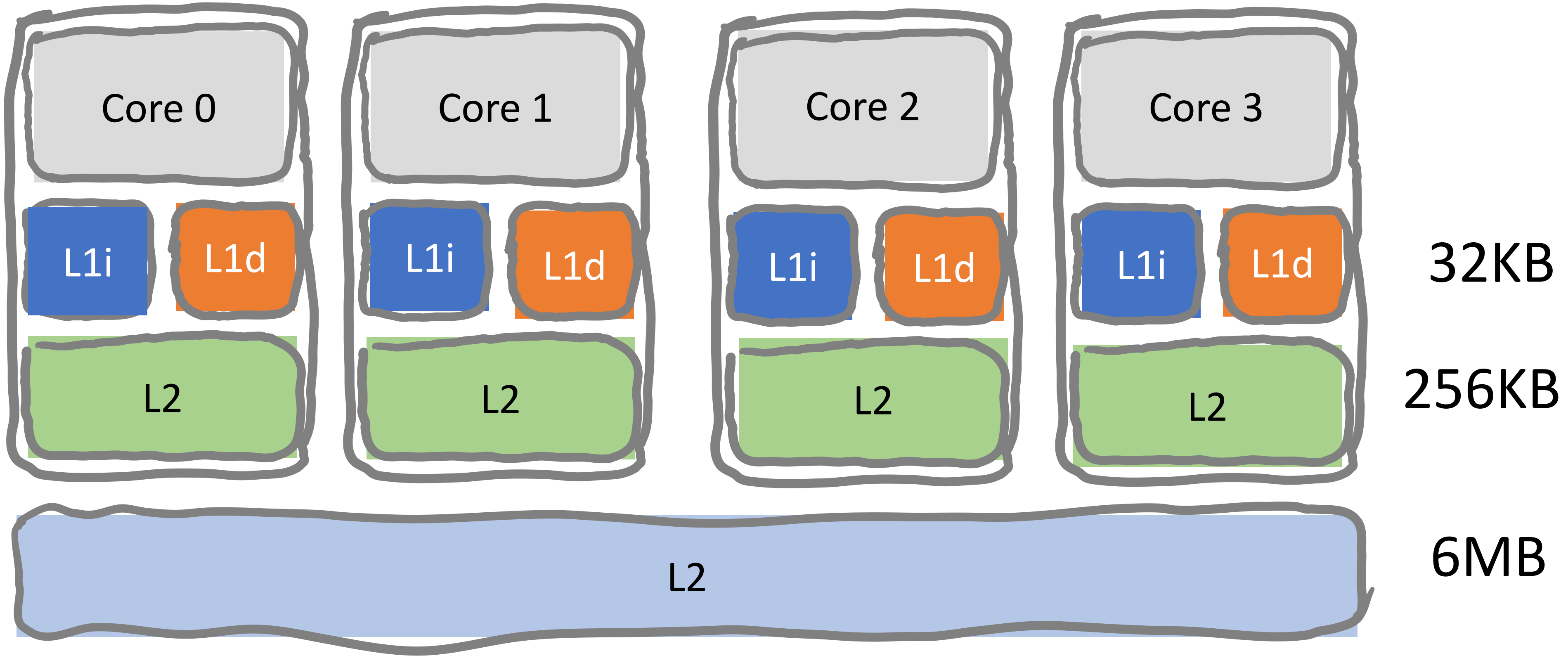

Kaveh Razavi


Cristiano Giuffrida


Herbert Bos

Side channels

Only possible because of shared resources



Brief sketch of the cache side channels (again)

Cache side channels
(Note: processes share cache)

- memory accesses depend on secret
- signing with RSA: compute $m^d \ (mod\ n)$
- to do so efficiently: square and multiply
  - iterate over all bits in key
  - square: always
  - multiply if bit is 1

Assume shared code

Attacker and victim share a crypto library

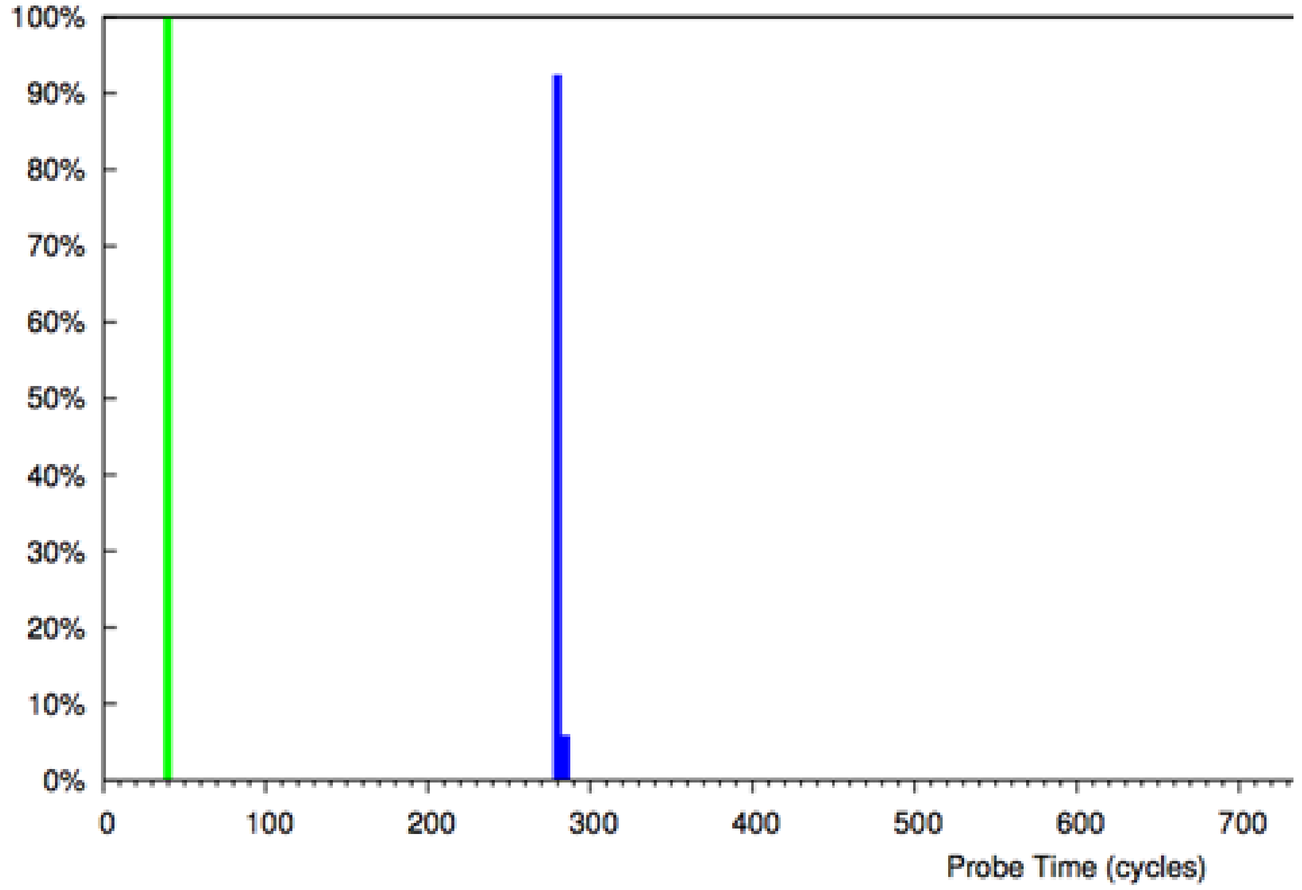
Only stored in memory once

Square and multiply at different addresses

Flush + Reload

Flush + Reload

Flush + Reload



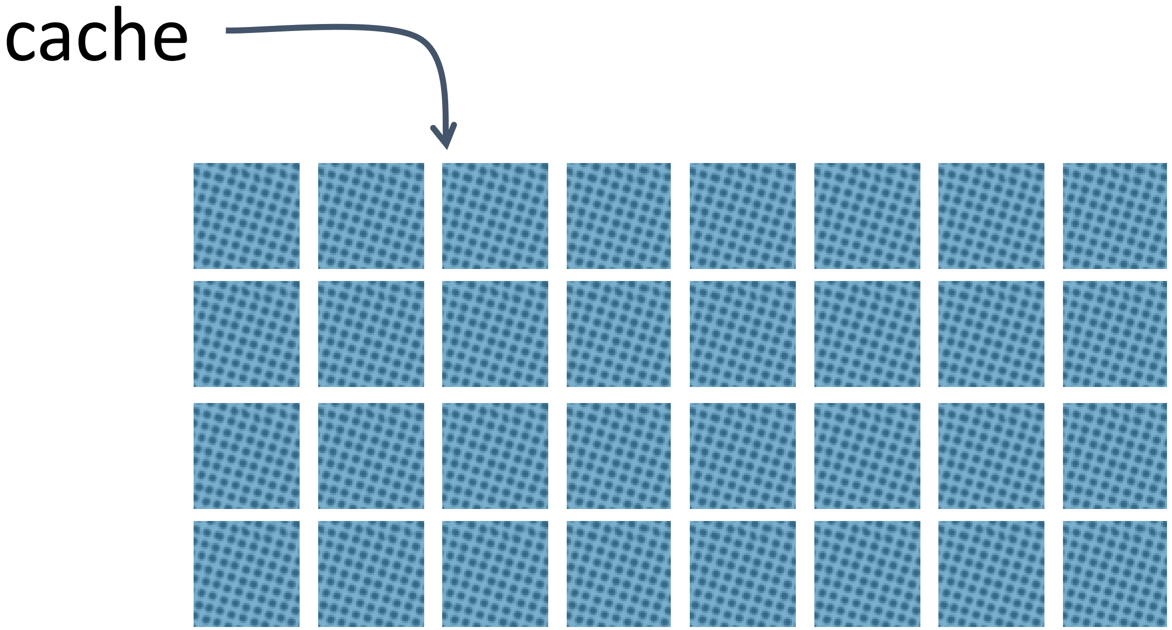
Flush + Reload

- Can also attack AES implementation with T tables

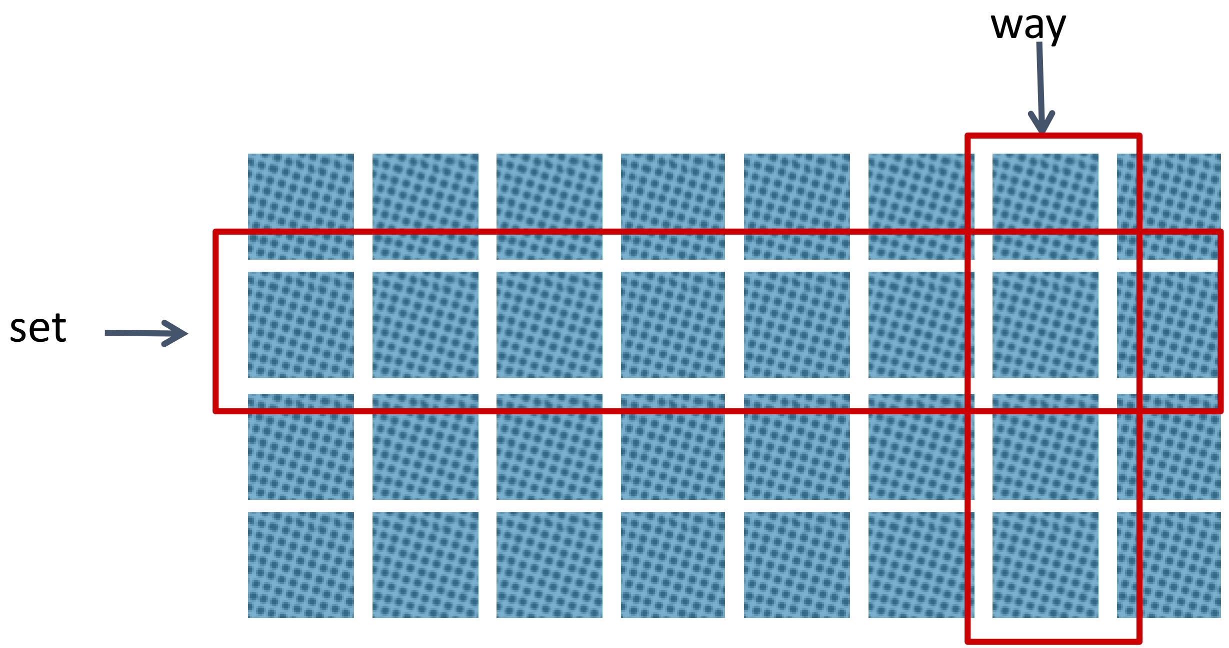
- A table lookup happens $T_j\ [x_i = p_i \oplus k_i\ ]$
  - where p_i is a plaintext byte, k_i a key byte,

Defenses

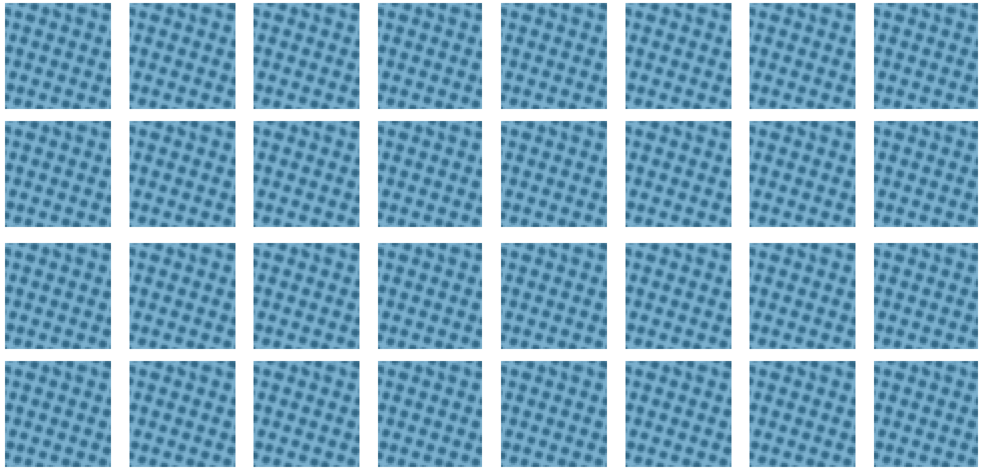
Partitioning



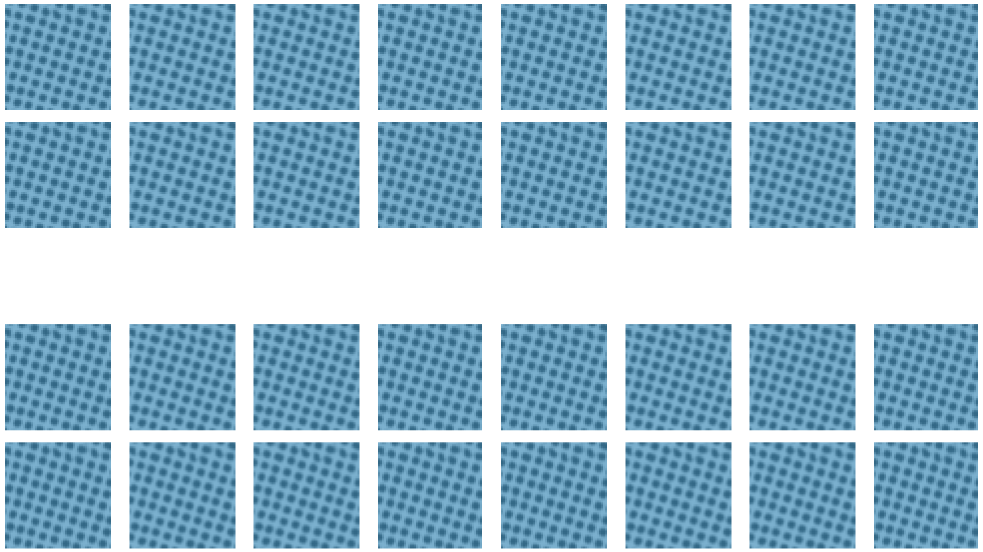
Partitioning



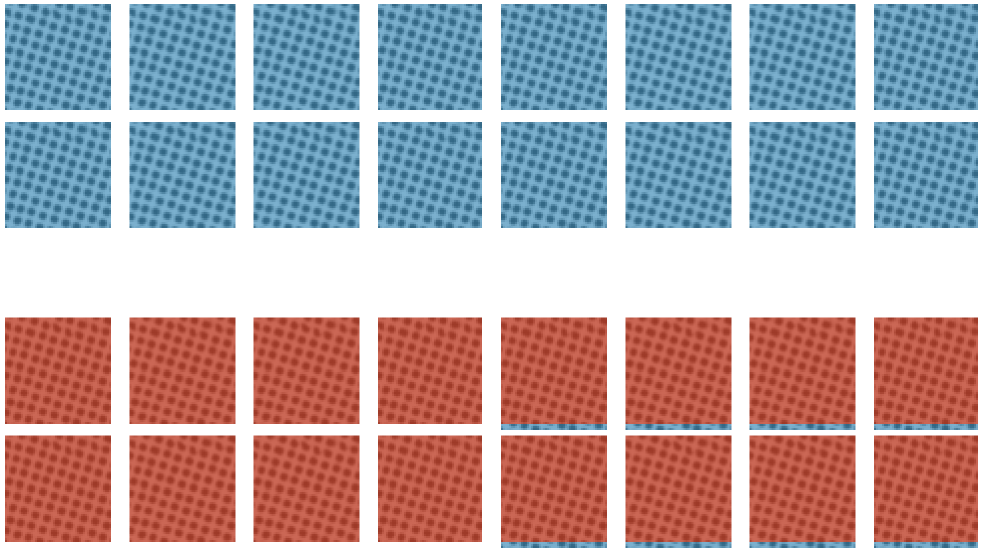
Partitioning



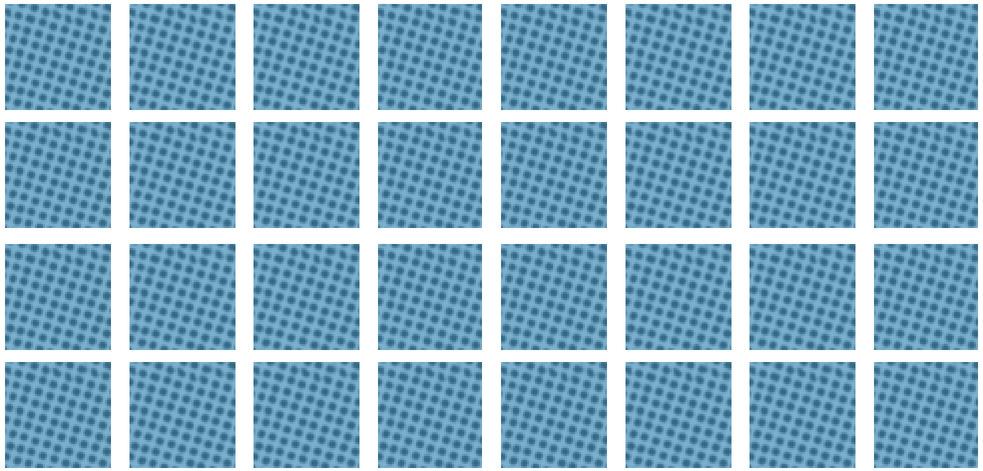
Partitioning



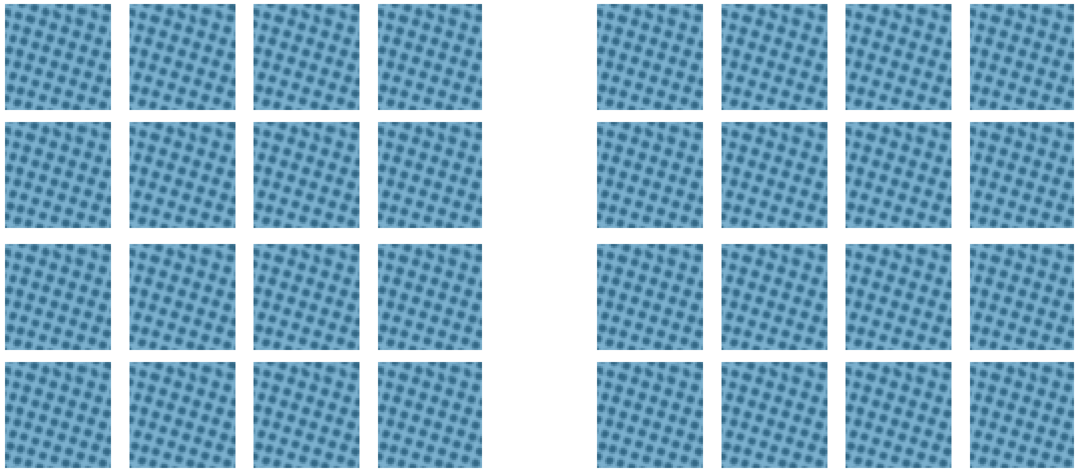
Partitioning



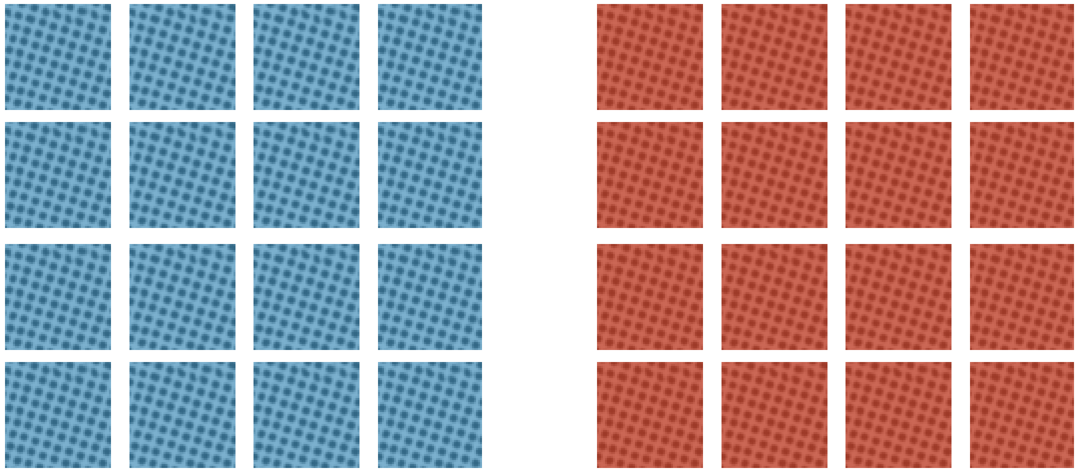
Partitioning



Partitioning



Partitioning



A third “defense”

Defenses

Set partitioning: cache colouring

Way partitioning: Intel CAT

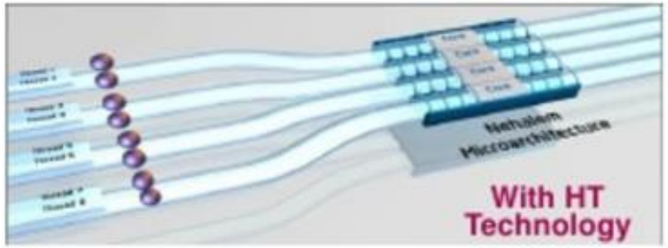
Transactions: TSX

- Intended for hardware transactional memory
- But relies on unshared cache activity
- Transactions fit in cache, otherwise auto-abort
- We can use this as a defense

Hyper Threading

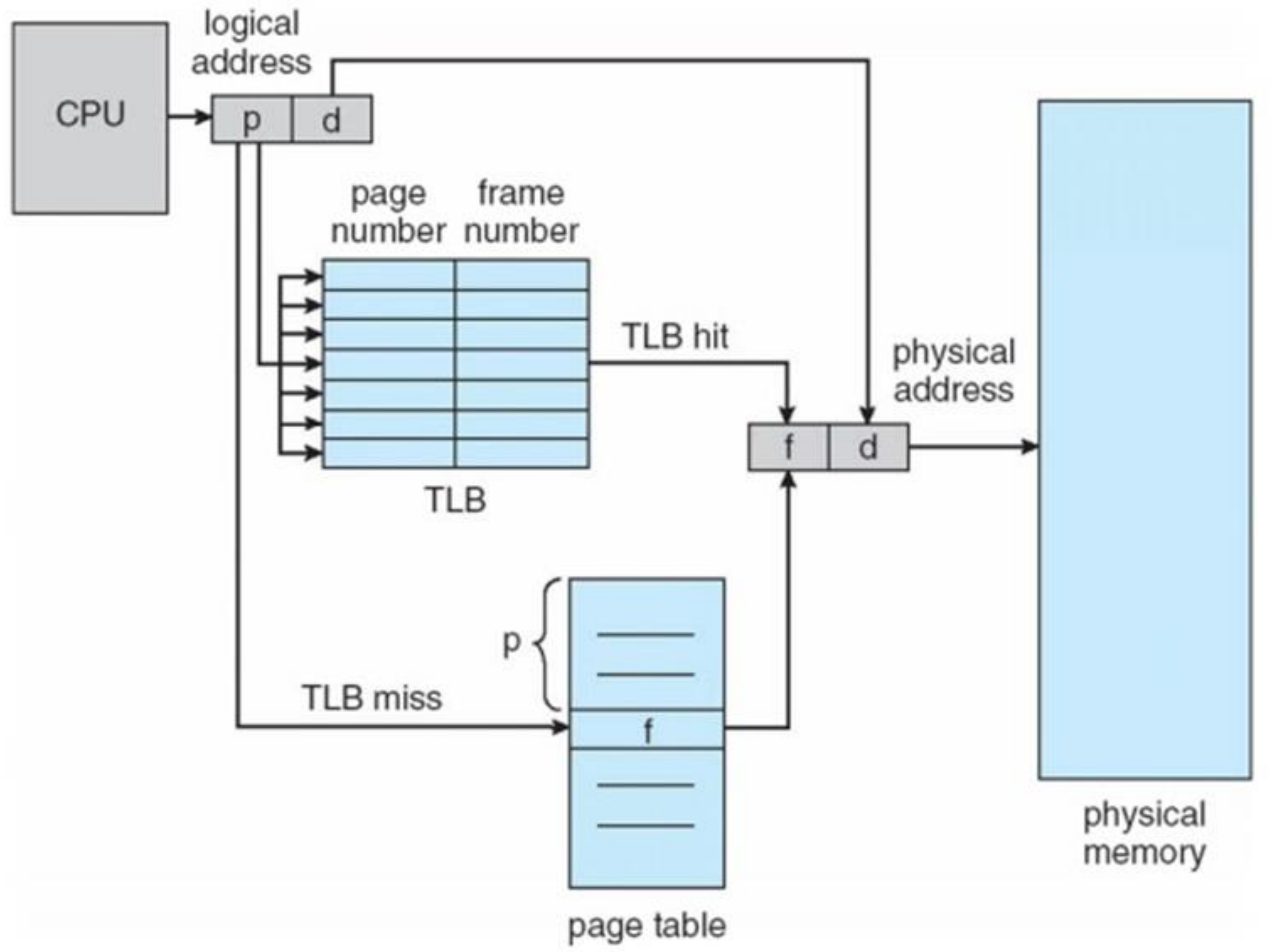
Intel® Hyper-Threading Technology

- Nehalem is a scalable multi-core architecture
- Hyper-Threading Technology augments benefits
  - Power-efficient way to boost performance in all form factors: higher multi-threaded performance, faster multi-tasking response

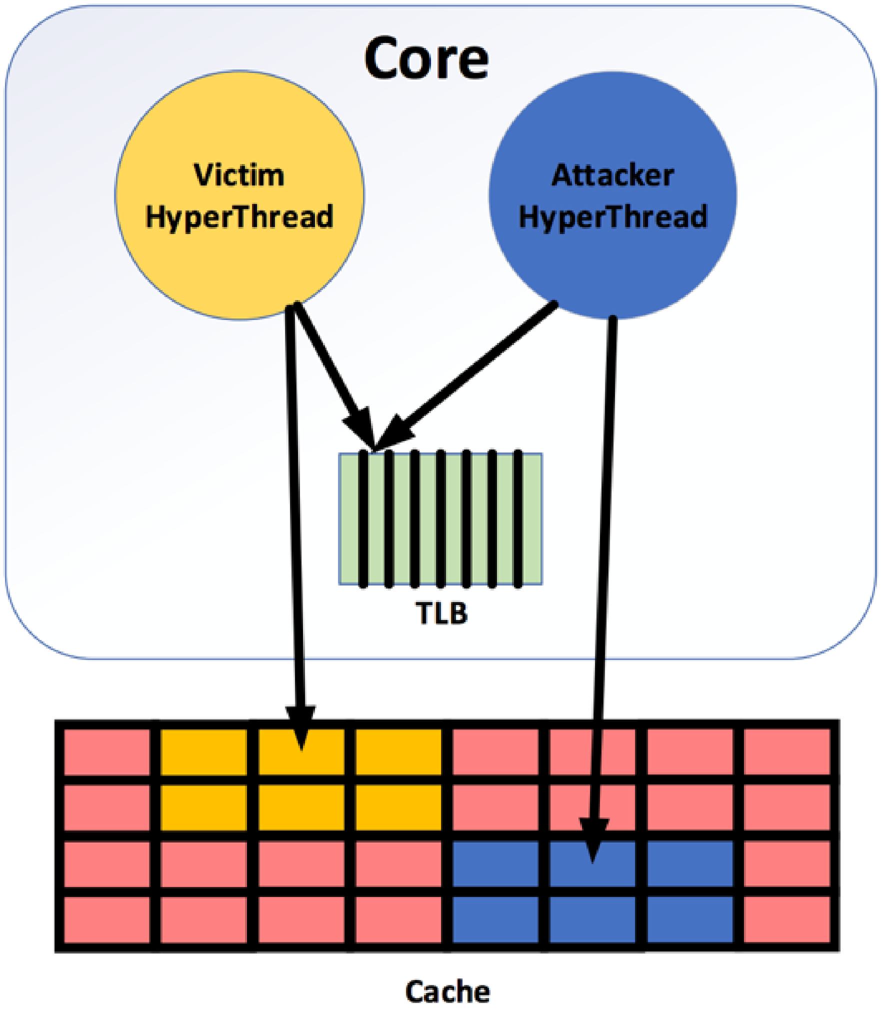


	Hyper-Threading		Multi-cores
	Shared or Partitioned	Replicated	Replicated
Register State		X	X
Return Stack		X	X
Reorder Buffer	X		X
Instruction TLB	X		X
Reservation Stations	X		X
Cache (L1, L2)	X		X
Data TLB	X		X
Execution Units	X		X

TLB


CPU
logical
address
p
d
page
number
frame
number
TLB hit
physical
address
f
d
TLB
p
TLB miss
f
page table
physical
memory

TLBleed:
TLB as shared state?



Very complicated

Many things unknown

We have L1**i**TLB, L1**d**TLB, L2**s**TLB

How are they structured (ways, sets)?

How are they filled?

⇒ Reverse engineering!

But are they suitable?

Many things unknown

We have L1**i**TLB, L1**d**TLB, L2**s**TLB

How are they structured (ways, sets)?

How are they filled?

⇒ Reverse engineering!

TLB

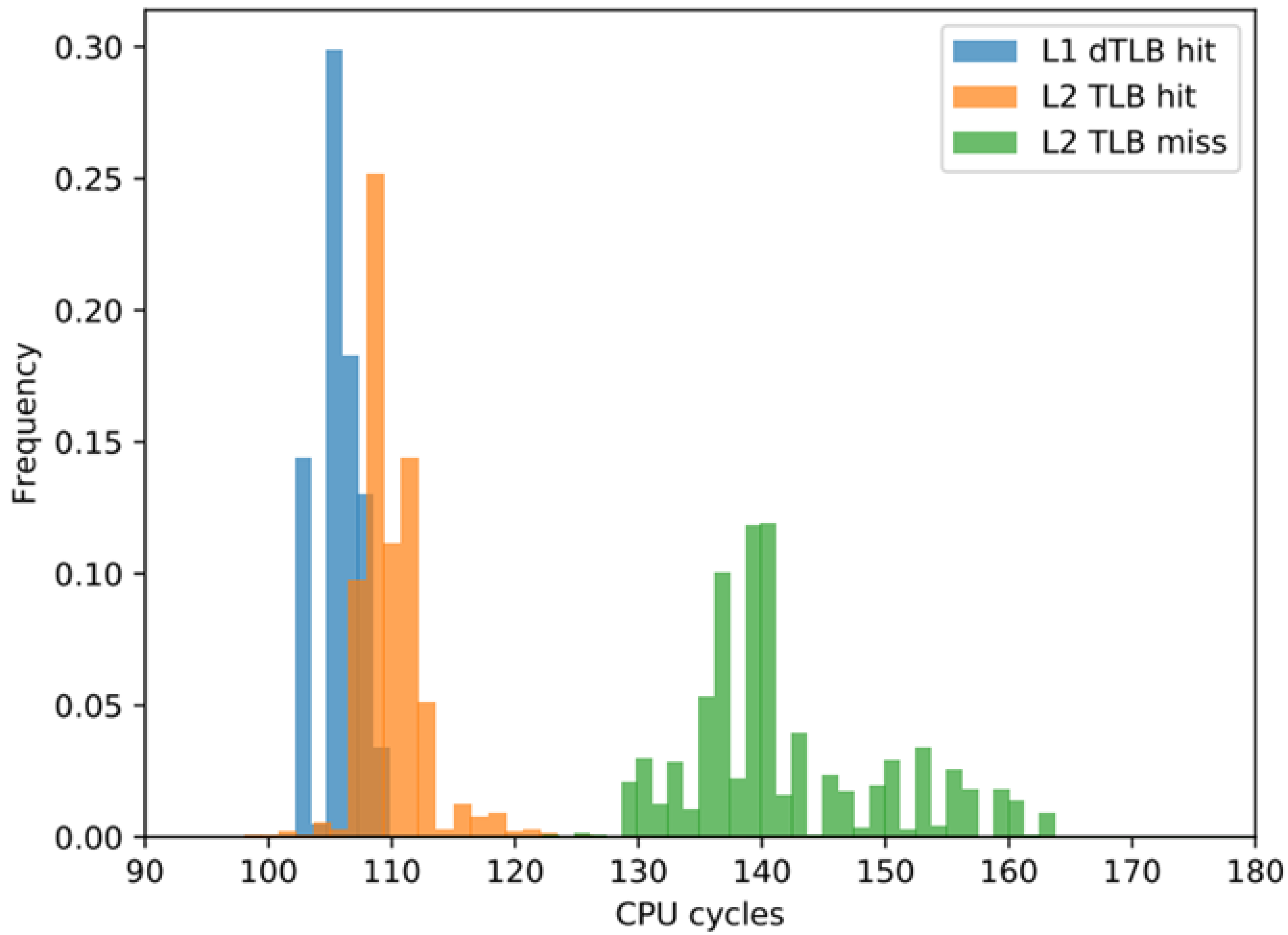
L1iTLB ⇒ not shared

L1dTLB ⇒ shared

L2sTLB ⇒ shared

Can we use latency as side channel?

Can we use latency as side channel?



Let’s do it

EdDSA ECC key multiplication

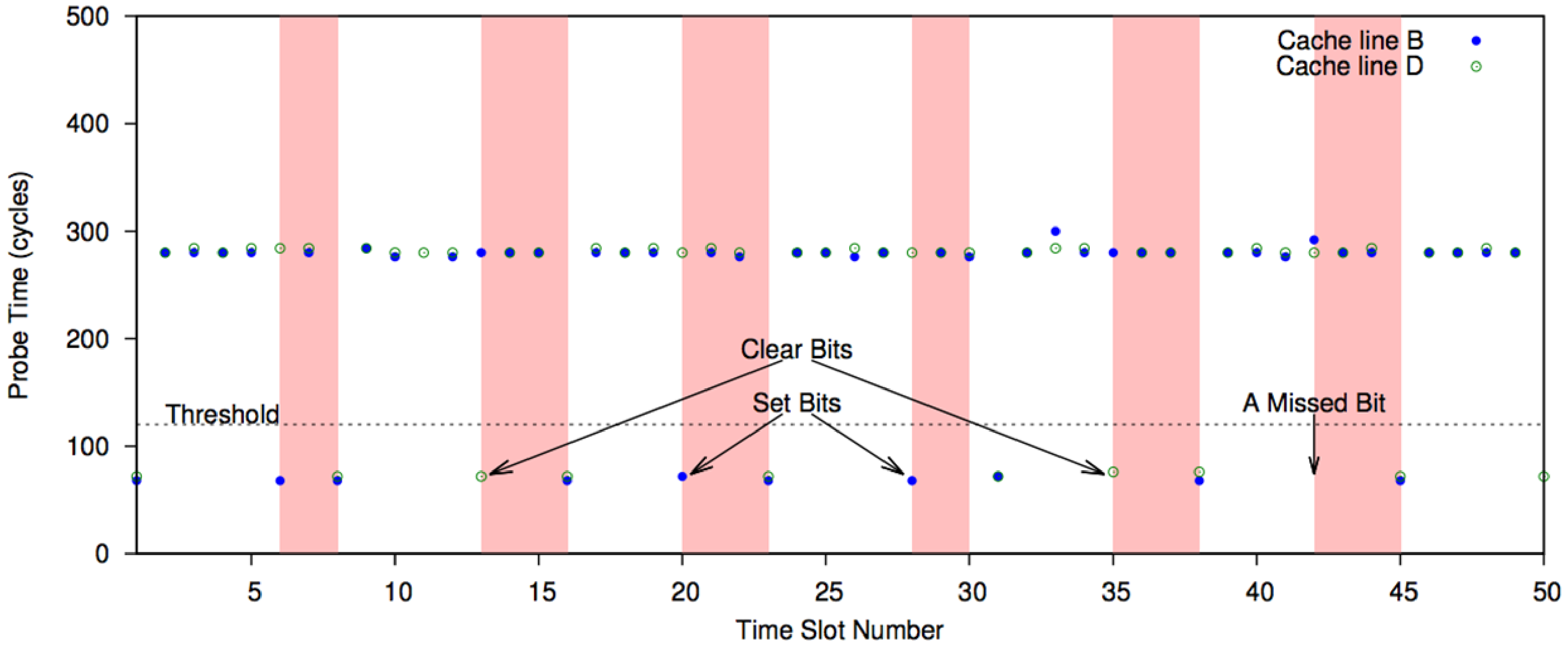
- Scalar is secret and ADD only happens if there's a 1

- But: we can not use code information! Only data..!

```
void _gcry_mpi_ec_mul_point (mpi_point_t result,
 gcry_mpi_t scalar, mpi_point_t point,
 mpi_ec_t ctx)
{
 ...
 for (j=nbits-1; j >= 0; j--) {
  _gcry_mpi_ec_dup_point (result, result, ctx);
  if (mpi_test_bit (scalar, j))
   _gcry_mpi_ec_add_points(result,result,point,ctx);
 }
 ...
}
```

Remember Flush+Reload

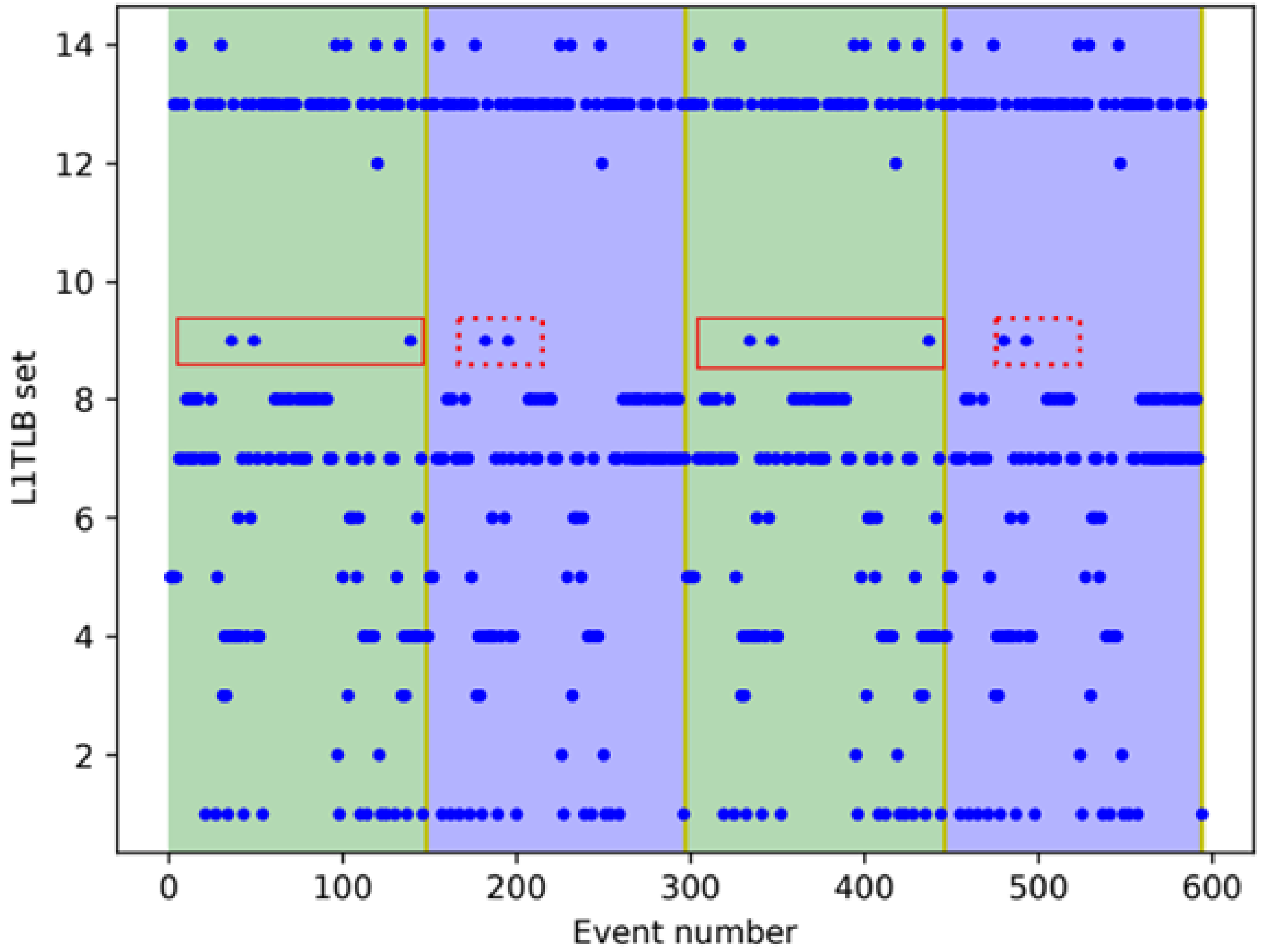
Traditional attack relies on *spatial separation*

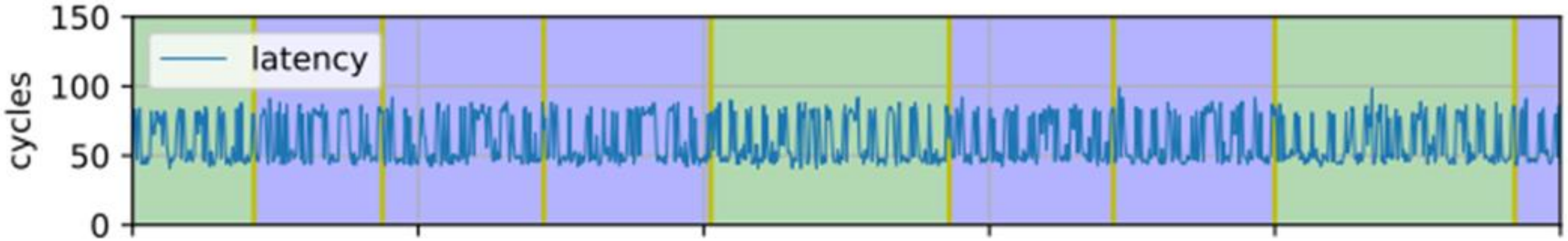


Let’s try this for the TLB

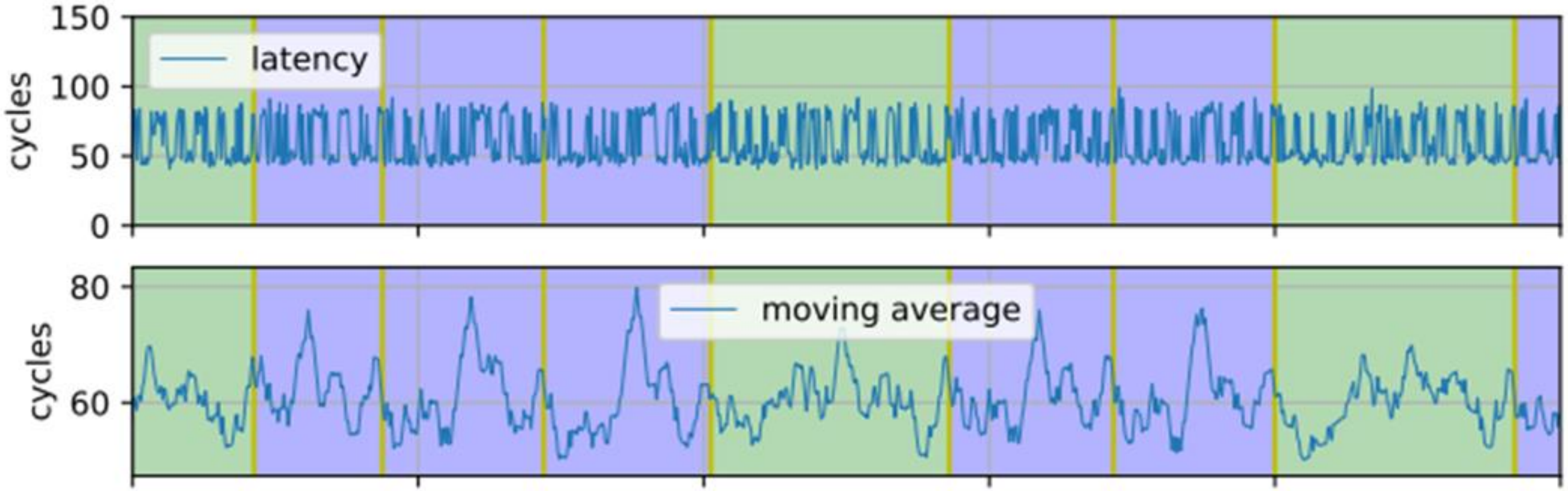
Let’s find the spatial L1 DTLB separation

There isn’t any



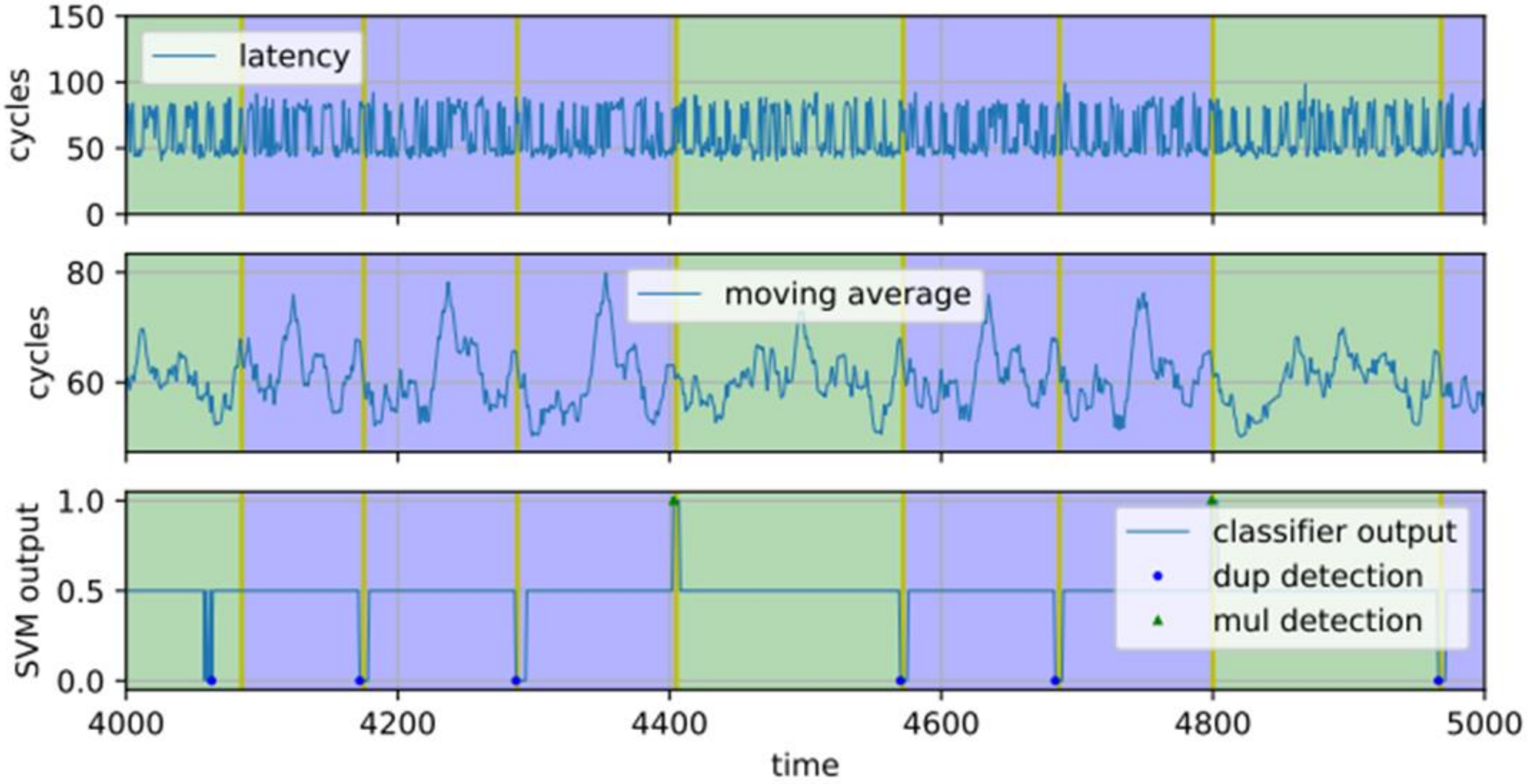


150
100
50
0
cycles
latency



150
100
50
0
cycles
latency
80
60
cycles
moving average

Monitor single TLB set for temporal information



Evaluation

Reliability

Microarchitecture	Trials	Success	Median BF
Skylake	500	0.998	$2^{1.6}$
Broadwell	500	0.982	$2^{3.0}$
Coffeelake	500	0.998	$2^{2.6}$
Total	1500	0.993	

With cache protection

Microarchitecture	Trials	Success	Median BF
Broadwell (CAT)	50	0.94	2^{12}
Broadwell	500	0.982	$2^{3.0}$

https://www.vusec.net/projects/tlbleed/

TLBs are caches too!

Data works as well as code

Temporal attacks work as well as spatial

Reconsider defenses

https://vusec.net

Sharing

is not caring

VUSec

https://vusec.net

Conclusion

Still no terrace...

Conclusion

We *suck* at bounty programs

Summary

We can launch Rowhammer attacks from

- CPU ➔ Javascript on x86, native on ARM
- GPU (!) ➔ Javascript on *anything*
- Remote devices (!)

We can target PCs, Clouds, Mobile, servers, …

ECC is not enough

[Use Emacs, not vi]

https://vusec.net

Summary

Systems full of active components accessing memory
GPU, MMU, co-processors, devices, … ➔ large attack surface

Also, tremendous amount of shared state
Caches, TLB, BPU state, power, … ➔ large new attack surface

[No really, Emacs]

Rethink Systems Security

Software security defenses

black hat
USA 2016

[Aug 4, 12:00] **Microsoft**: “*Thanks to our mitigation improvements, since releasing Edge one year ago, there have been no zero day exploits targeting Edge*”

Rethink Systems Security

Software security defenses

[Aug 4, 12:00] **Microsoft**: “*Thanks to our mitigation improvements, since releasing Edge one year ago, there have been no zero day exploits targeting Edge*”

[Aug 4, 17:00] **VUSec**: “*Dedup Est Machina: exploit the latest Microsoft Edge with all the defenses up, even in absence of software/configuration bugs*”

Rethink Systems Security

Formally verified systems

Microsoft Research
@MSFTResearch

Follow

Feel better. Hacker-proof code has been confirmed. quantamagazine.org/20160920-forma ... via @KSHartnett

Formally verified systems

Microsoft Research
@MSFTResearch

Follow

Feel better. Hacker-proof code has been confirmed. quantamagazine.org/20160920-forma … via @KSHartnett

[Aug 10] **VUSec**: "*Flip Feng Shui: Reliable exploitation of bug-free software systems*"

Conclusion

We find vulnerabilities because we are looking

Once found, however basic, a vulnerability quickly expands to cover “everything”

VUSec

https://vusec.net

[Emacs rules!]

The House is Built on Sand

VUSec
https://vusec.net