Reverse Engineering Hardware for Fun and Profit

Kaveh Razavi

About VUSec

~20 members, 3 faculty

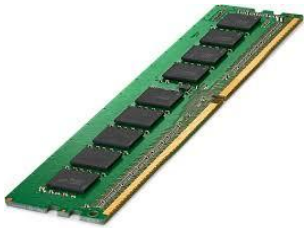
- Software protections
- Binary analysis
- Fuzzing
- Network security
- Hardware and OS security

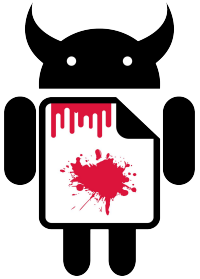
Assuming secure software,

what is still possible?

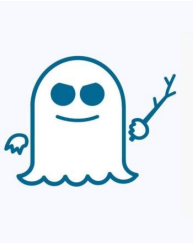
and what can we do about it?

General-purpose Hardware Attacks (2015-)



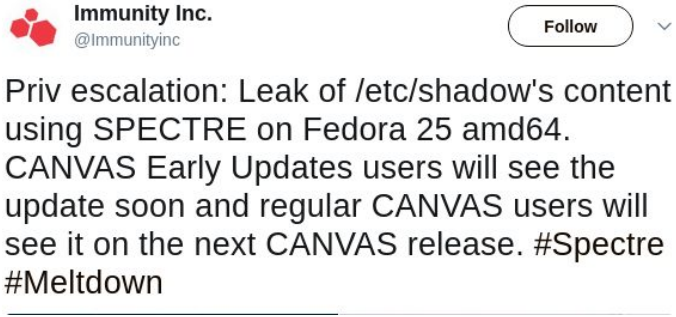


Drammer



Spectre/MDS

A government entity in a certain country: “can we please have the Drammer exploit?”

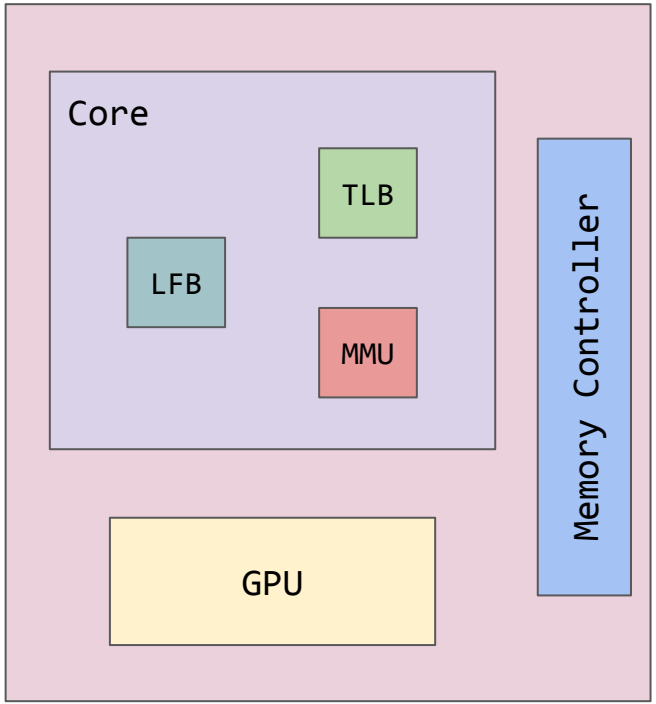


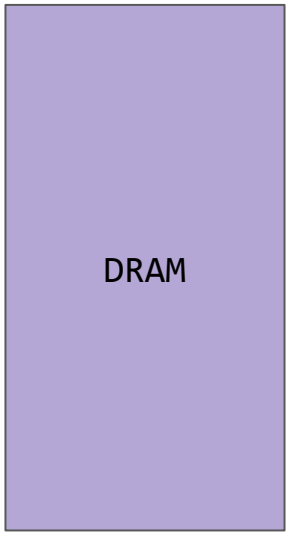
Understanding These Issues

Hardware is (almost) always closed.

Reverse Engineering!

Hardware Reverse Engineering at VUSec



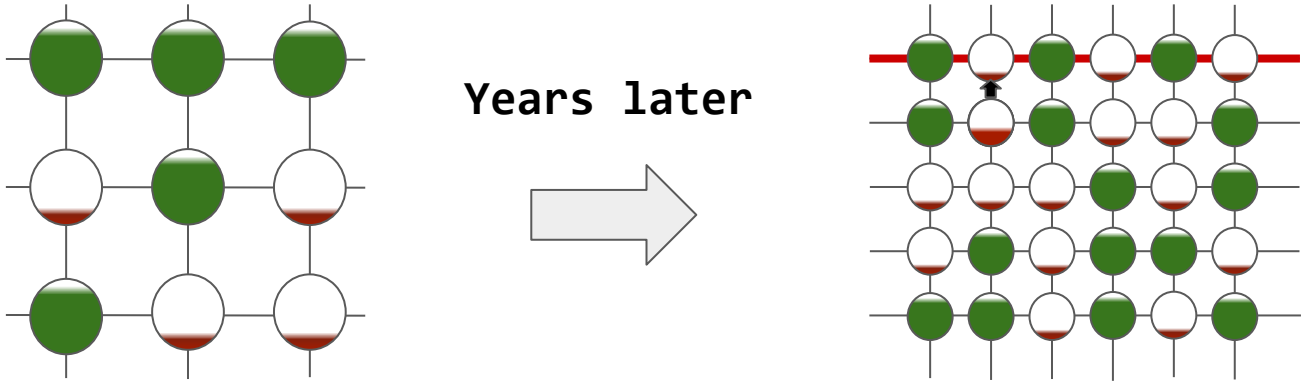


Rowhammer

Side channels

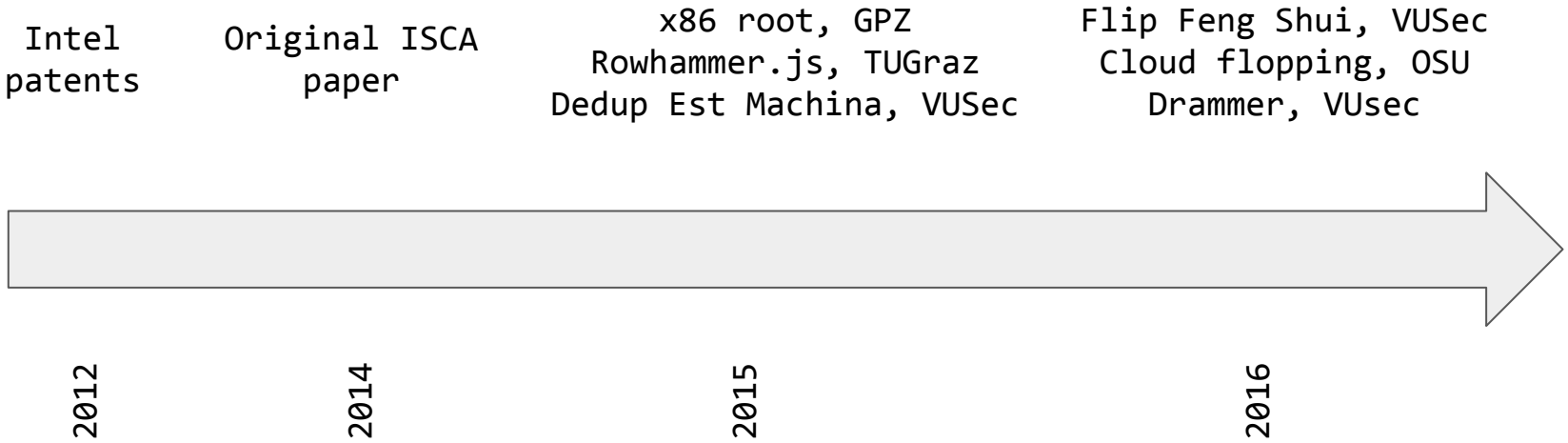
The Rowhammer Problem

We have reduced transistor without caring for reliability/security

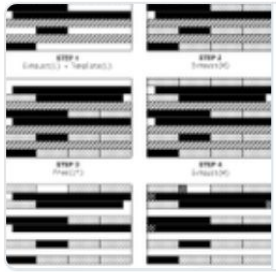


Rowhammer: affects 87% of deployed DDR3 memory, DDR4 as well.

Kim et al., “Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors,” ISCA’14



Browser exploitation: hard on x86, not possible on ARM



Drammer: Flip Feng Shui Goes Mobile - VUSec

Drammer is the first instance of mobile Rowhammer and comprehends a deterministic Android root exploit that does not rely on any software vulnerability.

vusec.net

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

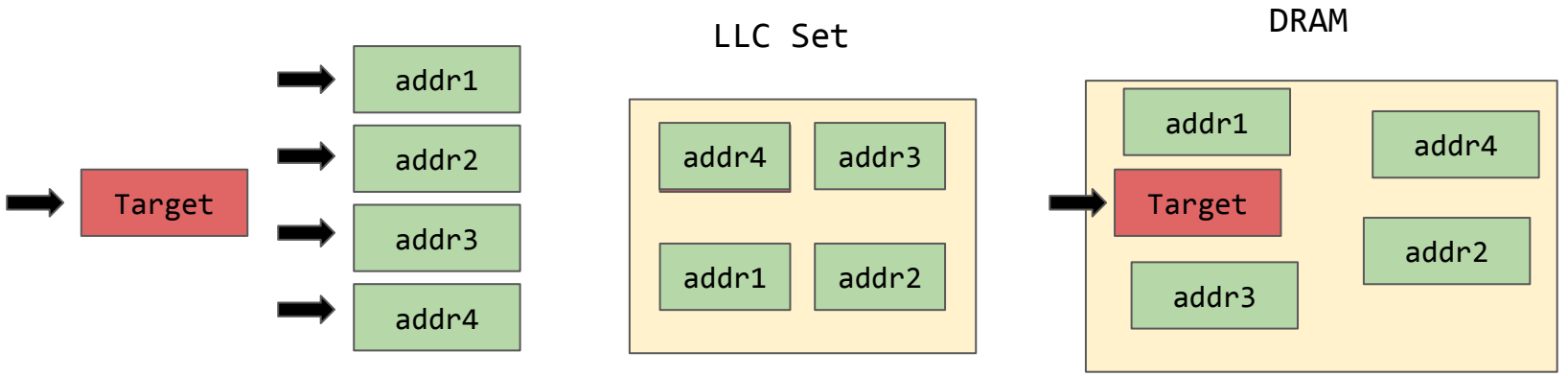
love to see it happen. :)

Native Rowhammering

x86: clflush ARM: DMA memory

From JavaScript: eviction sets

Eviction Sets and Rowhammer

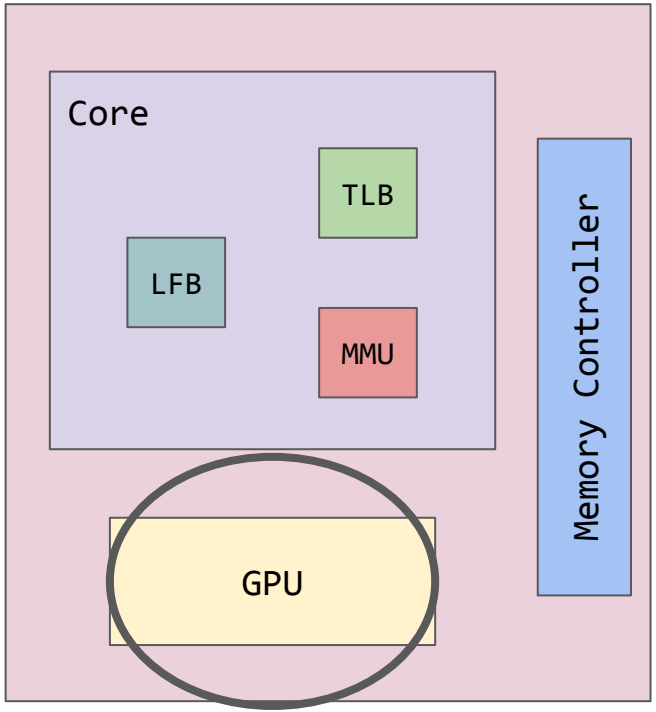


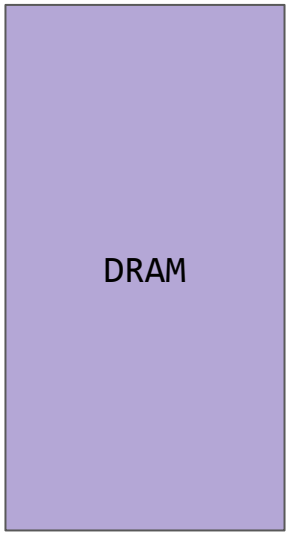
Building a Practical JS Rowhammer (on Mobile)

Eviction sets on x86 are slow → few flips

Eviction sets on ARM are **very** slow → no flips

Hardware Reverse Engineering at VUSec

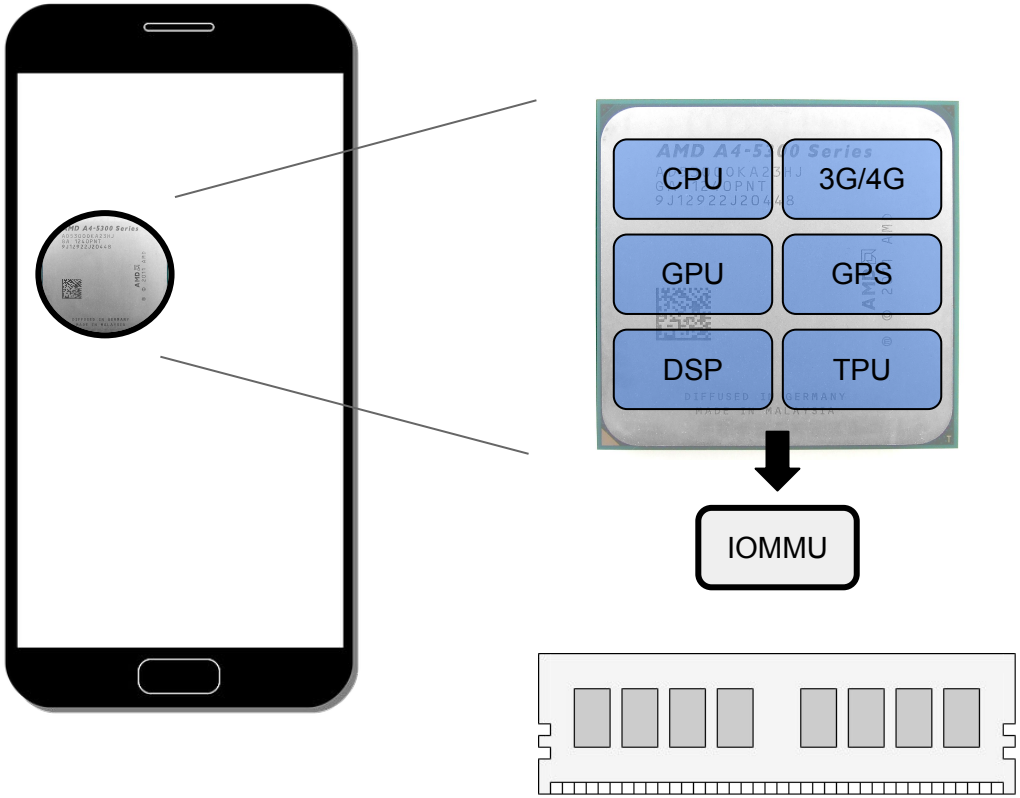




Rowhammer

Side channels

Inside a Phone’s SoC



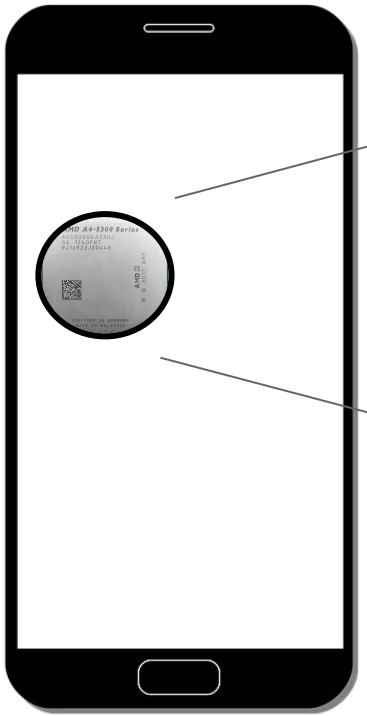
More co-processors

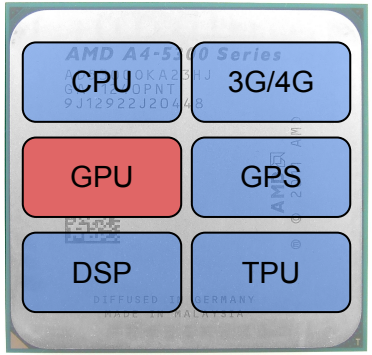
Greater attack surface

IOMMU

Fail to address uarch and Rowhammer attacks!

The GLitch Attack





Malicious website
or Ad

✚

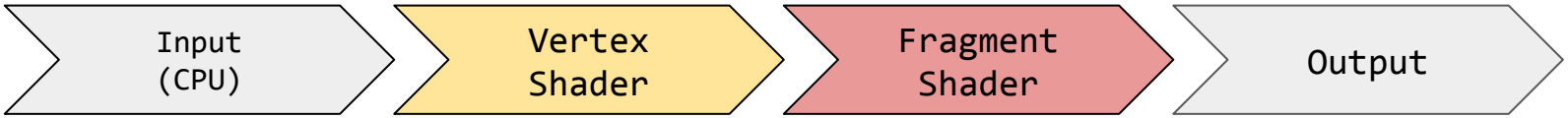
ARM (Android)

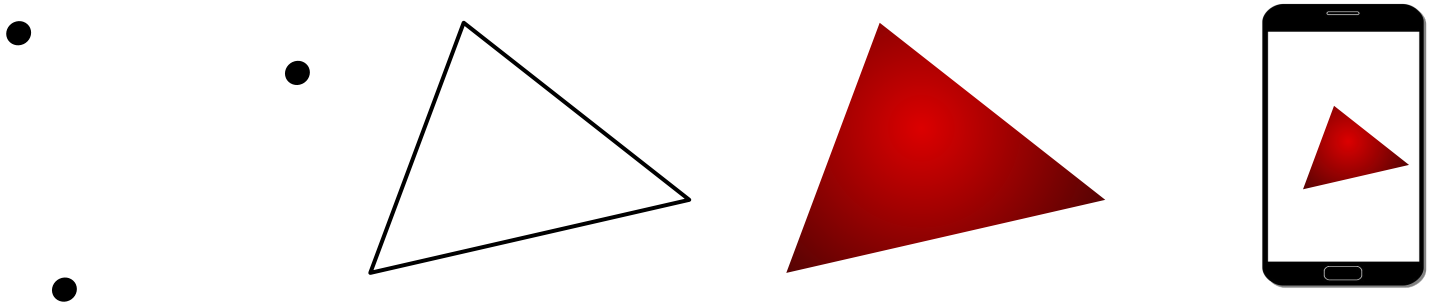
✚

Integrated GPU

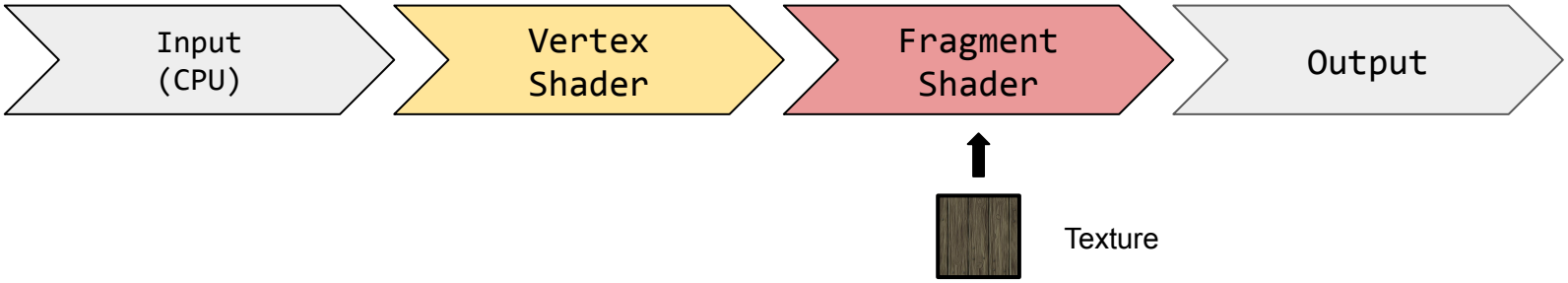
Frigo et al., “Grand Pwning Unit: Accelerating Microarchitectural Attacks with the GPU,” S&P’18

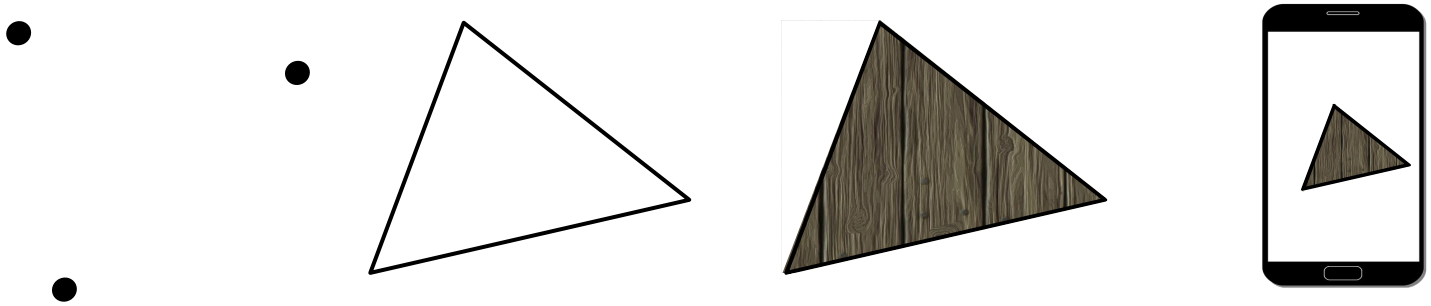
Understanding GPUs: The Rendering Pipeline



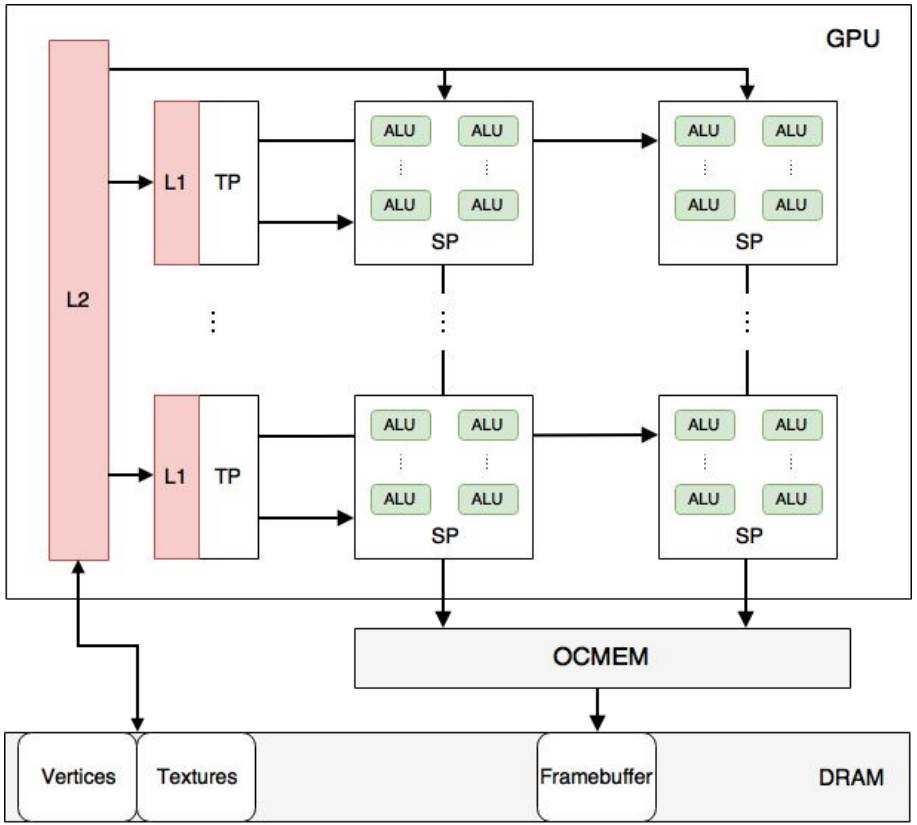


Understanding GPUs: The Rendering Pipeline

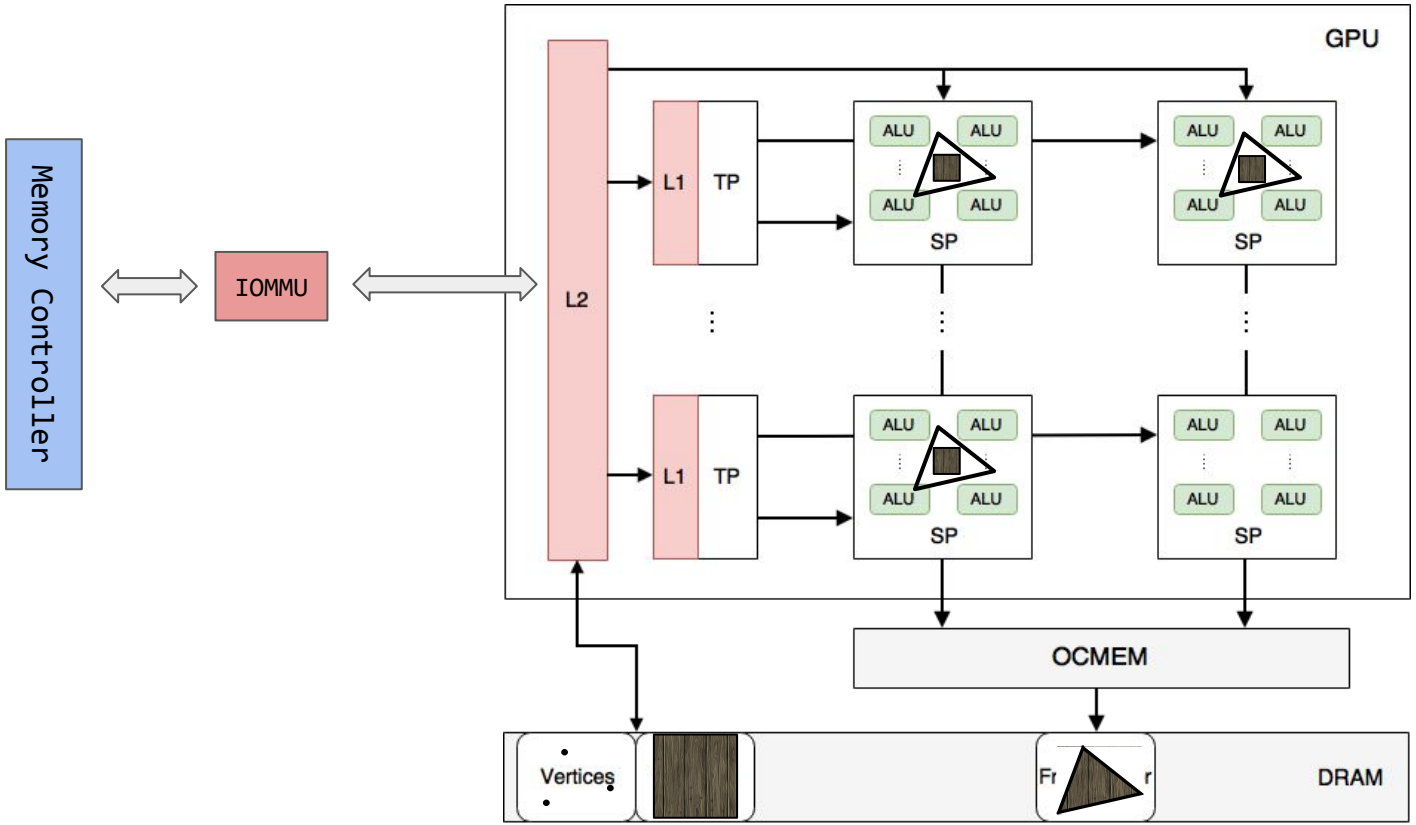




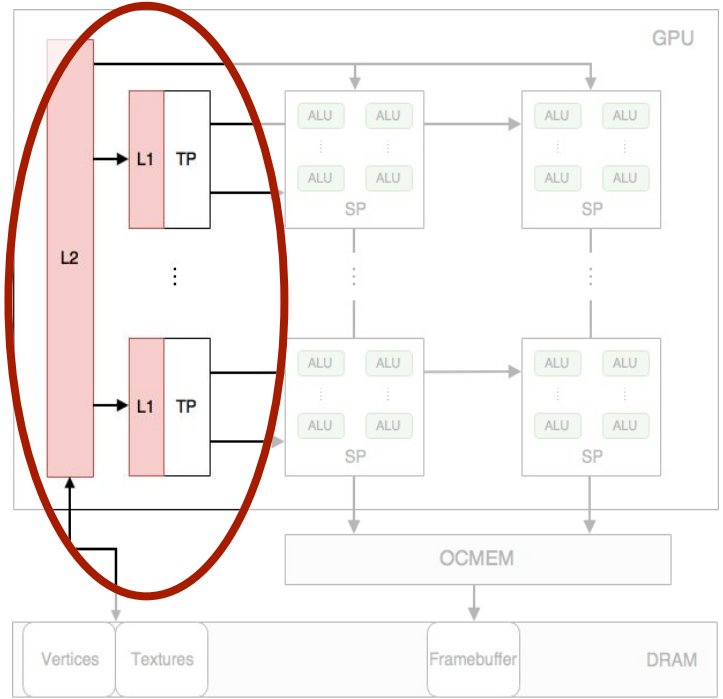
The Adreno 330 GPU Architecture



The Adreno 330 GPU Architecture



GPU Caches



1) Cacheline size
2) Size
3) Associativity
4) Replacement policy

Reverse engineering tool:

GPU performance counters:

L1_hit, L1_miss, L2_hit, L2_miss

Reverse Engineering Shader

```
1  #define MAX max // max offset
2  #define STRIDE stride // access stride
3
4  uniform sampler2D tex;
5
6  void main() {
7    vec4 val;
8    vec2 texCoord;
9    // external loop not required for (a)
10   for (int i=0; i<2; i++) {
11     for (int x=0; x < MAX; x += STRIDE) {
12       texCoord = offToPixel(x);
13       val += texture2D(tex, texCoord);
14     }
15   }
16   gl_Position = val;
17 }
```

Cacheline Sizes

STRIDE = 1

MAX = 1 .. N

L1_miss != 1

L2_miss != 1

L1 cacheline = 16 bytes

L2 cacheline = 64 bytes

```
1  #define MAX max // max offset
2  #define STRIDE stride // access stride
3
4  uniform sampler2D tex;
5
6  void main() {
7    vec4 val;
8    vec2 texCoord;
9    // external loop not required for (a)
10   for (int i=0; i<2; i++) {
11     for (int x=0; x < MAX; x += STRIDE) {
12       texCoord = offToPixel(x);
13       val += texture2D(tex, texCoord);
14     }
15   }
16   gl_Position = val;
17 }
```

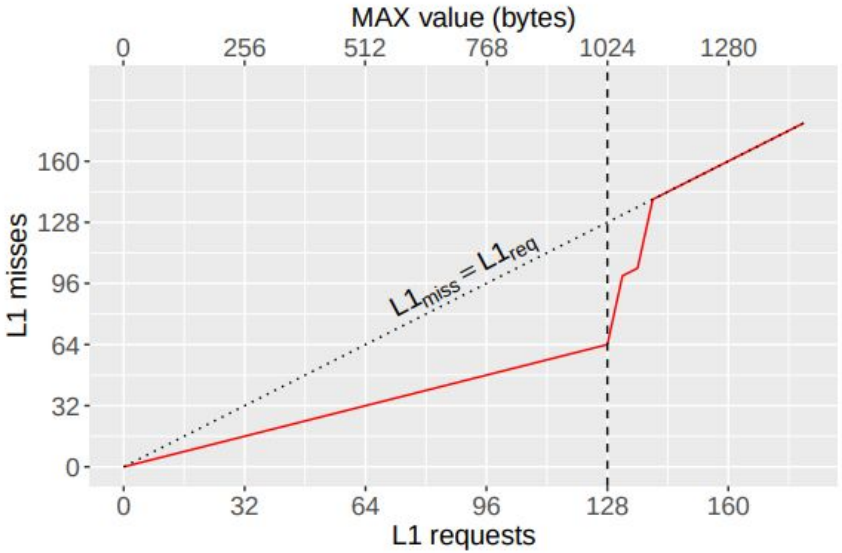
Cache Sizes

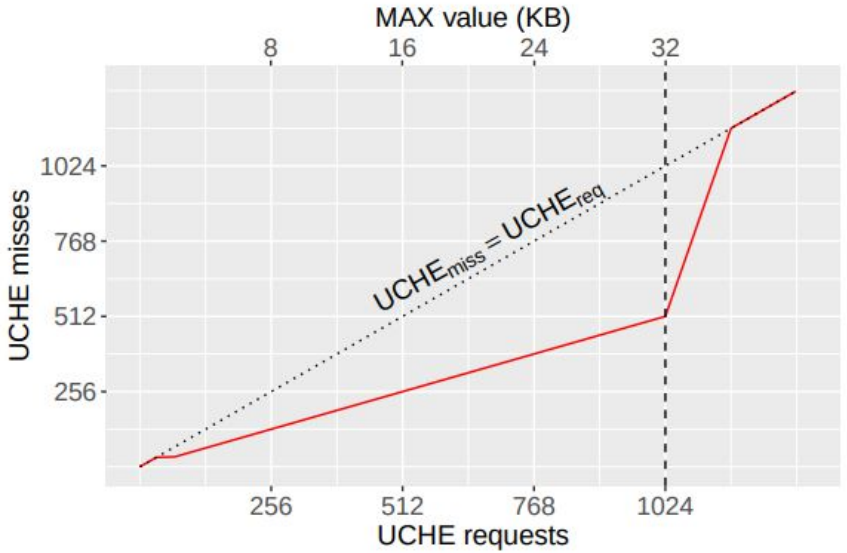
STRIDE = cacheline size

MAX = 1 .. N

```
1  #define MAX max // max offset
2  #define STRIDE stride // access stride
3
4  uniform sampler2D tex;
5
6  void main() {
7    vec4 val;
8    vec2 texCoord;
9    // external loop not required for (a)
10   for (int i=0; i<2; i++) {
11     for (int x=0; x < MAX; x += STRIDE) {
12       texCoord = offToPixel(x);
13       val += texture2D(tex, texCoord);
14     }
15   }
16   gl_Position = val;
17 }
```

Cache Sizes


MAX value (bytes)
0
256
512
768
1024
1280
160
128
96
64
32
0
L1 misses
$L1_{miss} = L1_{req}$
0
32
64
96
128
160
L1 requests


MAX value (KB)
8
16
24
32
1024
768
512
256
UCHE misses
$UCHE_{miss} = UCHE_{req}$
256
512
768
1024
UCHE requests

Replacement Policy

Typically LRU or some variant

addr1 + addr2 + addr3 + addr4 + addr1 + target

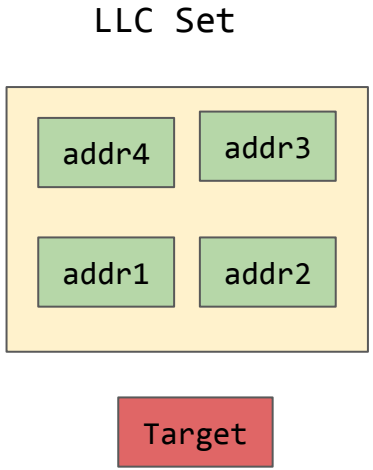
LLC set: addr3 + addr4 + addr1 + target

On GPU: FIFO

addr1 + addr2 + addr3 + addr4 + addr1 + target

LLC set: addr2 + addr3 + addr4 + target

Complicates Rowhammer - See paper.



Associativity

Goal: find addresses that map to the same cache set

Methodology: Oren et al. eviction set building algorithm

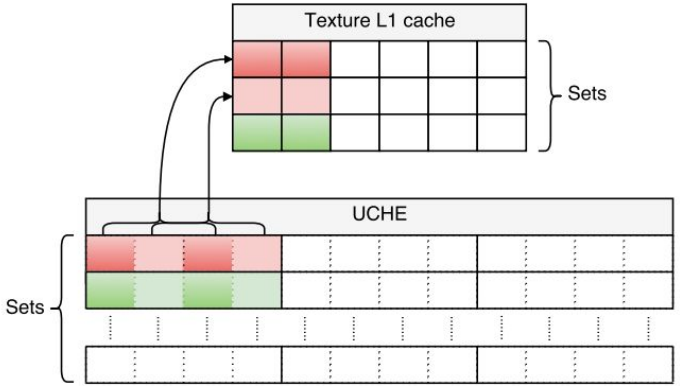
- Pick a pool virtual addresses
- Pick an observer address
- Reduce the pool to a minimal set that evicts observer

Associativity (and address mapping)

L1 = 16 way set associative

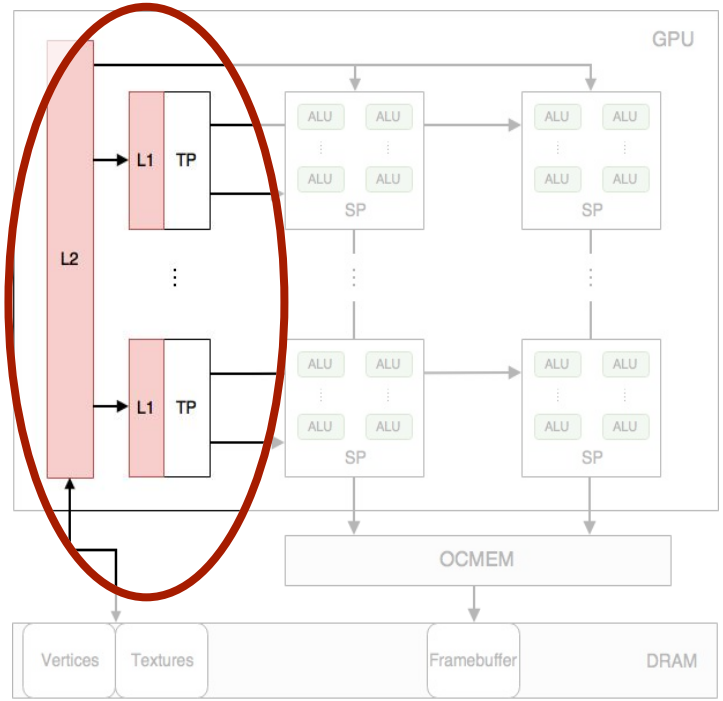
L2 = 8 way set associative

Non-inclusive L2



How to efficiently flush L1 with L2 lines?

GPU Caches

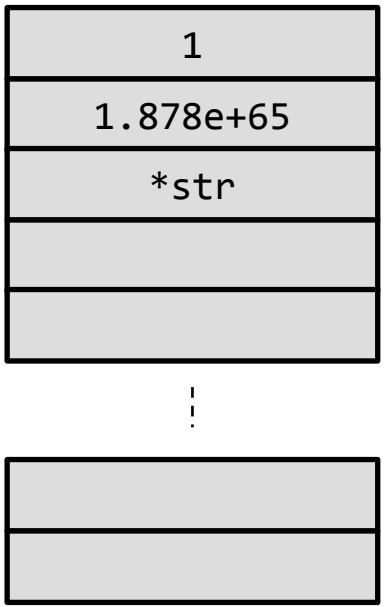


GPU caches	vs.	CPU caches (ARM)
+ Small and fast		- Large and slow
+ Deterministic		- Random

Putting All Together

GLitch

Exploitation: NaN-boxing



int

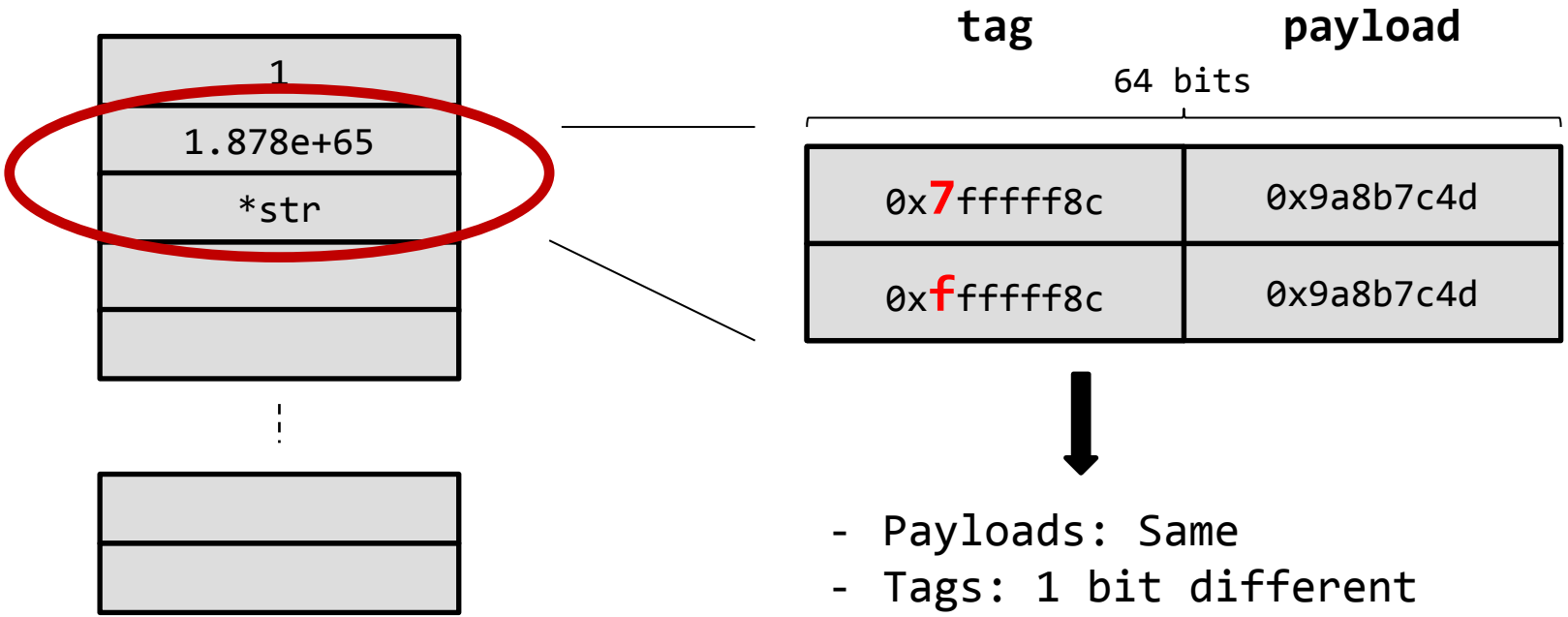
double

string

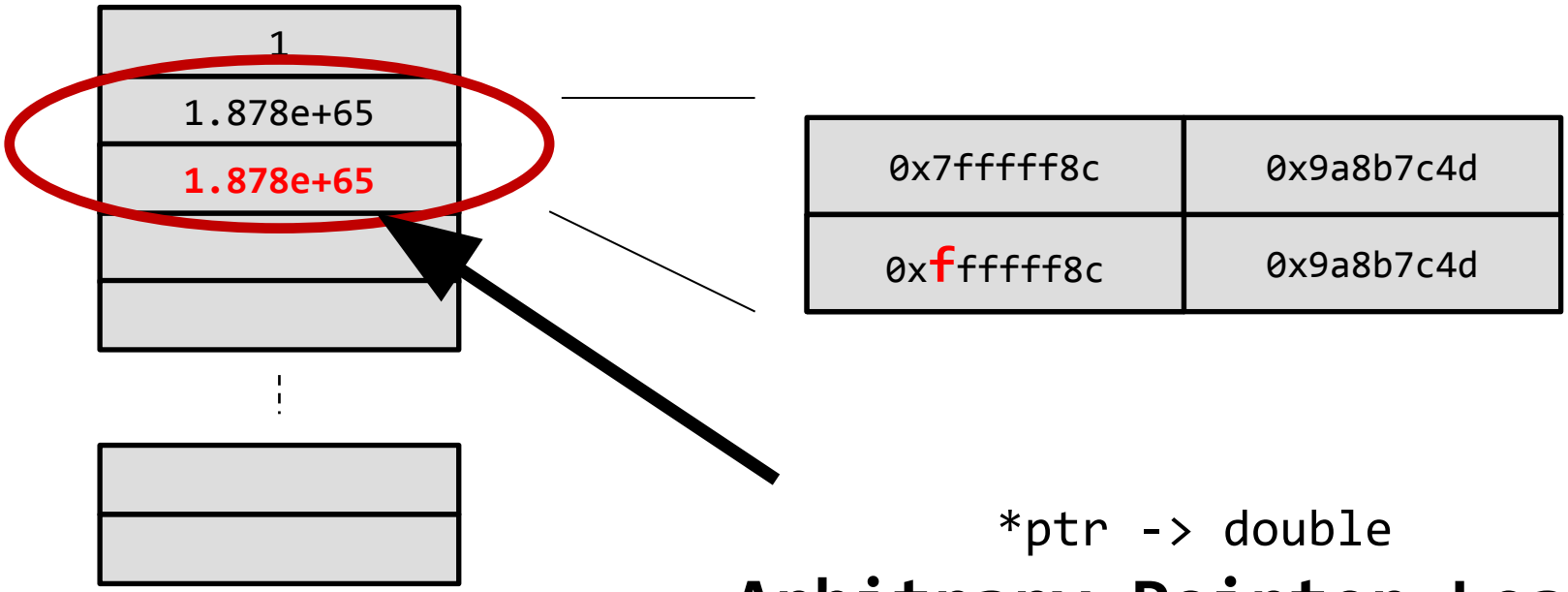
```
1 var arr = new Array(100);
2 arr[0] = 1 // int
3 arr[1] = 1.878e+65 // double
4 arr[3] = “Hello World” // string
...
```

NaN-boxing

Exploitation: NaN-boxing



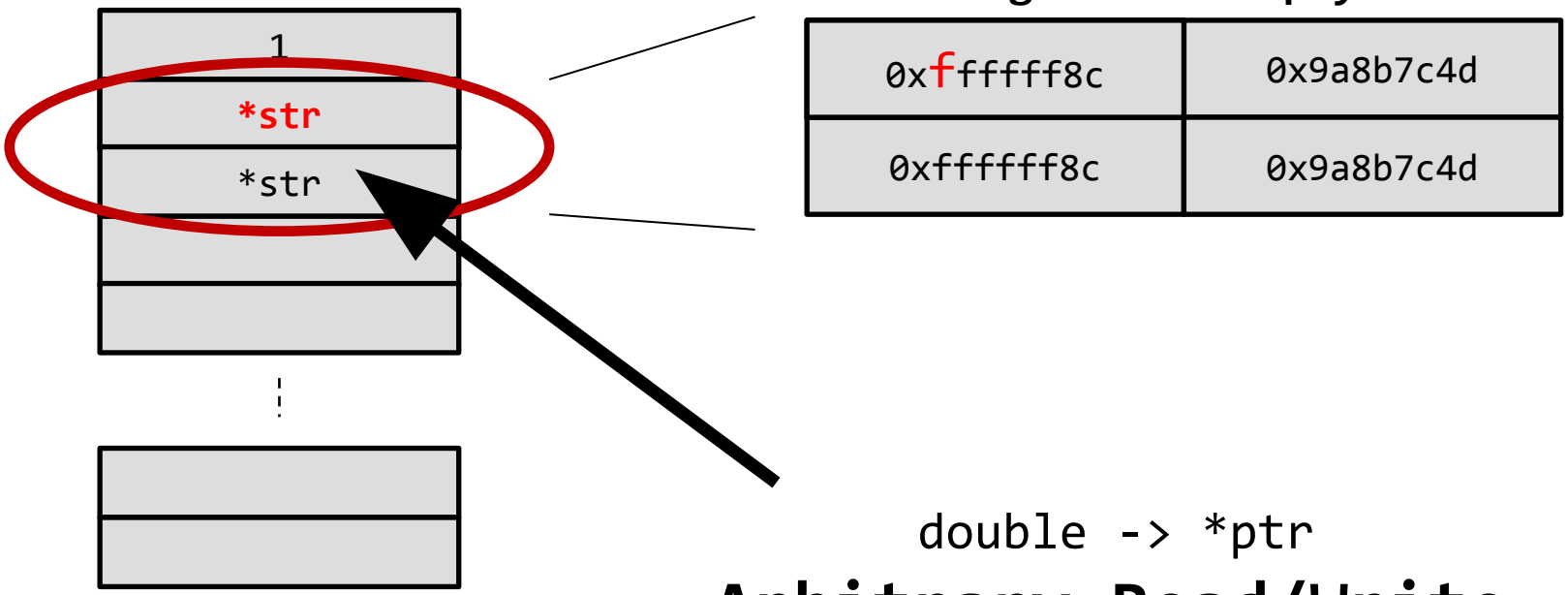
Exploitation: Type Flipping



*ptr -> double

Arbitrary Pointer Leak

Exploitation: Type Flipping



double -> *ptr

Arbitrary Read/Write

End to End Exploitation with GLitch

Attack	Compromise	Breaking ASLR
GLitch	116 s	27 s
Dedup Est Machina [7]	823 s	743 s
Rowhammer.js [20]	*	-
AnC [18]	-	114 s

2018 Code Blue young researcher award!

2019 best NL security master thesis award!

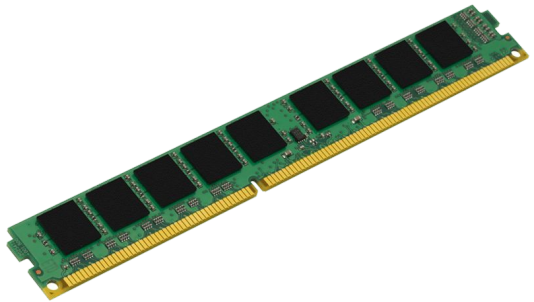
Meanwhile After Every Rowhammer Talk...

Question from audience:

Doesn’t ECC memory fix all of this?

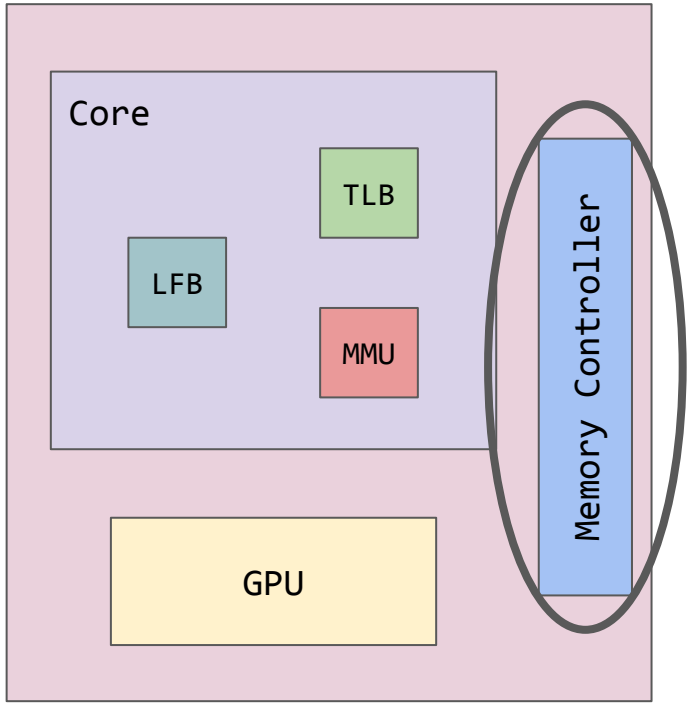
Error-correction Codes (SECDED)

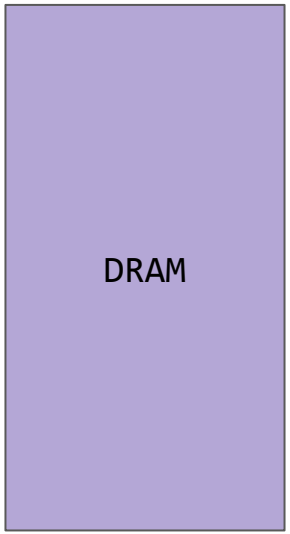
- Original paper demonstrated SECDED not to be enough
- … but exploitation turned out to be difficult
  - ECC implementation is closed (guarantees unknown)
  - 1 bit flips not visible, 2 bit flips crash the system



ECC DRAM as a practical secure defense.

Hardware Reverse Engineering at VUSec



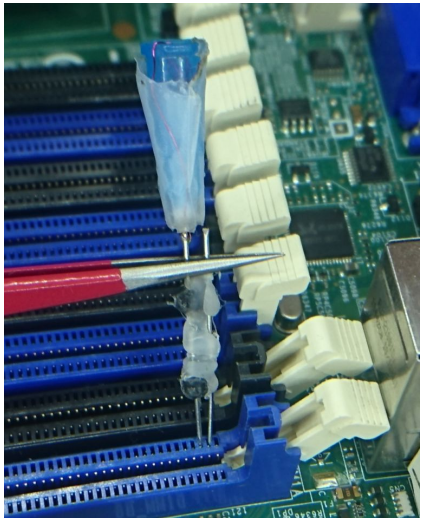


Rowhammer

Side channels

Recovering ECC Functions

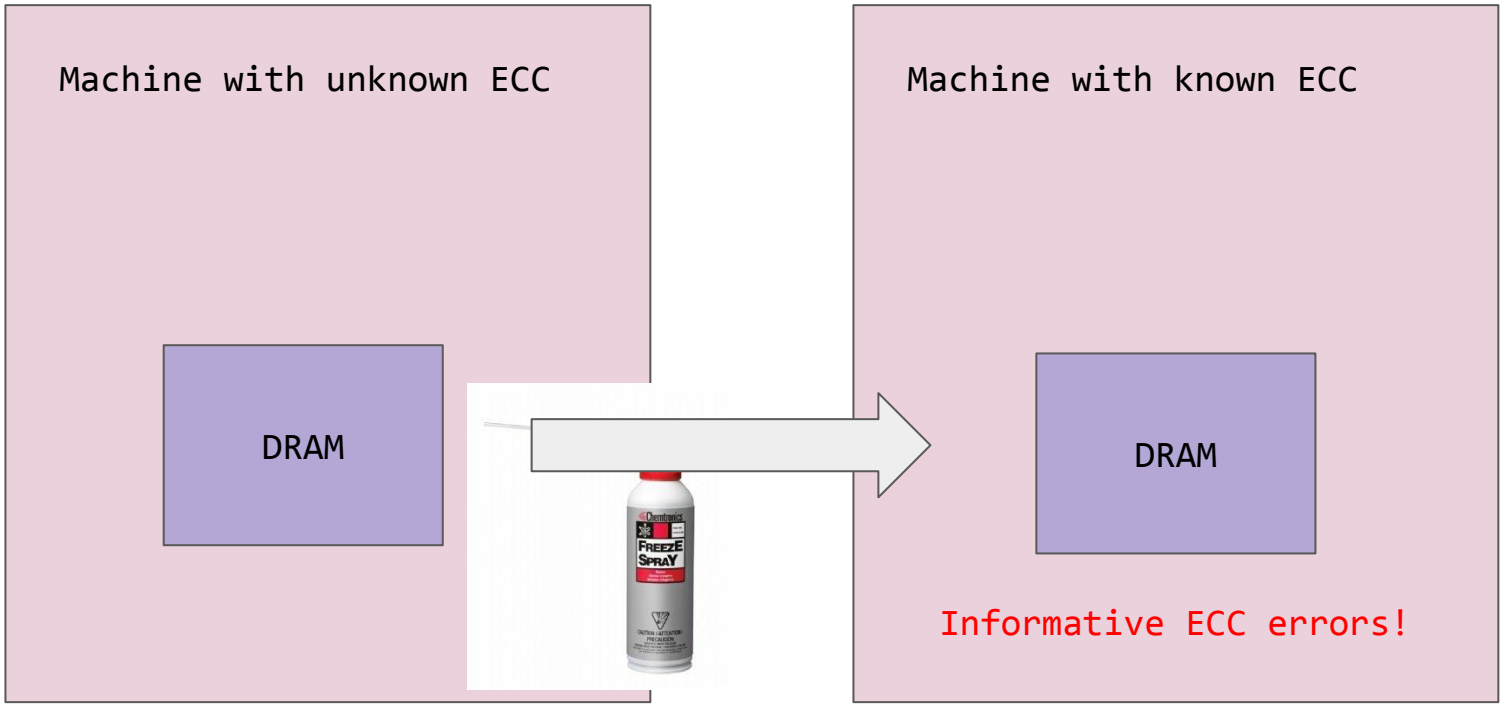
- Observing signals are not easy at 1Ghz+
  - Need custom interposer
  - Expensive logic analyzer
- Fault injection with syringe needles!
- Short-circuit data lines with Vss
  - High-to-low voltage flips
- ECC error reports allows for ECC function recovery



Cojocar et al., “Exploiting Correcting Codes: On the Effectiveness of ECC Memory Against Rowhammer Attacks,” S&P’19

Demo

Needle Flip Injection Is Too Painful...



Demo

Results

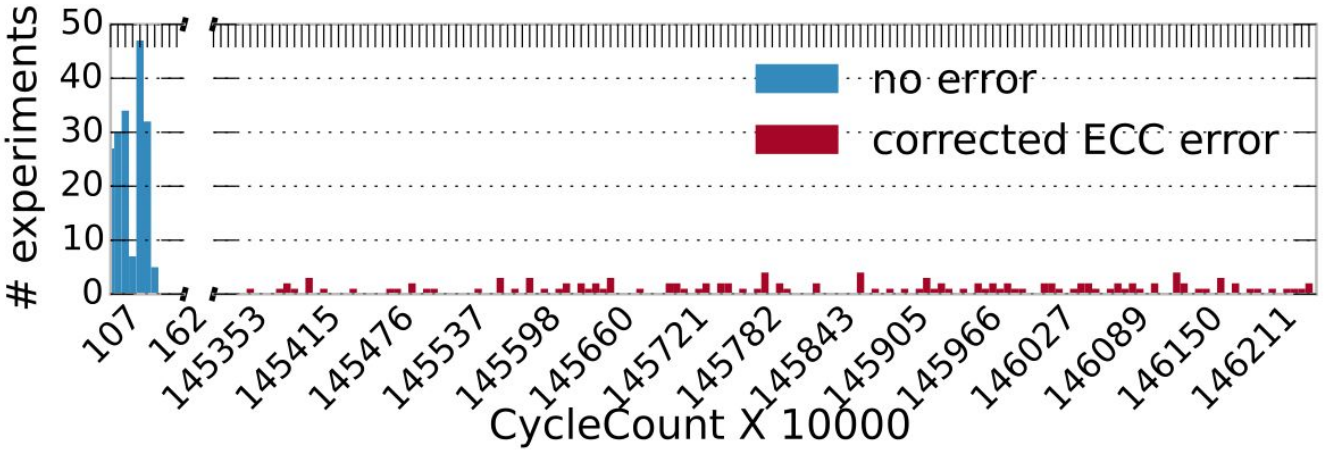
TABLE V: Error patterns that can circumvent ECC.

ID	Pattern	Config.	# flips	Flips location
AMD-1	$[\mathcal{P}_1]$	Ideal	3-BF-16	3 symbols, 1 in control bits
AMD-1	$[\mathcal{P}_2]$	Ideal	4-BF-16	Min. 2 symbols
Intel-1	$[\mathcal{P}_3]$	Ideal	4-BF-8	Min. 2 symbols
Intel-1	$[\mathcal{P}_4]$	Default	2-BF-8	Min. 2 symbols

TABLE VI: Percentages of rows with corruptions in an ECC DIMM.

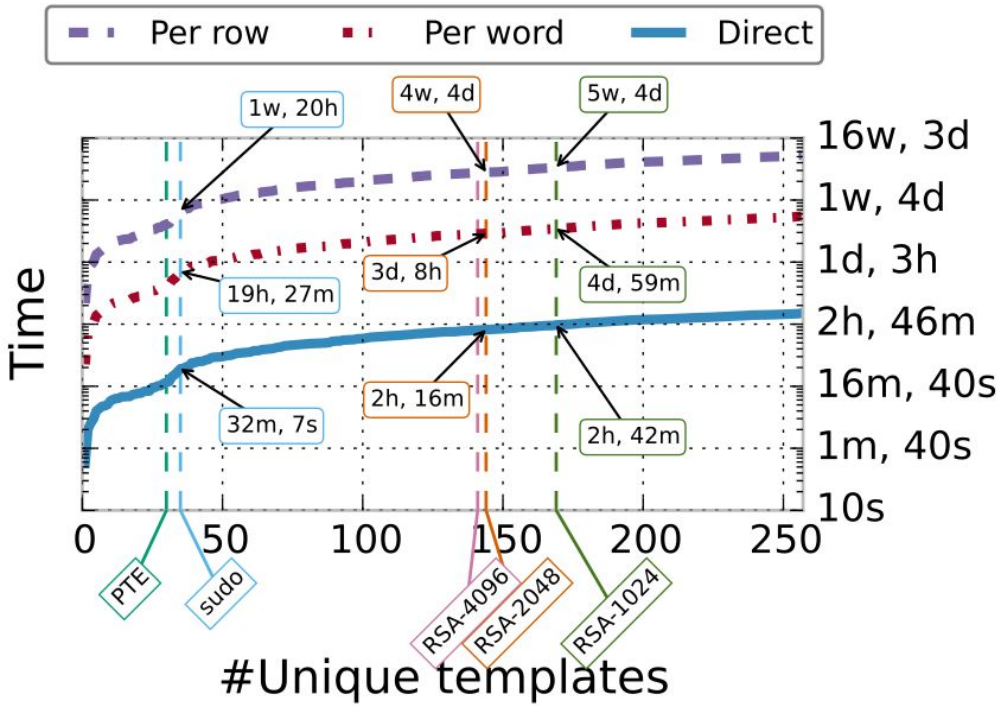
$[\mathcal{P}_1]$	$[\mathcal{P}_2]$	$[\mathcal{P}_3]$	$[\mathcal{P}_4]$
0.12%	0.12%	0.06%	0.60%

Avoiding Crashes



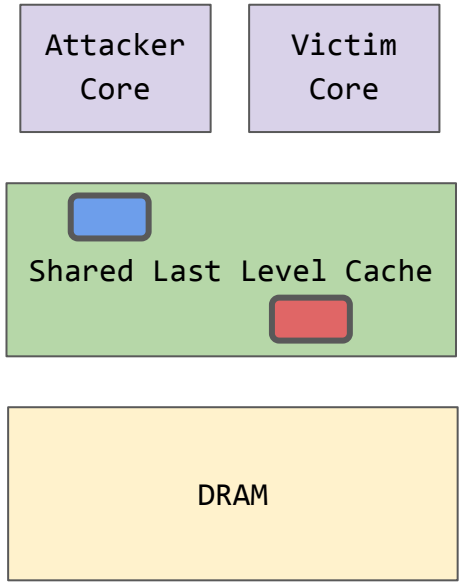
Detect single flips and merge them for silent corruptions.

ECC: Replicating Existing Attacks



Distinguished Practical Paper Award at S&P'19

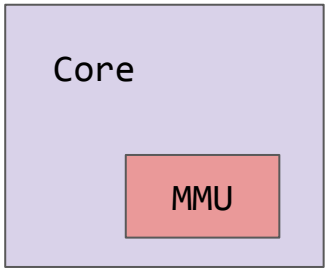
Traditional Cache Attacks




```
if (secret_key[i]) == 1)
{
     something();
}
else
{

     something_else();
}
```

Attacking CPU-internal Components

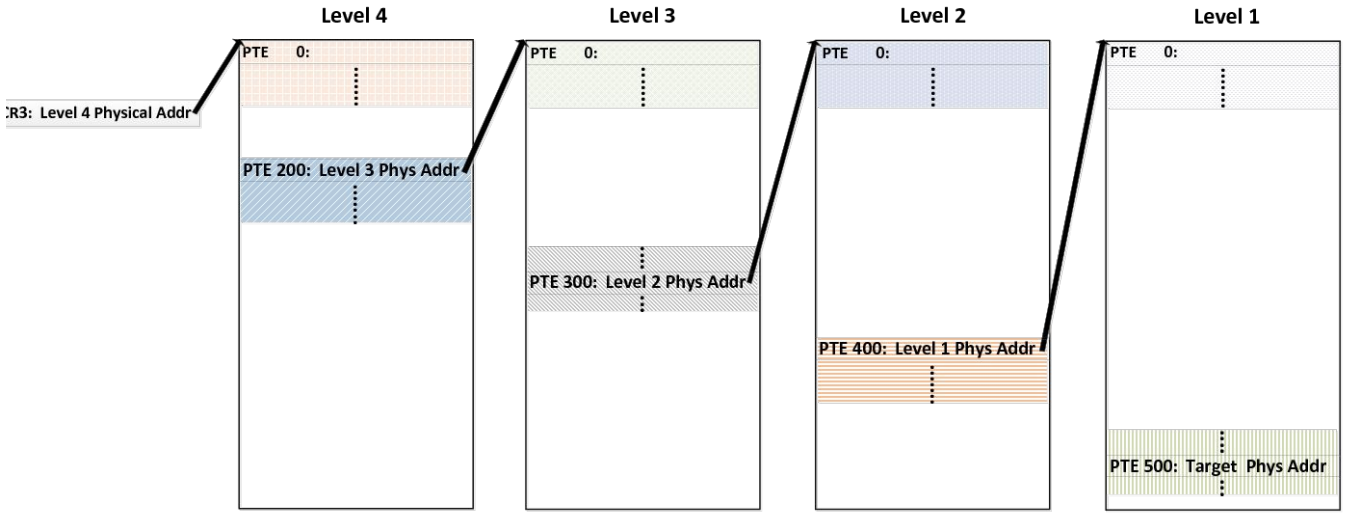


AnC

ASLR leak

2017

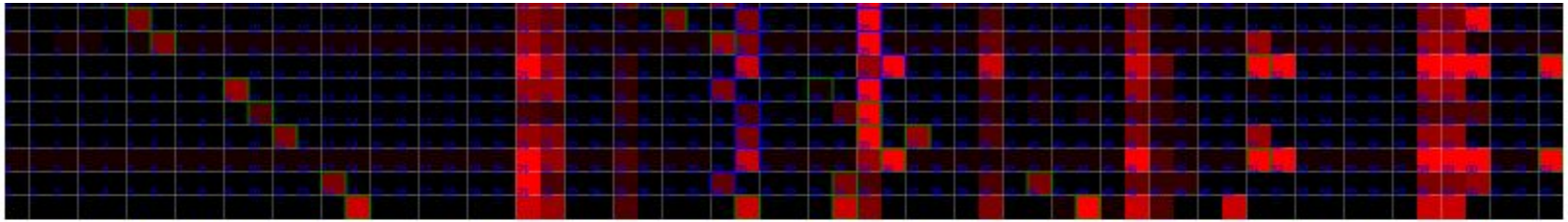
AnC: MMU Leaves a Trace in the CPU Caches



Secret: randomized virtual address

Gras/Razavi et al., “ASLR on the Line: Practical Cache Attacks on the MMU,” NDSS’17

AnC from JavaScript

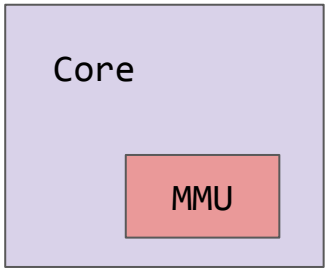


```
24.457 got level 4 - start slot 148, address 0x94000
24.993 got level 3 - start slot 295, address 0x24e94000
24.993 estimated remaining entropy 6 slot solutions: -1,-1,295,148
68.737 got level 4 - start slot 0, address 0x0
69.502 got level 3 - start slot 359, address 0x2ce00000
70.259 got level 2 - start slot 411, address 0x66ece00000
88.041 got level 1 - start slot 238, address 0x7766ece00000
88.041 estimated remaining entropy 0 slot solutions: 238,411,359,0
data: 0x7766ece00000, code slots: -1,-1,295,148, code: 0x7966e4e94000
```

Affected Architectures

CPU	Year
Intel Core i7-7500U (Kaby Lake) @ 2.70GHz	2016
Intel Core m3-6Y30 (Skylake) @ 0.90GHz	2015
Intel Xeon E3-1240 v5 (Skylake) @ 3.50GHz	2015
Intel Core i7-6700K (Skylake) @ 4.00GHz	2015
Intel Celeron N2840 (Silvermont) @ 2.16GHz	2014
Intel Core i7-4500U (Haswell) @ 1.80GHz	2013
Intel Core i7-3632QM (Ivy Bridge) @ 2.20GHz	2012
Intel Core i7-2620QM (Sandy Bridge) @ 2.00GHz	2011
Intel Core i5 M480 (Westmere) @ 2.67GHz	2010
Intel Core i7 920 (Nehalem) @ 2.67GHz	2008
AMD Ryzen 7 1700 8-Core (Zen) @ 3.3GHz	2017
AMD Ryzen 5 1600X 6-Core (Zen) @ 3.6GHz	2017
AMD FX-8350 8-Core (Piledriver) @ 4.0GHz	2012
AMD FX-8320 8-Core (Piledriver) @ 3.5GHz	2012
AMD FX-8120 8-Core (Bulldozer) @ 3.4GHz	2011
AMD Athlon II 640 X4 (K10) @ 3.0GHz	2010
AMD E-350 (Bobcat) @ 1.6GHz	2010
AMD Phenom 9550 4-Core (K10) @ 2.2GHz	2008
Rockchip RK3399 (ARM Cortex A72) @ 2.0GHz	2017
Rockchip RK3399 (ARM Cortex A53) @ 1.4GHz	2017
Allwinner A64 (ARM Cortex A53) @ 1.2GHz	2016
Samsung Exynos 5800 (ARM Cortex A15) @ 2.1GHz	2014
Nvidia Tegra K1 CD580M-A1 (ARM Cortex A15) @ 2.3GHz	2014
Nvidia Tegra K1 CD570M-A1 (ARM Cortex A15; LPAE) @ 2.1GHz	2014
Samsung Exynos 5800 (ARM Cortex A7) @ 1.3GHz	2014
Samsung Exynos 5250 (ARM Cortex A15) @ 1.7GHz	2012

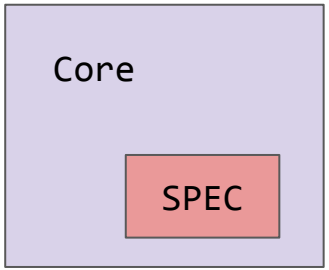
Attacking CPU-internal Components



AnC

ASLR leak

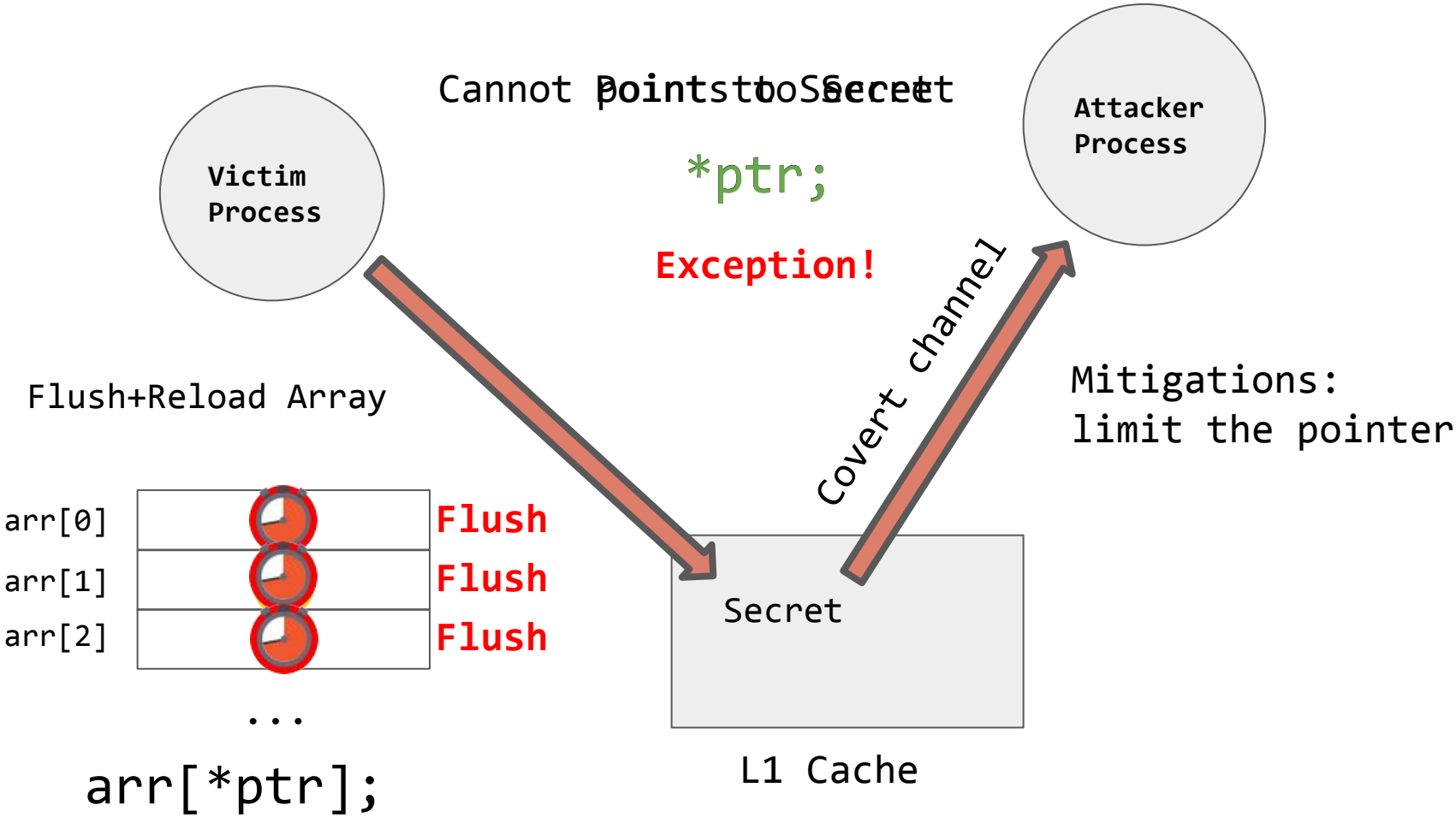
2017



Spectre

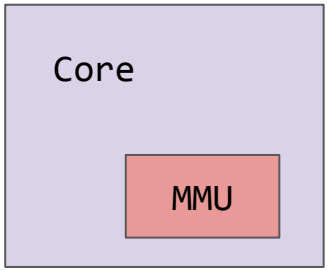
Arbitrary leak

2018


Victim Process
Cannot Point to Secret
*ptr;
Exception!
Attacker Process
Covert channel
Mitigations:
limit the pointer
Flush+Reload Array
arr[0]
arr[1]
arr[2]
Flush
Flush
Flush
...
arr[*ptr];
Secret
L1 Cache

Are these spot mitigations enough?

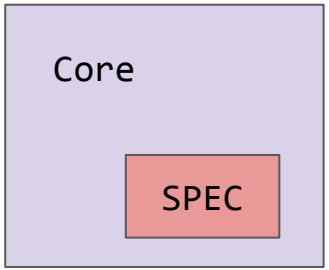
Attacking CPU-internal Components



AnC

ASLR leak

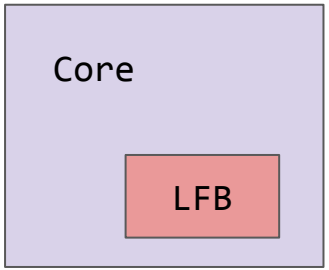
2017



Spectre

Arbitrary leak

2018



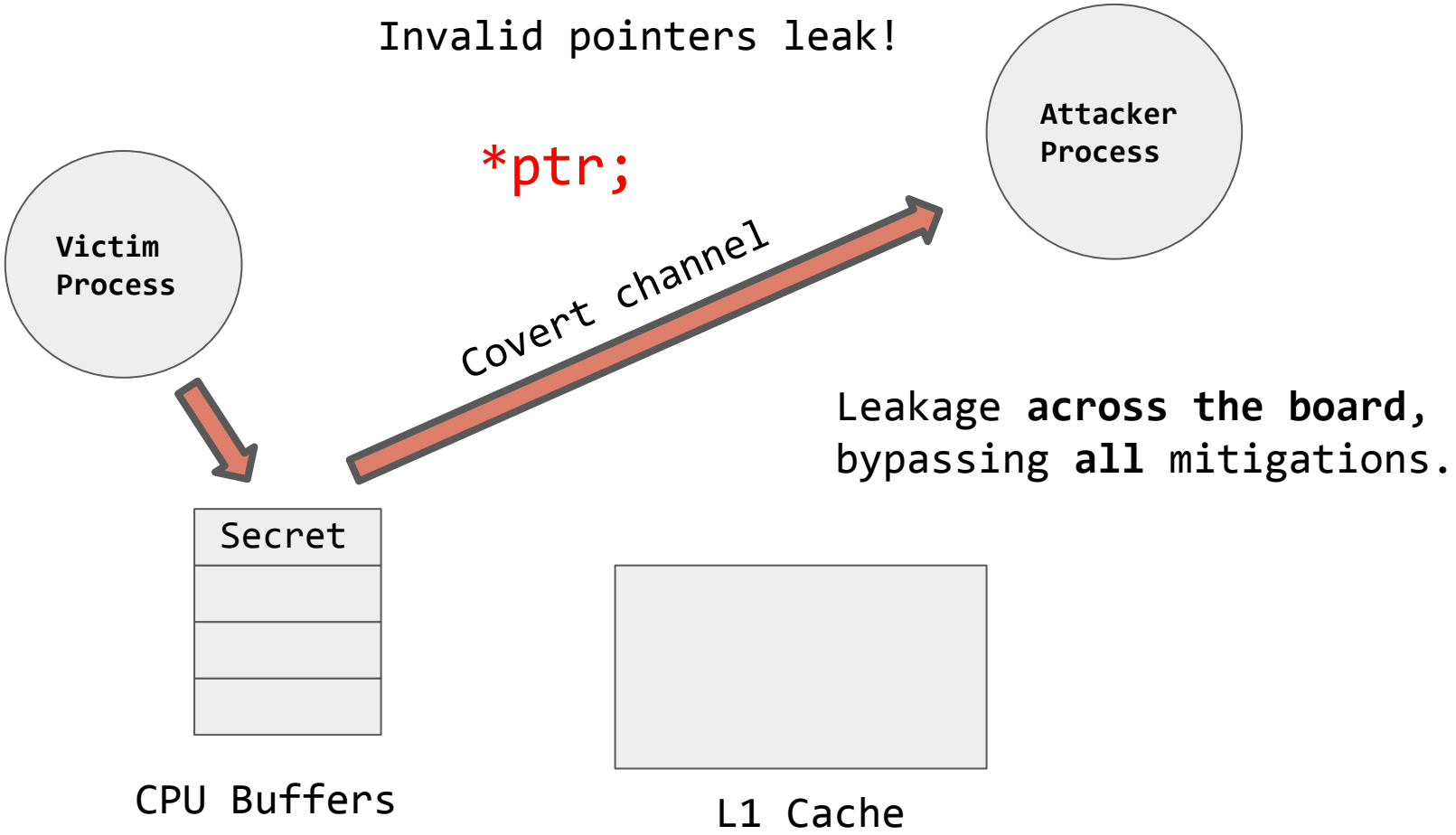
RIDL

Arbitrary leak

2019

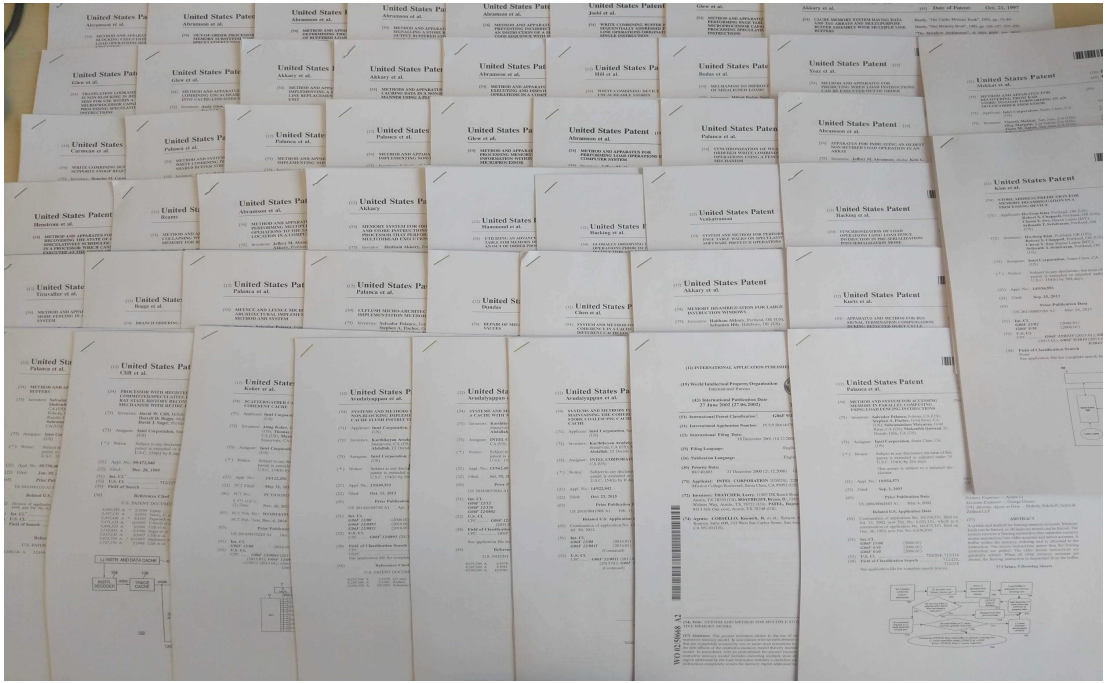
Van Schaik et al., “RIDL: Rogue Inflight Data Load,” S&P’19

Invalid pointers leak!



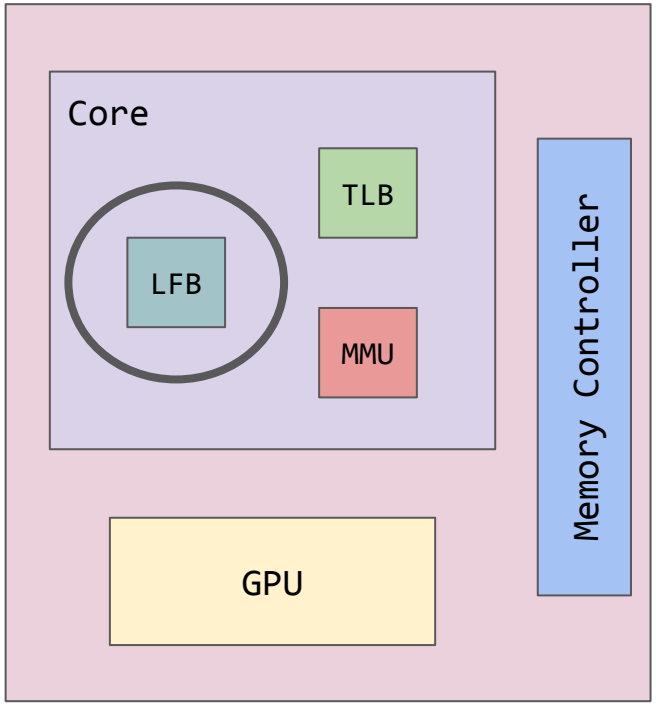
Leakage **across the board**, bypassing **all** mitigations.

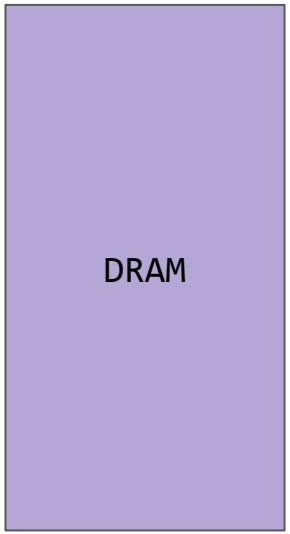
What CPU Buffers?



\+ Performance counters and leakage

Hardware Reverse Engineering at VUSec

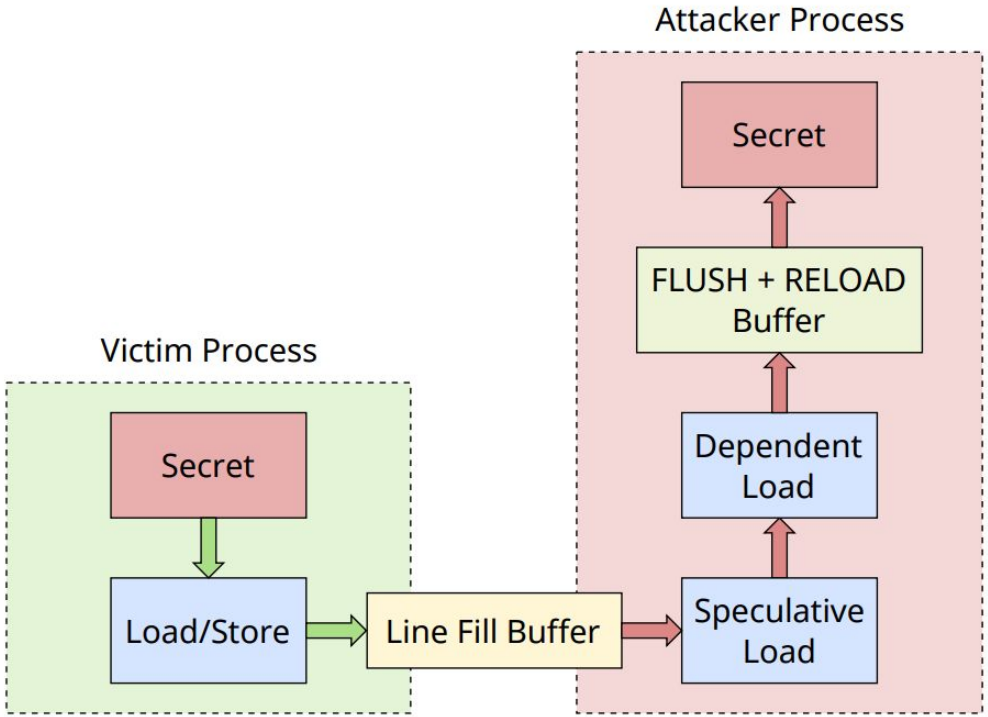




Rowhammer

Side channels

Theory: Leaking through LFB



RIDLing

① **FLUSH**

```
for (i = 0; i < 256; ++i) {
    _mm_clflush(probe + i * 4096);
}
```

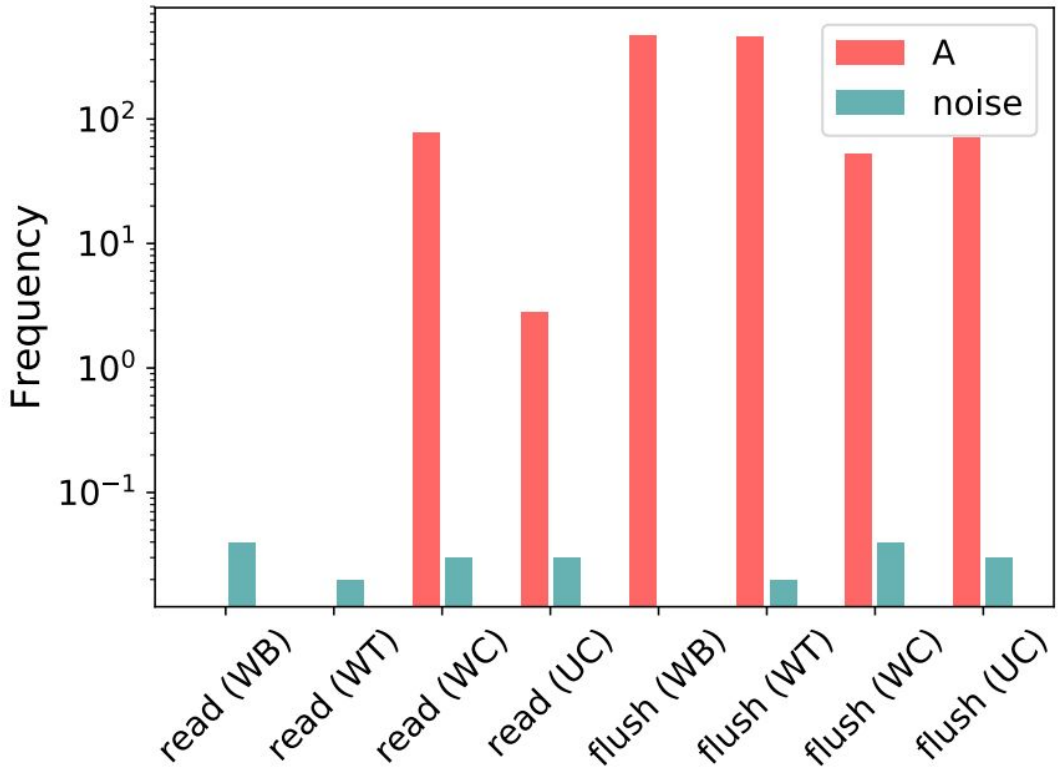
② **RIDL**

```
if (_xbegin() == _XBEGIN_STARTED) {
  char byte = *(volatile char *)NULL;
  char *p = probe + byte * 4096;
  *(volatile char *)p;
  _xend();
}
```

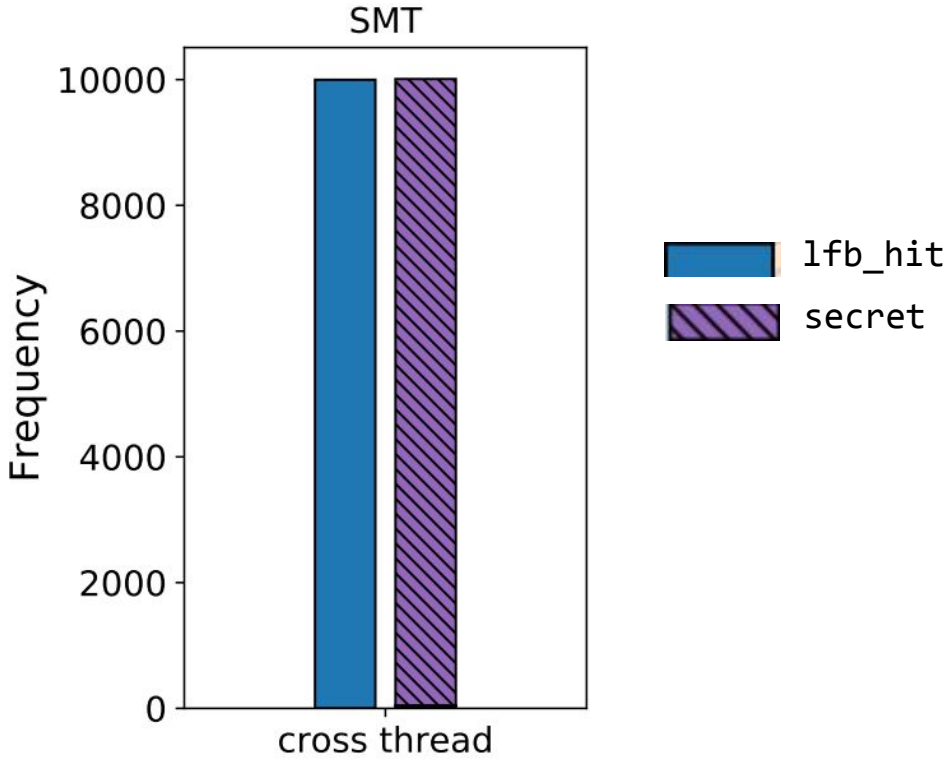
③ **RELOAD**

```
for (i = 0; i < 256; ++i) {
    t0 = __rdtsc();
    *(volatile char *)(probe + i * 4096);
    dt = __rdtsc() - t0;
}
```

Not Through CPU Caches

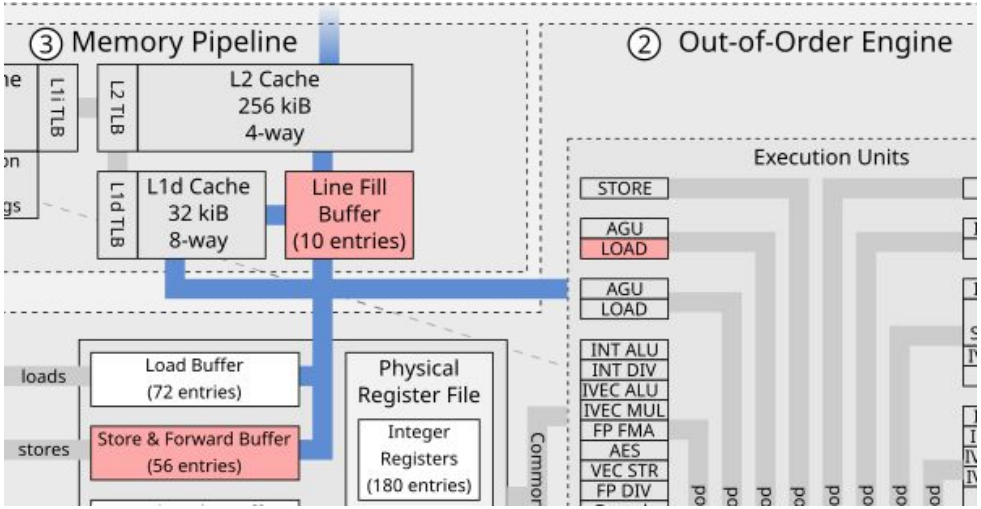


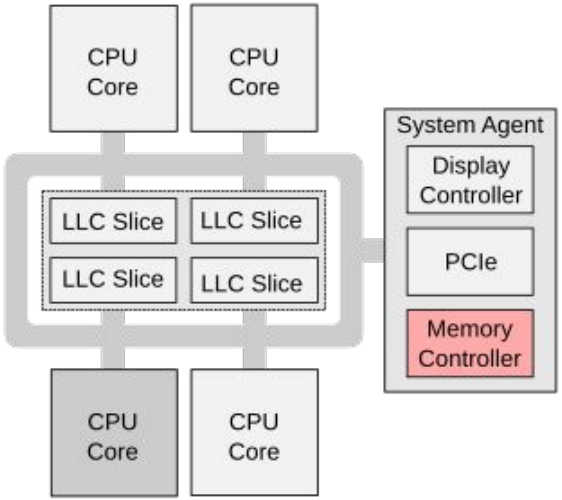
LFB Leaks


SMT
10000
8000
6000
4000
2000
0
Frequency
cross thread
lfb_hit
secret

Sometimes We Get It Wrong..

- RIDL paper: leakage is primarily through LFBs
- Public disclosure: May 14
- Intel on May 10:
  - There are 4 variants, you only leak through LFB
  - !&#&* (heated exchange)
  - You are leaking through LFB, UC, LP





More information on: mdsattacks.com

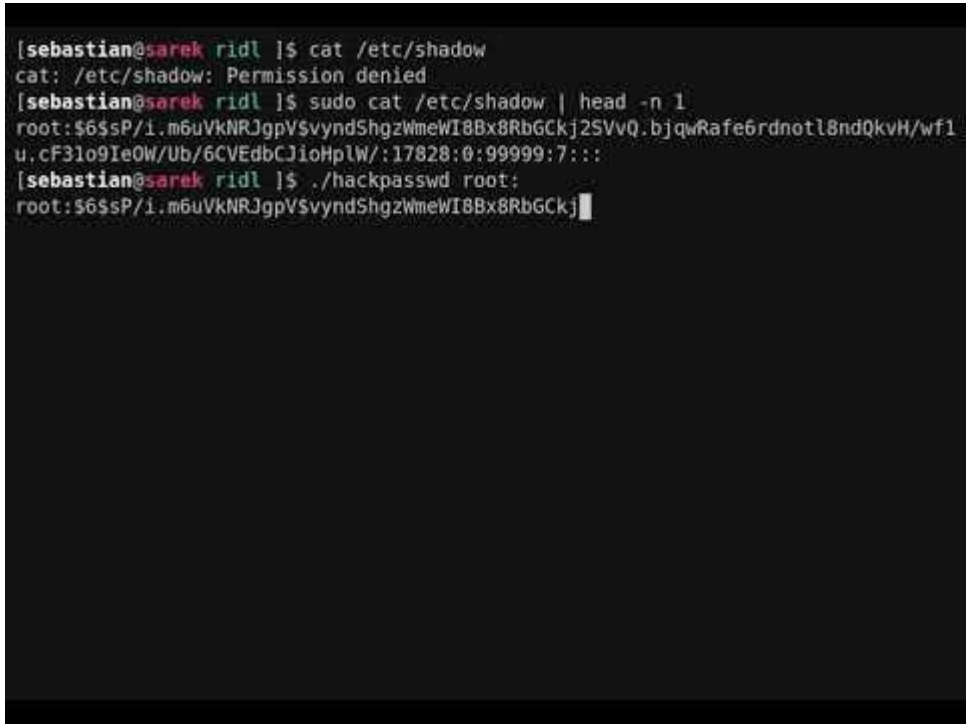
Which CPUs Are Vulnerable?

- ✓ Intel Core i9-9900K (Coffee Lake R) - 2018
- ✓ Intel Xeon Silver 4110 (Skylake SP) - 2017
- ✓ Intel Core i7-8700K (Coffee Lake) - 2017
- ✓ Intel Core i7-7800X (Skylake X) - 2017
- ✓ Intel Core i7-7700K (Kaby Lake) - 2017
- ✓ Intel Core i7-6700K (Skylake) - 2015
- ✓ Intel Core i7-5775C (Broadwel) - 2015
- ✓ Intel Core i7-4790 (Haswell) - 2014
- ✓ Intel Core i7-3770K (Ivy Bridge) - 2012
- ✓ Intel Core i7-2600 (Sandy Bridge) - 2011
- ✓ Intel Core i3-550 (Westmere) - 2010
- ✓ Intel Core i7-920 (Nehalem) - 2008
- ✗ AMD Ryzen 5 2500U (Raven Ridge) - 2018
- ✗ AMD Ryzen 7 2600X (Pinnacle Ridge) - 2018
- ✗ AMD Ryzen 7 1600X (Summit Ridge) - 2017

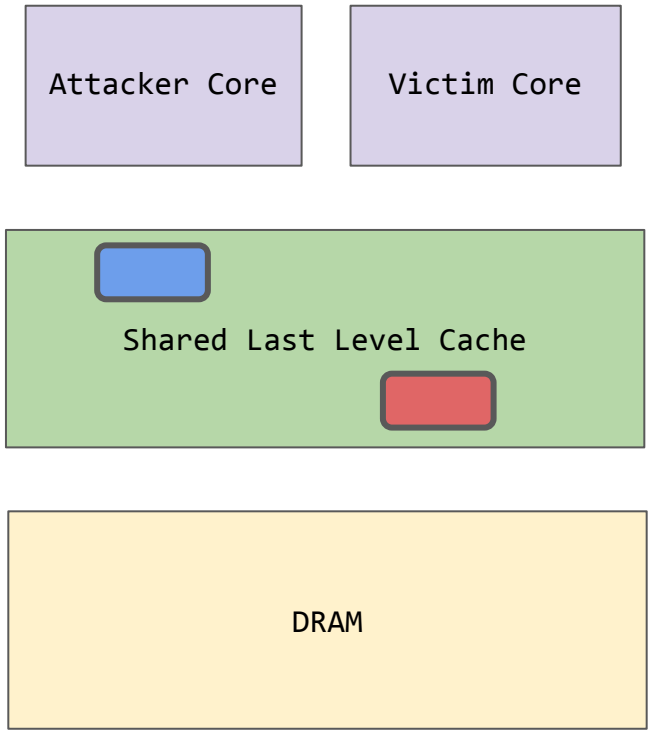
1 Year of CVD with Intel

$100,000 bounty award

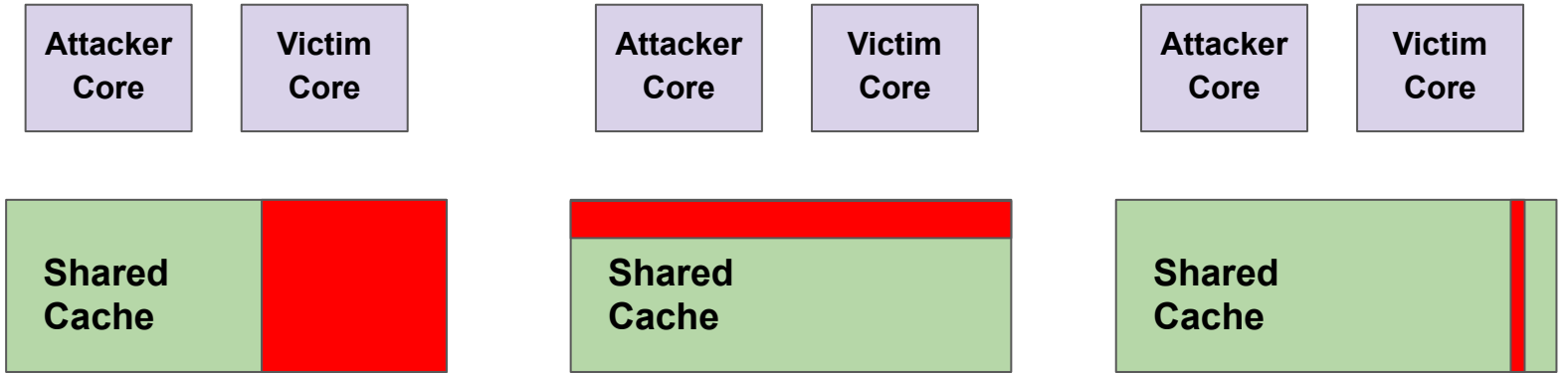
Demo


[sebastian@sarek ridl]$ cat /etc/shadow
cat: /etc/shadow: Permission denied
[sebastian@sarek ridl]$ sudo cat /etc/shadow | head -n 1
root:6sP/i.m6uVkNRJgpV$vyndShgzWmeWI8Bx8RbGCkj2SVvQ.bjqwRafe6rdnotl8ndQkvH/wf1
u.cF3lo9IeOW/Ub/6CVEdbCJioHplW/:17828:0:99999:7:::
[sebastian@sarek ridl]$ ./hackpasswd root:
root:6sP/i.m6uVkNRJgpV$vyndShgzWmeWI8Bx8RbGCkj

Traditional Cache Attacks



Proposed Defenses: Cache Partitioning



COLORIS
Page Coloring
PACT'14

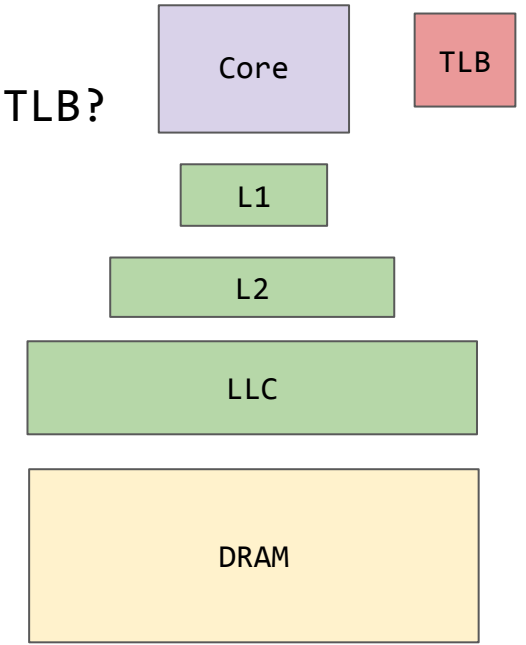
CATalyst
Intel CAT
HPCA'17

Cloak
Intel TSX
SEC'17

Issue: Imperfect Partitioning

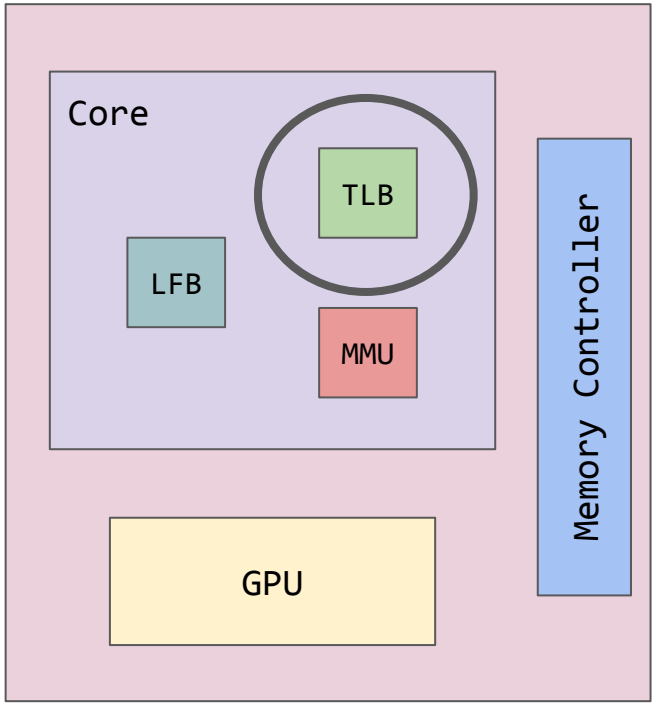
Can we make meaningful attacks against the TLB?

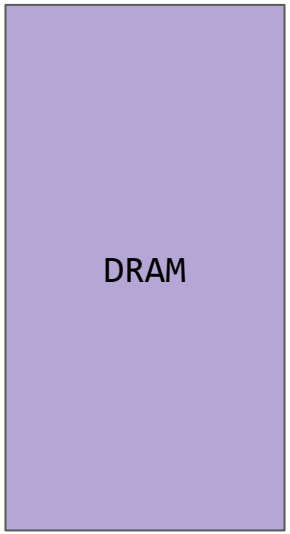
- Architecture of the TLB is unknown
- Spatial attacks are difficult
  (4KB page size)



Gras et al., “Translation Leakaside Buffer: Defeating Cache Side-channel Protections with TLB Attacks,” SEC’18

Hardware Reverse Engineering at VUSec





Rowhammer

Side channels

Reverse Engineering the TLB Architecture

Goal: find virtual addresses that prime a TLB set

Methodology: Oren et al. eviction set building algorithm

- Pick a pool virtual addresses
- Pick an observer address
- Reduce the pool to a minimal set that evicts observer

		L1 dTLB					L1 iTLB					L2 sTLB				
Name	year	set	w	pn	hsh	shr	set	w	pn	hsh	shr	set	w	pn	hsh	shr
Sandybridge	2011	16	4	7.0	lin	✓	16	4	50.0	lin	✗	128	4	16.3	lin	✓
Ivybridge	2012	16	4	7.1	lin	✓	16	4	49.4	lin	✗	128	4	18.0	lin	✓
Haswell	2013	16	4	8.0	lin	✓	8	8	27.4	lin	✗	128	8	17.1	lin	✓
HaswellXeon	2014	16	4	7.9	lin	✓	8	8	28.5	lin	✗	128	8	16.8	lin	✓
Skylake	2015	16	4	9.0	lin	✓	8	8	2.0	lin	✗	128	12	212.0	XOR-7	✓
BroadwellXeon	2016	16	4	8.0	lin	✓	8	8	18.2	lin	✗	256	6	272.4	XOR-8	✓
Coffeelake	2017	16	4	9.1	lin	✓	8	8	26.3	lin	✗	128	12	230.3	XOR-7	✓

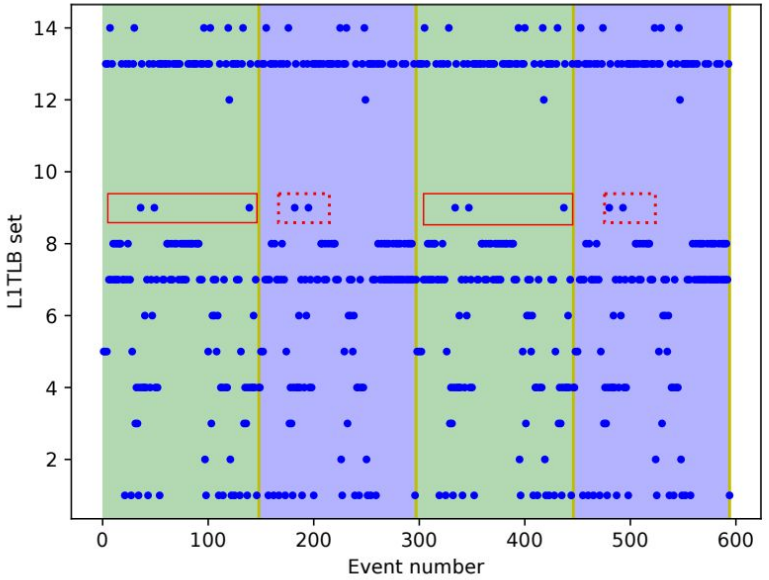
Temporal Attacks with PRIME+PROBE

Timing of misses in a TLB set rather than different sets.

- Help from a SVM classifier

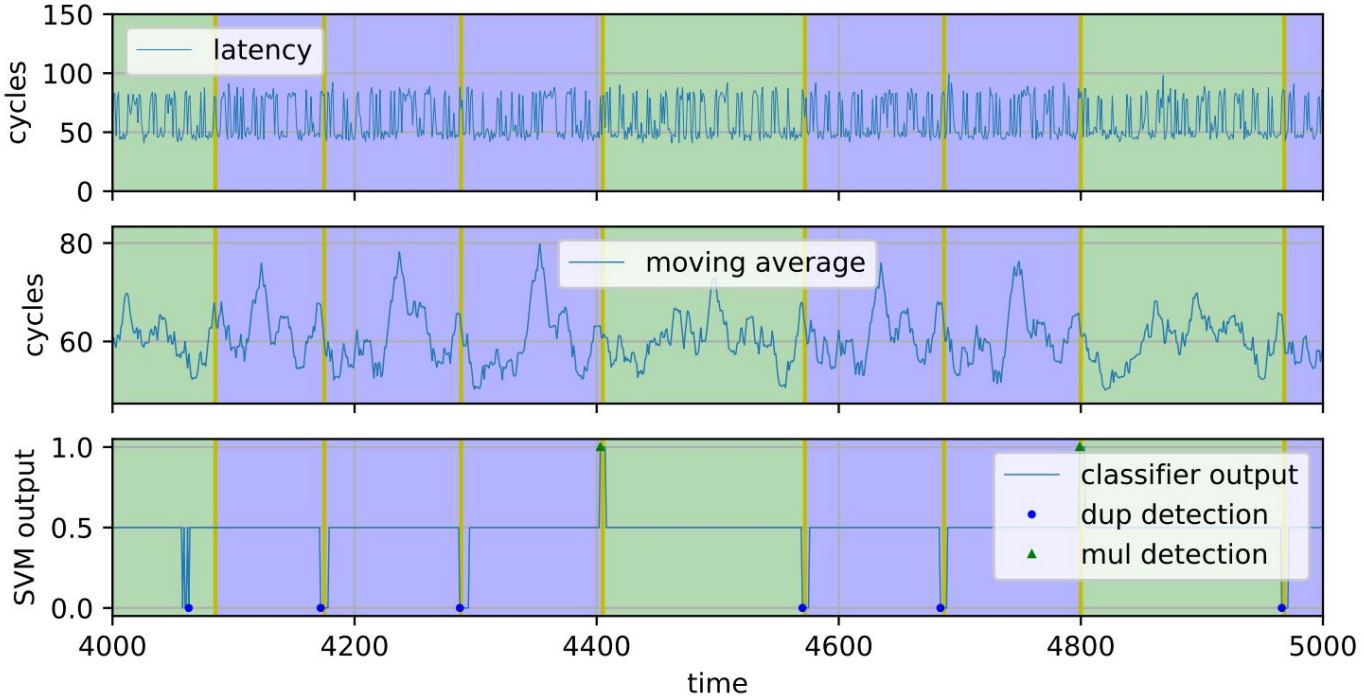
Table 3: TLBleed compromising Intel CAT.

Microarchitecture	Trials	Success	Median BF
Broadwell (CAT)	500	0.960	$2^{2.6}$
Broadwell	500	0.982	$2^{3.0}$



Elliptic curve point multiplication in libgcrypt.

Example SVM Output



Impact

Security

Meet TLBleed: A crypto-key-leaking CPU attack that Intel reckons we shouldn't worry about

How to extract 256-bit keys with 99.8% success

By Chris Williams, Editor in Chief 22 Jun 2018 at 22:44 60 SHARE ▼

Kav
@kavehrazavi

Our upcoming #TLBleed paper leads to (finally) disabling SMT in security-sensitive environments (#OpenBSD in this case).

XLATE Attacks (SEC'18) similarly bypasses imperfect partitioning

Van Schaik et al., "Malicious Management Unit: Why Stopping Cache Attacks in Software is Harder than You Think," SEC'18

Conclusion

Hardware is the new software except it is harder to fix

Reverse engineering is our main tool in academia

We need to invest in open hardware

Ben Gras, Pietro Frigo, Lucian Cojocar, Stephan van Schaik, Alyssa Milburn, Sebastian Osterlund, Cristiano Giuffrida, Herbert Bos