Oh my Cache! 2
More fun with caches.

Daniel Gruss
Graz University of Technology

October 13, 2017 — QSP Lab

Whoami

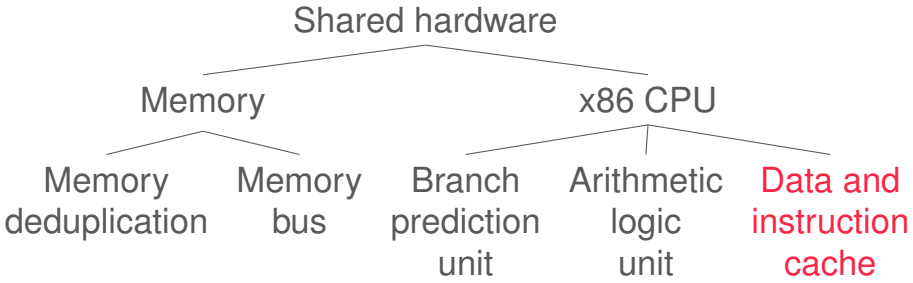
- Daniel Gruss
- Post-Doc @ Graz University of Technology
- Twitter: `@lavados`
- Email: `daniel.gruss@iaik.tugraz.at`

Get your computer ready!

Within the first two hours we will:

- Checkout `https://github.com/IAIK/cache_template_attacks`
- Make a histogram
- Key stroke attack on an editor
- Try to establish a covert channel

Information leakage



Why targeting the cache?

- shared across cores
- fast

Why targeting the cache?

- shared across cores
- fast

→ fast cross-core attacks!

Timing differences

- caches improve performance
- SRAM is expensive $\rightarrow$ small caches
- different timings for memory accesses
  - data is cached $\rightarrow$ cache hit $\rightarrow$ fast
  - data is not cached $\rightarrow$ cache miss $\rightarrow$ slow

Mesuring timing leakage

How every timing attack works:

- learn timing of different corner cases

Mesuring timing leakage

How every timing attack works:

- learn timing of different corner cases
- later, we recognize these corner cases by timing only

Calibration

```
git clone https://github.com/IAIK/cache_template_attacks.git
cd calibration
make
./calibration
```

Steps

1. build two cases: cache hits and cache misses
2. time each case many times (get rid of noise)

Steps

1. build two cases: cache hits and cache misses
2. time each case many times (get rid of noise)
3. we have a histogram!

Steps

1. build two cases: cache hits and cache misses
2. time each case many times (get rid of noise)
3. we have a histogram!
4. find a threshold to distinguish the two cases

Step 1.1. Cache hits

Loop:

1. measure time
2. access variable (always cache hit)
3. measure time
4. update histogram with delta

Step 1.2. Cache misses

Loop:

1. measure time
2. access variable (always cache miss)
3. measure time
4. update histogram with delta
5. flush variable (`clflush` instruction)

Step 2. Accurate timings

- very short timings
- `rdtsc` instruction: cycle-accurate timestamps

```
[...]
rdtsc
function()
rdtsc
[...]
```

Step 2. Accurate timings

- do you measure what you *think* you measure?
- out-of-order execution → what is really executed

```
rdtsc
function()
[...]
rdtsc
```

```
rdtsc
[...]
rdtsc
function()
```

```
rdtsc
rdtsc
function()
[...]
```

Step 2. Accurate timings

- use pseudo-serializing instruction `rdtscp` (recent CPUs)

Step 2. Accurate timings

- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cpuid`

Step 2. Accurate timings

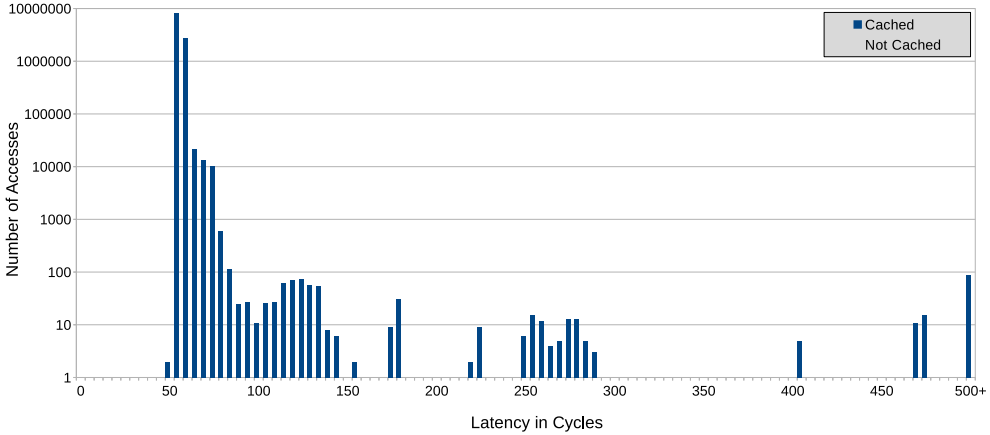
- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cpuid`
- and/or use fences like `mfence`

Step 2. Accurate timings

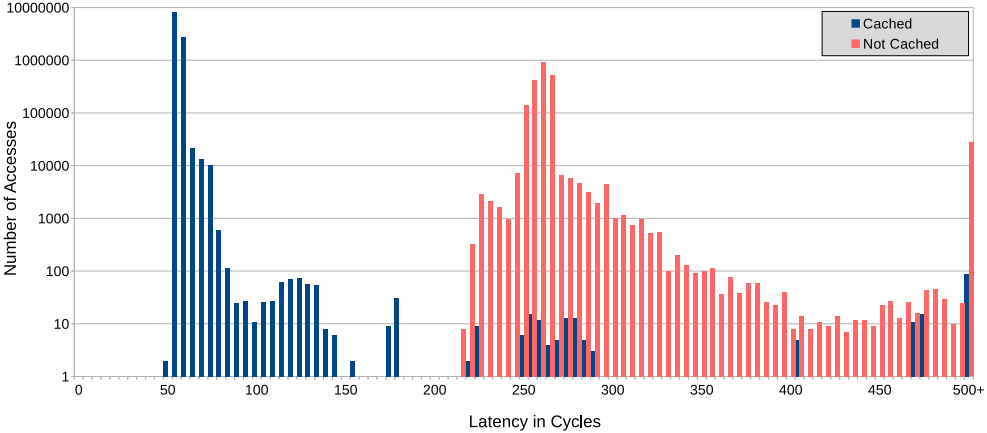
- use pseudo-serializing instruction `rdtscp` (recent CPUs)
- and/or use serializing instructions like `cpuid`
- and/or use fences like `mfence`

Intel, *How to Benchmark Code Execution Times on Intel IA-32 and IA-64 Instruction Set Architectures White Paper*, December 2010.

Step 3. Histogram



Step 3. Histogram



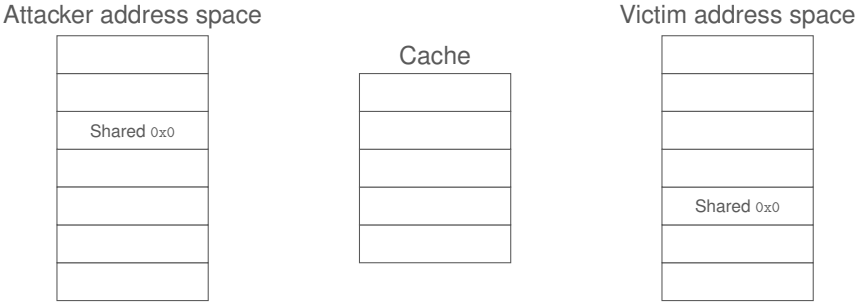
Step 4. Find threshold

- as high as possible
- most cache hits are below
- no cache miss below

Side-channel attack on user input

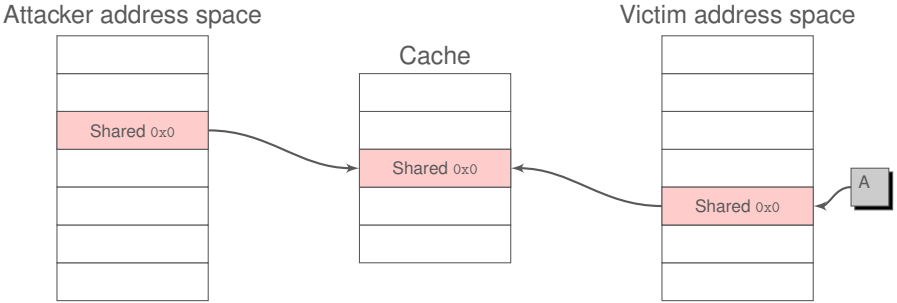
- locate key-dependent memory accesses
- with cache template attacks

Profiling Phase: one event



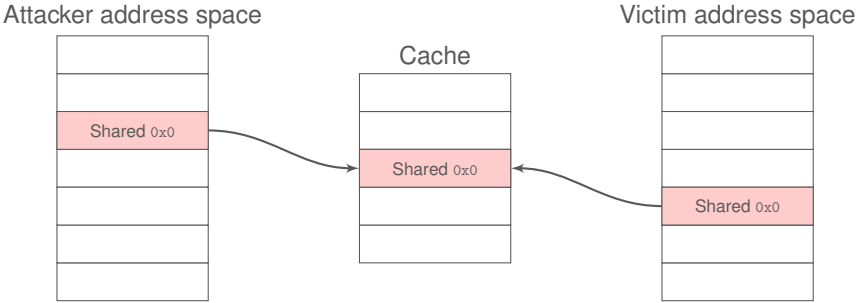
Cache is empty

Profiling Phase: one event



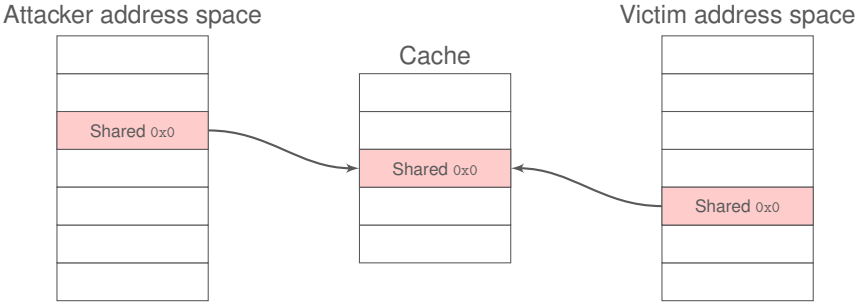
Attacker triggers an event

Profiling Phase: one event



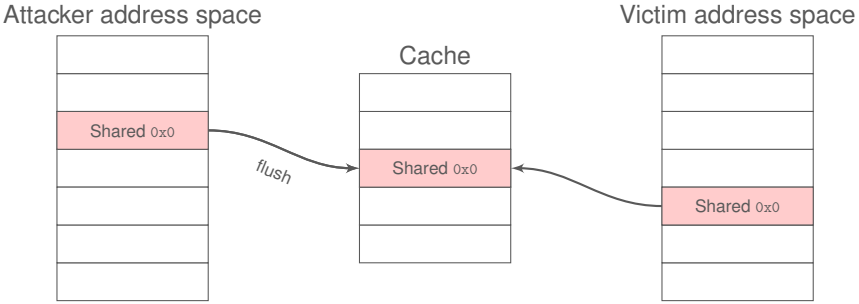
Attacker checks one address for cache hits (“Reload”)

Profiling Phase: one event



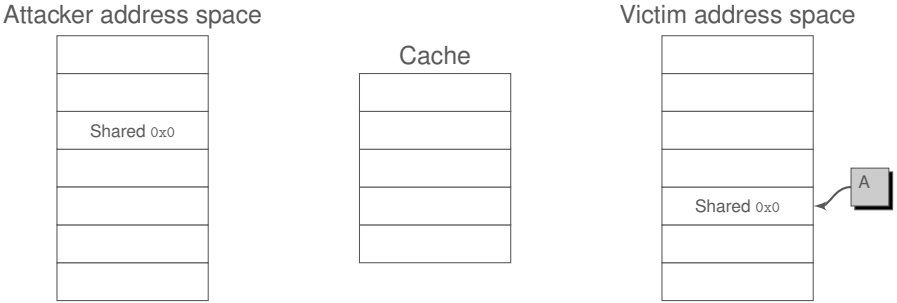
Update number of cache hits per event

Profiling Phase: one event



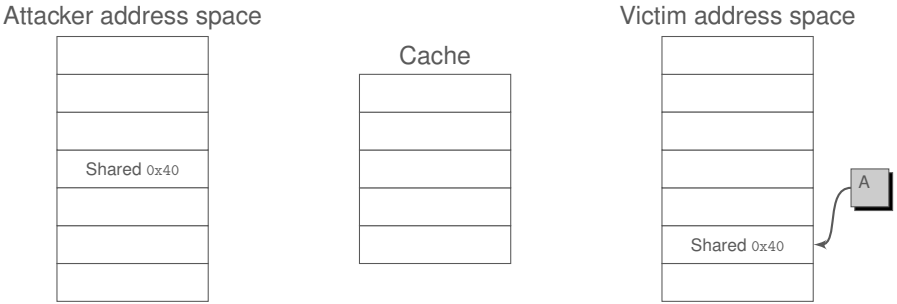
Attacker flushes shared memory

Profiling Phase: one event



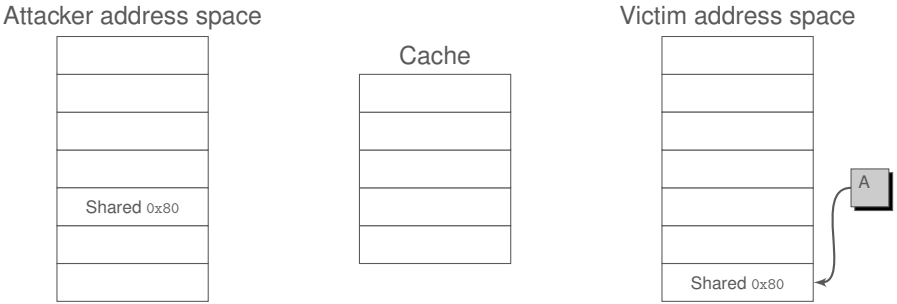
Repeat for higher accuracy

Profiling Phase: one event



Continue with next address

Profiling Phase: one event



Continue with next address

What to profile?

```
# ps -A | grep gedit
# cat /proc/pid/maps
00400000-00489000 r-xp 00000000 08:11 396356
/usr/bin/gedit
7f5a96991000-7f5a96a51000 r-xp 00000000 08:11 399365
/usr/lib/x86_64-linux-gnu/libgdk-3.so.0.1400.14
...
```

memory range, access rights, offset, –, –, file name

Profiling a single event

```
cd ../profiling/generic_low_frequency_example
# put the threshold into spy.c (MIN_CACHE_MISS_CYCLES)
make
./spy
# start the targeted program
sleep 2; ./spy 200             400000-489000  --    20000
-- -- /usr/bin/gedit
```

... and hold down key in the targeted program
save addresses with peaks!

Exploitation phase

```
cd ../exploitation/generic
# put the threshold into spy.c (MIN_CACHE_MISS_CYCLES)
make
./spy file offset
```

Directly mapped cache

Memory Address

Directly mapped cache

Memory Address

Cache

Directly mapped cache

Memory Address

Cache

Tag	Data

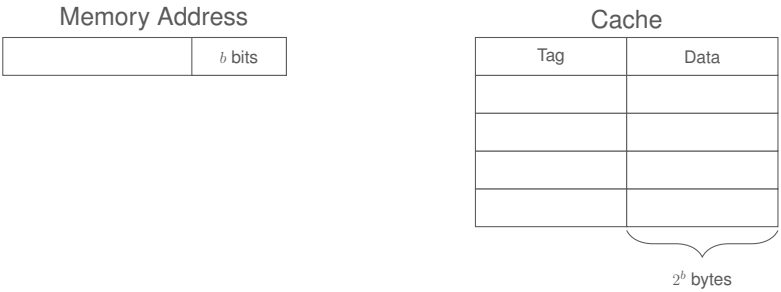
Directly mapped cache

Memory Address

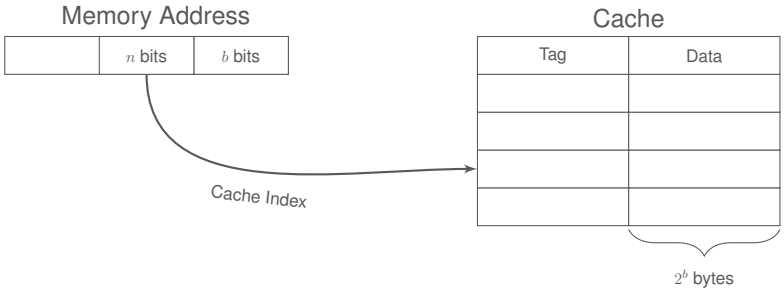
Cache

Tag	Data

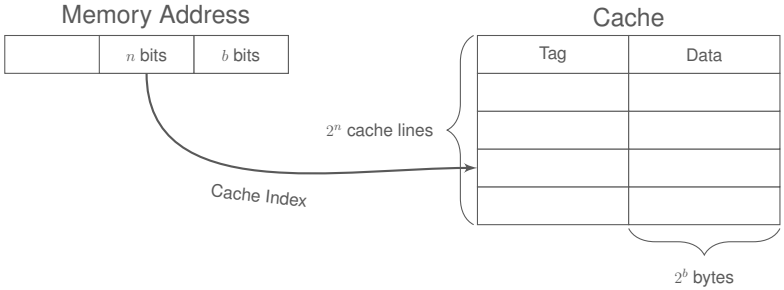
Directly mapped cache



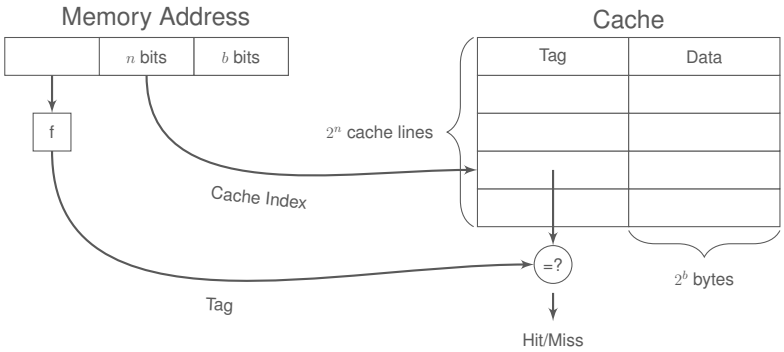
Directly mapped cache



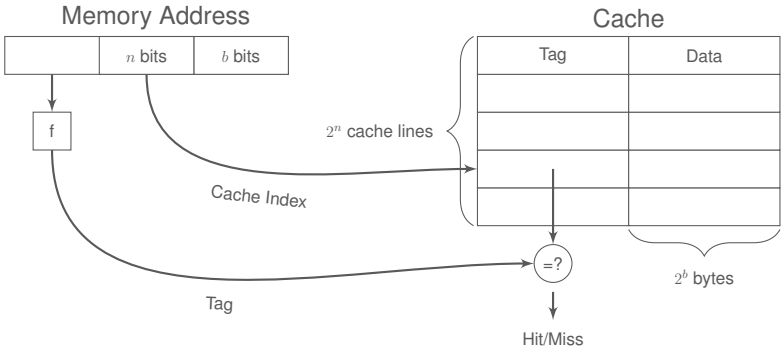
Directly mapped cache



Directly mapped cache

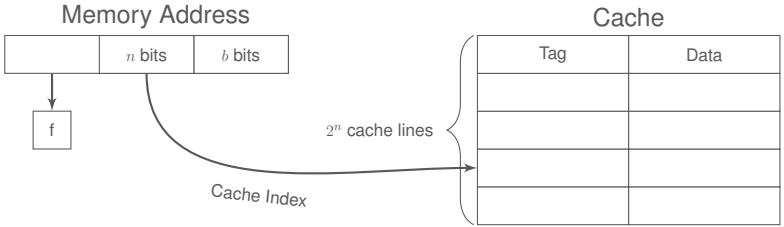


Directly mapped cache

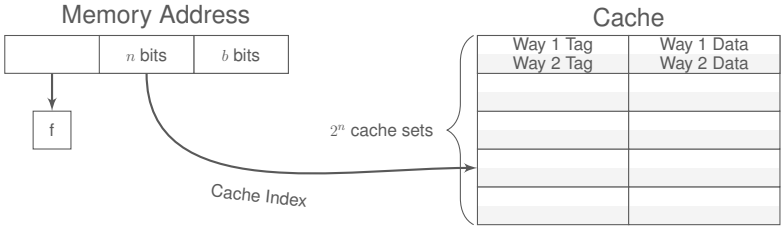


Problem: working on congruent addresses

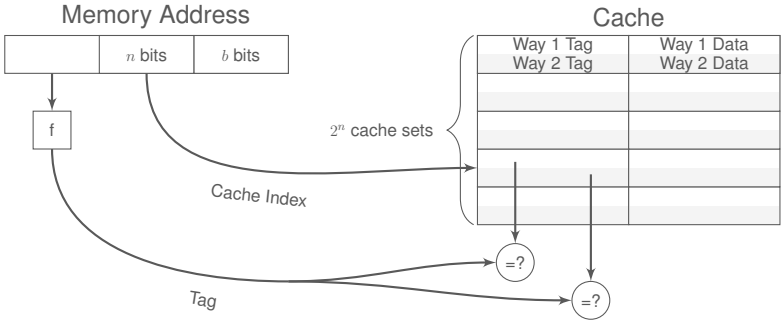
2-way set associativity



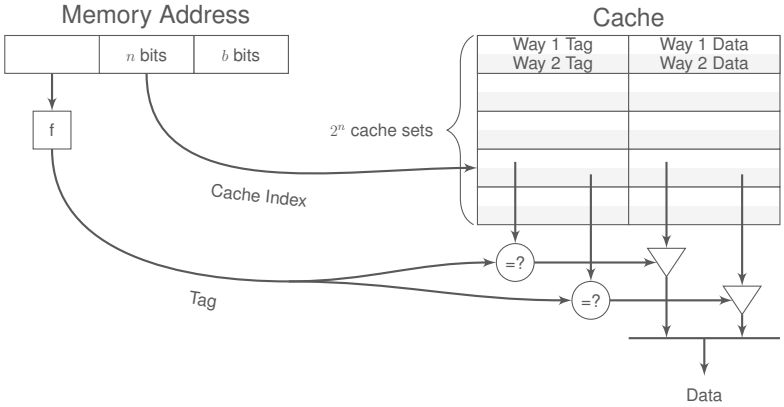
2-way set associativity



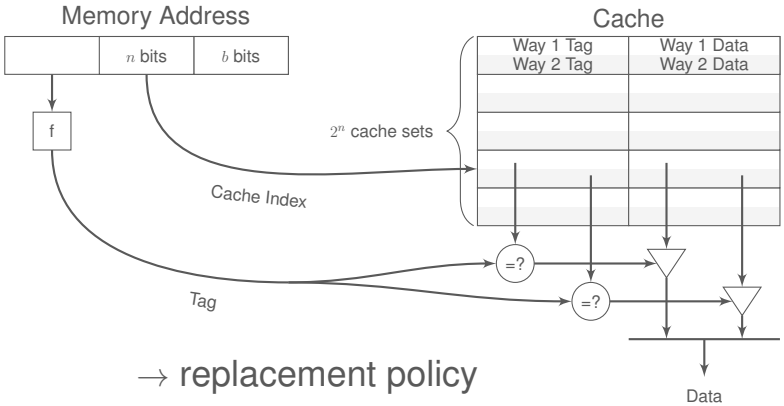
2-way set associativity



2-way set associativity

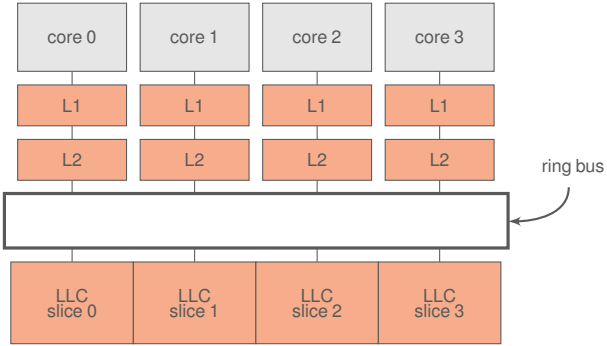


2-way set associativity



$\rightarrow$ replacement policy

Caches today



- L1 and L2 are private
- last-level cache:
  - divided in slices
  - shared across cores
  - inclusive

Cache levels: Latency comparison

On current Intel CPUs:

Cache levels: Latency comparison

On current Intel CPUs:

- L1 cache: 4 cycles

Cache levels: Latency comparison

On current Intel CPUs:

- L1 cache: 4 cycles
- L2 cache: 12 cycles

Cache levels: Latency comparison

On current Intel CPUs:

- L1 cache: 4 cycles
- L2 cache: 12 cycles
- L3 cache: 26-31 cycles

Cache levels: Latency comparison

On current Intel CPUs:

- L1 cache: 4 cycles
- L2 cache: 12 cycles
- L3 cache: 26-31 cycles
- DRAM memory: $>$120 cycles

(Unprivileged) cache maintainance

User programs can optimize cache usage:

- `prefetch`: suggest CPU to load data into cache
- `clflush`: throw out data from from all caches

... based on virtual addresses

CPU cache attacks

- cache-based keylogging
- crypto key recovery
  - various implementations (AES, RSA, ECC, ...)
  - up to 97% key bits recovered after 1 encryption
- cross-VM, cross-core, even cross-CPU
- any CPU vendor

Cross-core attacks?

- using the inclusive property

Cross-core attacks?

- using the inclusive property
- last-level cache is a superset of L1 and L2

Cross-core attacks?

- using the inclusive property
- last-level cache is a superset of L1 and L2
- data evicted from last-level cache $\rightarrow$ evicted from L1 and L2

Cross-core attacks?

- using the inclusive property
- last-level cache is a superset of L1 and L2
- data evicted from last-level cache $\rightarrow$ evicted from L1 and L2
- a core can evict lines in the private L1 of another core

Access-driven attacks

Attacker monitors its own activity to find sets accessed by victim.

Flush+Reload	Prime+Probe
Gullasch et al. 2011	Percival 2005
Yarom and Falkner 2014	Liu et al. 2015
Gruss, Spreitzer, et al. 2015	Maurice, Neumann, et al. 2015

Same techniques for covert and side channels

Flush+Reload: Building Blocks

- Shared Library / load binary twice / page deduplication

Flush+Reload: Building Blocks

- Shared Library / load binary twice / page deduplication
- `clflush` throws data out of cache

→ We can throw other shared code out of the cache

Flush+Reload: Building Blocks

- Shared Library / load binary twice / page deduplication
- `clflush` throws data out of cache

→ We can throw other shared code out of the cache

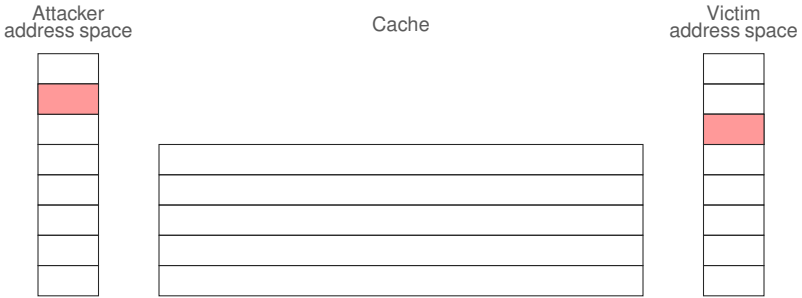
- `rdtsc` / `rdtscp` give accurate timing information

→ We can measure whether shared code is in the cache

Flush+Reload: First steps

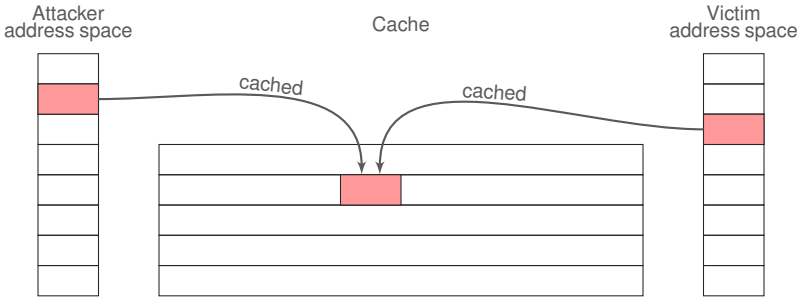
- Measure timing of cached memory
- Measure timing of non-cached memory (flush before measuring)
- Draw a histogram

Flush+Reload



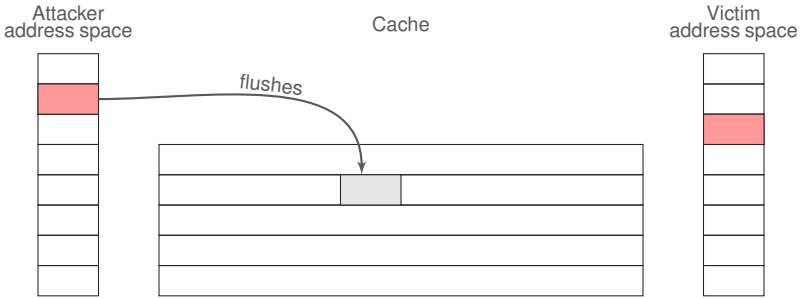
step 0: attacker maps shared library → shared memory, shared in cache

Flush+Reload



step 0: attacker maps shared library → shared memory, shared in cache

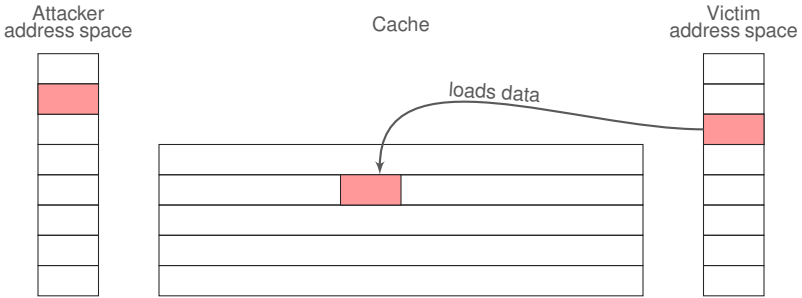
Flush+Reload



step 0: attacker maps shared library → shared memory, shared in cache

step 1: attacker flushes the shared line

Flush+Reload

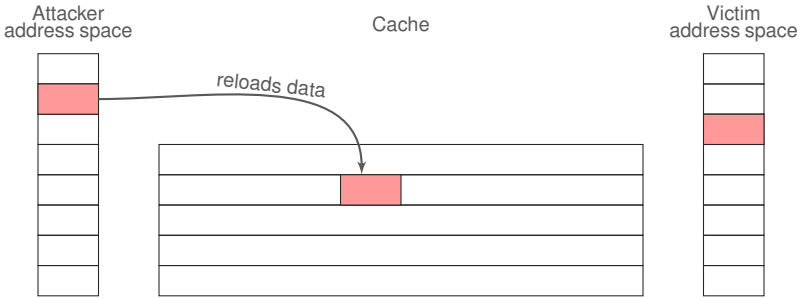


step 0: attacker maps shared library → shared memory, shared in cache

step 1: attacker flushes the shared line

step 2: victim loads data while performing encryption

Flush+Reload



step 0: attacker maps shared library → shared memory, shared in cache

step 1: attacker flushes the shared line

step 2: victim loads data while performing encryption

step 3: attacker reloads data → fast access if the victim loaded the line

Flush+Reload

Pros: fine granularity (1 line)

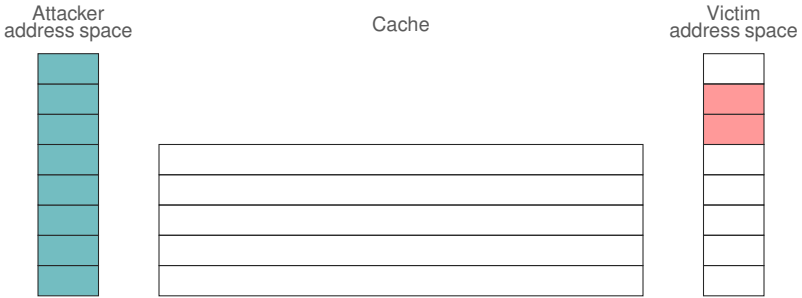
Cons: restrictive

1. needs `clflush` instruction (not available e.g., in JS)
2. needs shared memory

Variants of Flush+Reload

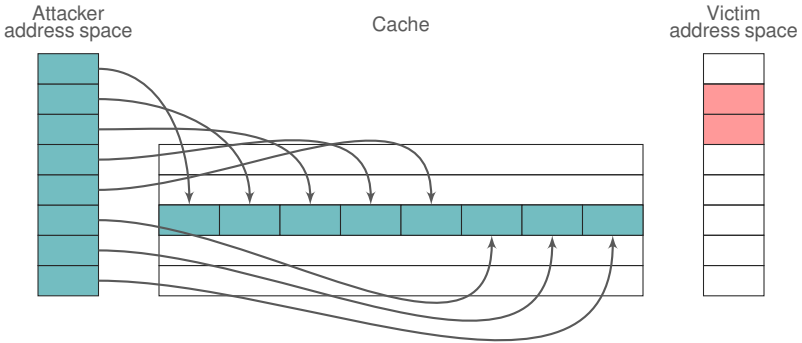
- Flush+Flush Gruss, Maurice, Wagner, et al. 2016
- Evict+Reload Gruss, Spreitzer, et al. 2015 on ARM Lipp et al. 2016

Prime+Probe



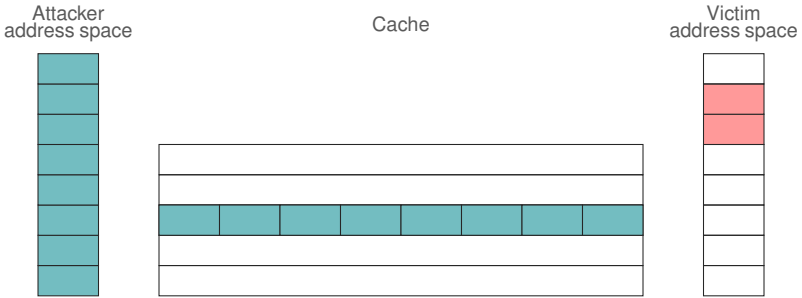
step 0: attacker fills the cache (prime)

Prime+Probe



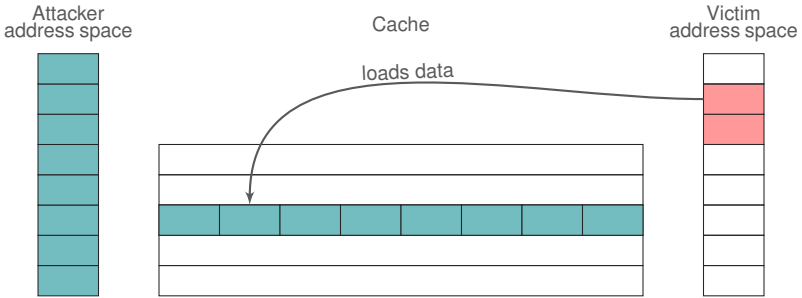
step 0: attacker fills the cache (prime)

Prime+Probe



step 0: attacker fills the cache (prime)

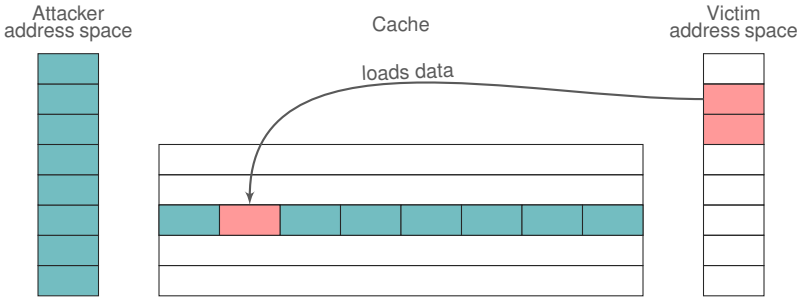
Prime+Probe



step 0: attacker fills the cache (prime)

step 1: victim evicts cache lines while performing encryption

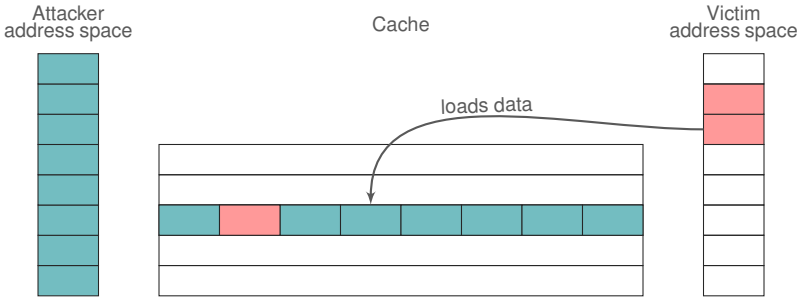
Prime+Probe



step 0: attacker fills the cache (prime)

step 1: victim evicts cache lines while performing encryption

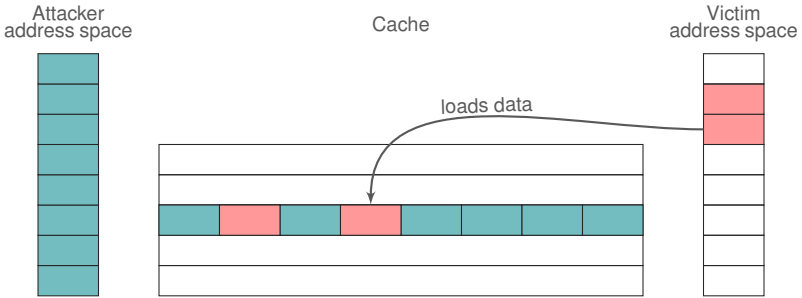
Prime+Probe



step 0: attacker fills the cache (prime)

step 1: victim evicts cache lines while performing encryption

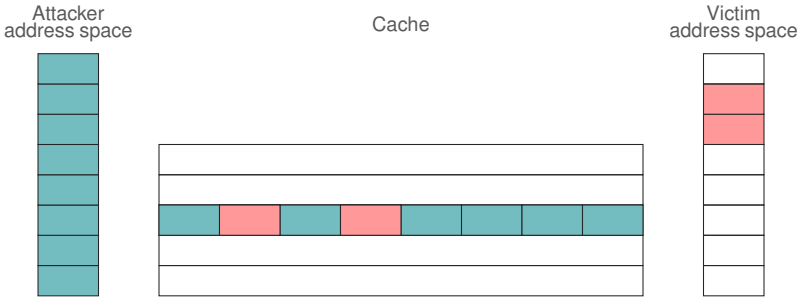
Prime+Probe



step 0: attacker fills the cache (prime)

step 1: victim evicts cache lines while performing encryption

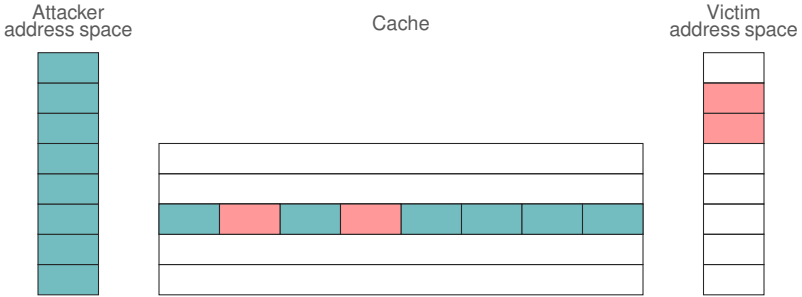
Prime+Probe



step 0: attacker fills the cache (prime)

step 1: victim evicts cache lines while performing encryption

Prime+Probe

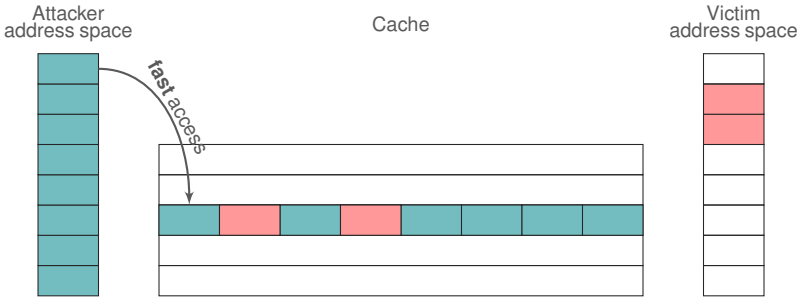


step 0: attacker fills the cache (prime)

step 1: victim evicts cache lines while performing encryption

step 2: attacker probes data to determine if the set was accessed

Prime+Probe

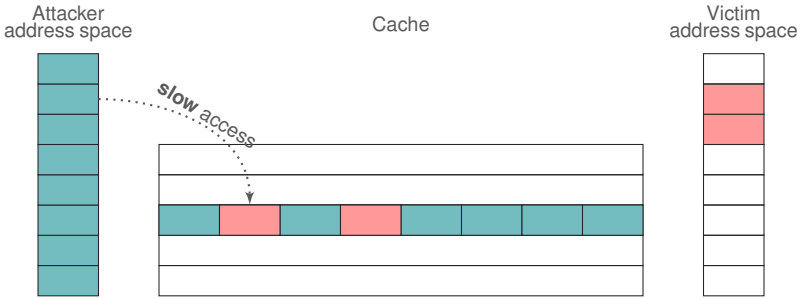


step 0: attacker fills the cache (prime)

step 1: victim evicts cache lines while performing encryption

step 2: attacker probes data to determine if the set was accessed

Prime+Probe



step 0: attacker fills the cache (prime)

step 1: victim evicts cache lines while performing encryption

step 2: attacker probes data to determine if the set was accessed

Prime+Probe

Pros: less restrictive

1. no need for `clflush` instruction (not available e.g., in JS)
2. no need for shared memory

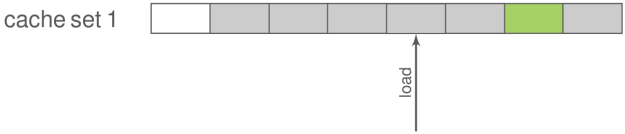
Cons: coarser granularity (1 set)

Issues with Prime+Probe

We need to evict caches lines without `clflush` or shared memory:

1. which addresses do we access to have congruent cache lines?
2. without any privilege?
3. and in which order do we access them?

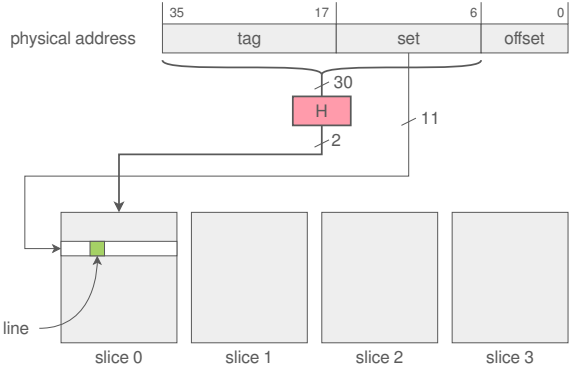
#1.1: Which physical addresses to access?



"LRU eviction":

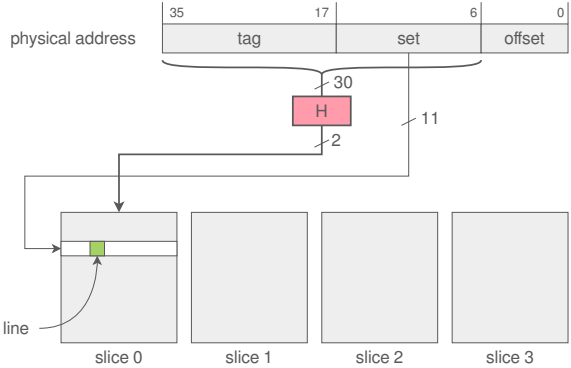
- assume that cache uses LRU replacement
- accessing n addresses from the same cache set to evict an n-way set
- eviction from last level $\rightarrow$ from whole hierarchy (it's inclusive!)

#1.2: Which addresses map to the same set?



- function H that maps slices is undocumented
- reverse-engineered by Maurice, Le Scouarnec, et al. 2015; Inci et al. 2015; Yarom, Ge, et al. 2015

#1.2: Which addresses map to the same set?



- function H that maps slices is undocumented
- reverse-engineered by Maurice, Le Scouarnec, et al. 2015; Inci et al. 2015; Yarom, Ge, et al. 2015
- hash function basically an XOR of address bits

#1.2: Which addresses map to the same set?

3 functions, depending on the number of cores

		Address bit																																	
		37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06
2 cores	o_0						$\oplus$		$\oplus$		$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$		$\oplus$		$\oplus$		$\oplus$	$\oplus$	$\oplus$		$\oplus$		$\oplus$		$\oplus$				$\oplus$
4 cores	o_0					$\oplus$	$\oplus$		$\oplus$		$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$		$\oplus$		$\oplus$		$\oplus$	$\oplus$	$\oplus$		$\oplus$		$\oplus$		$\oplus$				$\oplus$
	o_1				$\oplus$	$\oplus$		$\oplus$		$\oplus$	$\oplus$		$\oplus$		$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$		$\oplus$		$\oplus$		$\oplus$		$\oplus$				$\oplus$	
8 cores	o_0		$\oplus$	$\oplus$		$\oplus$	$\oplus$		$\oplus$		$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$		$\oplus$		$\oplus$		$\oplus$	$\oplus$	$\oplus$		$\oplus$		$\oplus$		$\oplus$				$\oplus$
	o_1	$\oplus$		$\oplus$	$\oplus$	$\oplus$		$\oplus$		$\oplus$	$\oplus$		$\oplus$		$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$	$\oplus$		$\oplus$		$\oplus$		$\oplus$		$\oplus$				$\oplus$	
	o_2	$\oplus$	$\oplus$	$\oplus$	$\oplus$			$\oplus$	$\oplus$			$\oplus$	$\oplus$			$\oplus$	$\oplus$			$\oplus$			$\oplus$			$\oplus$	$\oplus$				$\oplus$		

#2: Obtain information without root privileges

- last-level cache is physically indexed

#2: Obtain information without root privileges

- last-level cache is physically indexed
- root privileges needed for physical addresses

#2: Obtain information without root privileges

- last-level cache is physically indexed
- root privileges needed for physical addresses
- use 2 MB pages $\rightarrow$ lowest 21 bits are the same as virtual address

#2: Obtain information without root privileges

- last-level cache is physically indexed
- root privileges needed for physical addresses
- use 2 MB pages → lowest 21 bits are the same as virtual address

→ enough to compute the cache set

#3.1: Replacement policy on older CPUs

“LRU eviction” memory accesses

#3.1: Replacement policy on older CPUs

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first

#3.1: Replacement policy on older CPUs

“LRU eviction” memory accesses

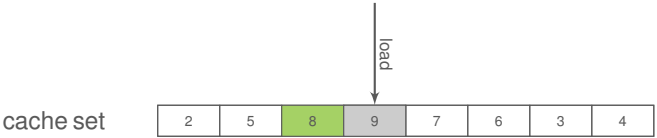
cache set

2	5	8	1	7	6	3	4

- LRU replacement policy: oldest entry first
- timestamps for every cache line

#3.1: Replacement policy on older CPUs

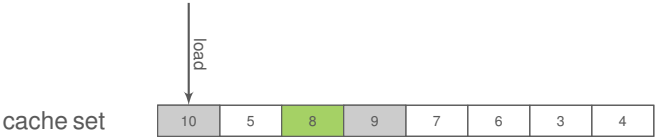
“LRU eviction” memory accesses



- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

#3.1: Replacement policy on older CPUs

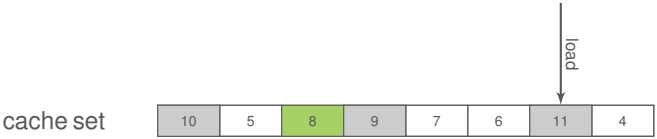
“LRU eviction” memory accesses



- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

#3.1: Replacement policy on older CPUs

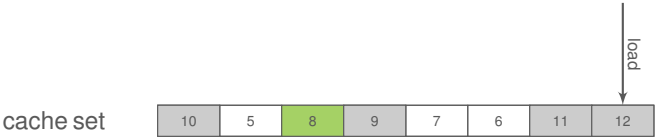
“LRU eviction” memory accesses



- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

#3.1: Replacement policy on older CPUs

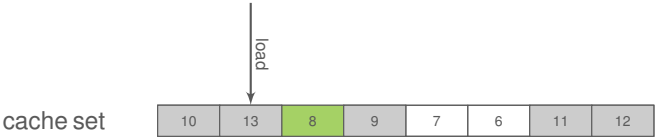
“LRU eviction” memory accesses



- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

#3.1: Replacement policy on older CPUs

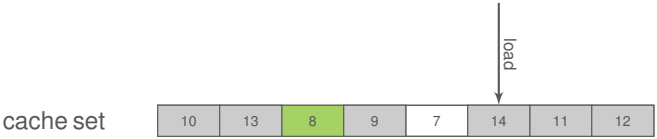
“LRU eviction” memory accesses



- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

#3.1: Replacement policy on older CPUs

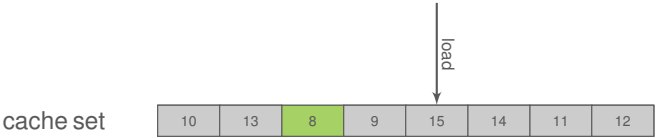
“LRU eviction” memory accesses



- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

#3.1: Replacement policy on older CPUs

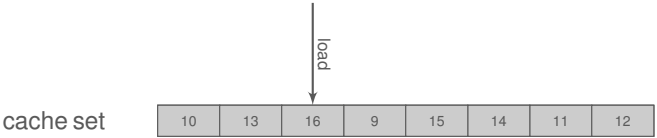
“LRU eviction” memory accesses



- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

#3.1: Replacement policy on older CPUs

“LRU eviction” memory accesses



- LRU replacement policy: oldest entry first
- timestamps for every cache line
- access updates timestamp

#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

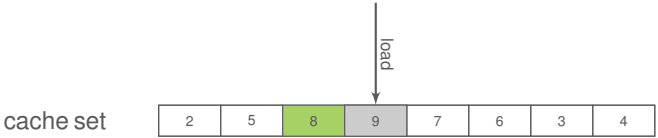
cache set

2	5	8	1	7	6	3	4

- no LRU replacement

#3.2: Replacement policy on recent CPUs

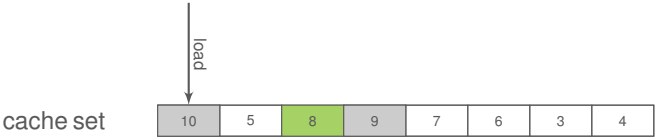
“LRU eviction” memory accesses



- no LRU replacement

#3.2: Replacement policy on recent CPUs

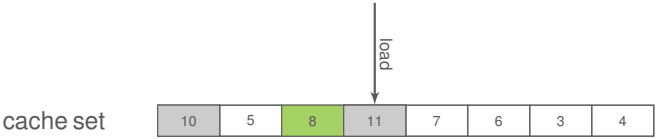
“LRU eviction” memory accesses



- no LRU replacement

#3.2: Replacement policy on recent CPUs

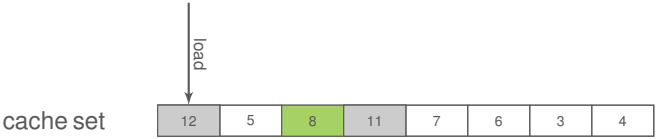
“LRU eviction” memory accesses



- no LRU replacement

#3.2: Replacement policy on recent CPUs

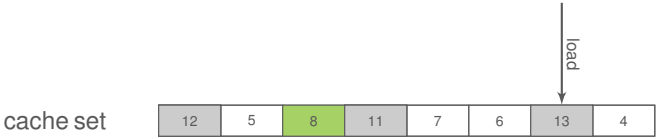
“LRU eviction” memory accesses



- no LRU replacement

#3.2: Replacement policy on recent CPUs

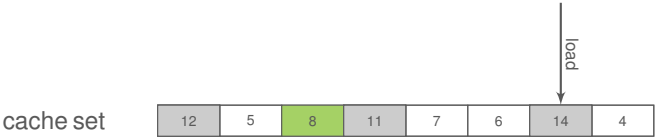
“LRU eviction” memory accesses



- no LRU replacement

#3.2: Replacement policy on recent CPUs

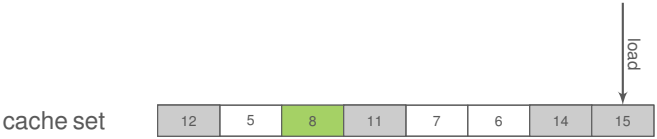
“LRU eviction” memory accesses



- no LRU replacement

#3.2: Replacement policy on recent CPUs

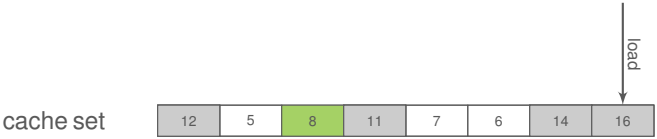
“LRU eviction” memory accesses



- no LRU replacement

#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses



- no LRU replacement

#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

cache set

12	5	8	11	7	6	14	16

- no LRU replacement
- only 75% success rate on Haswell

#3.2: Replacement policy on recent CPUs

“LRU eviction” memory accesses

cache set

12	5	8	11	7	6	14	16

- no LRU replacement
- only 75% success rate on Haswell
- more accesses → higher success rate, but too slow

#3.3: Cache eviction strategy

Figure: Fast and effective on Haswell. Eviction rate >99.97%.

Side channels vs covert channels

- side channel: attacker spies a victim process
- covert channel: communication between two processes
  - that are not supposed to communicate
  - that are collaborating

1-bit cache covert channels

ideas for 1-bit channels:

1-bit cache covert channels

ideas for 1-bit channels:

- Prime+Probe: use one cache set to transmit

  0: sender does not access the set $\rightarrow$ low access time in receiver
  1: sender does access the set $\rightarrow$ high access time in receiver

1-bit cache covert channels

ideas for 1-bit channels:

- Prime+Probe: use one cache set to transmit

  0: sender does not access the set → low access time in receiver
  1: sender does access the set → high access time in receiver

- Flush+Reload/Flush+Flush/Evict+Reload: use one address to transmit

  0: sender does not access the address → high access time in receiver
  1: sender does access the address → low access time in receiver

1-bit covert channels

- 1 bit data, 0 bit control?

1-bit covert channels

- 1 bit data, 0 bit control?
- idea: divide time into slices (e.g., $50\mu s$ frames)
- synchronize sender and receiver with a shared clock

Problems of 1-bit covert channels

- errors?

Problems of 1-bit covert channels

- errors? $\rightarrow$ error-correcting codes
- retransmission may be more efficient (less overhead)
- desynchronization
- optimal transmission duration may vary

Multi-bit covert channels

- combine multiple 1-bit channels

Multi-bit covert channels

- combine multiple 1-bit channels
- avoid interferences

$\rightarrow$ higher performance

Multi-bit covert channels

- combine multiple 1-bit channels
- avoid interferences

→ higher performance

- use 1-bit for sending = true/false

Packets / frames

Organize data in packets / frames:

- some data bits
- check sum
- sequence number

→ keep sender and receiver synchronous

→ check whether retransmission is necessary

State of the art

method	raw capacity	err. rate	true capacity	env.
F+F Gruss, Maurice, Wagner, et al. 2016	3968Kbps	0.840%	3690Kbps	native
F+R Gruss, Maurice, Wagner, et al. 2016	2384Kbps	0.005%	2382Kbps	native
E+R Lipp et al. 2016	1141Kbps	1.100%	1041Kbps	native
P+P Liu et al. 2015	600Kbps	1.000%	552Kbps	virt

Cache Template Attacks

Profiling Phase

- Preprocessing step to find exploitable addresses automatically
  - w.r.t. “events” (keystrokes, encryptions, ...)
  - called “Cache Template”

Cache Template Attacks

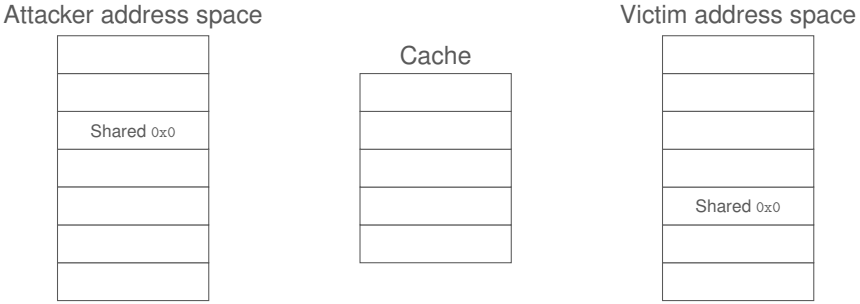
Profiling Phase

- Preprocessing step to find exploitable addresses automatically
  - w.r.t. “events” (keystrokes, encryptions, ...)
  - called “Cache Template”

Exploitation Phase

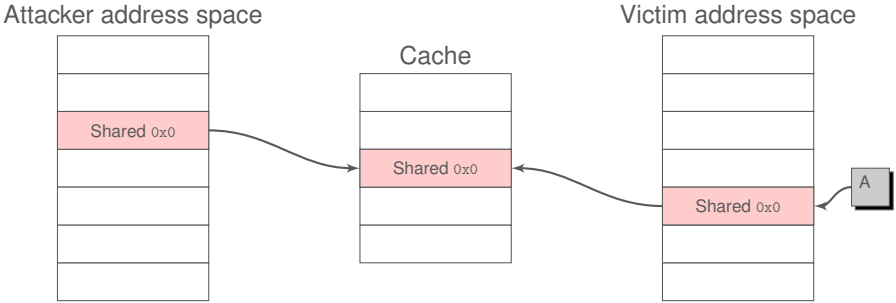
- Monitor exploitable addresses

Profiling Phase



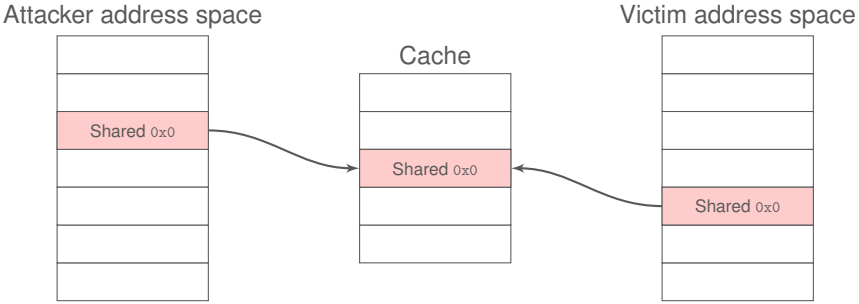
Cache is empty

Profiling Phase



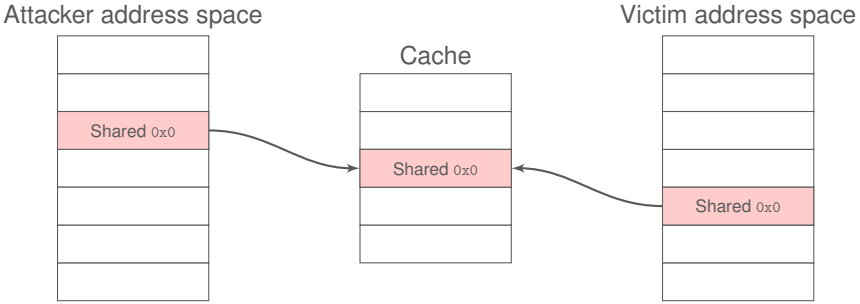
Attacker triggers an event

Profiling Phase



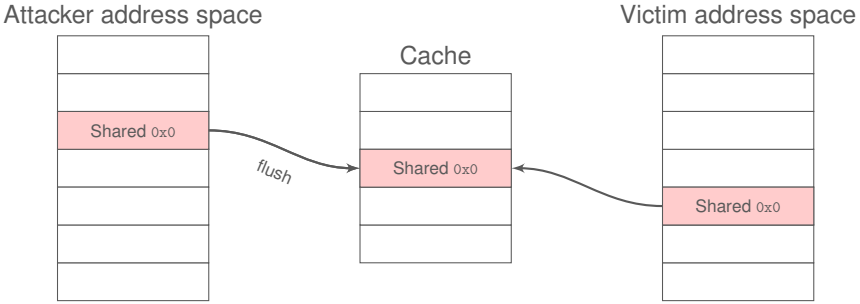
Attacker checks one address for cache hits (“Reload”)

Profiling Phase



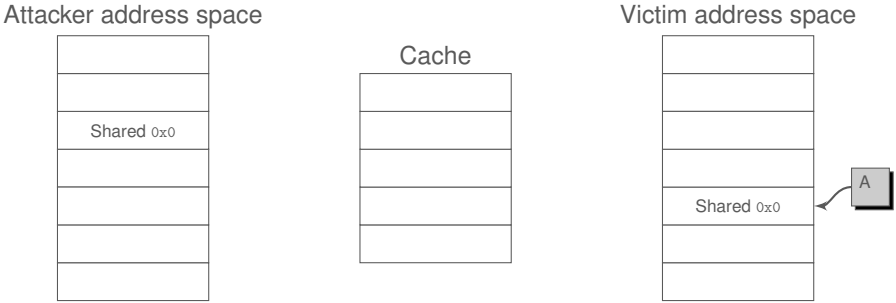
Update cache hit ratio (per event and address)

Profiling Phase



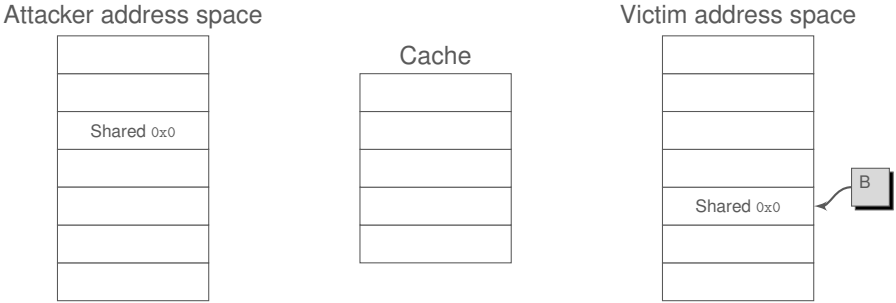
Attacker flushes shared memory

Profiling Phase



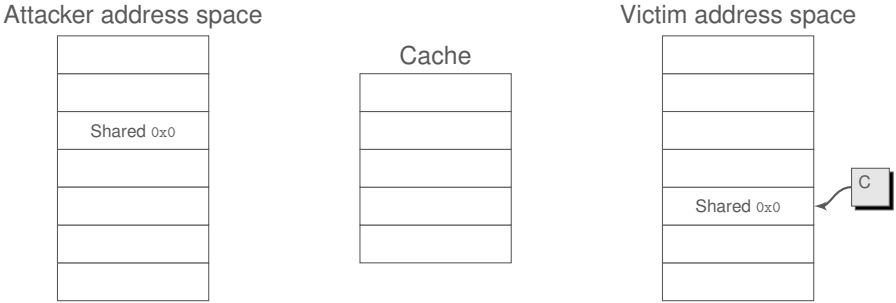
Repeat for higher accuracy

Profiling Phase



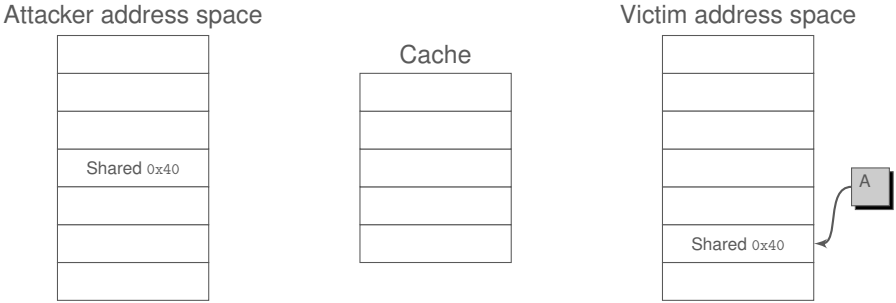
Repeat for all events

Profiling Phase



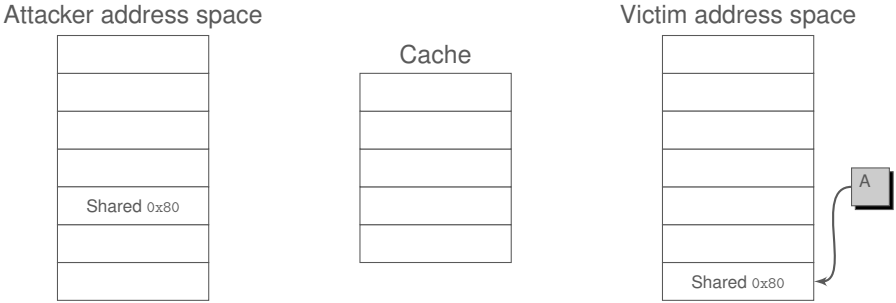
Repeat for all events

Profiling Phase



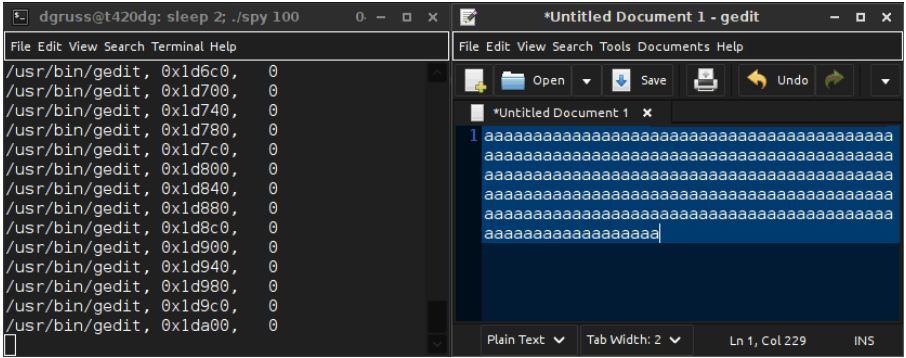
Continue with next address

Profiling Phase

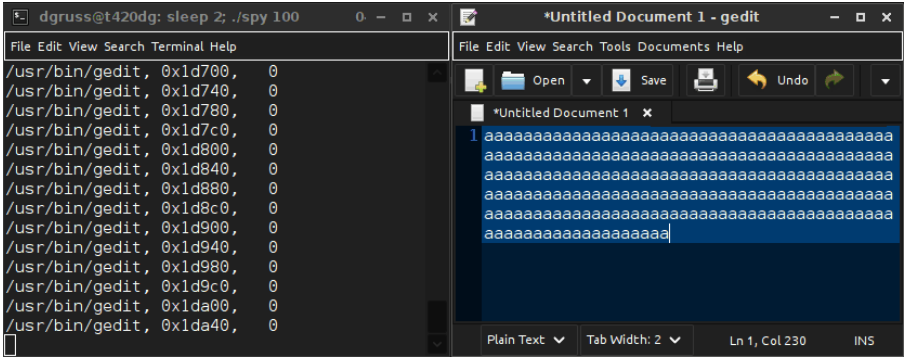


Continue with next address

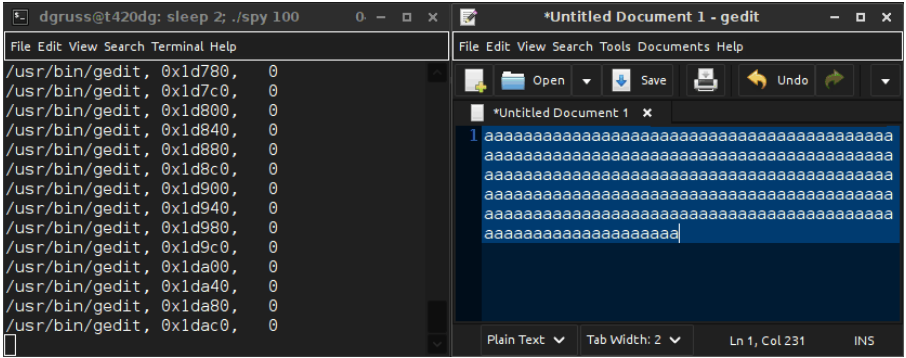
Profiling a Single Event



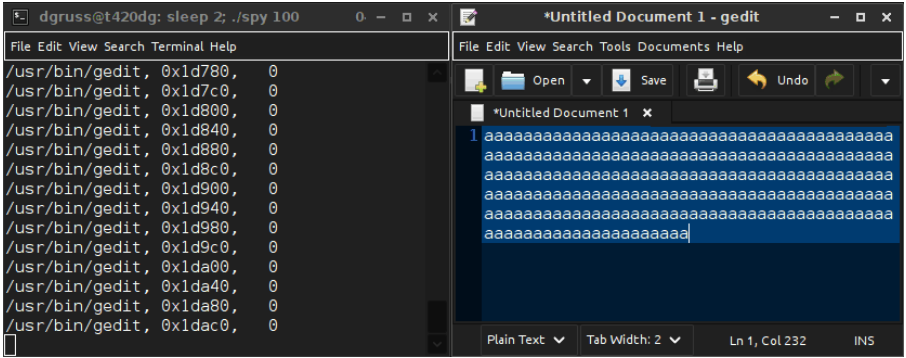
Profiling a Single Event



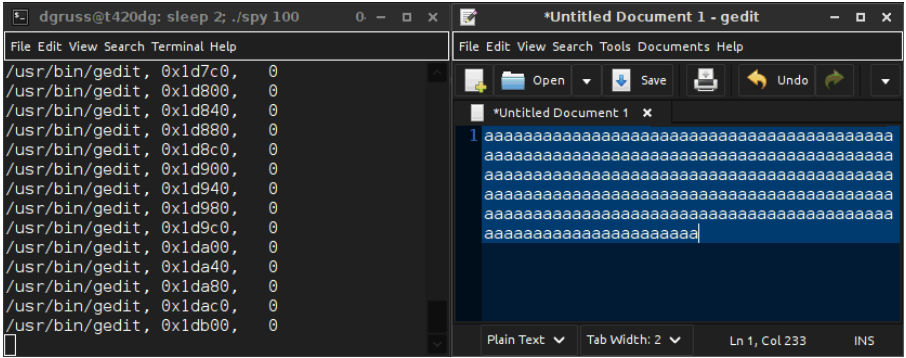
Profiling a Single Event



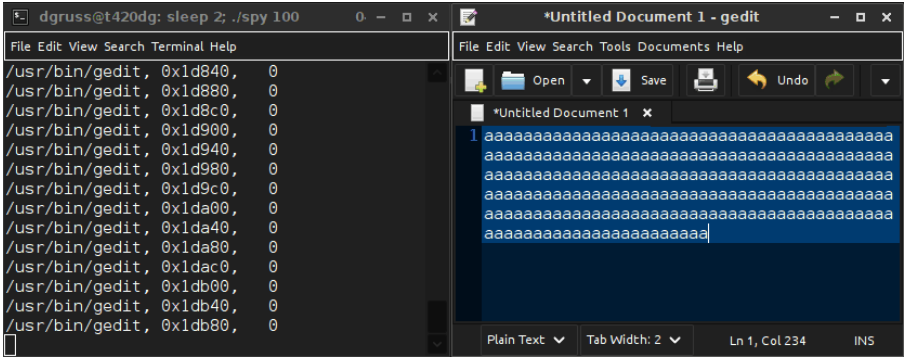
Profiling a Single Event



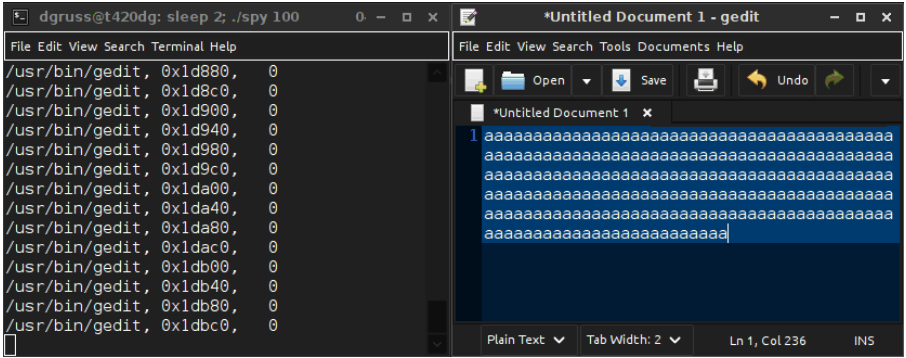
Profiling a Single Event



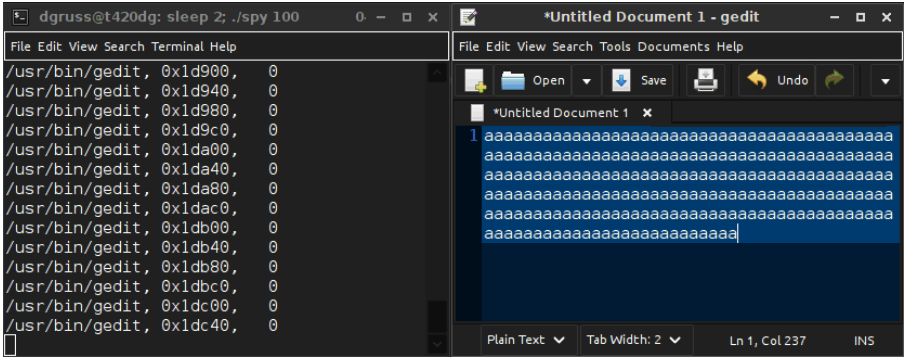
Profiling a Single Event



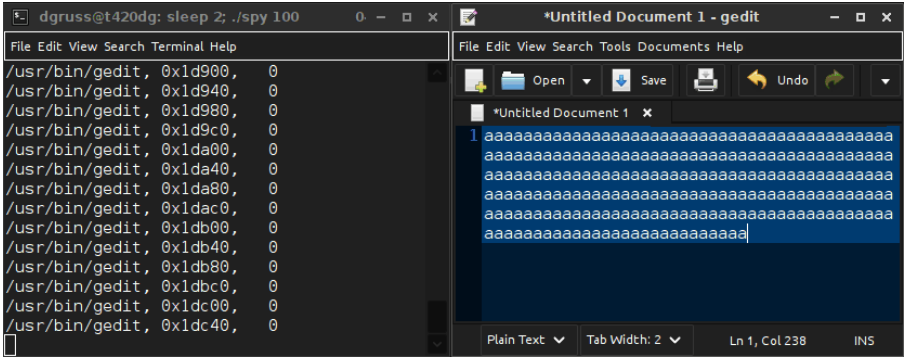
Profiling a Single Event



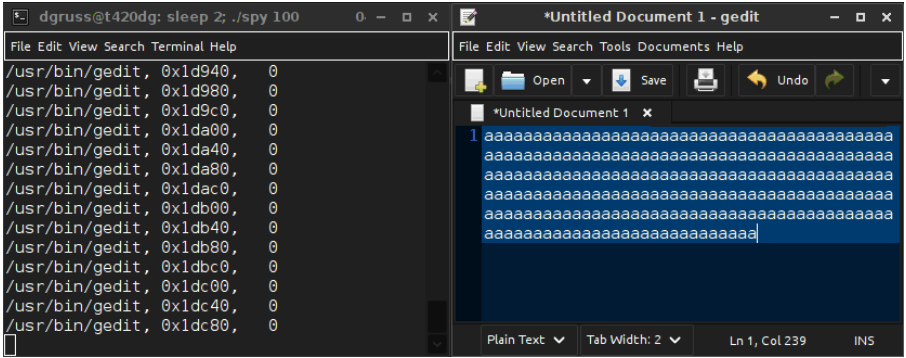
Profiling a Single Event



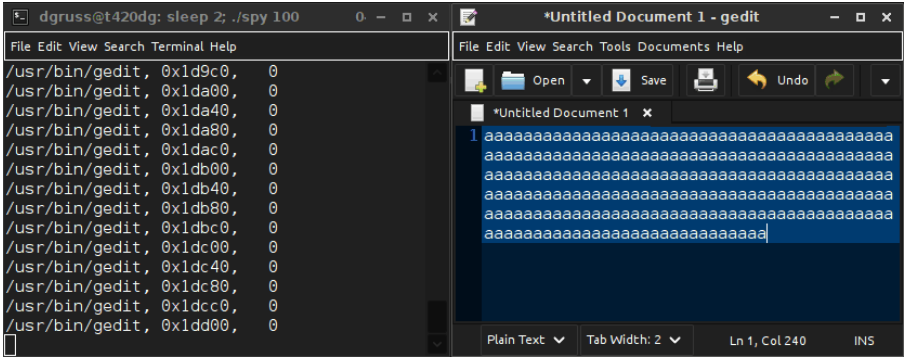
Profiling a Single Event



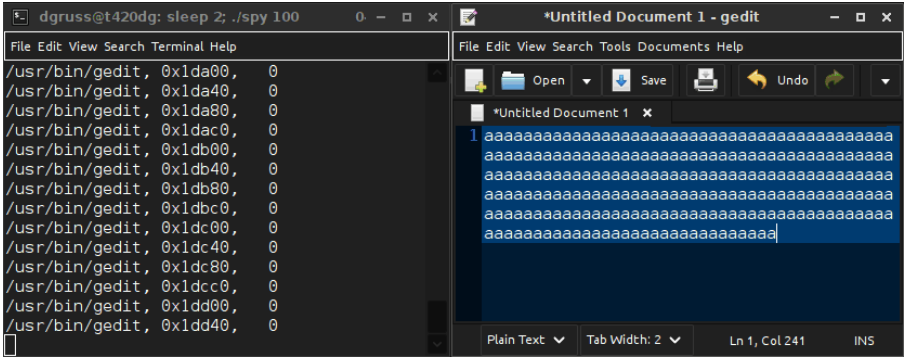
Profiling a Single Event



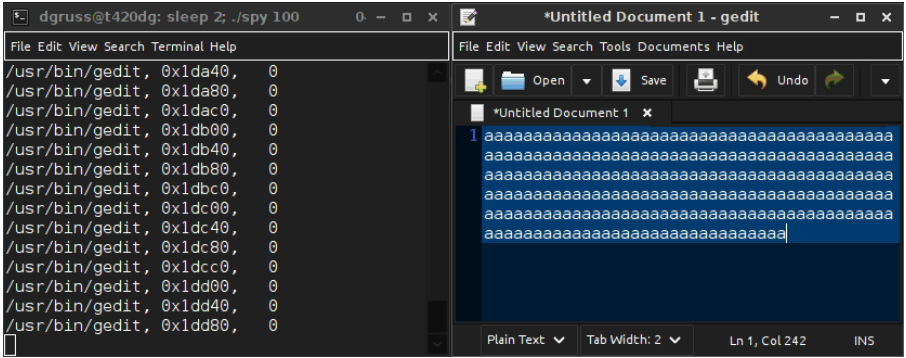
Profiling a Single Event



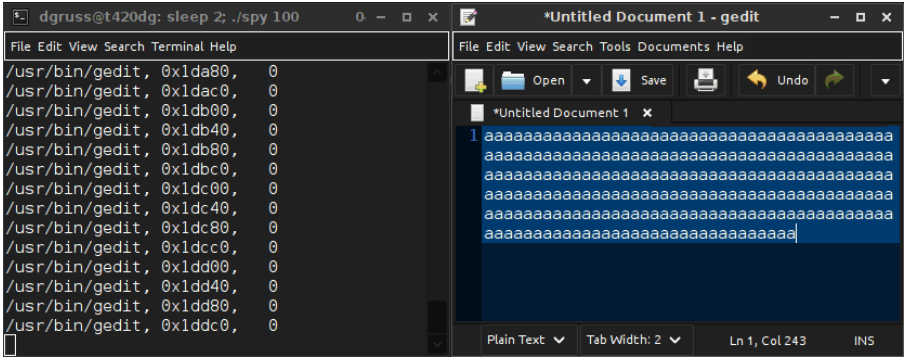
Profiling a Single Event



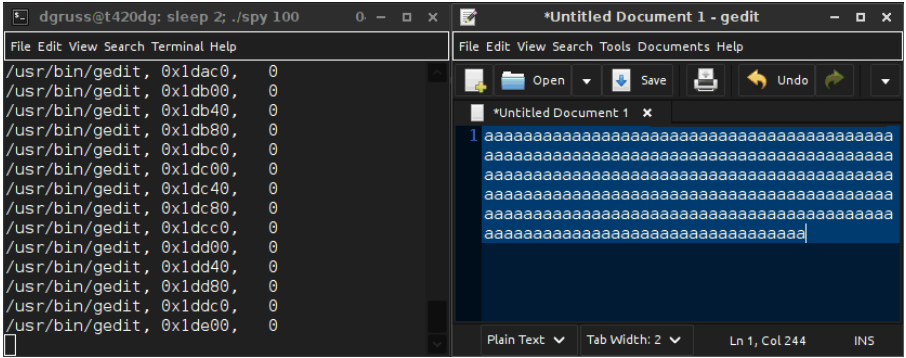
Profiling a Single Event



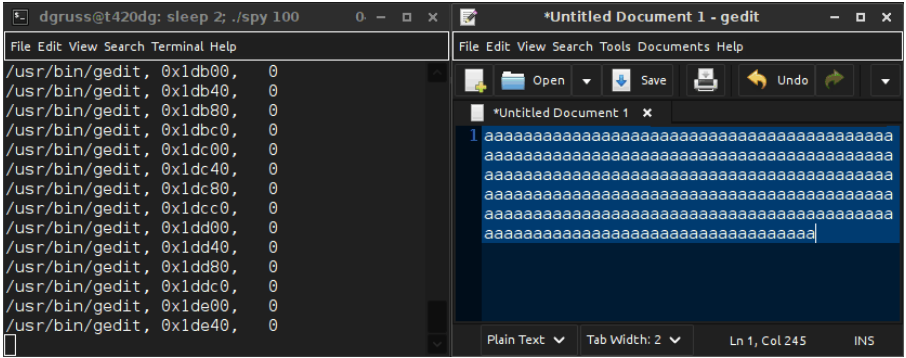
Profiling a Single Event



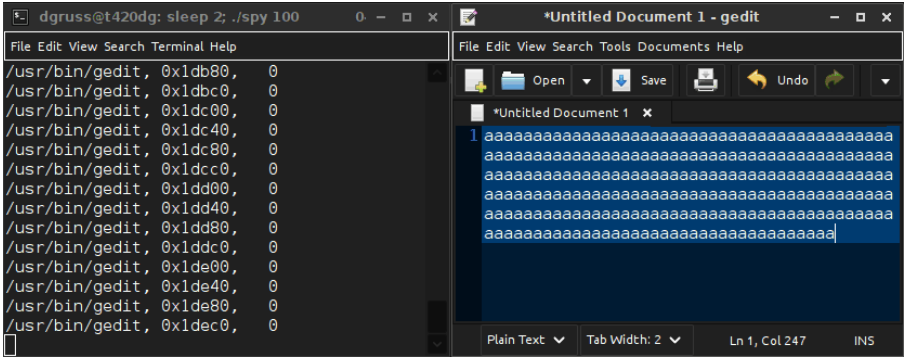
Profiling a Single Event



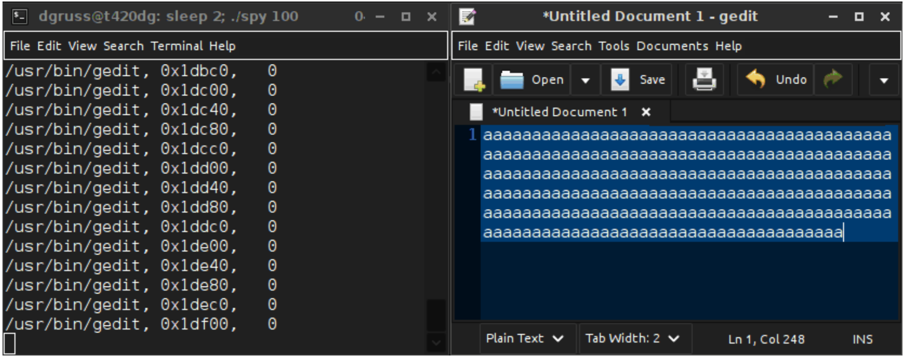
Profiling a Single Event



Profiling a Single Event



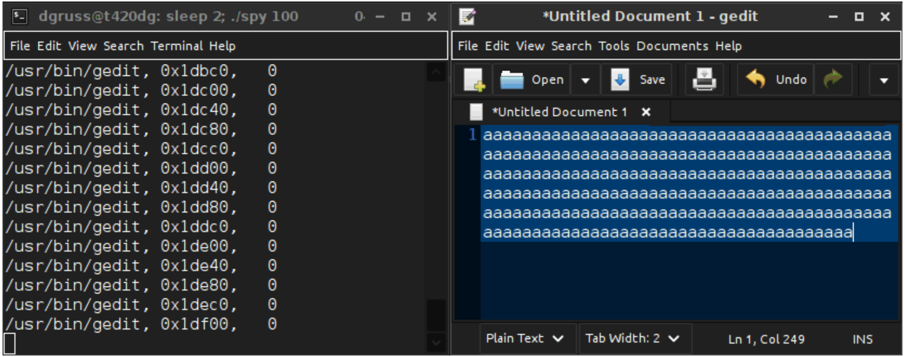
Profiling a Single Event



```
/usr/bin/gedit, 0x1dbc0,   0
/usr/bin/gedit, 0x1dc00,   0
/usr/bin/gedit, 0x1dc40,   0
/usr/bin/gedit, 0x1dc80,   0
/usr/bin/gedit, 0x1dcc0,   0
/usr/bin/gedit, 0x1dd00,   0
/usr/bin/gedit, 0x1dd40,   0
/usr/bin/gedit, 0x1dd80,   0
/usr/bin/gedit, 0x1ddc0,   0
/usr/bin/gedit, 0x1de00,   0
/usr/bin/gedit, 0x1de40,   0
/usr/bin/gedit, 0x1de80,   0
/usr/bin/gedit, 0x1dec0,   0
/usr/bin/gedit, 0x1df00,   0
```

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

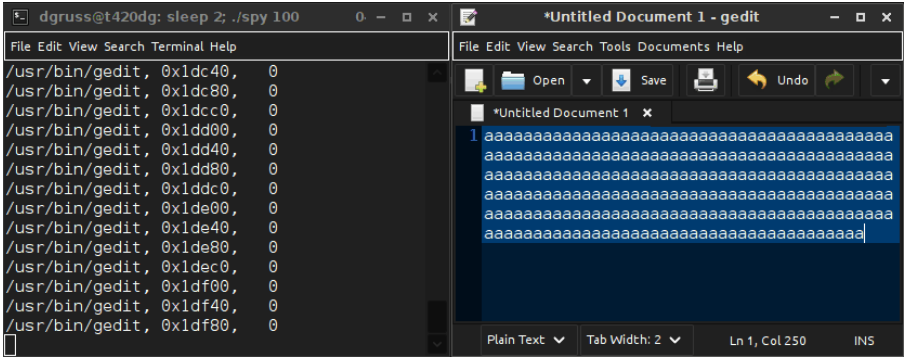
Profiling a Single Event



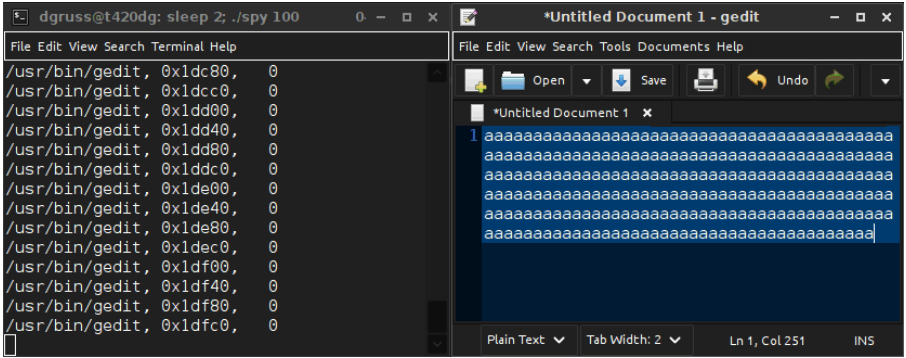
```
/usr/bin/gedit, 0x1dbc0,    0
/usr/bin/gedit, 0x1dc00,    0
/usr/bin/gedit, 0x1dc40,    0
/usr/bin/gedit, 0x1dc80,    0
/usr/bin/gedit, 0x1dcc0,    0
/usr/bin/gedit, 0x1dd00,    0
/usr/bin/gedit, 0x1dd40,    0
/usr/bin/gedit, 0x1dd80,    0
/usr/bin/gedit, 0x1ddc0,    0
/usr/bin/gedit, 0x1de00,    0
/usr/bin/gedit, 0x1de40,    0
/usr/bin/gedit, 0x1de80,    0
/usr/bin/gedit, 0x1dec0,    0
/usr/bin/gedit, 0x1df00,    0
```

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

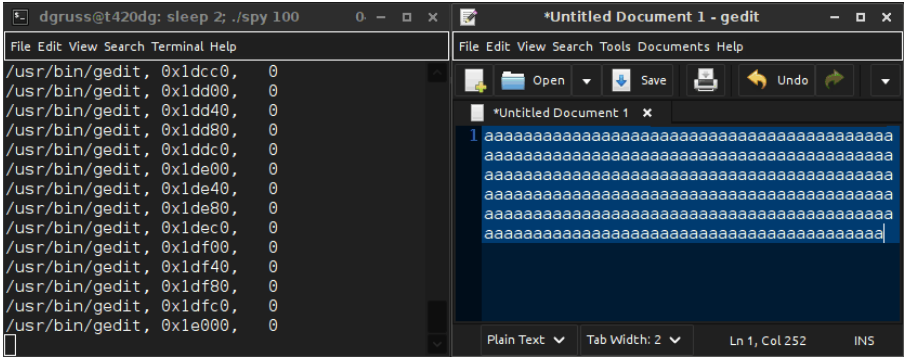
Profiling a Single Event



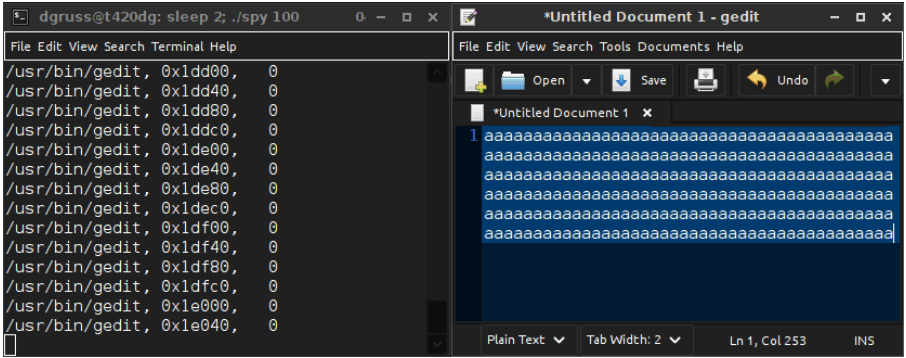
Profiling a Single Event



Profiling a Single Event

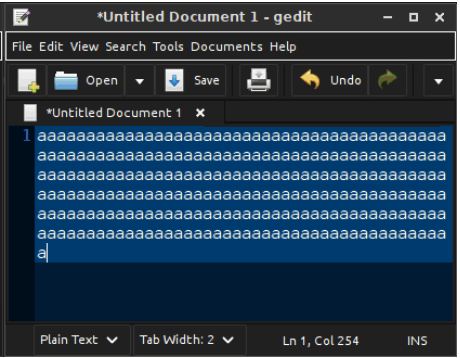


Profiling a Single Event

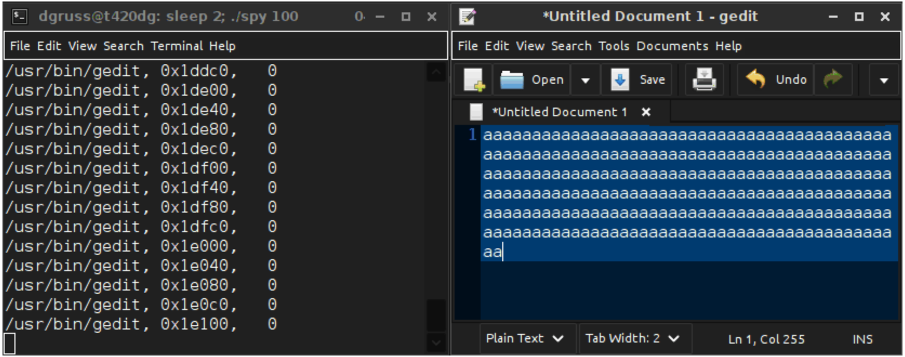


Profiling a Single Event

```
dgruss@t420dg: sleep 2; ./spy 100
File Edit View Search Terminal Help
/usr/bin/gedit, 0x1dd40,   0
/usr/bin/gedit, 0x1dd80,   0
/usr/bin/gedit, 0x1ddc0,   0
/usr/bin/gedit, 0x1de00,   0
/usr/bin/gedit, 0x1de40,   0
/usr/bin/gedit, 0x1de80,   0
/usr/bin/gedit, 0x1dec0,   0
/usr/bin/gedit, 0x1df00,   0
/usr/bin/gedit, 0x1df40,   0
/usr/bin/gedit, 0x1df80,   0
/usr/bin/gedit, 0x1dfc0,   0
/usr/bin/gedit, 0x1e000,   0
/usr/bin/gedit, 0x1e040,   0
/usr/bin/gedit, 0x1e080,   0
```



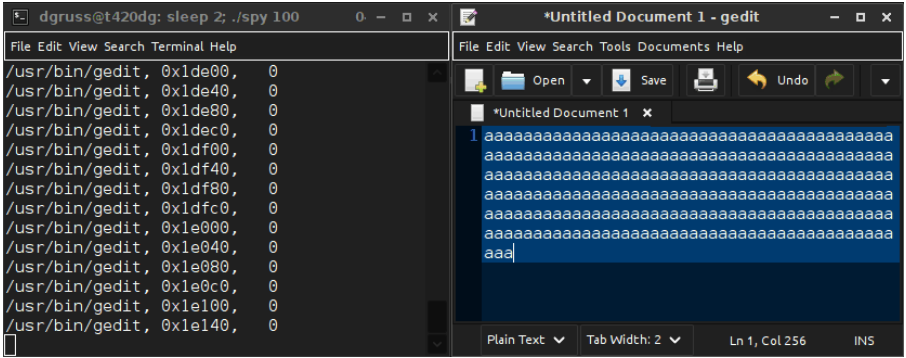
Profiling a Single Event



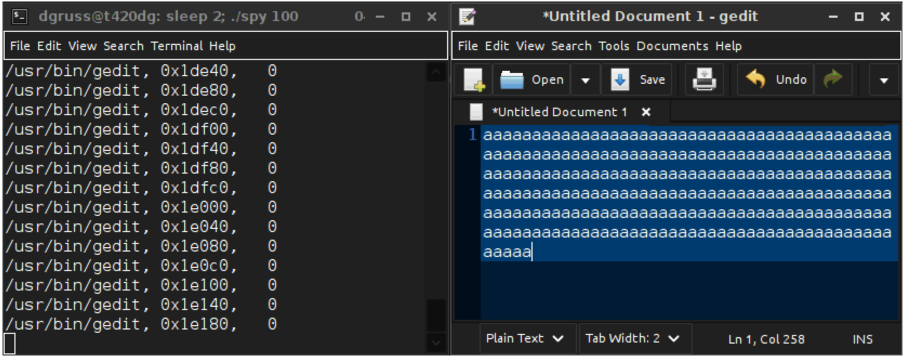
```
/usr/bin/gedit, 0x1ddc0,    0
/usr/bin/gedit, 0x1de00,    0
/usr/bin/gedit, 0x1de40,    0
/usr/bin/gedit, 0x1de80,    0
/usr/bin/gedit, 0x1dec0,    0
/usr/bin/gedit, 0x1df00,    0
/usr/bin/gedit, 0x1df40,    0
/usr/bin/gedit, 0x1df80,    0
/usr/bin/gedit, 0x1dfc0,    0
/usr/bin/gedit, 0x1e000,    0
/usr/bin/gedit, 0x1e040,    0
/usr/bin/gedit, 0x1e080,    0
/usr/bin/gedit, 0x1e0c0,    0
/usr/bin/gedit, 0x1e100,    0
```

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aa
```

Profiling a Single Event



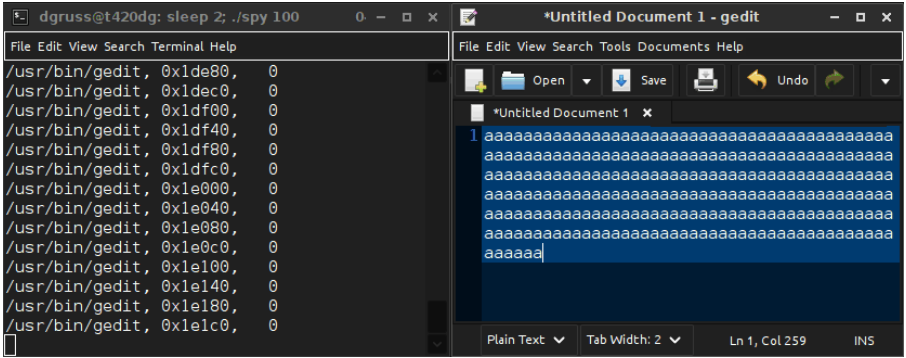
Profiling a Single Event



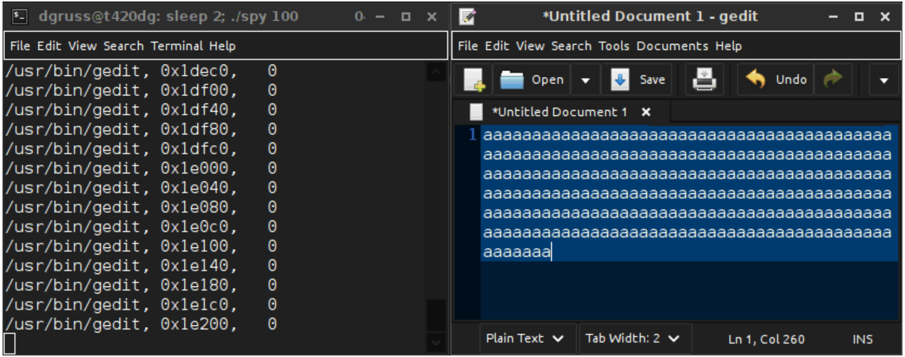
```
/usr/bin/gedit, 0x1de40,   0
/usr/bin/gedit, 0x1de80,   0
/usr/bin/gedit, 0x1dec0,   0
/usr/bin/gedit, 0x1df00,   0
/usr/bin/gedit, 0x1df40,   0
/usr/bin/gedit, 0x1df80,   0
/usr/bin/gedit, 0x1dfc0,   0
/usr/bin/gedit, 0x1e000,   0
/usr/bin/gedit, 0x1e040,   0
/usr/bin/gedit, 0x1e080,   0
/usr/bin/gedit, 0x1e0c0,   0
/usr/bin/gedit, 0x1e100,   0
/usr/bin/gedit, 0x1e140,   0
/usr/bin/gedit, 0x1e180,   0
```

```
1 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaa
```

Profiling a Single Event



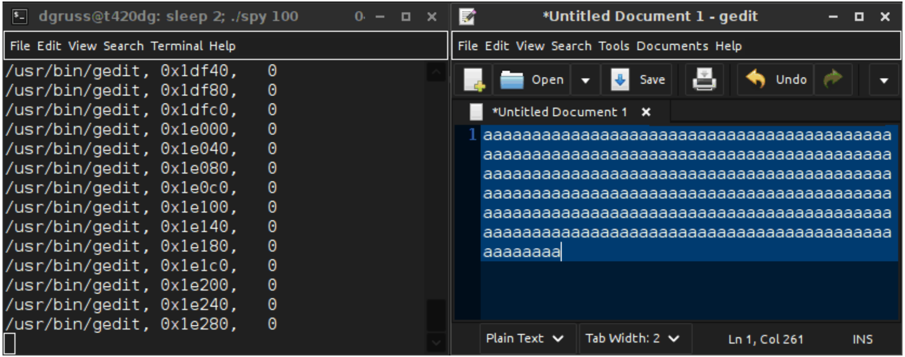
Profiling a Single Event



```
/usr/bin/gedit, 0x1dec0,    0
/usr/bin/gedit, 0x1df00,    0
/usr/bin/gedit, 0x1df40,    0
/usr/bin/gedit, 0x1df80,    0
/usr/bin/gedit, 0x1dfc0,    0
/usr/bin/gedit, 0x1e000,    0
/usr/bin/gedit, 0x1e040,    0
/usr/bin/gedit, 0x1e080,    0
/usr/bin/gedit, 0x1e0c0,    0
/usr/bin/gedit, 0x1e100,    0
/usr/bin/gedit, 0x1e140,    0
/usr/bin/gedit, 0x1e180,    0
/usr/bin/gedit, 0x1e1c0,    0
/usr/bin/gedit, 0x1e200,    0
```

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaa
```

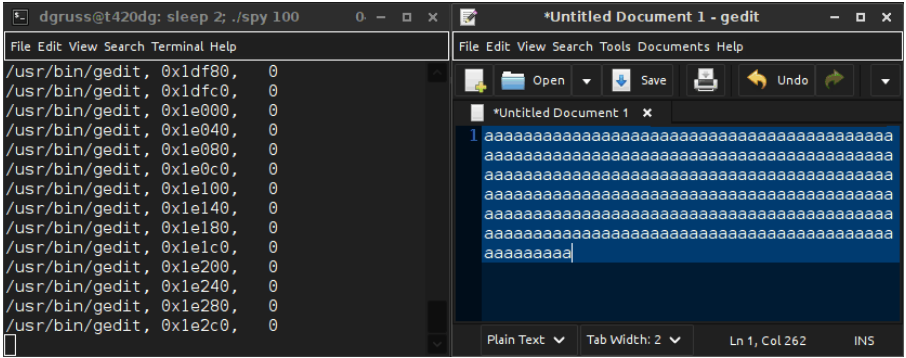
Profiling a Single Event



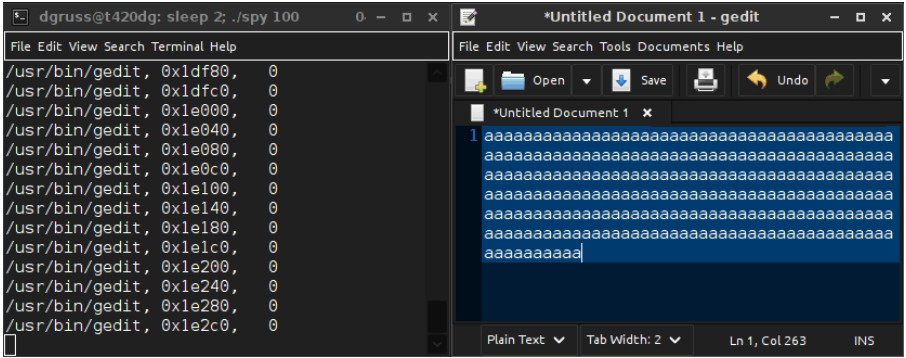
```
/usr/bin/gedit, 0x1df40,    0
/usr/bin/gedit, 0x1df80,    0
/usr/bin/gedit, 0x1dfc0,    0
/usr/bin/gedit, 0x1e000,    0
/usr/bin/gedit, 0x1e040,    0
/usr/bin/gedit, 0x1e080,    0
/usr/bin/gedit, 0x1e0c0,    0
/usr/bin/gedit, 0x1e100,    0
/usr/bin/gedit, 0x1e140,    0
/usr/bin/gedit, 0x1e180,    0
/usr/bin/gedit, 0x1e1c0,    0
/usr/bin/gedit, 0x1e200,    0
/usr/bin/gedit, 0x1e240,    0
/usr/bin/gedit, 0x1e280,    0
```

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaa
```

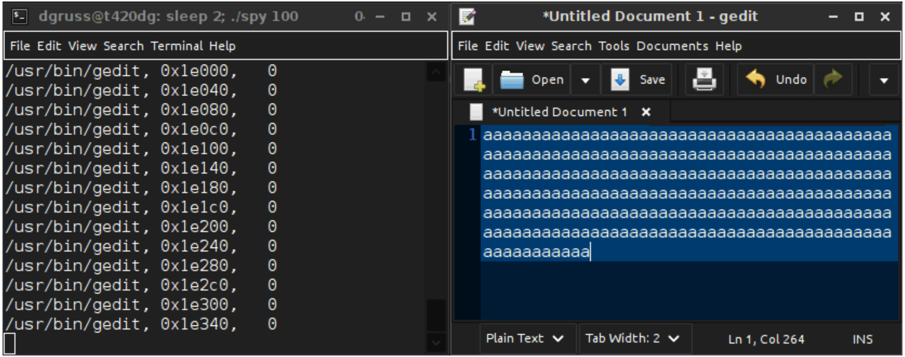
Profiling a Single Event



Profiling a Single Event



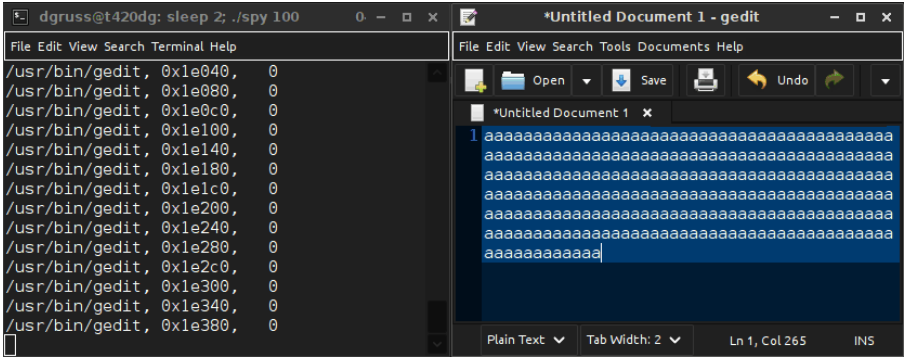
Profiling a Single Event



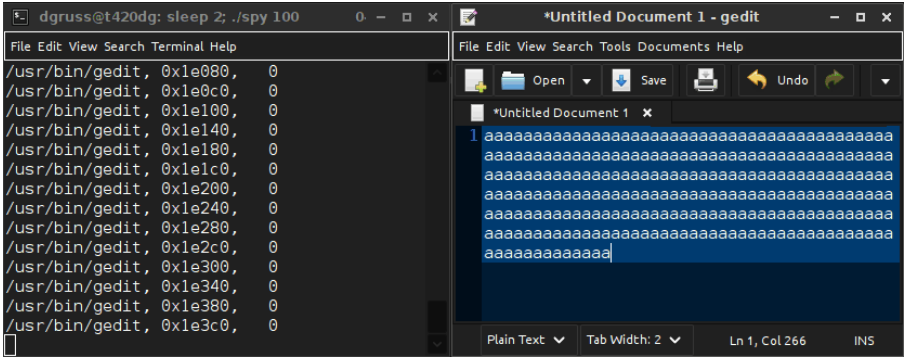
```
/usr/bin/gedit, 0x1e000,   0
/usr/bin/gedit, 0x1e040,   0
/usr/bin/gedit, 0x1e080,   0
/usr/bin/gedit, 0x1e0c0,   0
/usr/bin/gedit, 0x1e100,   0
/usr/bin/gedit, 0x1e140,   0
/usr/bin/gedit, 0x1e180,   0
/usr/bin/gedit, 0x1e1c0,   0
/usr/bin/gedit, 0x1e200,   0
/usr/bin/gedit, 0x1e240,   0
/usr/bin/gedit, 0x1e280,   0
/usr/bin/gedit, 0x1e2c0,   0
/usr/bin/gedit, 0x1e300,   0
/usr/bin/gedit, 0x1e340,   0
```

```
1 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaa
```

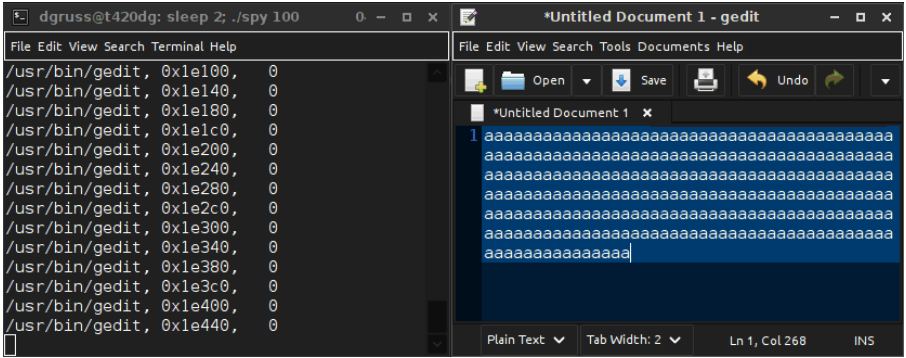
Profiling a Single Event



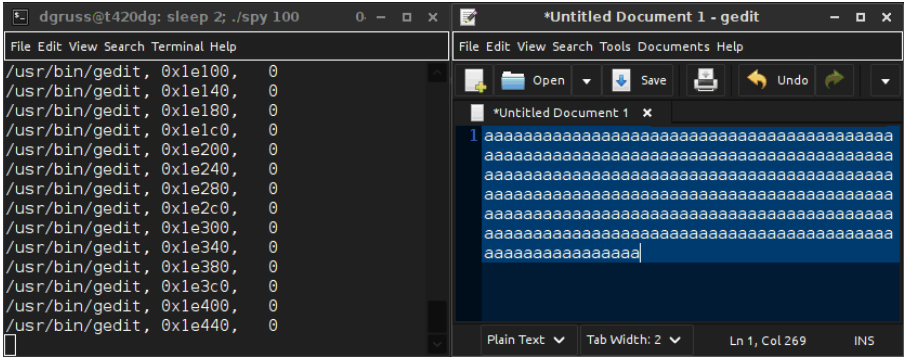
Profiling a Single Event



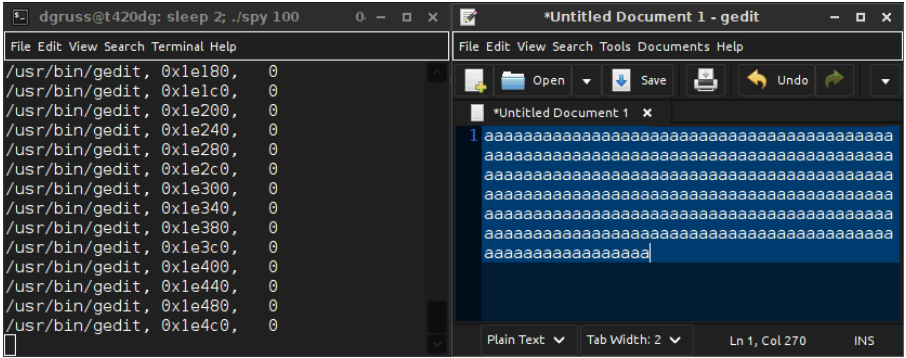
Profiling a Single Event



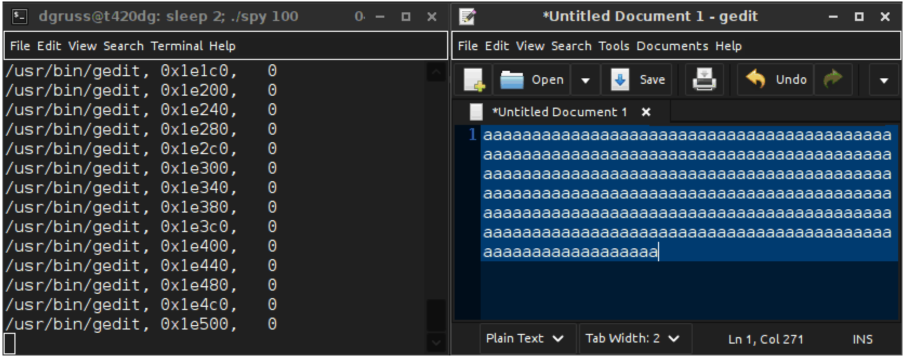
Profiling a Single Event



Profiling a Single Event



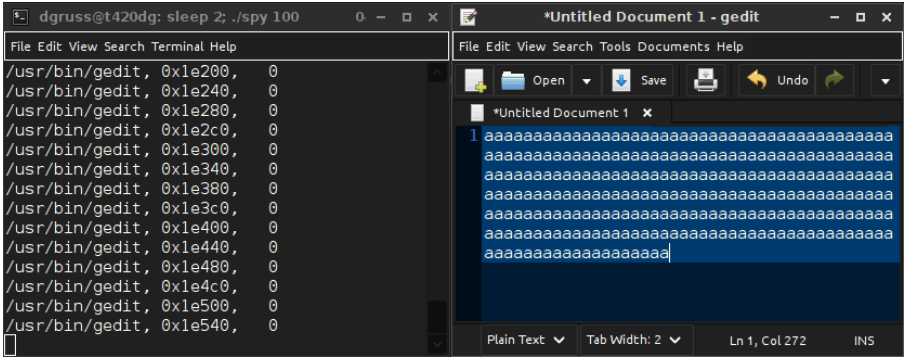
Profiling a Single Event



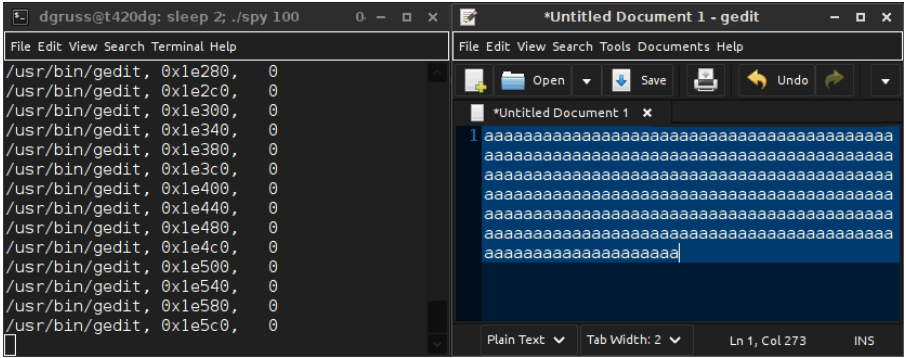
```
/usr/bin/gedit, 0x1e1c0,    0
/usr/bin/gedit, 0x1e200,    0
/usr/bin/gedit, 0x1e240,    0
/usr/bin/gedit, 0x1e280,    0
/usr/bin/gedit, 0x1e2c0,    0
/usr/bin/gedit, 0x1e300,    0
/usr/bin/gedit, 0x1e340,    0
/usr/bin/gedit, 0x1e380,    0
/usr/bin/gedit, 0x1e3c0,    0
/usr/bin/gedit, 0x1e400,    0
/usr/bin/gedit, 0x1e440,    0
/usr/bin/gedit, 0x1e480,    0
/usr/bin/gedit, 0x1e4c0,    0
/usr/bin/gedit, 0x1e500,    0
```

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaa
```

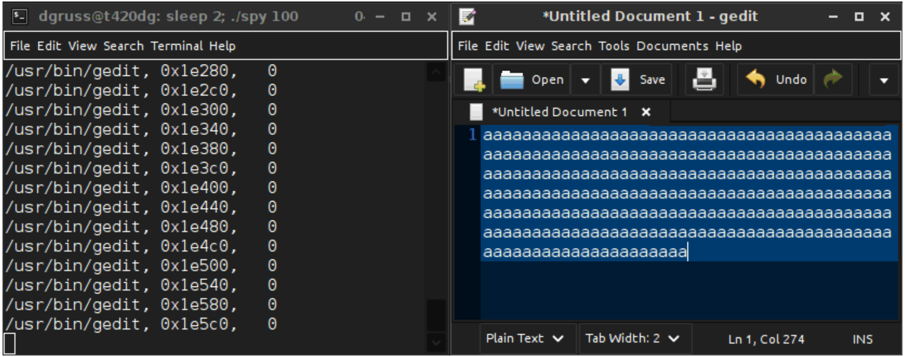
Profiling a Single Event



Profiling a Single Event



Profiling a Single Event

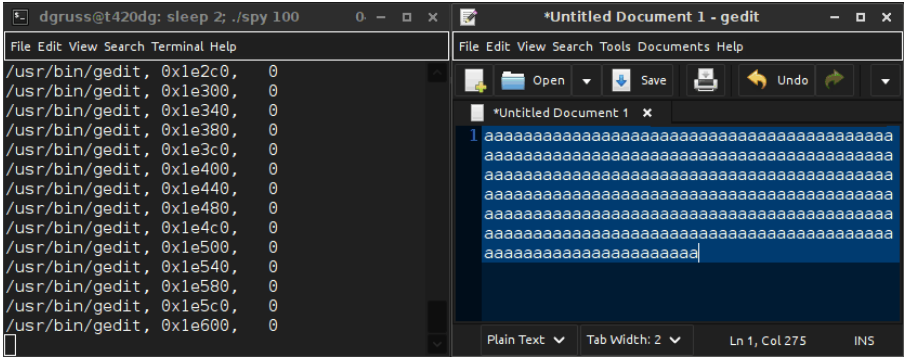


```
/usr/bin/gedit, 0x1e280,   0
/usr/bin/gedit, 0x1e2c0,   0
/usr/bin/gedit, 0x1e300,   0
/usr/bin/gedit, 0x1e340,   0
/usr/bin/gedit, 0x1e380,   0
/usr/bin/gedit, 0x1e3c0,   0
/usr/bin/gedit, 0x1e400,   0
/usr/bin/gedit, 0x1e440,   0
/usr/bin/gedit, 0x1e480,   0
/usr/bin/gedit, 0x1e4c0,   0
/usr/bin/gedit, 0x1e500,   0
/usr/bin/gedit, 0x1e540,   0
/usr/bin/gedit, 0x1e580,   0
/usr/bin/gedit, 0x1e5c0,   0
```

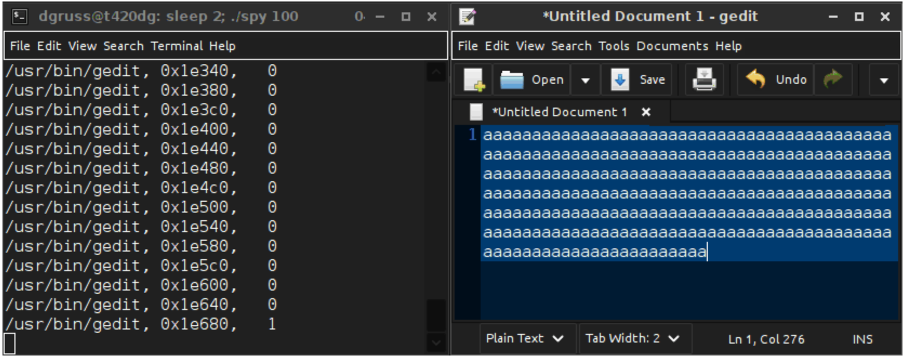
```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaa
```

Plain Text
Tab Width: 2
Ln 1, Col 274
INS

Profiling a Single Event



Profiling a Single Event

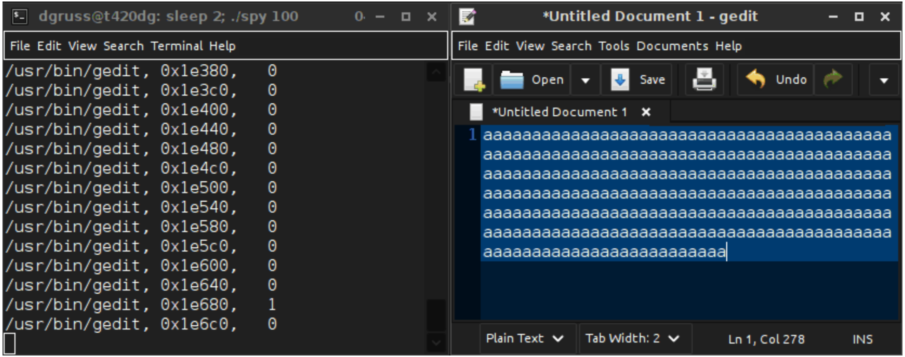


```
/usr/bin/gedit, 0x1e340,   0
/usr/bin/gedit, 0x1e380,   0
/usr/bin/gedit, 0x1e3c0,   0
/usr/bin/gedit, 0x1e400,   0
/usr/bin/gedit, 0x1e440,   0
/usr/bin/gedit, 0x1e480,   0
/usr/bin/gedit, 0x1e4c0,   0
/usr/bin/gedit, 0x1e500,   0
/usr/bin/gedit, 0x1e540,   0
/usr/bin/gedit, 0x1e580,   0
/usr/bin/gedit, 0x1e5c0,   0
/usr/bin/gedit, 0x1e600,   0
/usr/bin/gedit, 0x1e640,   0
/usr/bin/gedit, 0x1e680,   1
```

```
1 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaa
```

Plain Text ˅ Tab Width: 2 ˅ Ln 1, Col 276 INS

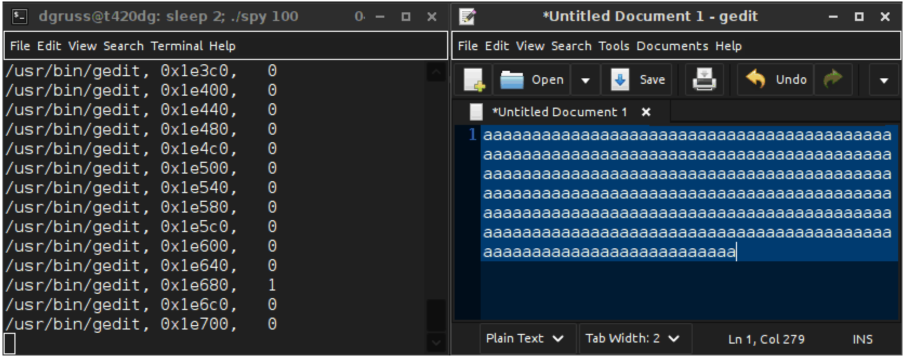
Profiling a Single Event



```
/usr/bin/gedit, 0x1e380,   0
/usr/bin/gedit, 0x1e3c0,   0
/usr/bin/gedit, 0x1e400,   0
/usr/bin/gedit, 0x1e440,   0
/usr/bin/gedit, 0x1e480,   0
/usr/bin/gedit, 0x1e4c0,   0
/usr/bin/gedit, 0x1e500,   0
/usr/bin/gedit, 0x1e540,   0
/usr/bin/gedit, 0x1e580,   0
/usr/bin/gedit, 0x1e5c0,   0
/usr/bin/gedit, 0x1e600,   0
/usr/bin/gedit, 0x1e640,   0
/usr/bin/gedit, 0x1e680,   1
/usr/bin/gedit, 0x1e6c0,   0
```

```
1 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaa
```

Profiling a Single Event



```
/usr/bin/gedit, 0x1e3c0,   0
/usr/bin/gedit, 0x1e400,   0
/usr/bin/gedit, 0x1e440,   0
/usr/bin/gedit, 0x1e480,   0
/usr/bin/gedit, 0x1e4c0,   0
/usr/bin/gedit, 0x1e500,   0
/usr/bin/gedit, 0x1e540,   0
/usr/bin/gedit, 0x1e580,   0
/usr/bin/gedit, 0x1e5c0,   0
/usr/bin/gedit, 0x1e600,   0
/usr/bin/gedit, 0x1e640,   0
/usr/bin/gedit, 0x1e680,   1
/usr/bin/gedit, 0x1e6c0,   0
/usr/bin/gedit, 0x1e700,   0
```

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaa
```

Profiling a Single Event

dgruss@t420dg: sleep 2; ./spy 100

File Edit View Search Terminal Help

```
/usr/bin/gedit, 0x1e400,   0
/usr/bin/gedit, 0x1e440,   0
/usr/bin/gedit, 0x1e480,   0
/usr/bin/gedit, 0x1e4c0,   0
/usr/bin/gedit, 0x1e500,   0
/usr/bin/gedit, 0x1e540,   0
/usr/bin/gedit, 0x1e580,   0
/usr/bin/gedit, 0x1e5c0,   0
/usr/bin/gedit, 0x1e600,   0
/usr/bin/gedit, 0x1e640,   0
/usr/bin/gedit, 0x1e680,   1
/usr/bin/gedit, 0x1e6c0,   0
/usr/bin/gedit, 0x1e700,   0
/usr/bin/gedit, 0x1e740,   0
```

*Untitled Document 1 - gedit

File Edit View Search Tools Documents Help

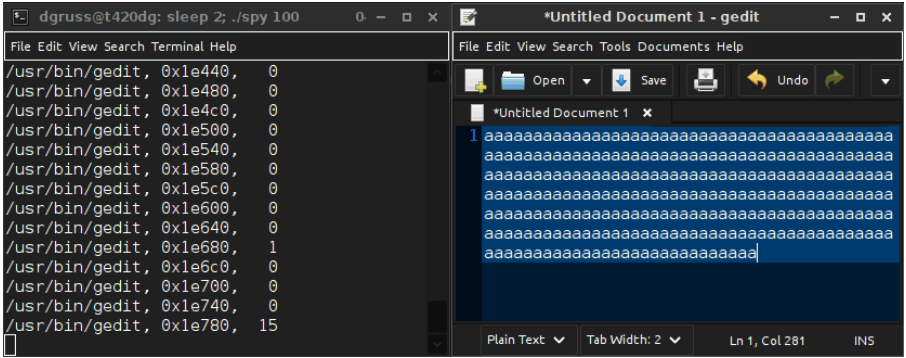
Open Save Undo

*Untitled Document 1

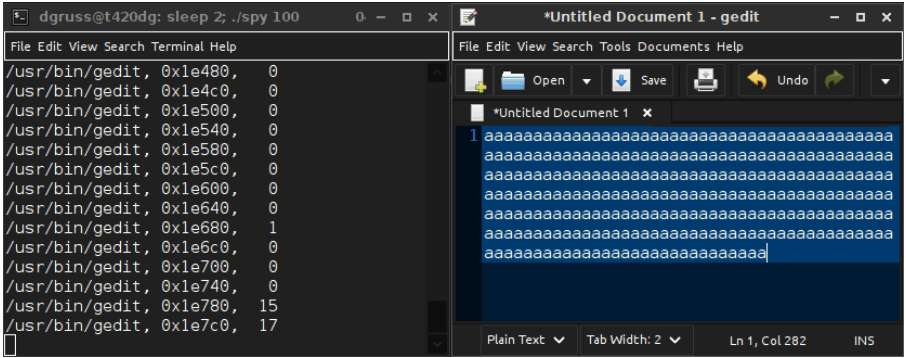
```
1 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
  aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
```

Plain Text Tab Width: 2 Ln 1, Col 280 INS

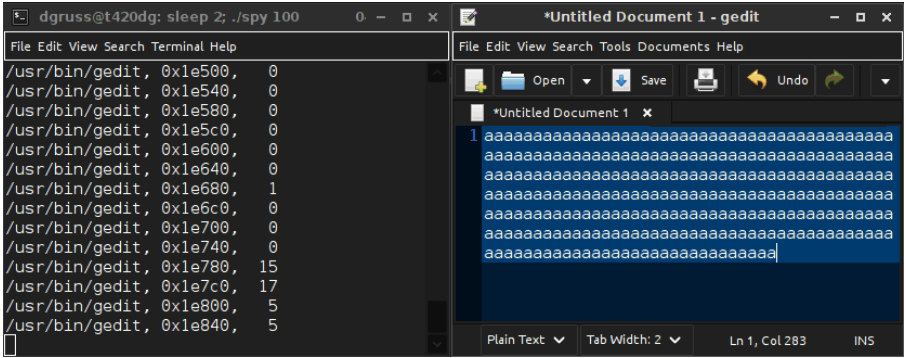
Profiling a Single Event



Profiling a Single Event



Profiling a Single Event



Profiling a Single Event

dgruss@t420dg: sleep 2; ./spy 100

File Edit View Search Terminal Help

```
/usr/bin/gedit, 0x1e540,   0
/usr/bin/gedit, 0x1e580,   0
/usr/bin/gedit, 0x1e5c0,   0
/usr/bin/gedit, 0x1e600,   0
/usr/bin/gedit, 0x1e640,   0
/usr/bin/gedit, 0x1e680,   1
/usr/bin/gedit, 0x1e6c0,   0
/usr/bin/gedit, 0x1e700,   0
/usr/bin/gedit, 0x1e740,   0
/usr/bin/gedit, 0x1e780,  15
/usr/bin/gedit, 0x1e7c0,  17
/usr/bin/gedit, 0x1e800,   5
/usr/bin/gedit, 0x1e840,   5
/usr/bin/gedit, 0x1e880,   0
```

*Untitled Document 1 - gedit

File Edit View Search Tools Documents Help

Open Save Undo

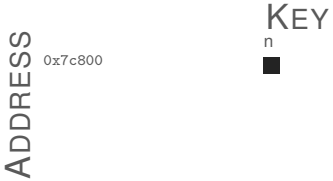
*Untitled Document 1

Plain Text Tab Width: 2 Ln 1, Col 284 INS

Profiling Phase: 1 Event, 1 Address

Address	Key
0x7c800	n

Profiling Phase: 1 Event, 1 Address



Example: Cache Hit Ratio for $(\texttt{0x7c800}, \texttt{n})$: 200 / 200

Profiling Phase: All Events, 1 Address

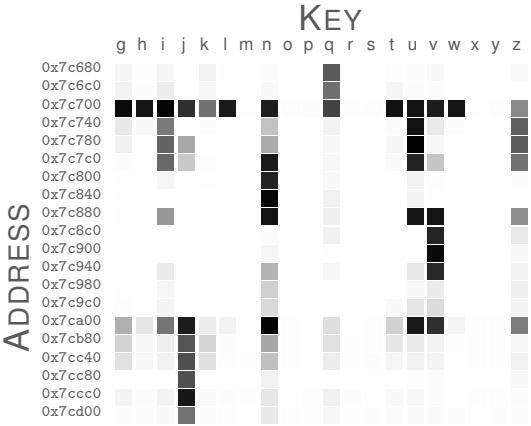
Profiling Phase: All Events, 1 Address

Example: Cache Hit Ratio for $(\texttt{0x7c800}, \texttt{u})$: 13 / 200

Profiling Phase: All Events, 1 Address

Distinguish `n` from other keys by monitoring `0x7c800`

Profiling Phase: All Events, All Addresses



Exploitation Phase

- Monitor addresses from Cache Template

Exploitation Phase

- Monitor addresses from Cache Template
- Report to log file / attacker

Exploitation Phase

- Monitor addresses from Cache Template
- Report to log file / attacker
- Manual analysis of log file
  - Find password in keypress log, etc.

Example Attacks

Attack 1: Keystroke Timings

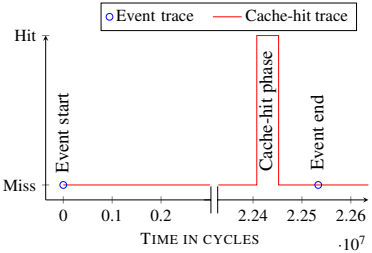
- Spy on keystroke timings on Linux, Windows and OS X

Attack 1: Keystroke Timings

- Spy on keystroke timings on Linux, Windows and OS X
- Sub-microsecond accuracy

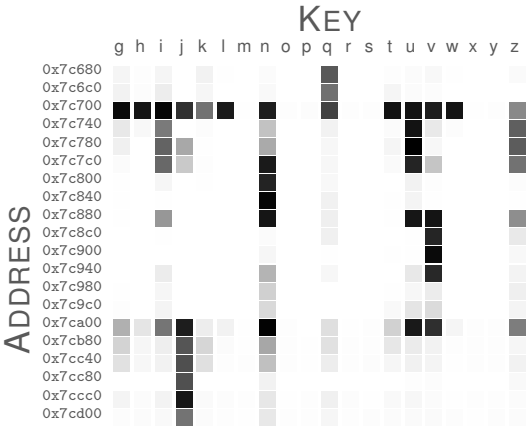
Attack 1: Keystroke Timings

- Spy on keystroke timings on Linux, Windows and OS X
- Sub-microsecond accuracy
- Derive text input from timings



Attack 2: Keylogging

- Linux with GTK: monitor keystrokes of specific keys
- Detect groups of keys
- Some keys distinct



Attack 3: Locate AES T-Tables

AES uses T-Tables (precomputed from S-Boxes)

- 4 T-Tables
- $$T_0\left[k_{\{0,4,8,12\}} \oplus p_{\{0,4,8,12\}}\right]$$
  $$T_1\left[k_{\{1,5,9,13\}} \oplus p_{\{1,5,9,13\}}\right]$$
  ...
- If we know which entry of T is accessed, we know the result of $k_i \oplus p_i$.
- Known-plaintext attack (p_i is known) $\rightarrow$ k_i can be determined

Attack 3: Locate AES T-Tables

AES T-Table implementation from OpenSSL 1.0.2

Attack 3: Locate AES T-Tables

AES T-Table implementation from OpenSSL 1.0.2

- Most addresses in two groups:
  - Cache hit ratio 100% (always cache hits)
  - Cache hit ratio 0% (no cache hits)

Attack 3: Locate AES T-Tables

AES T-Table implementation from OpenSSL 1.0.2

- Most addresses in two groups:
  - Cache hit ratio 100% (always cache hits)
  - Cache hit ratio 0% (no cache hits)
- One 4096 byte memory block:
  - Cache hit ratio of 92%
  - Cache hits depend on key value and plaintext value
  - The T-Tables

Attack 4: AES T-Table Template Attack

AES T-Table implementation from OpenSSL 1.0.2

- Known-plaintext attack
- Events: encryption with only one fixed key byte

Attack 4: AES T-Table Template Attack

AES T-Table implementation from OpenSSL 1.0.2

- Known-plaintext attack
- Events: encryption with only one fixed key byte
- Profile each event

Attack 4: AES T-Table Template Attack

AES T-Table implementation from OpenSSL 1.0.2

- Known-plaintext attack
- Events: encryption with only one fixed key byte
- Profile each event
- Exploitation phase:
  - Eliminate key candidates

Attack 4: AES T-Table Template Attack

AES T-Table implementation from OpenSSL 1.0.2

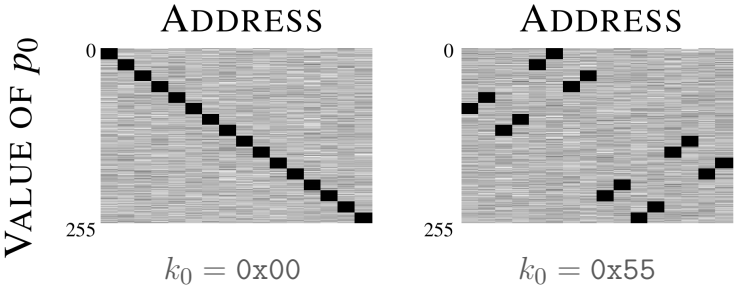
- Known-plaintext attack
- Events: encryption with only one fixed key byte
- Profile each event
- Exploitation phase:
  - Eliminate key candidates
  - Reduction of key space in first-round attack:
    - 64 bits after 16–160 encryptions

Attack 4: AES T-Table Template Attack

AES T-Table implementation from OpenSSL 1.0.2

- Known-plaintext attack
- Events: encryption with only one fixed key byte
- Profile each event
- Exploitation phase:
  - Eliminate key candidates
  - Reduction of key space in first-round attack:
    - 64 bits after 16–160 encryptions
  - State of the art: full key recovery after 30000 encryptions

Attack 4: AES T-Table Template



(transposed)

Conclusion

- Novel technique to find any cache side-channel leakage
  - Attacks
  - Detect vulnerabilities

Conclusion

- Novel technique to find any cache side-channel leakage
  - Attacks
  - Detect vulnerabilities
- Works on virtually all Intel CPUs
- Works even with unknown binaries

Conclusion

- Novel technique to find any cache side-channel leakage
  - Attacks
  - Detect vulnerabilities
- Works on virtually all Intel CPUs
- Works even with unknown binaries
- Marks a change of perspective:

Conclusion

- Novel technique to find any cache side-channel leakage
  - Attacks
  - Detect vulnerabilities
- Works on virtually all Intel CPUs
- Works even with unknown binaries
- Marks a change of perspective:
  - Large scale analysis of binaries
  - Large scale automated attacks

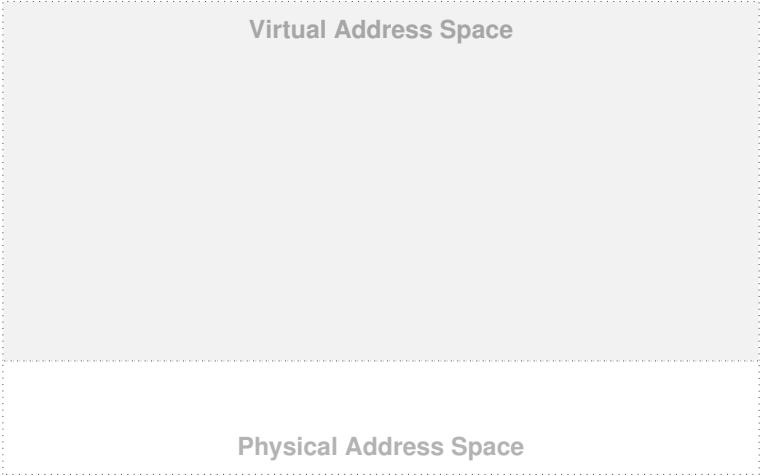
Yet another cache: Linux page cache

- Files buffered page-wise in “page cache”
- Lower access time for frequently accessed data

Yet another cache: Linux page cache

- Files buffered page-wise in “page cache”
- Lower access time for frequently accessed data
- Use up all the memory
- Pages are freed on demand
- Deduplicate pages (copy-on-write)

Copy-on-Write

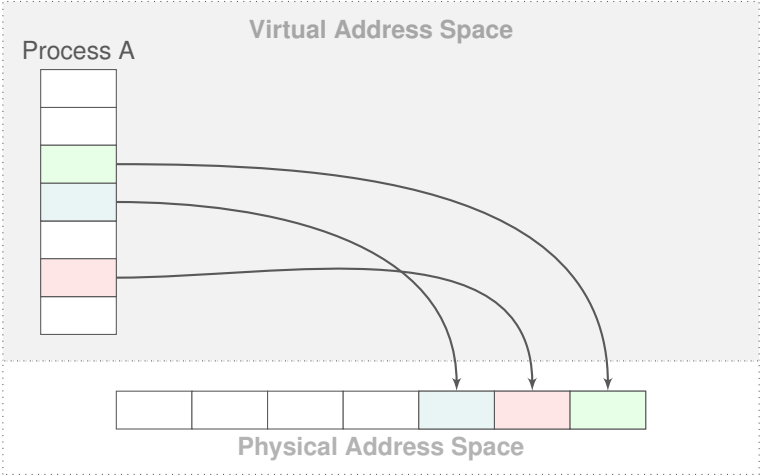


Copy-on-Write

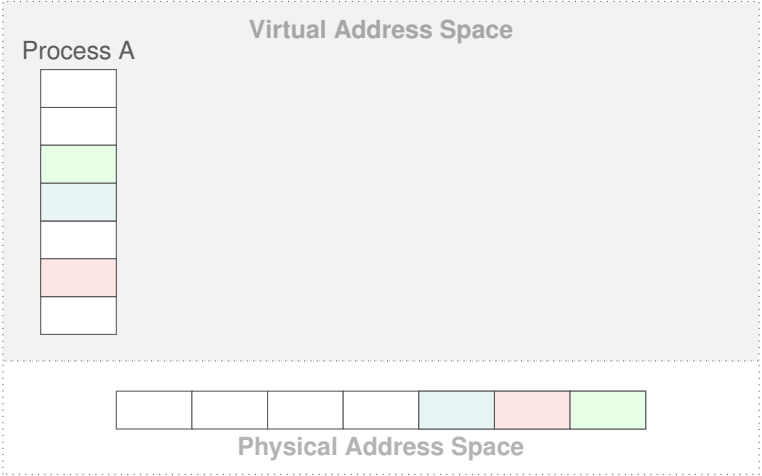
Copy-on-Write

Copy-on-Write

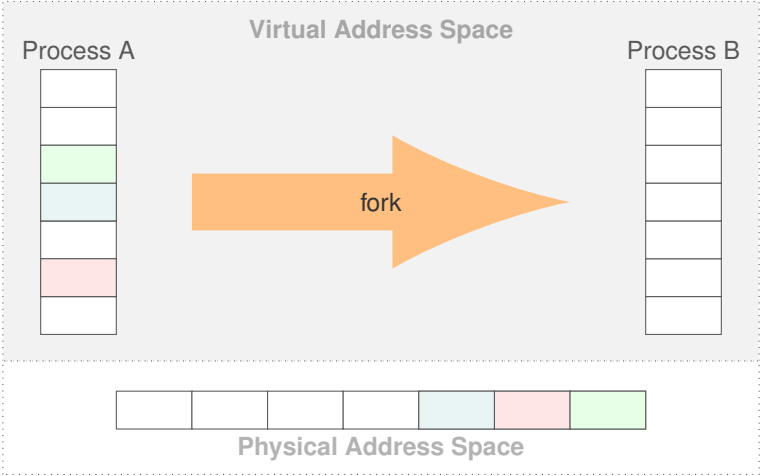
Copy-on-Write



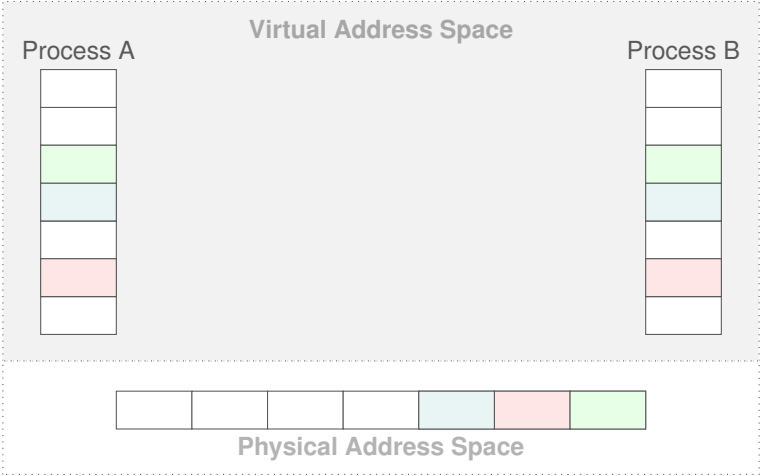
Copy-on-Write



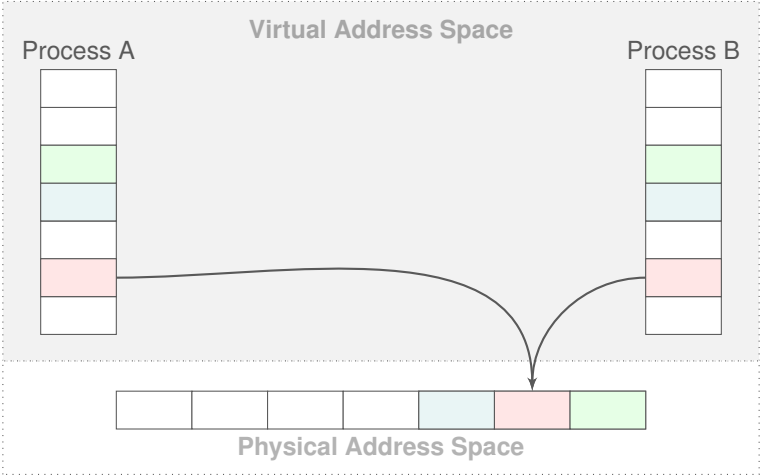
Copy-on-Write



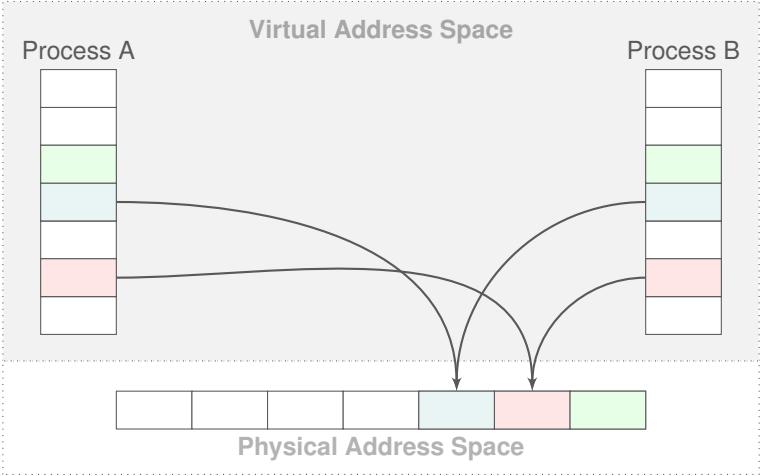
Copy-on-Write



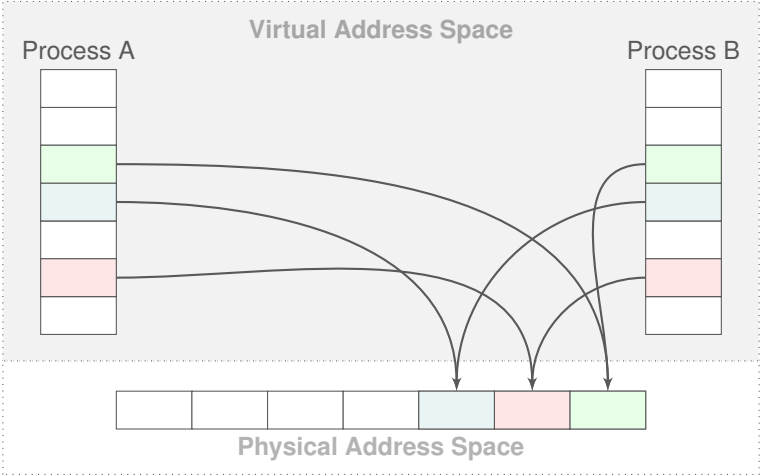
Copy-on-Write



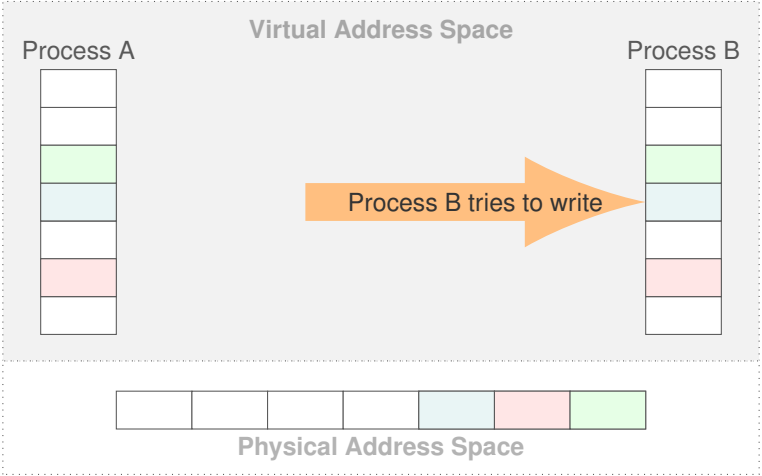
Copy-on-Write



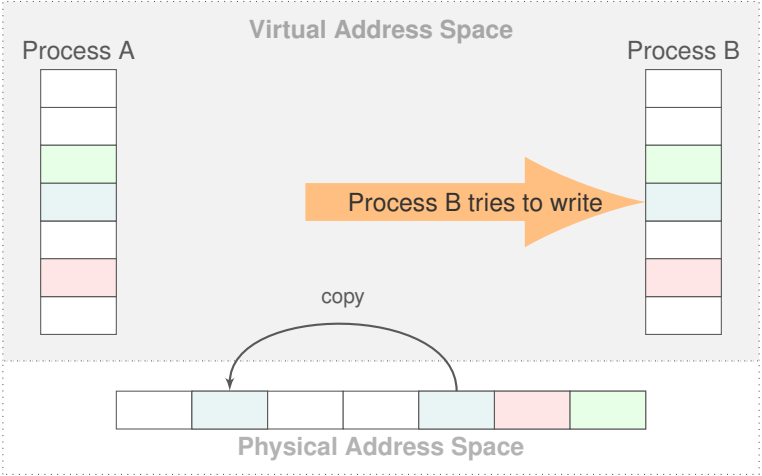
Copy-on-Write



Copy-on-Write



Copy-on-Write



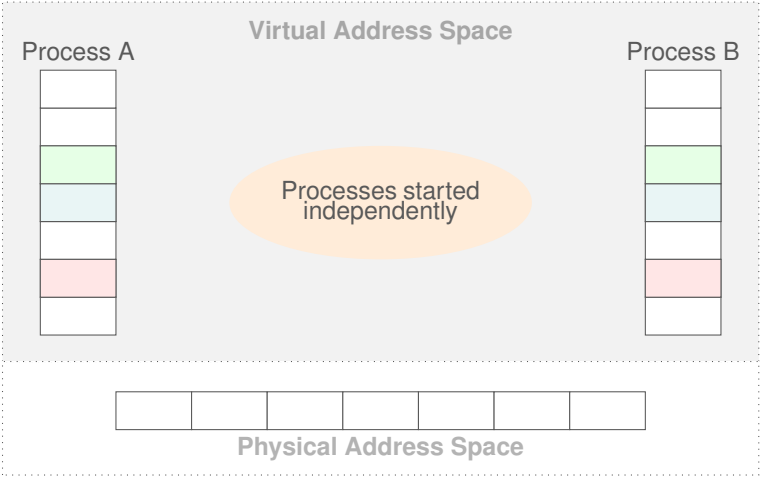
Copy-on-Write

Write vs. Copy-on-Write

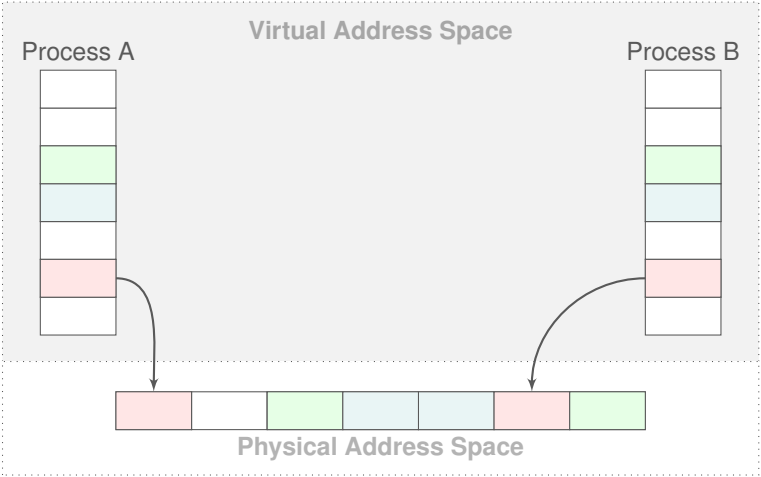
- Regular write access $< 0.2\mu s$
- Write access with copy-on-write pagefault $> 3.0\mu s$
- Clearly distinguishable

Page Deduplication

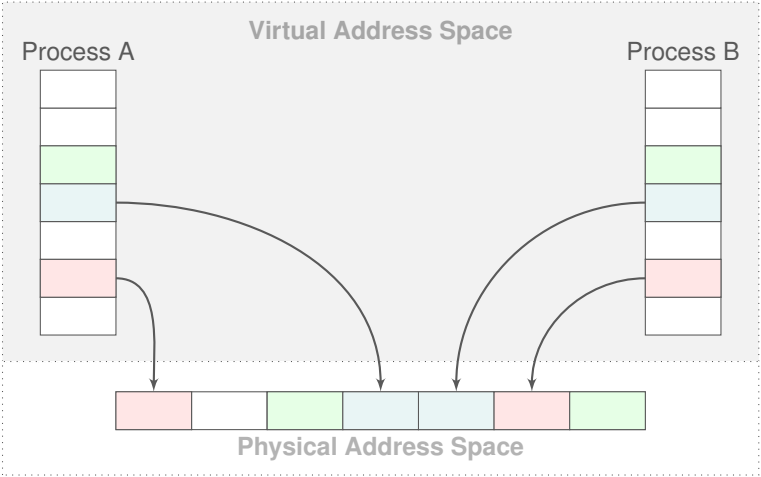
Page Deduplication



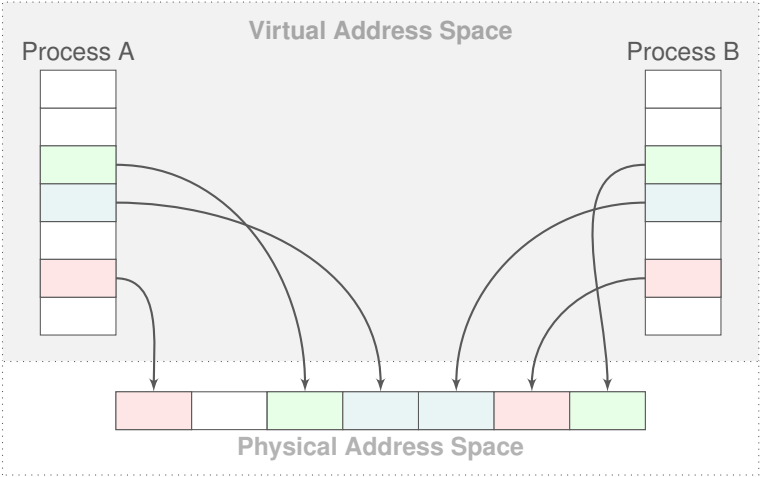
Page Deduplication



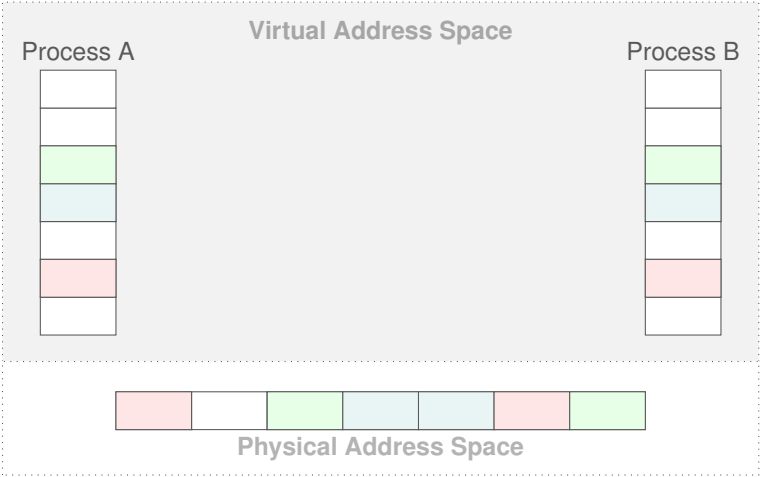
Page Deduplication



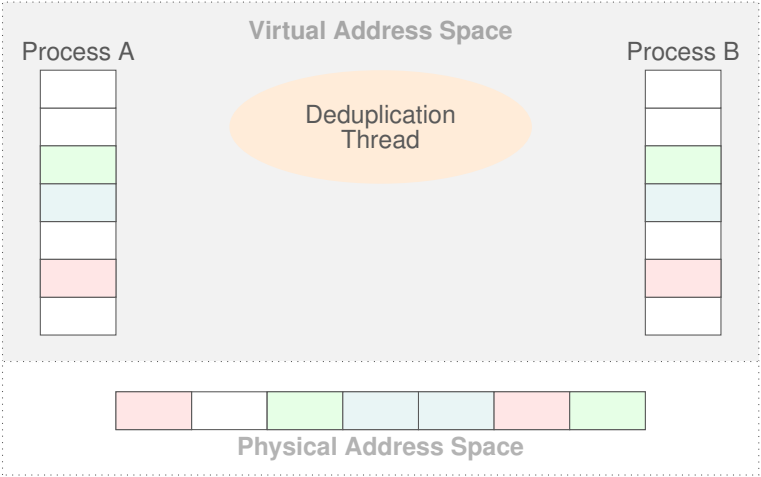
Page Deduplication



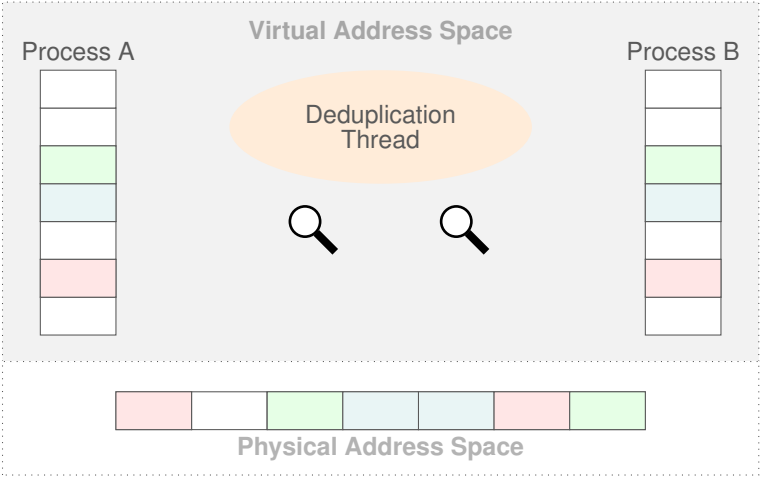
Page Deduplication



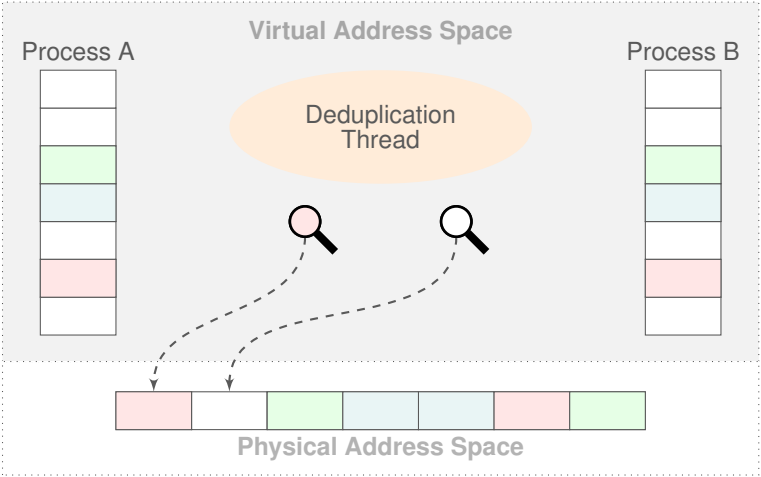
Page Deduplication



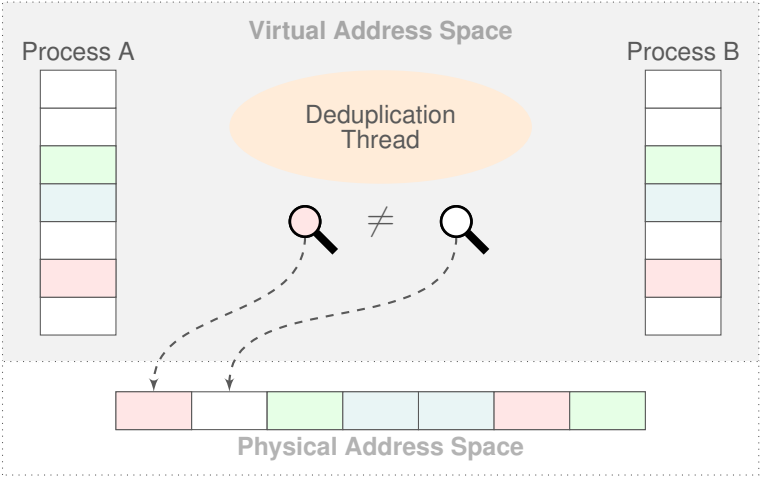
Page Deduplication



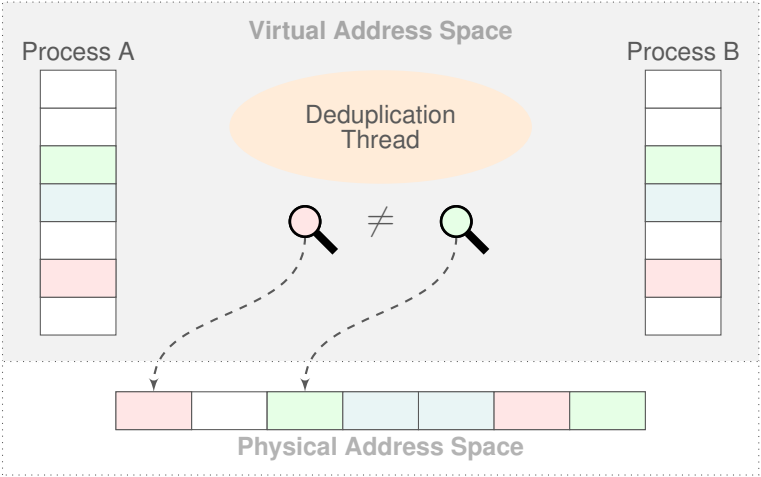
Page Deduplication



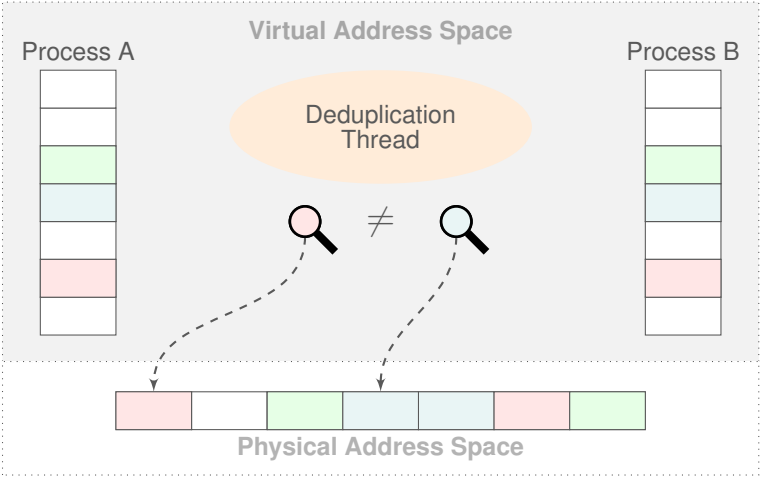
Page Deduplication



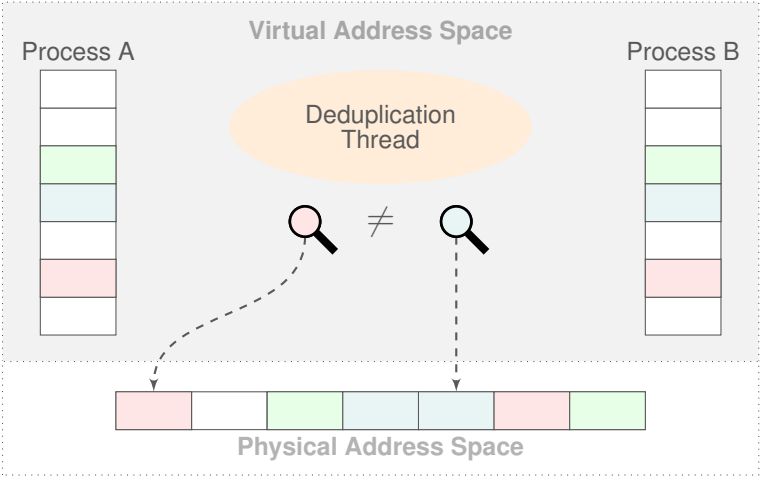
Page Deduplication



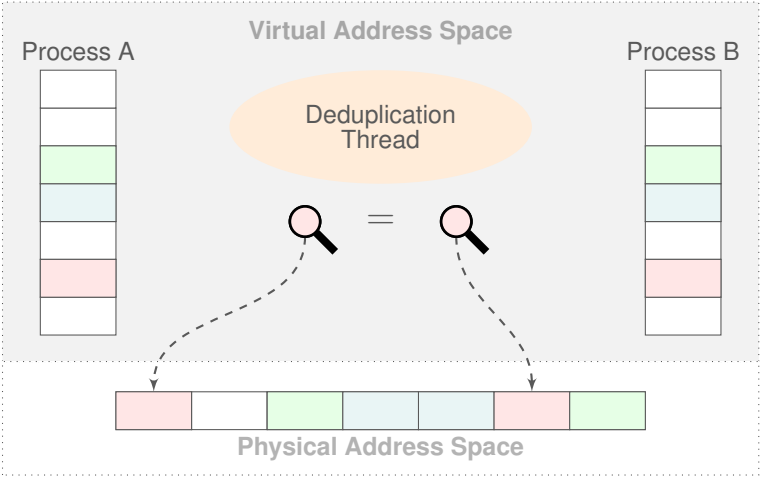
Page Deduplication



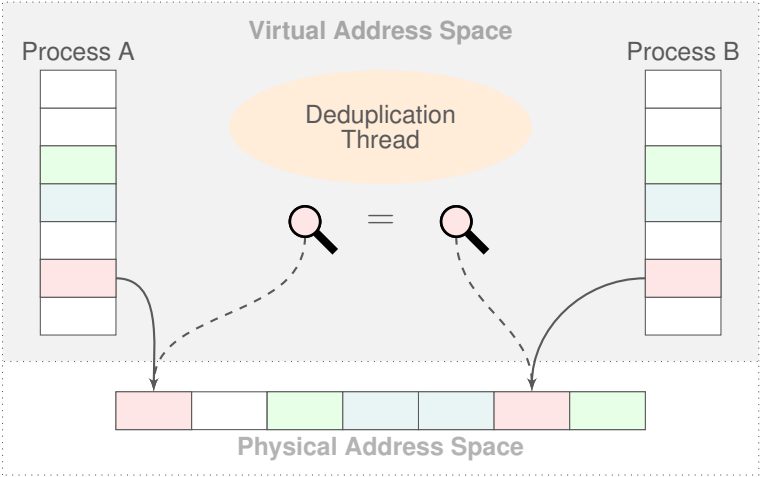
Page Deduplication



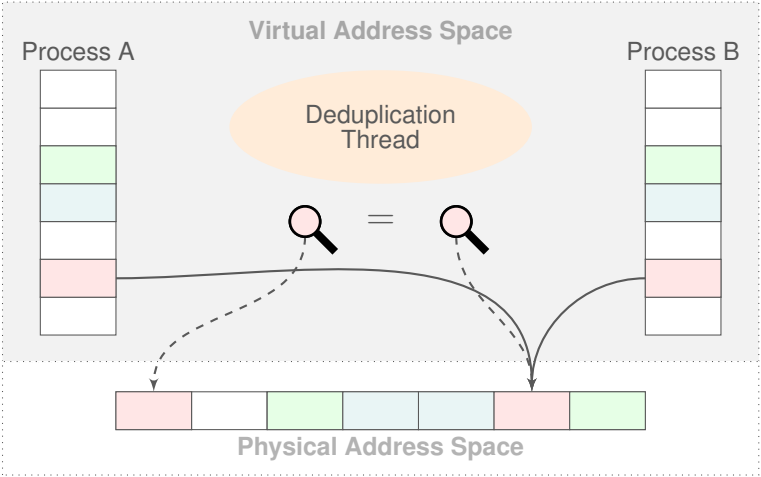
Page Deduplication



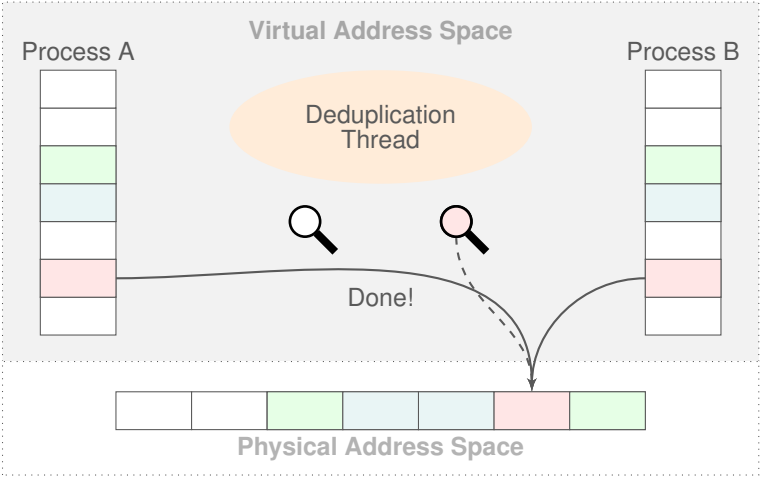
Page Deduplication



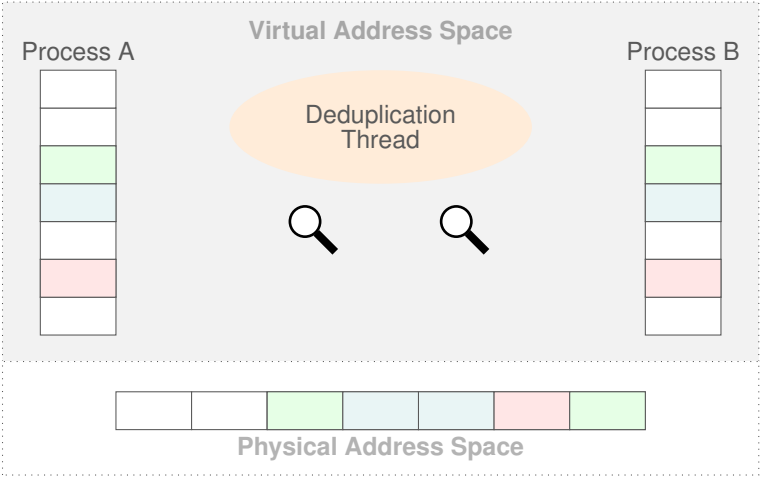
Page Deduplication



Page Deduplication



Page Deduplication



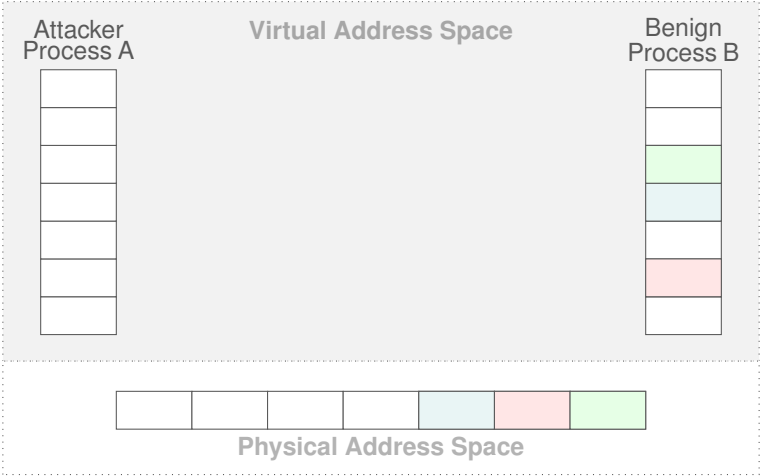
Page Deduplication (without fork)

- Deduplication between processes:
  1. in same OS instance (Android, Windows)
  2. in different VMs (KVM, VMWare, ...)

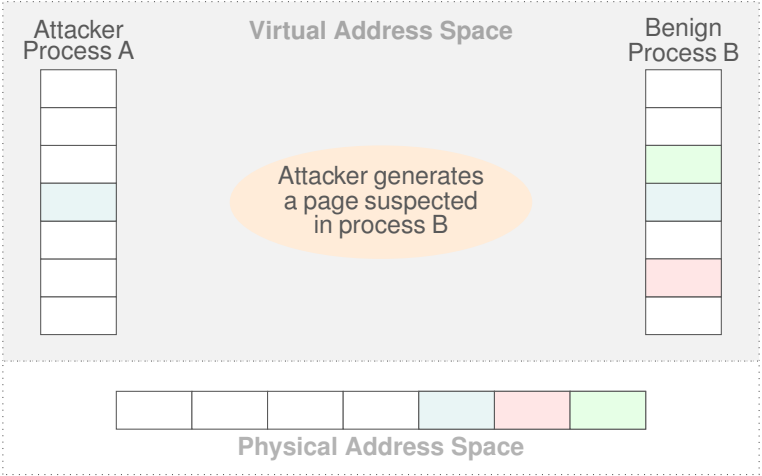
Page Deduplication (without fork)

- Deduplication between processes:
  1. in same OS instance (Android, Windows)
  2. in different VMs (KVM, VMWare, ...)
- Code pages, data pages - even kernel pages
- Time until deduplication 2-45 minutes
  - depends on system configuration

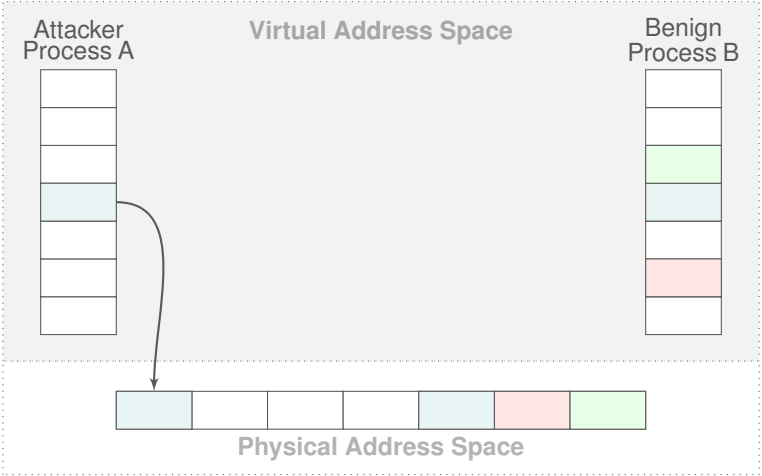
Page Deduplication Attack



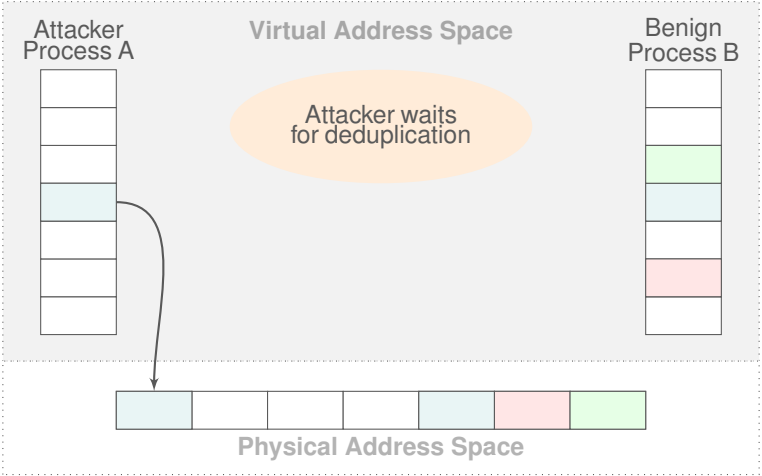
Page Deduplication Attack



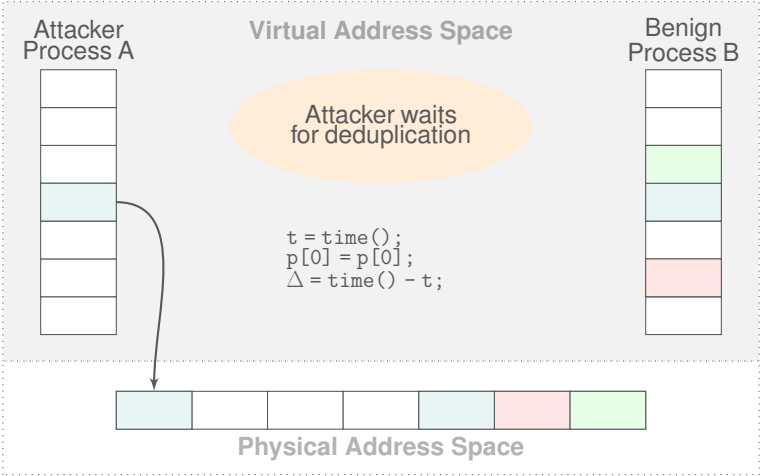
Page Deduplication Attack



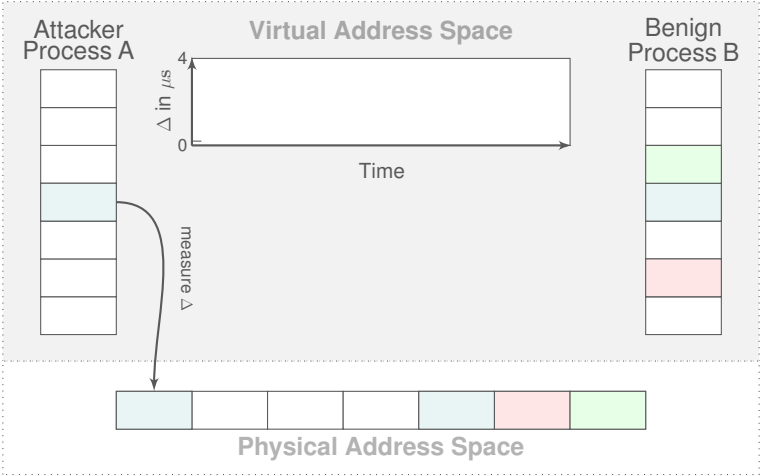
Page Deduplication Attack



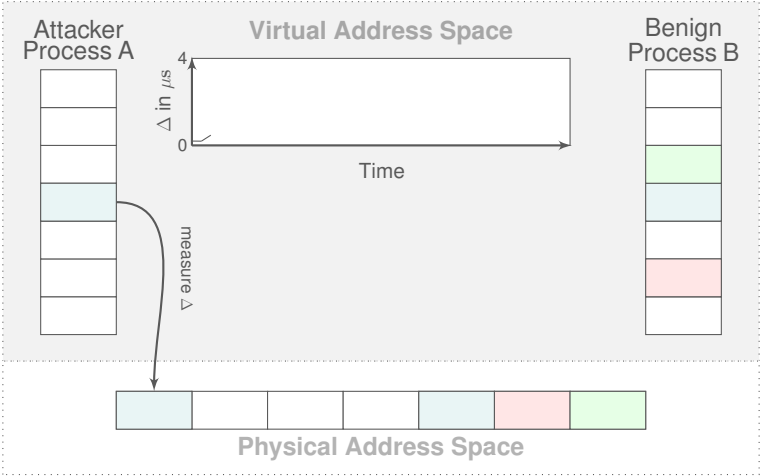
Page Deduplication Attack



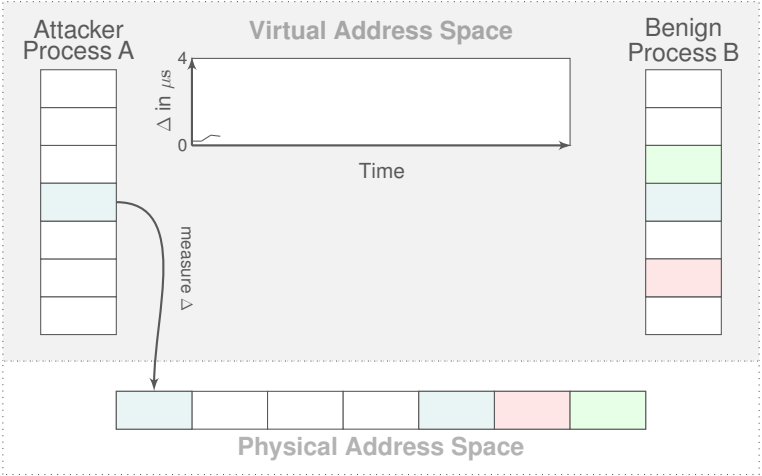
Page Deduplication Attack



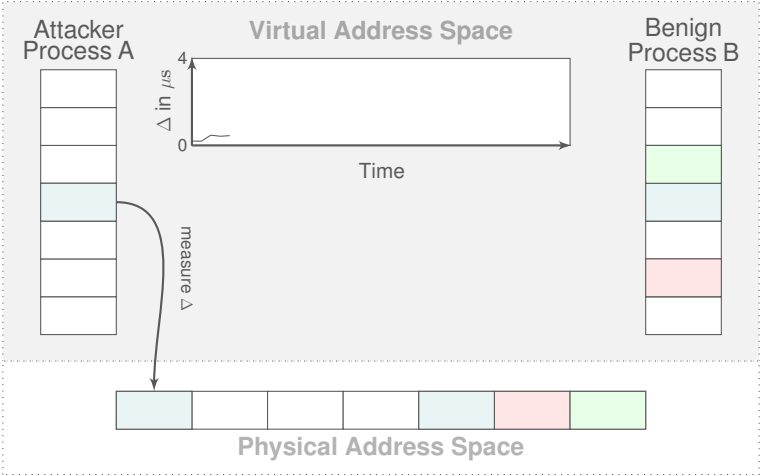
Page Deduplication Attack



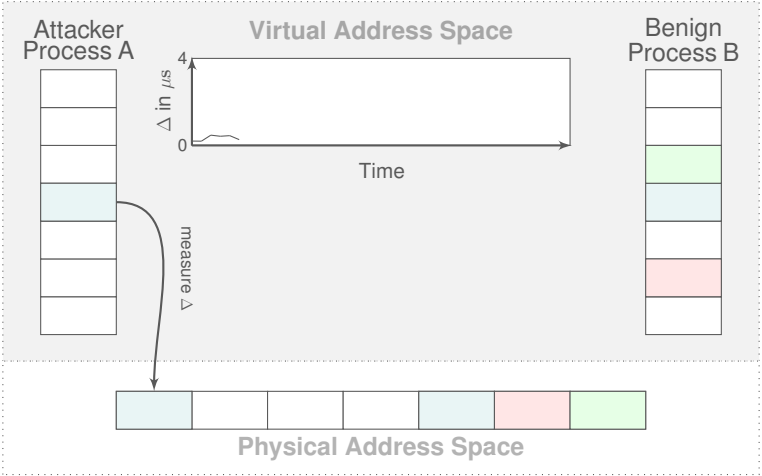
Page Deduplication Attack



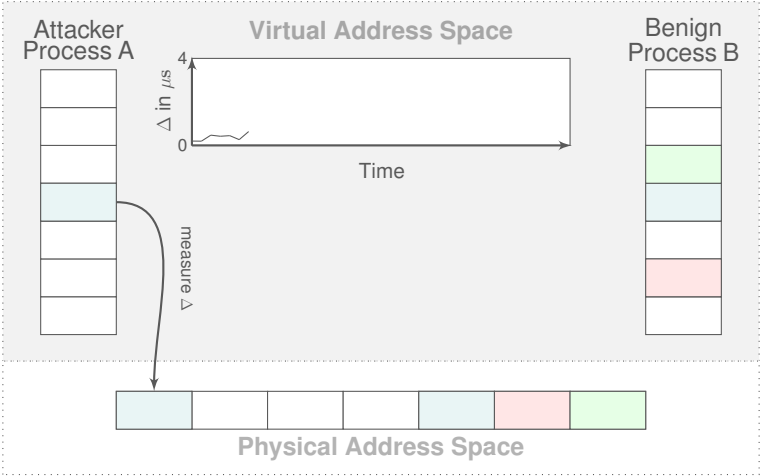
Page Deduplication Attack



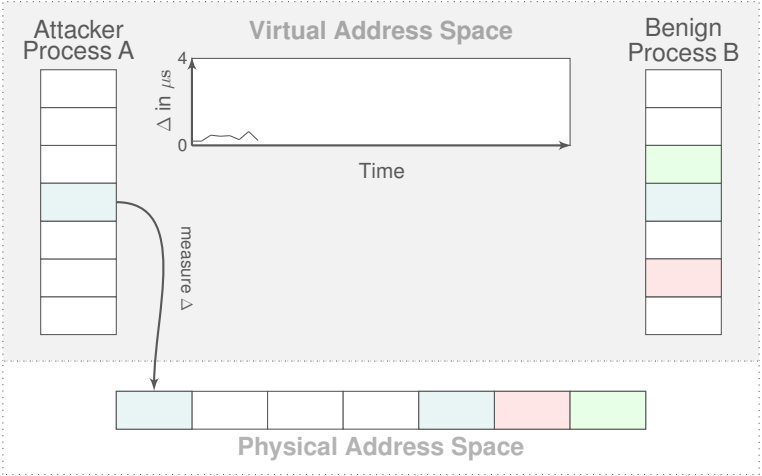
Page Deduplication Attack



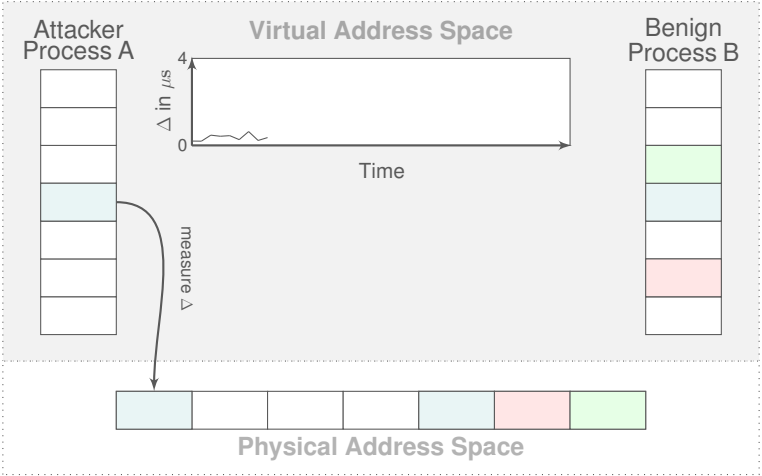
Page Deduplication Attack



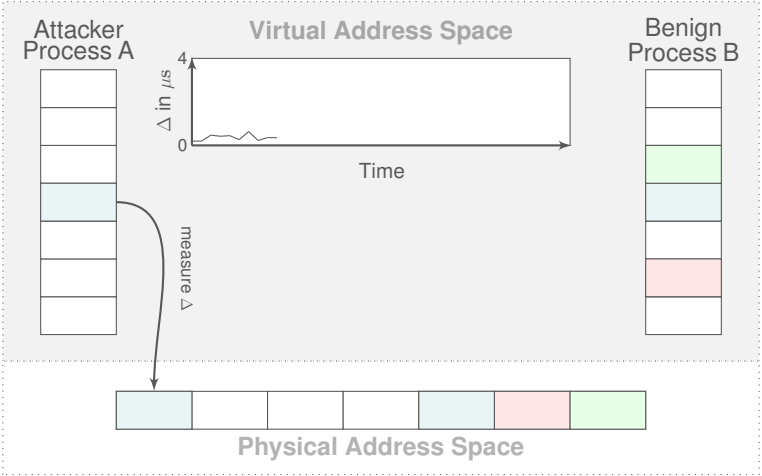
Page Deduplication Attack



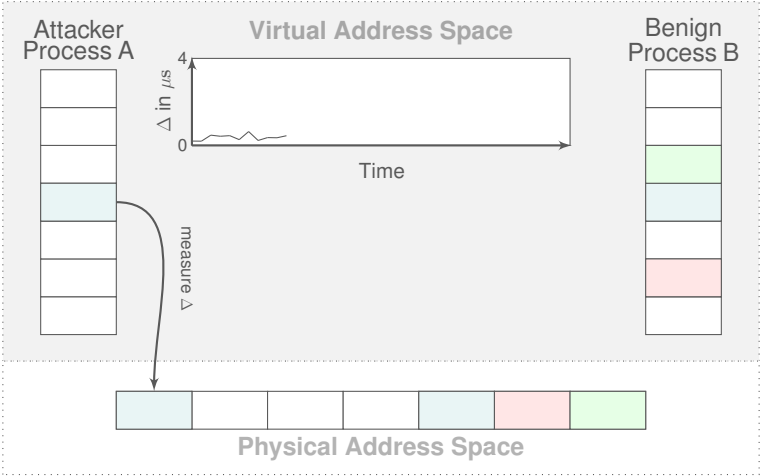
Page Deduplication Attack



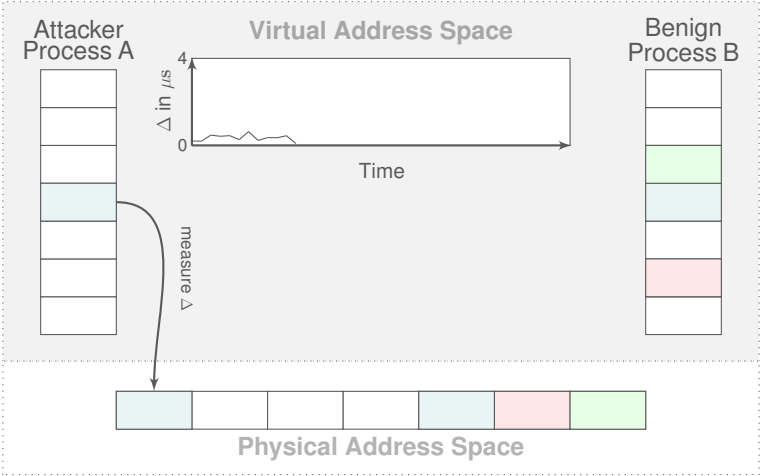
Page Deduplication Attack



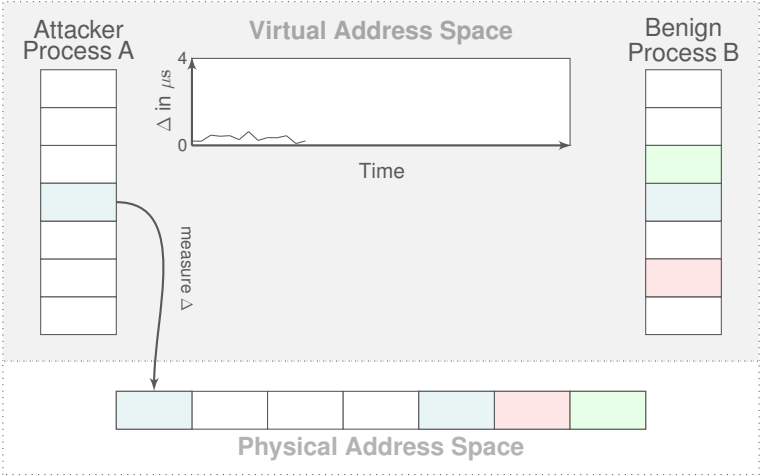
Page Deduplication Attack



Page Deduplication Attack

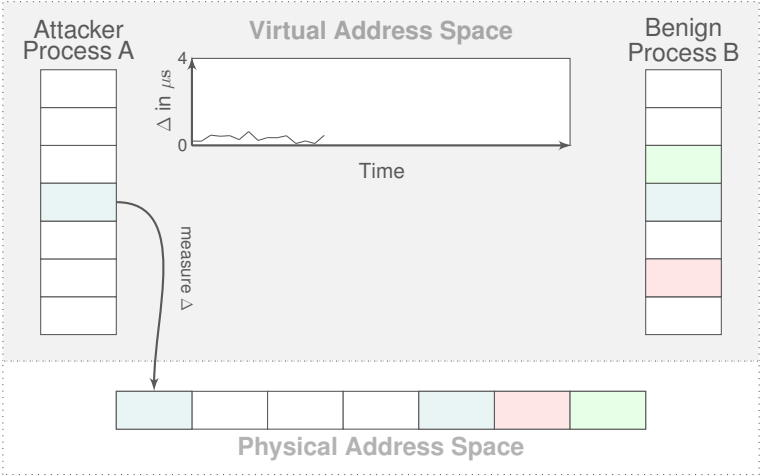


Page Deduplication Attack



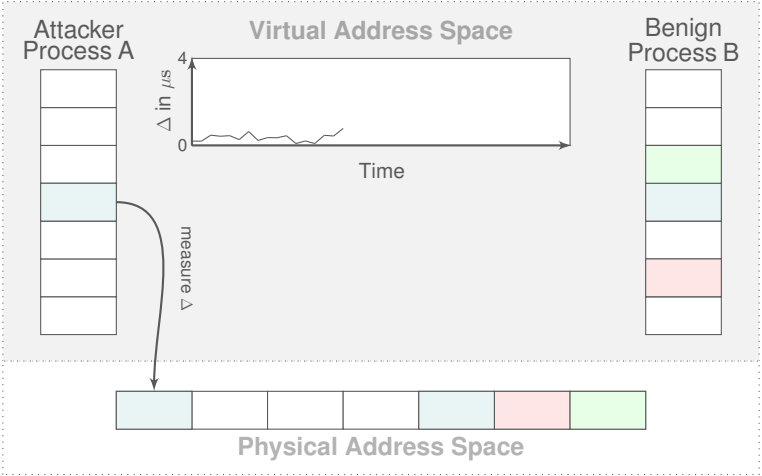
Page Deduplication Attack

Page Deduplication Attack

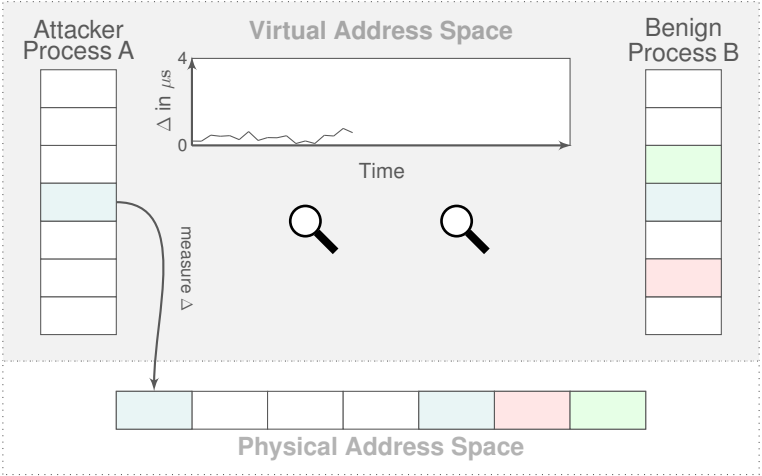


Page Deduplication Attack

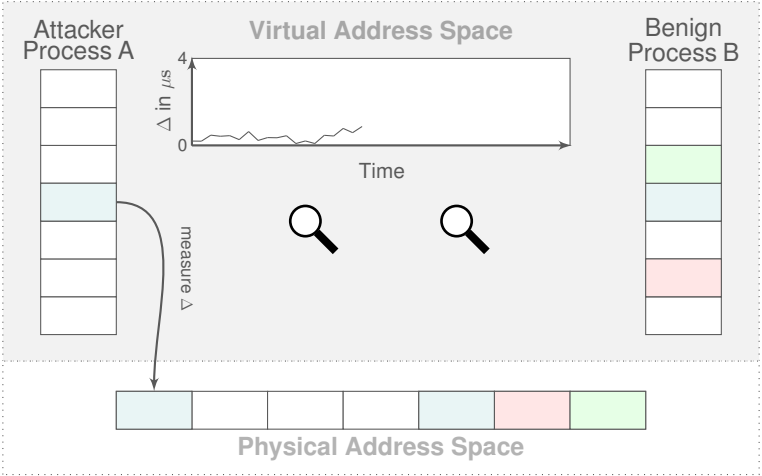
Page Deduplication Attack



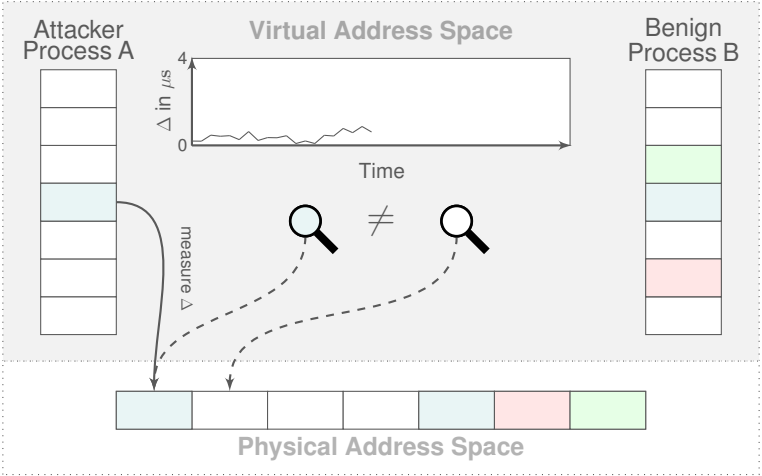
Page Deduplication Attack



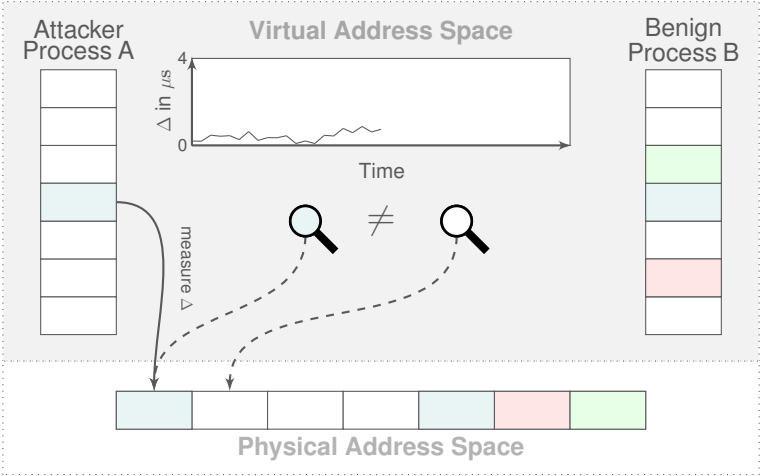
Page Deduplication Attack



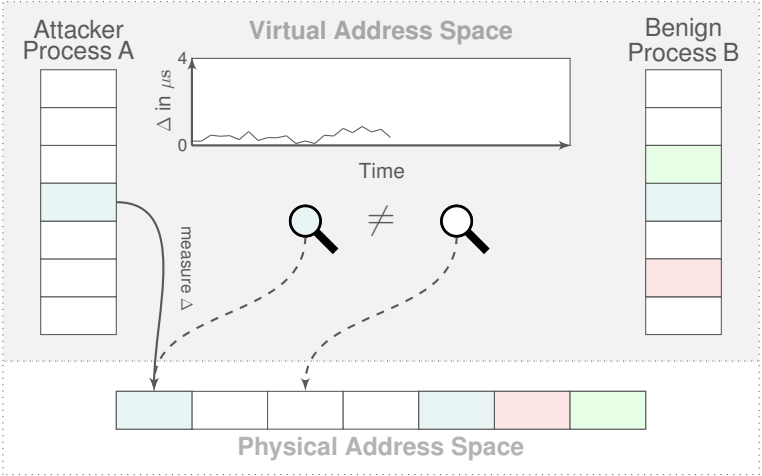
Page Deduplication Attack



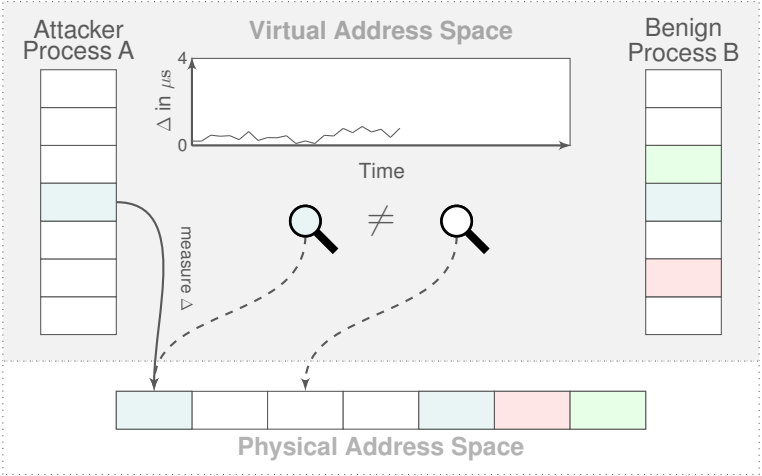
Page Deduplication Attack



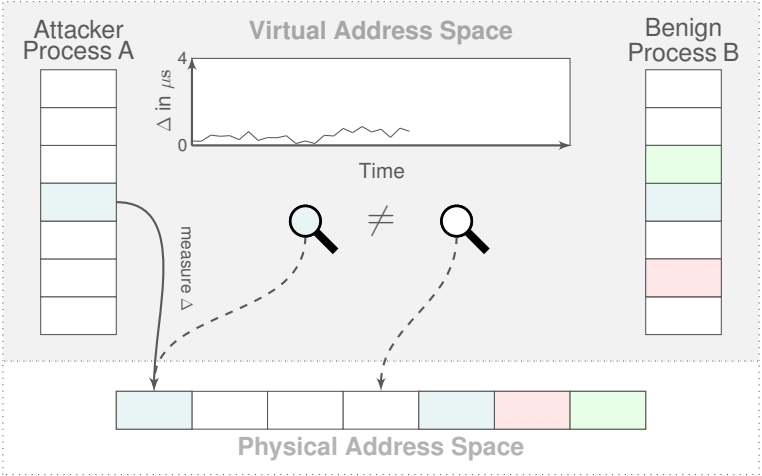
Page Deduplication Attack



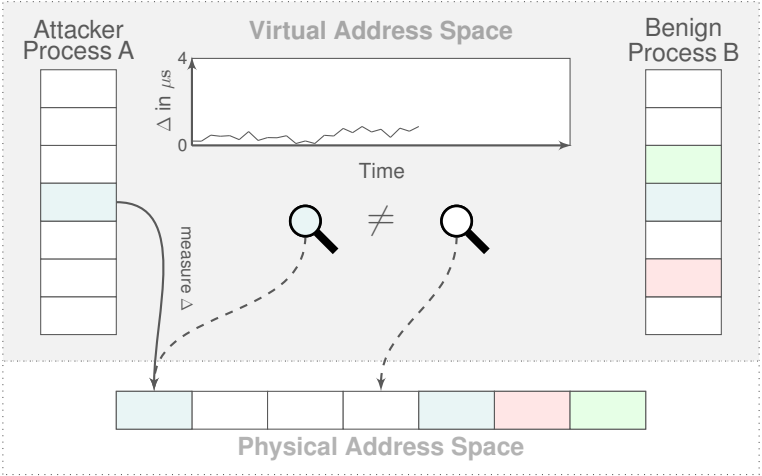
Page Deduplication Attack



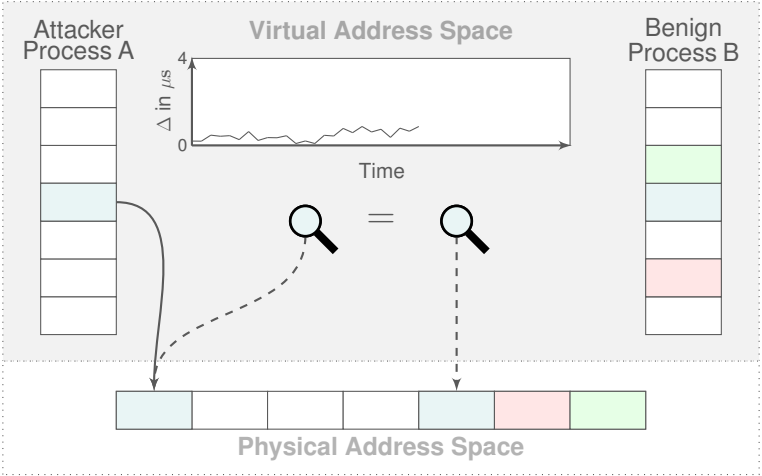
Page Deduplication Attack



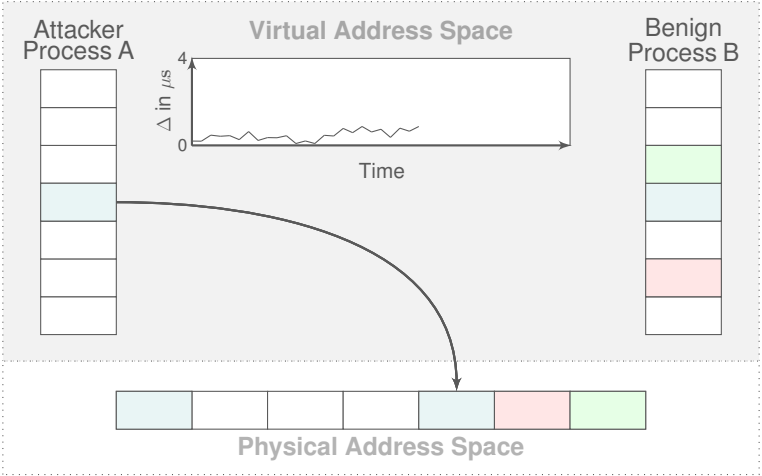
Page Deduplication Attack



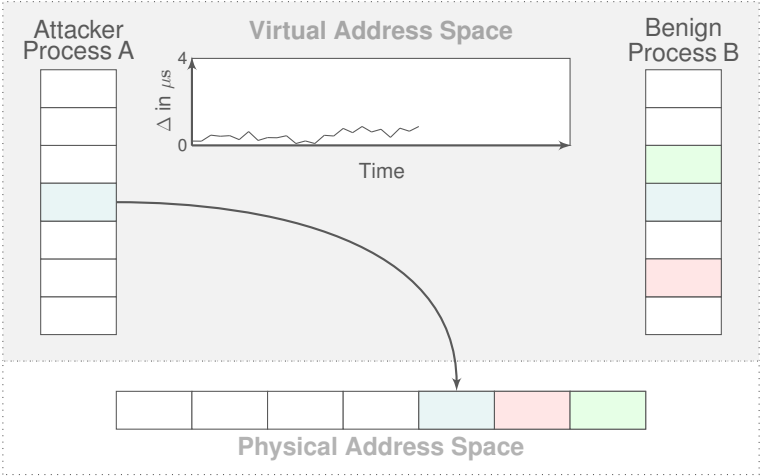
Page Deduplication Attack



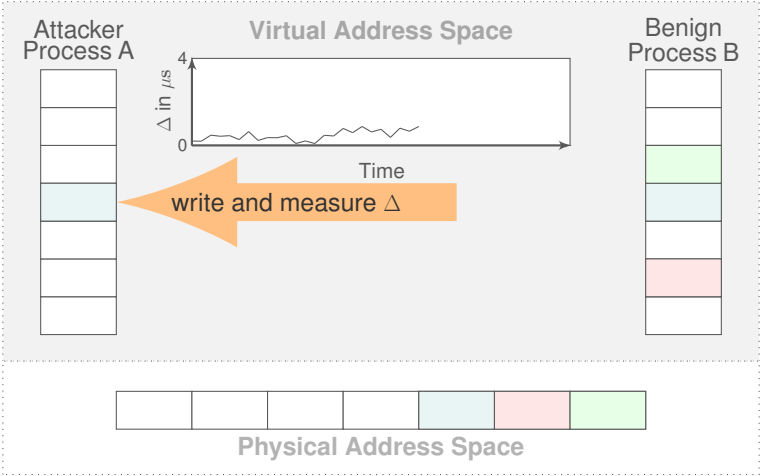
Page Deduplication Attack



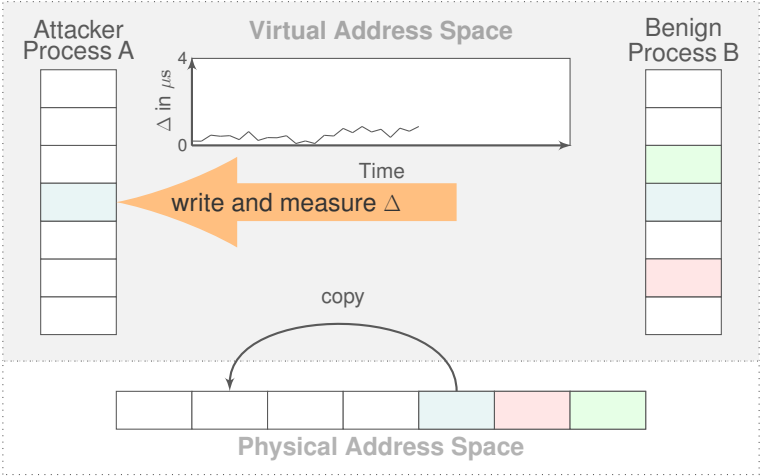
Page Deduplication Attack



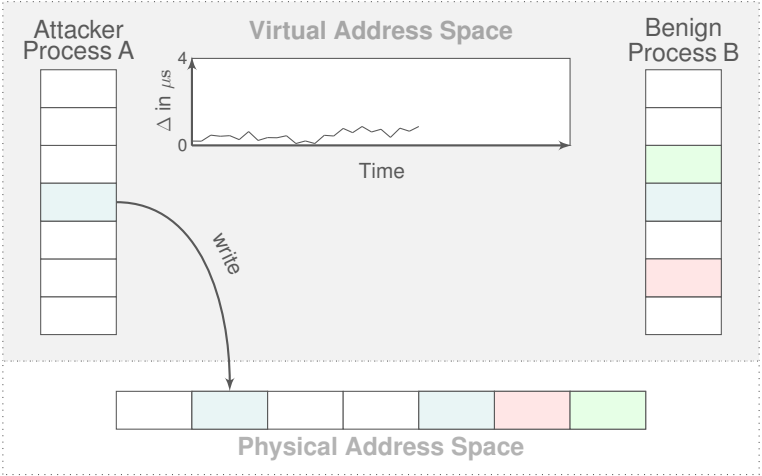
Page Deduplication Attack



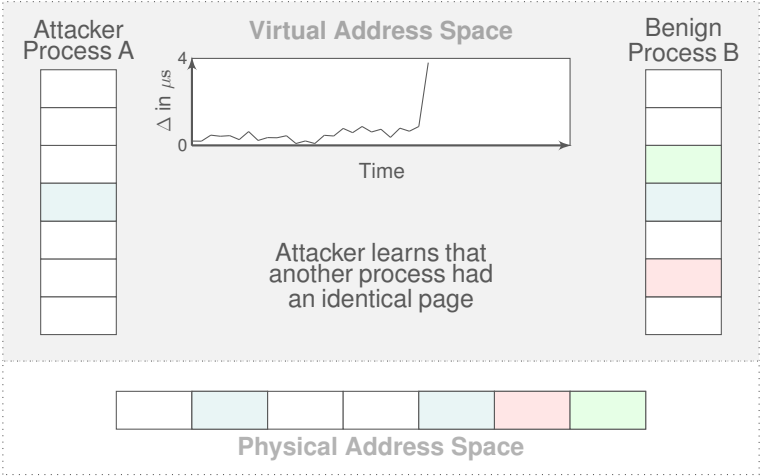
Page Deduplication Attack



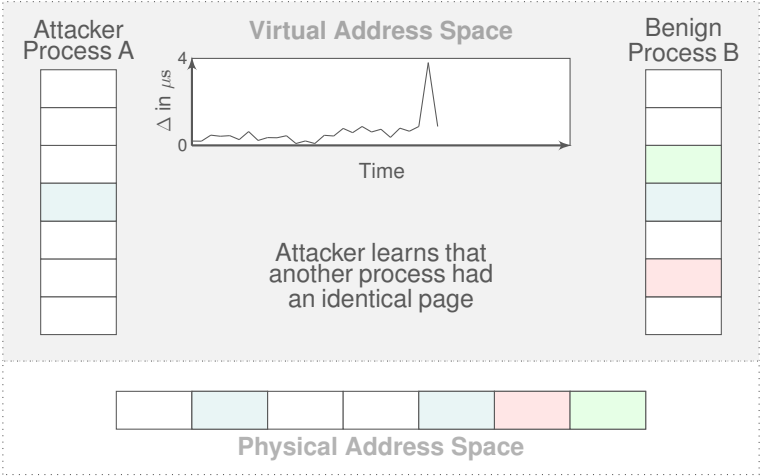
Page Deduplication Attack



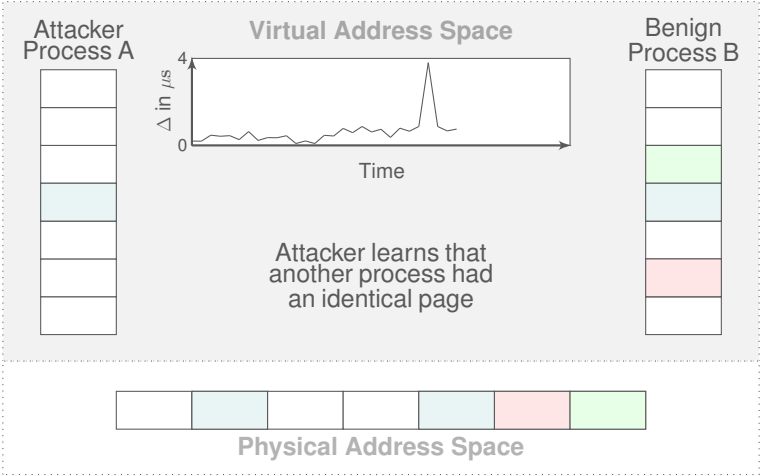
Page Deduplication Attack



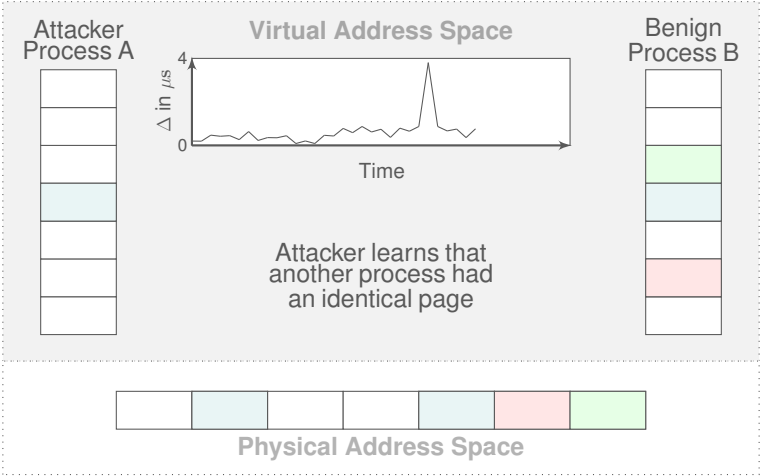
Page Deduplication Attack



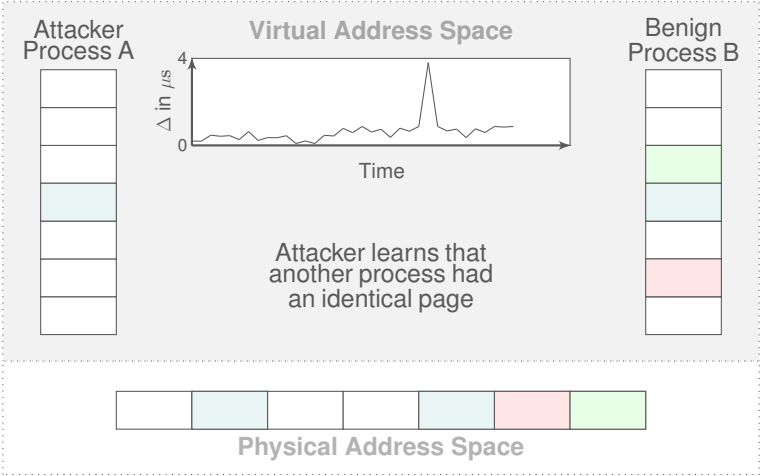
Page Deduplication Attack



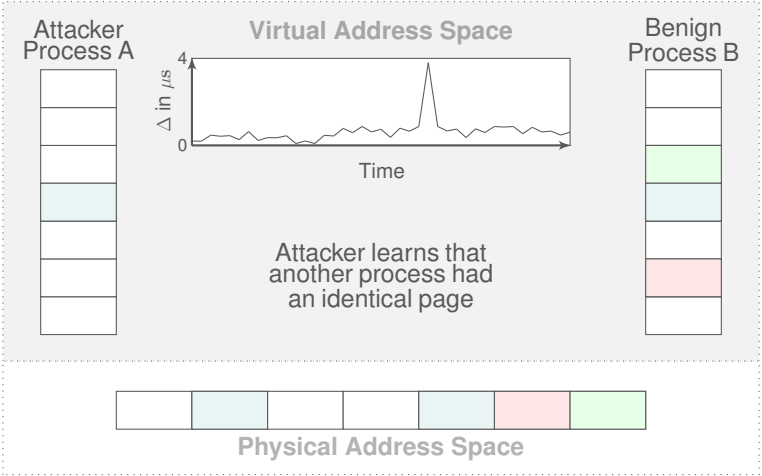
Page Deduplication Attack



Page Deduplication Attack



Page Deduplication Attack



What can be attacked?

- Detect binary versions in co-located VMs
- Detect downloaded image in Firefox under certain conditions

→ Attacks on hypervisors

- Native code only

Suzaki et al. 2011; Owens and Wang 2011; J. Xiao et al. 2013; J. Xiao et al. 2012

What can be attacked?

- Detect CSS files and images of opened websites
  - Chrome, Firefox and Internet Explorer
- Perform the attack in JavaScript

→ Attacks on KVM, Windows 8.1 and Android

Attacking Browsers

- Images and CSS files are page-aligned in memory
- Load them into memory for all websites of interest
- Detect deduplication

→ Malicious ad networks: alternative to tracking pixels?

Detect Image (Native, Cross-VM, KVM)

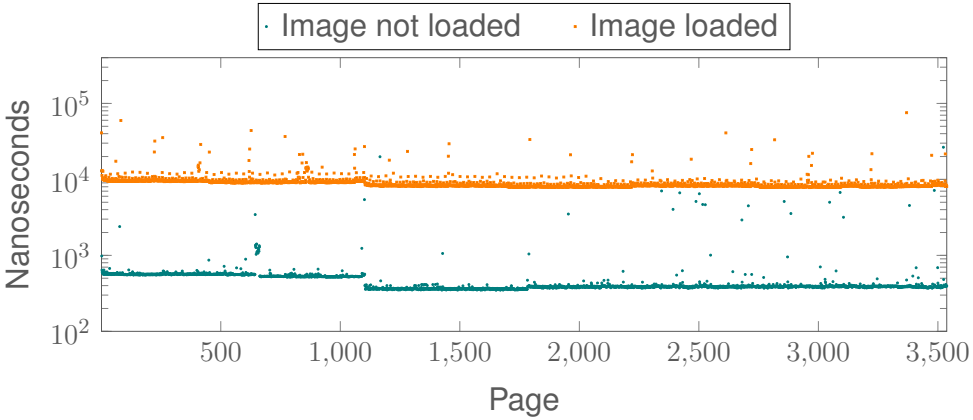
Challenges of JavaScript-based attacks

- No cycle counting (`rdtsc`)
- No access to virtual addresses

Page Deduplication Attacks in JavaScript

- Only require microsecond accuracy
  - `performance.now()` is accurate enough
  - Can even work with millisecond accuracy
    - Accumulate time difference
    - Only possible with enough image/CSS data
- Large typed arrays are allocated page-aligned

Detect Image (JavaScript, Cross-VM, KVM)



Detection of Open Websites

- Attacker chosen set of websites
- Load website images and CSS files into arrays
- Reuse HTTP headers of system under attack

Countermeasures

JavaScript:

- Reduce timer accuracy?
- Prevent page-aligned arrays?
- Website diversification?
- Prevent control over full pages
  - Every n-th byte not part of JavaScript array

Countermeasures

JavaScript:

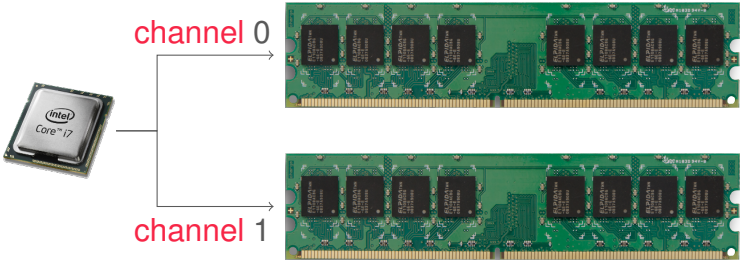
- Reduce timer accuracy?
- Prevent page-aligned arrays?
- Website diversification?
- Prevent control over full pages
  - Every n-th byte not part of JavaScript array

Generic:

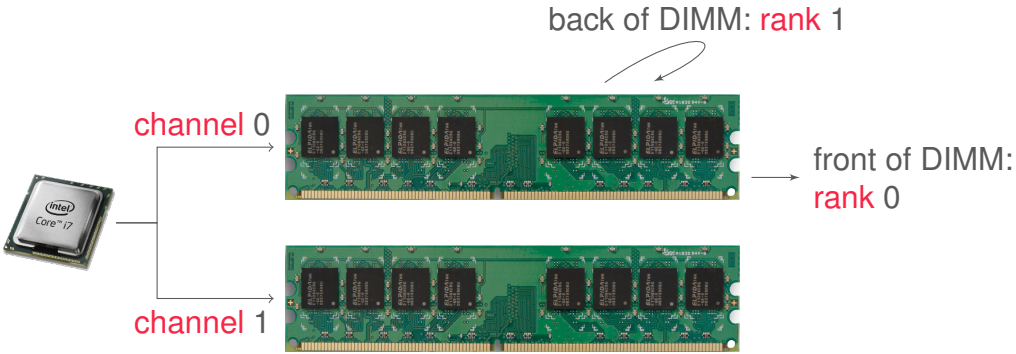
- Disable page deduplication (for writable pages)

DRAM organization

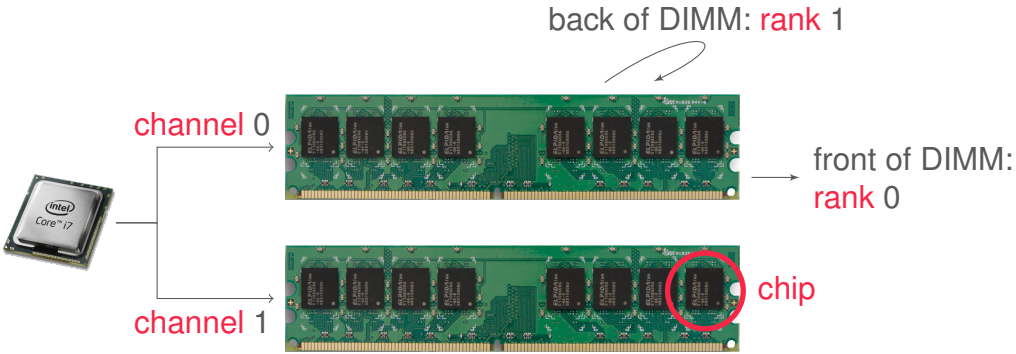
DRAM organization



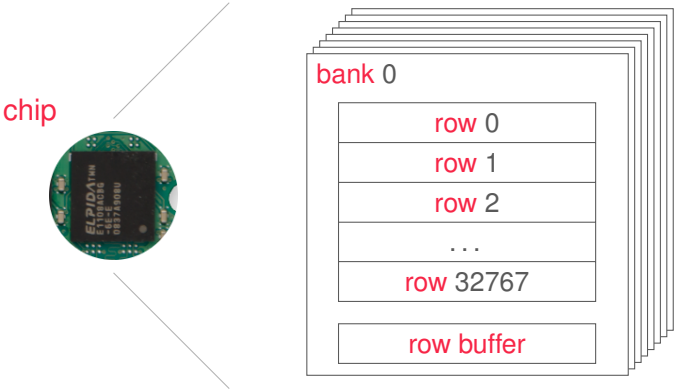
DRAM organization



DRAM organization

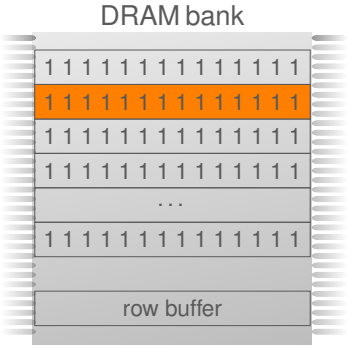


DRAM organization



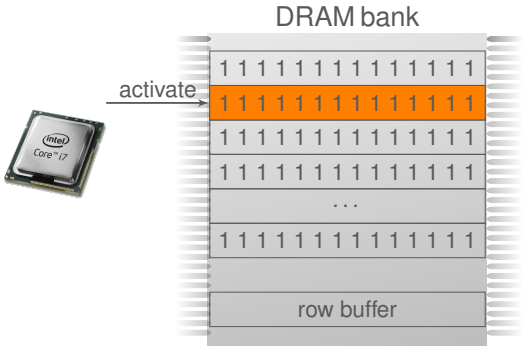
- bits in cells in rows
- access: activate row, copy to row buffer

How reading from DRAM works



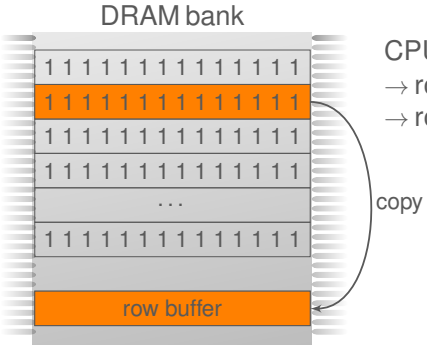
CPU wants to access row 1

How reading from DRAM works



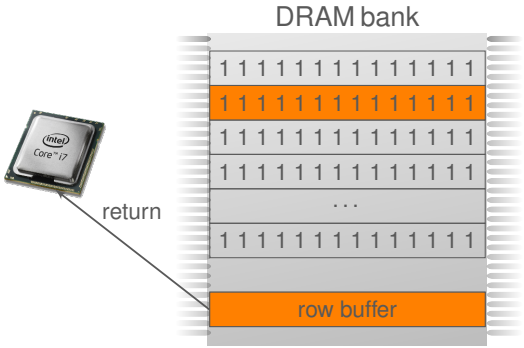
CPU wants to access row 1
→ row 1 activated

How reading from DRAM works



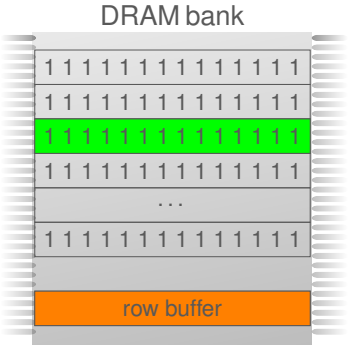
CPU wants to access row 1
→ row 1 activated
→ row 1 copied to row buffer

How reading from DRAM works



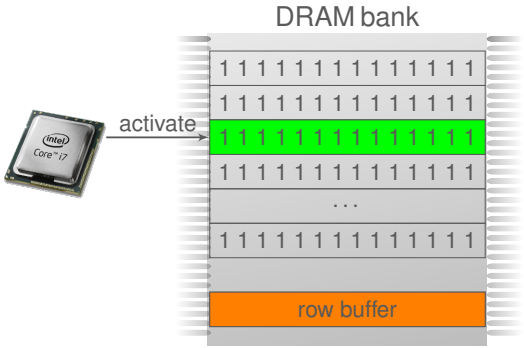
CPU wants to access row 1
→ row 1 activated
→ row 1 copied to row buffer

How reading from DRAM works



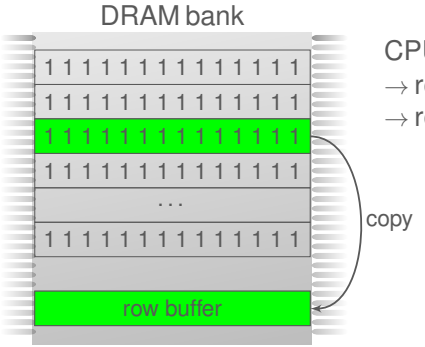
CPU wants to access row 2

How reading from DRAM works



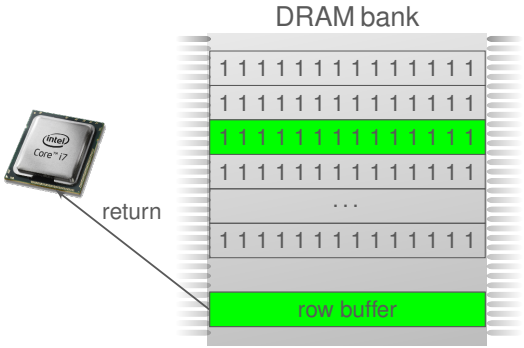
CPU wants to access row 2
→ row 2 activated

How reading from DRAM works



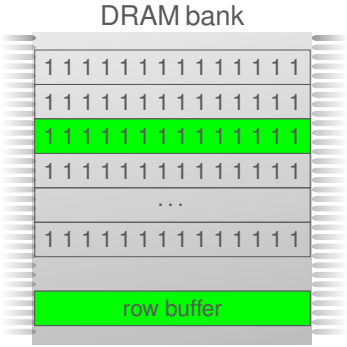
CPU wants to access row 2
→ row 2 activated
→ row 2 copied to row buffer

How reading from DRAM works



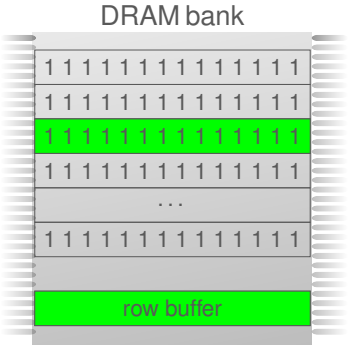
CPU wants to access row 2
→ row 2 activated
→ row 2 copied to row buffer

How reading from DRAM works



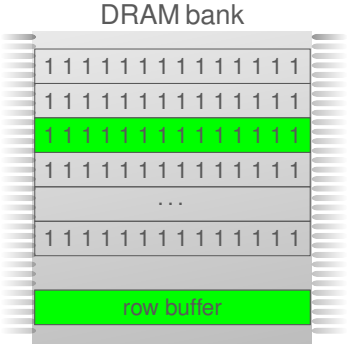
CPU wants to access row 2
→ row 2 activated
→ row 2 copied to row buffer
→ slow (row conflict)

How reading from DRAM works



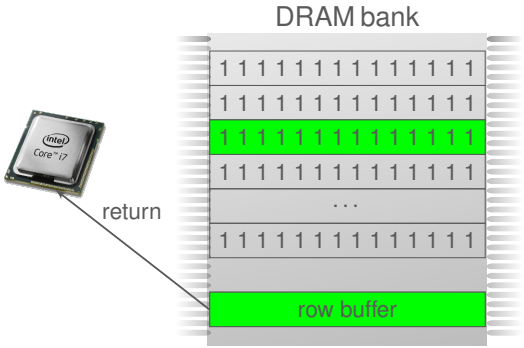
CPU wants to access row 2—again

How reading from DRAM works



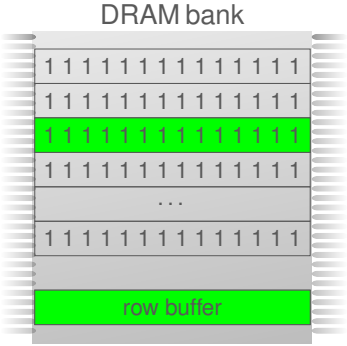
CPU wants to access row 2—again
→ row 2 already in row buffer

How reading from DRAM works



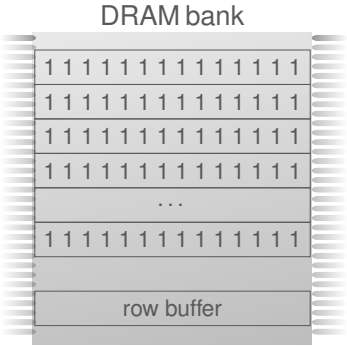
CPU wants access row 2—again
→ row 2 already in row buffer

How reading from DRAM works



CPU wants access row 2—again
→ row 2 already in row buffer
→ fast (row hit)

How reading from DRAM works



row buffer = cache

DRAM refresh

- cells leak $\rightarrow$ repetitive refresh necessary
- refresh $\approx$ reading (destructive) + writing same data again
- maximum interval between refreshes to guarantee data integrity

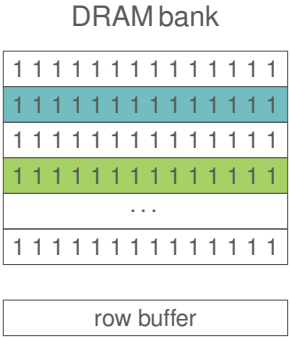
DRAM refresh

- cells leak $\rightarrow$ repetitive refresh necessary
- refresh $\approx$ reading (destructive) + writing same data again
- maximum interval between refreshes to guarantee data integrity

- cells leak faster upon proximate accesses $\rightarrow$ Rowhammer

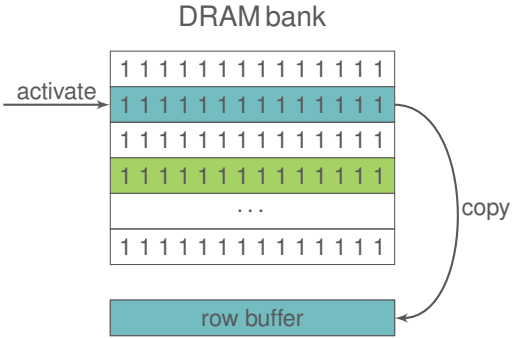
Rowhammer

"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



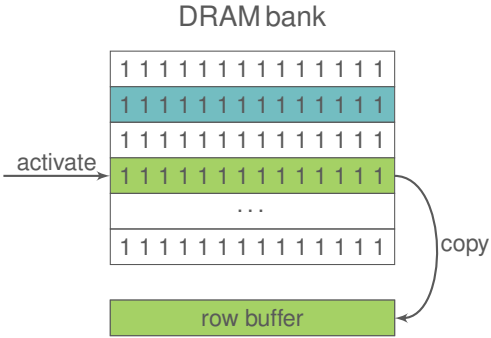
Rowhammer

"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



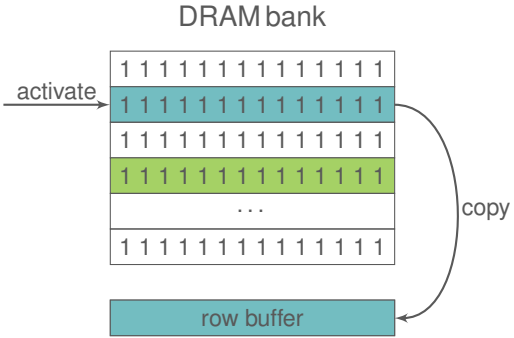
Rowhammer

"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



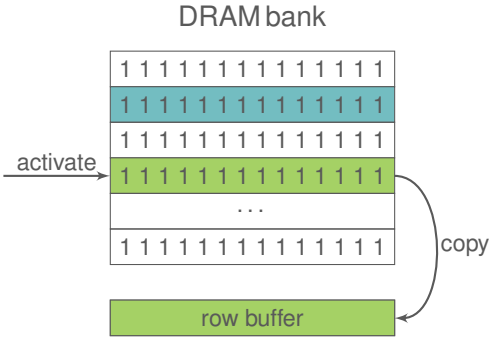
Rowhammer

"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



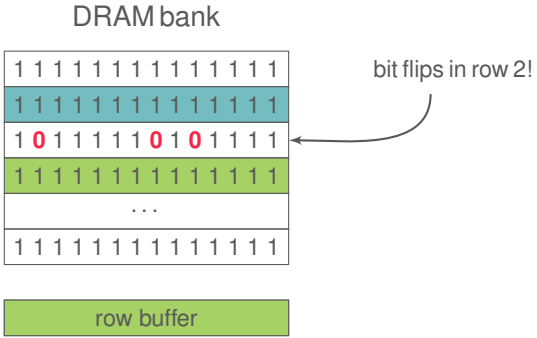
Rowhammer

"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



Rowhammer

"It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after" – Motherboard Vice



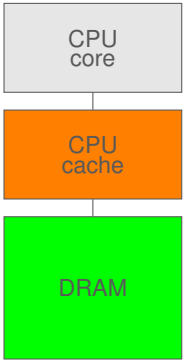
Requirements

Memory accesses must be

- uncached: reach DRAM
- fast: race against the next row refresh
- targeted: reach specific row

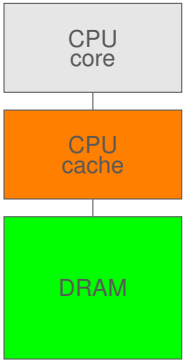
How do we get enough uncached accesses?

Impact of the CPU cache



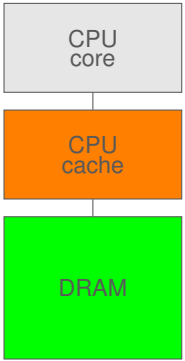
- only non-cached accesses reach DRAM

Impact of the CPU cache



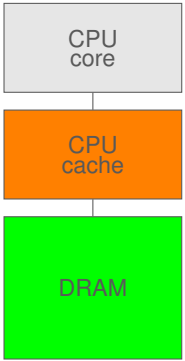
- only non-cached accesses reach DRAM
- either remove data from cache

Impact of the CPU cache



- only non-cached accesses reach DRAM
- either remove data from cache
- or don't put it there in the first place

Impact of the CPU cache



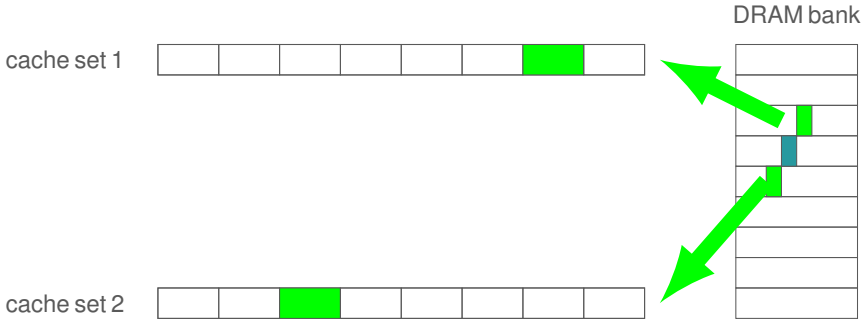
- only non-cached accesses reach DRAM
- either remove data from cache
- or don't put it there in the first place

→ next access will be served from DRAM

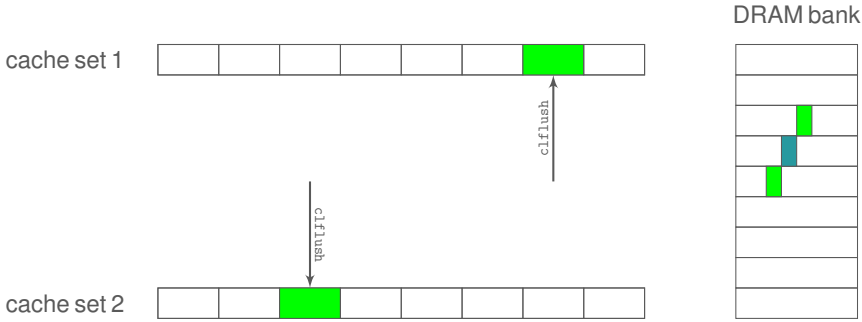
Access techniques

1. `clflush` instruction → original paper (Kim et al. 2014)
2. cache eviction (Gruss, Maurice, and Mangard 2016; Aweke et al. 2016)
3. non-temporal accesses (Qiao and Seaborn 2016)
4. uncached memory (Veen et al. 2016)

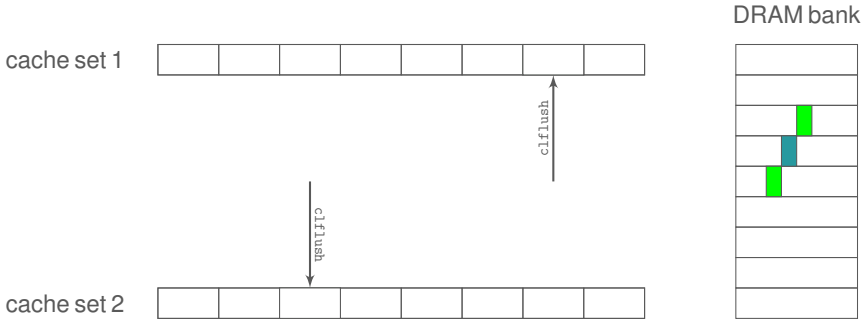
#1 Hammering with `clflush`



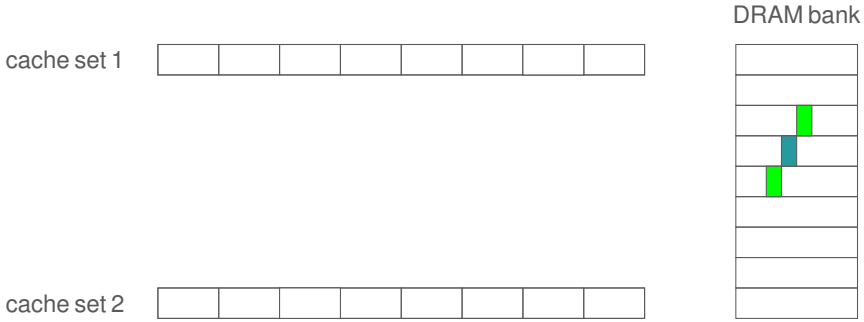
#1 Hammering with `clflush`



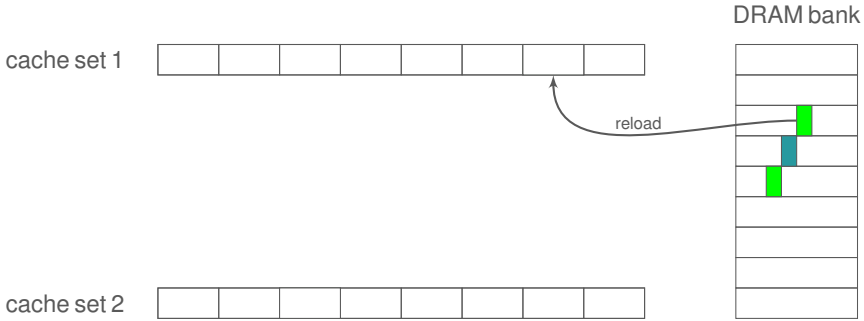
#1 Hammering with `clflush`



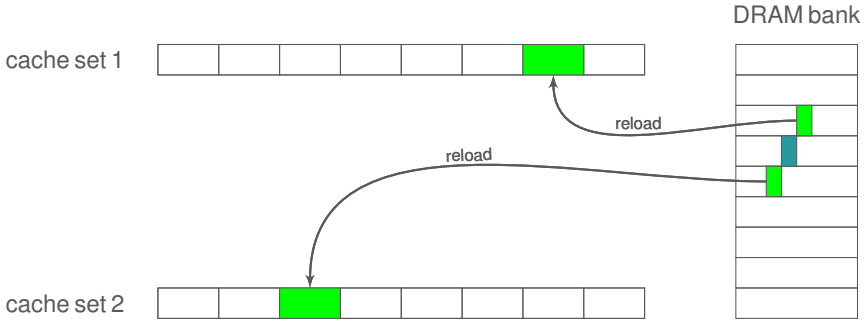
#1 Hammering with `clflush`



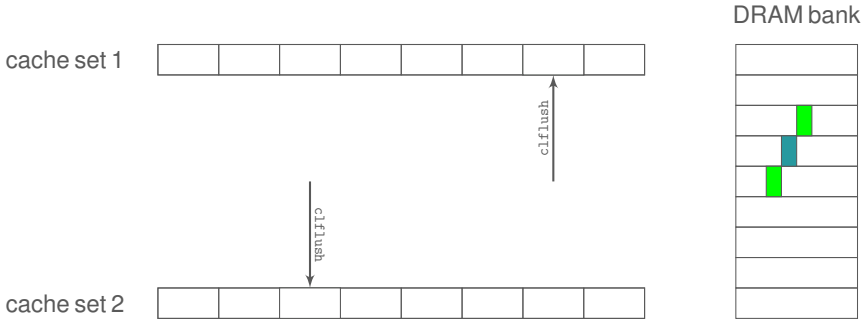
#1 Hammering with `clflush`



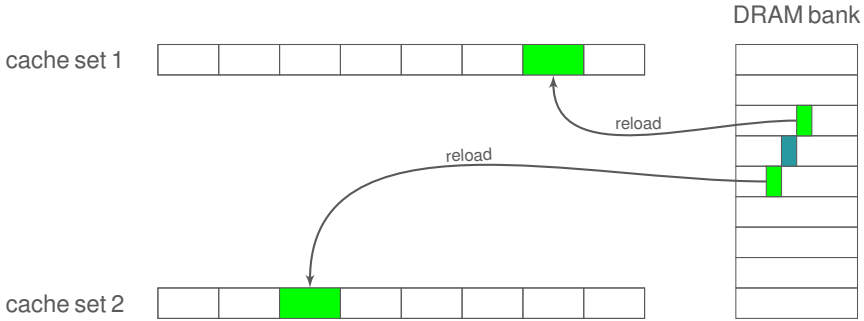
#1 Hammering with `clflush`



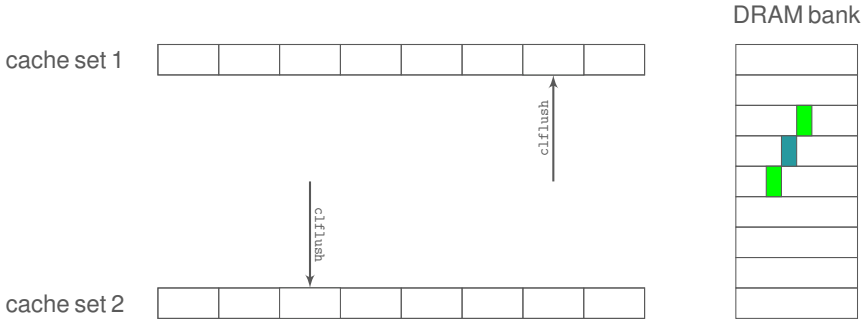
#1 Hammering with `clflush`



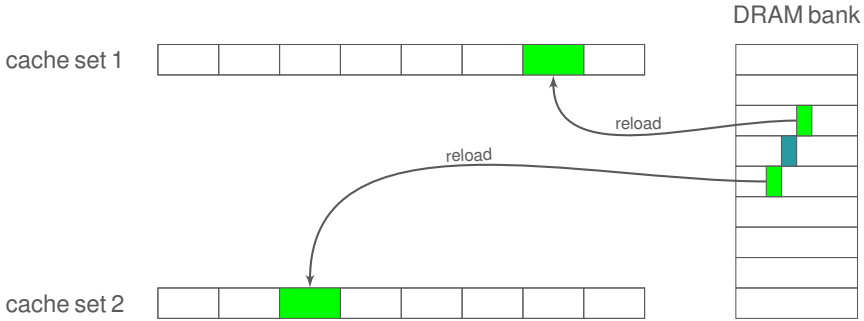
#1 Hammering with `clflush`



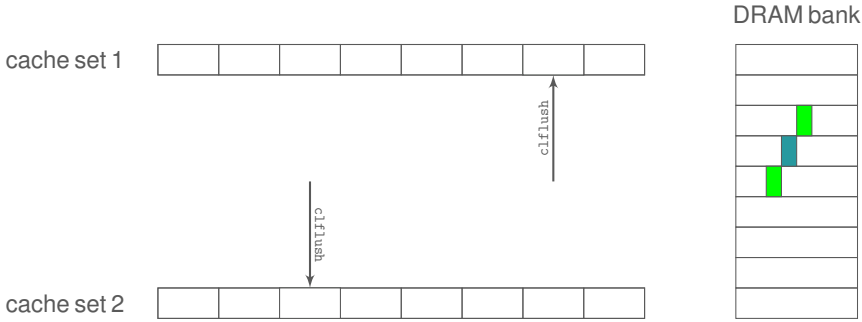
#1 Hammering with `clflush`



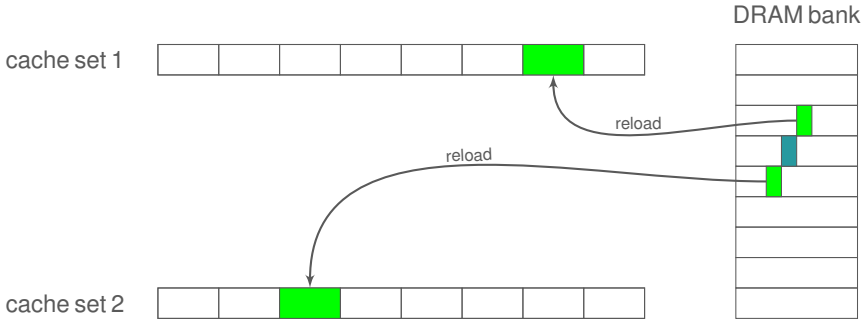
#1 Hammering with `clflush`



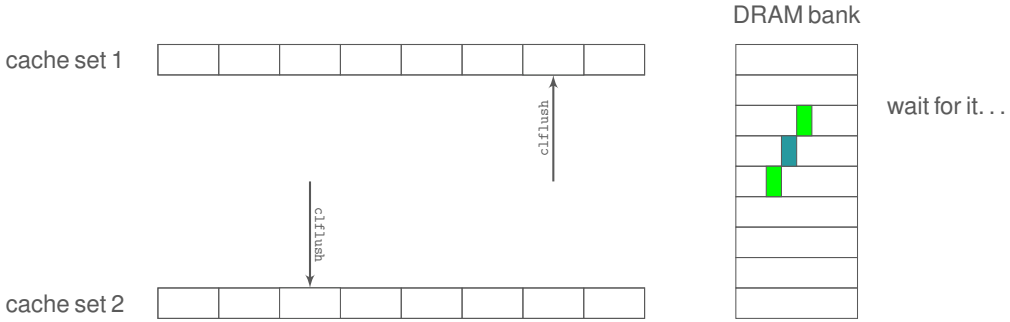
#1 Hammering with `clflush`



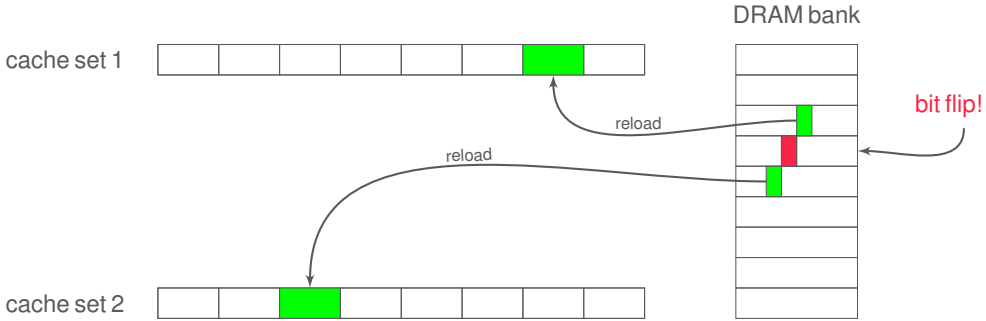
#1 Hammering with `clflush`



#1 Hammering with `clflush`



#1 Hammering with `clflush`



How widespread is the issue?

DDR3:

- Kim et al.: 110/129 modules from 3 vendors, all but 3 since mid-2011
- Seaborn and Dullien: 15/29 laptops

DDR4 believed to be safe:

- we showed bit flips (Pessl et al. 2016)

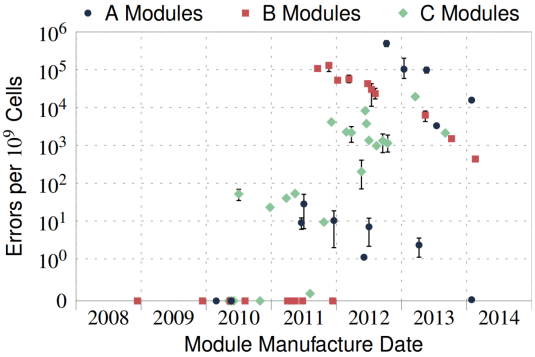


Figure: *

Flush, reload, flush, reload...

- the core of Rowhammer is essentially a Flush+Reload loop
- as much an attack on DRAM as on cache

#2 Hammering with cache eviction

- idea: avoid `clflush` to be independent of specific instructions

  $\rightarrow$ no `clflush` in JavaScript

#2 Hammering with cache eviction

- idea: avoid `clflush` to be independent of specific instructions

  → no `clflush` in JavaScript

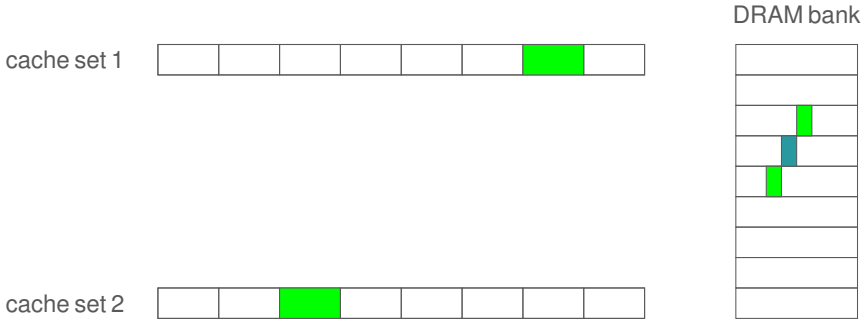
- our approach: use regular memory accesses for eviction

  → techniques from cache attacks!

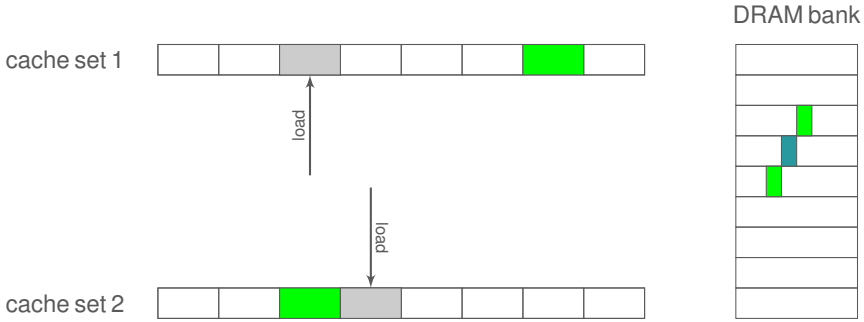
#2 Hammering with cache eviction

- idea: avoid `clflush` to be independent of specific instructions
  - → no `clflush` in JavaScript
- our approach: use regular memory accesses for eviction
  - → techniques from cache attacks!
  - → Rowhammer, Prime+Probe style!

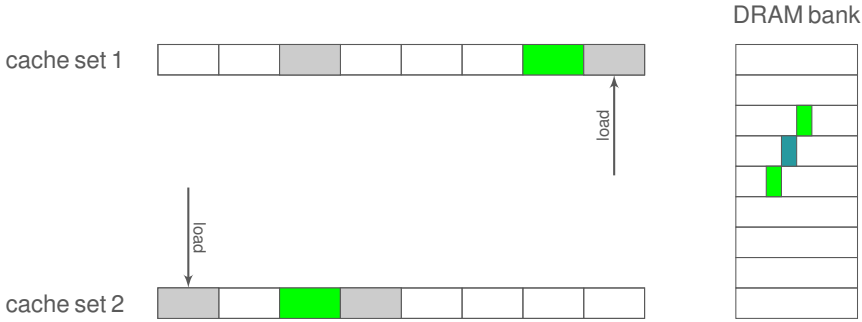
#2 Hammering with cache eviction



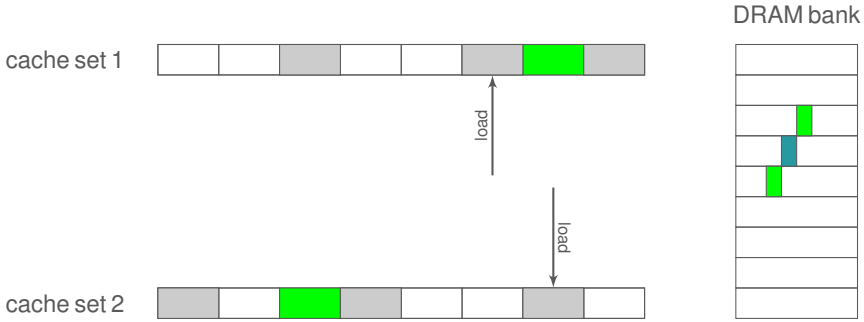
#2 Hammering with cache eviction



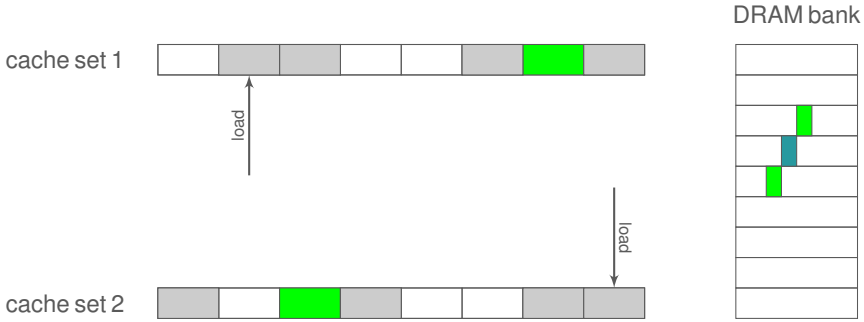
#2 Hammering with cache eviction



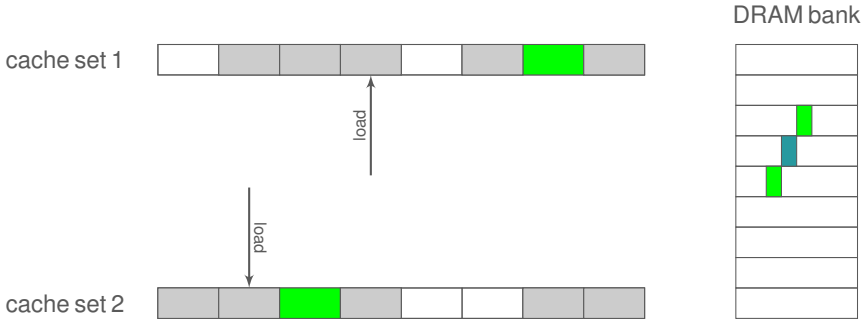
#2 Hammering with cache eviction



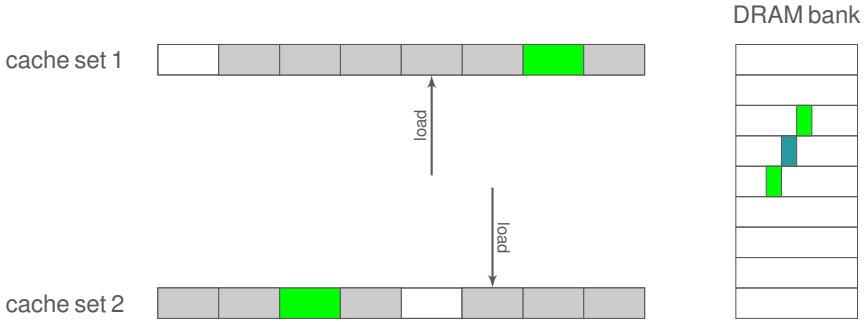
#2 Hammering with cache eviction



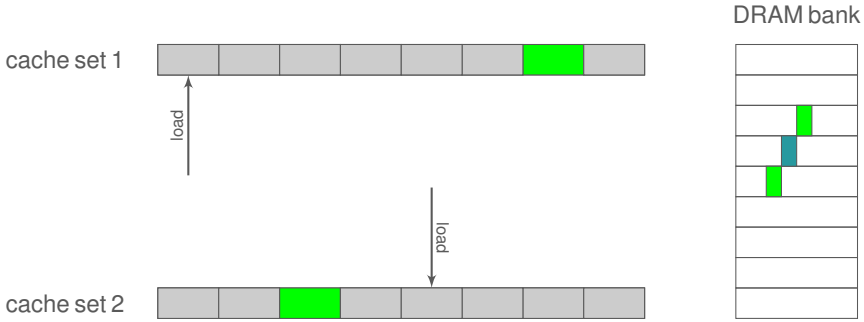
#2 Hammering with cache eviction



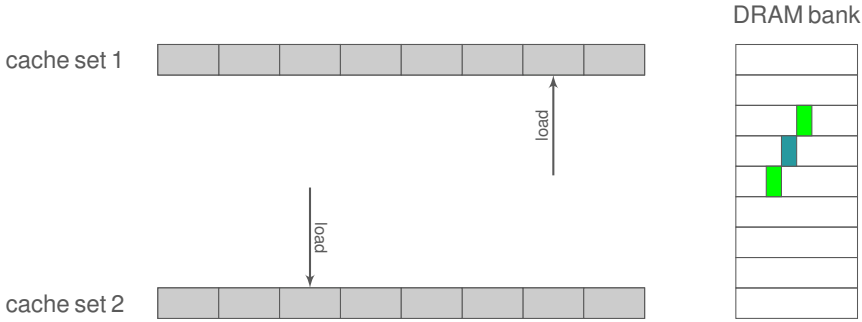
#2 Hammering with cache eviction



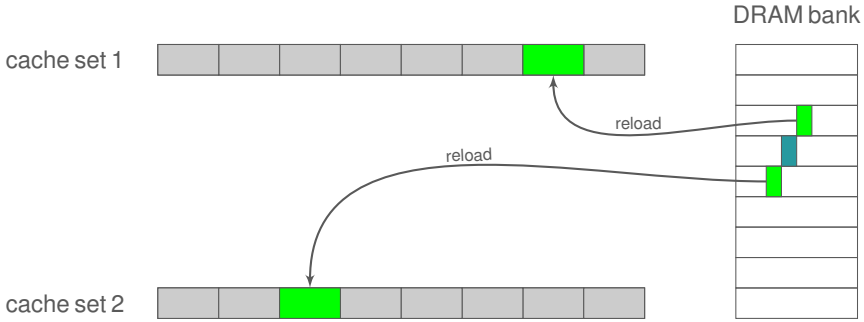
#2 Hammering with cache eviction



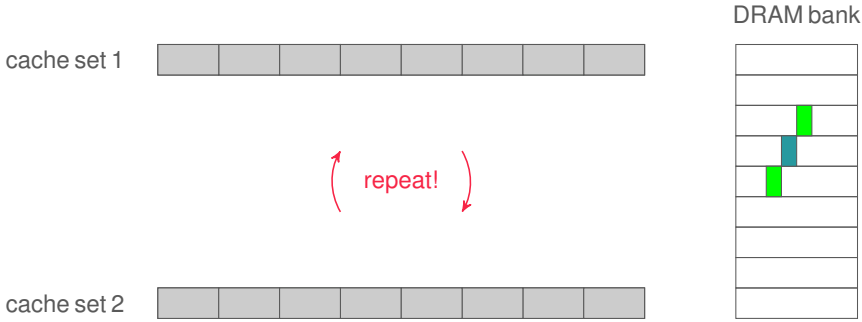
#2 Hammering with cache eviction



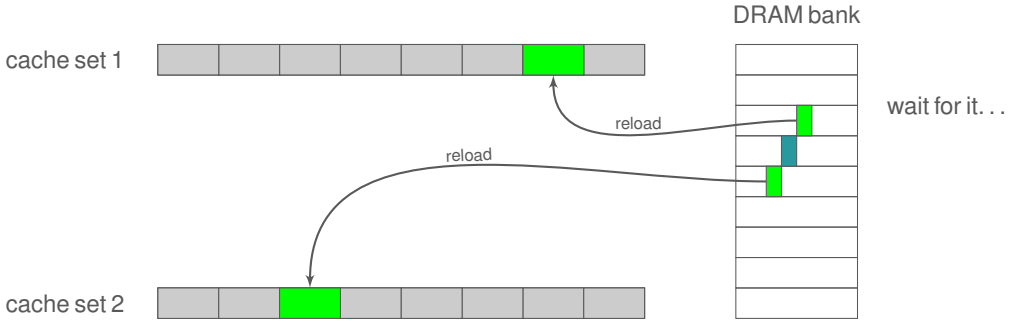
#2 Hammering with cache eviction



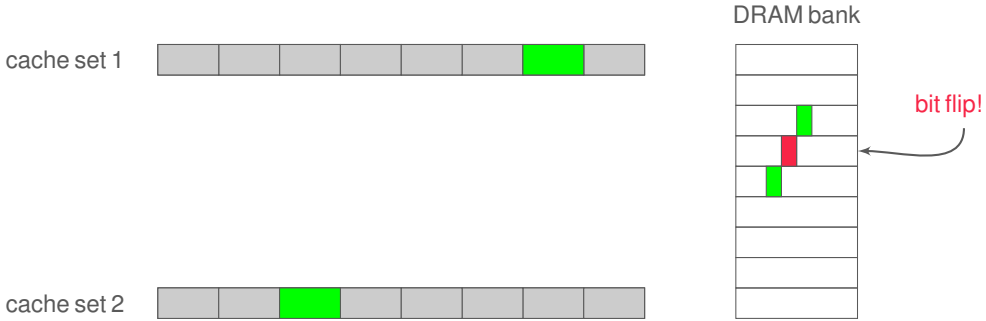
#2 Hammering with cache eviction



#2 Hammering with cache eviction



#2 Hammering with cache eviction

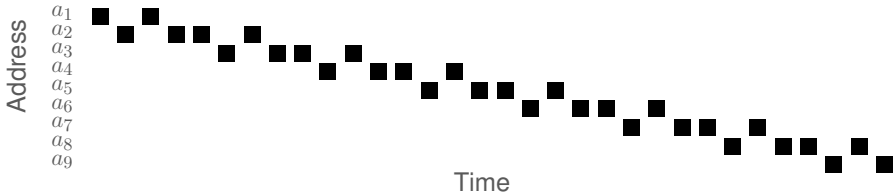


Cache eviction strategies

Not as simple as that $\rightarrow$ replacement policy is not LRU

Cache eviction strategies

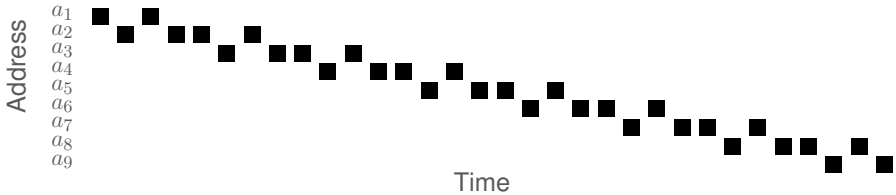
Not as simple as that → replacement policy is not LRU



→ fast and effective on Haswell: eviction rate >99.97%

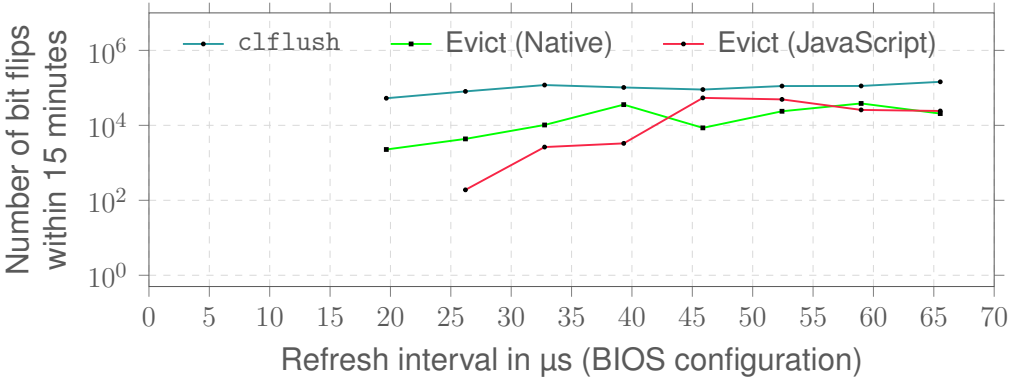
Cache eviction strategies

Not as simple as that → replacement policy is not LRU



→ fast and effective on Haswell: eviction rate >99.97%
→ we evaluated 10 000+ strategies to find the best one

Hammering with cache eviction on Haswell



#3 Hammering with non-temporal accesses

- non-temporal accesses: data accessed just once, not in the future
- NTA instructions $\rightarrow$ bypass cache to minimize cache pollution

#3 Hammering with non-temporal accesses

- non-temporal accesses: data accessed just once, not in the future
- NTA instructions → bypass cache to minimize cache pollution
- NT stores to 1 address are combined at WC buffer
- only last write goes to DRAM → rate not sufficient

#3 Hammering with non-temporal accesses

- non-temporal accesses: data accessed just once, not in the future
- NTA instructions $\rightarrow$ bypass cache to minimize cache pollution
- NT stores to 1 address are combined at WC buffer
- only last write goes to DRAM $\rightarrow$ rate not sufficient
- following cached access to same address

#3 Hammering with non-temporal accesses

```
begin:
  movnti %eax, (X)
  movnti %eax, (Y)
  mov %eax, (X)
  mov %eax, (Y)
  jmp begin
```

#4 Hammering with uncached memory

Sometimes, everything fails,

#4 Hammering with uncached memory

Sometimes, everything fails, e.g., on mobile devices

#4 Hammering with uncached memory

Sometimes, everything fails, e.g., on mobile devices

- ARMv7 flush instruction is privileged

#4 Hammering with uncached memory

Sometimes, everything fails, e.g., on mobile devices

- ARMv7 flush instruction is privileged
- cache eviction seems to be too slow

#4 Hammering with uncached memory

Sometimes, everything fails, e.g., on mobile devices

- ARMv7 flush instruction is privileged
- cache eviction seems to be too slow
- ARMv8 non-temporal stores are still cached in practice

#4 Hammering with uncached memory

- ION: memory management since Android 4.0
- apps can use `/dev/ion` for uncached, physically contiguous memory
- no privilege and no permission needed

How do we target accesses?

Physical addresses and DRAM

- fixed map: physical addresses $\rightarrow$ DRAM cells
- undocumented for Intel
- reverse-engineering for Sandy Bridge (Seaborn 2015)
- and by us for Sandy, Ivy, Haswell, Skylake, . . . (Pessl et al. 2016)
- using the timing difference between row hits and row conflicts

Rowhammer preparations

For starting it's easier with an empty file cache

```
sync && echo 3 | sudo tee /proc/sys/vm/drop_caches
```

and swap disabled

```
sudo swapoff -a
```

and with full CPU speed

```
sudo cpupower -c all set -b 0
```

How do I reverse my own DRAM?

https://github.com/IAIK/DRAMA

```
taskset 0x4 sudo ./measure -p 0.5 -s 16
# taskset core for stability
# sudo for pagemap access
# -p 0.5 allocate 50% of memory, the more the better
# -s I expect at least 16 sets (I have 32)
```

How do I flip bits?

https://github.com/IAIK/rowhammerjs

Copy functions from `measure` result

```
make ivy # or your microarchitecture
sudo ./rowhammer-ivy -d 2
# sudo for pagemap
# -d 2, for 2 DIMMs
sudo ./rowhammer-ivy -d 2 -f 0
# -f 0, only test offset 0 of every row
```

Demo

Demo!

Rowhammer without `clflush`?

- idea: avoid `clflush` to be independent of specific instructions
  $\rightarrow$ no `clflush` in JavaScript

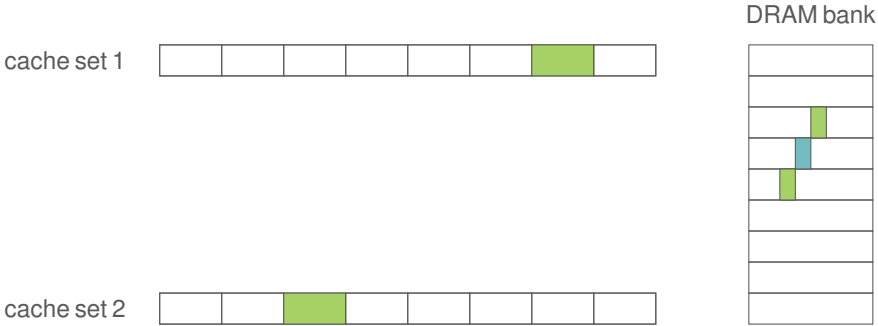
Rowhammer without `clflush`?

- idea: avoid `clflush` to be independent of specific instructions
  → no `clflush` in JavaScript
- our approach: use regular memory accesses for eviction
  → techniques from cache attacks!

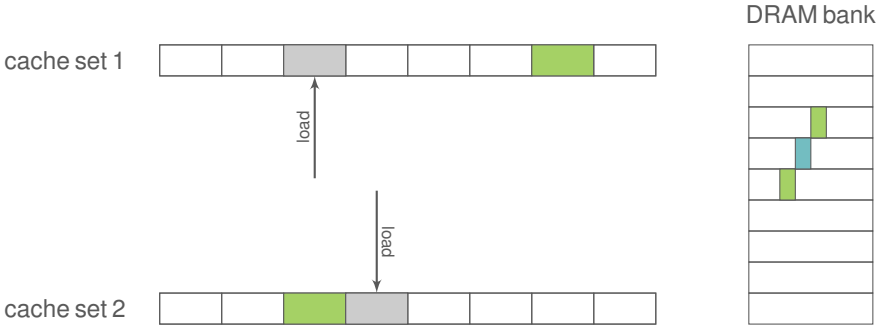
Rowhammer without `clflush`?

- idea: avoid `clflush` to be independent of specific instructions
  - → no `clflush` in JavaScript
- our approach: use regular memory accesses for eviction
  - → techniques from cache attacks!
  - → Rowhammer, Prime+Probe style!

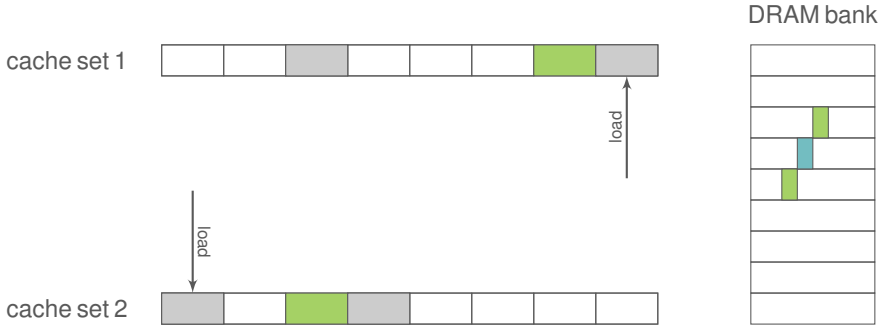
Rowhammer without `clflush`



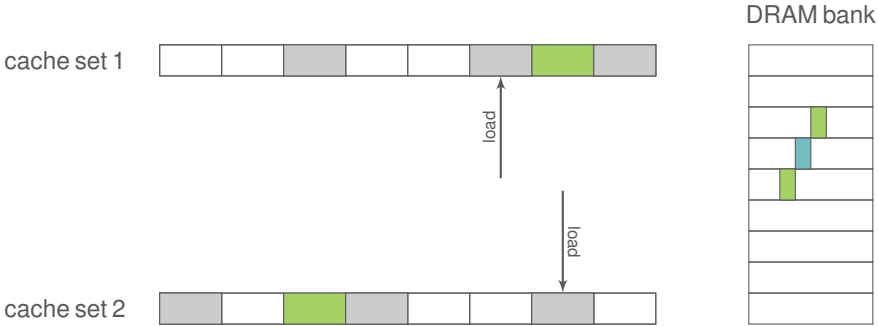
Rowhammer without `clflush`



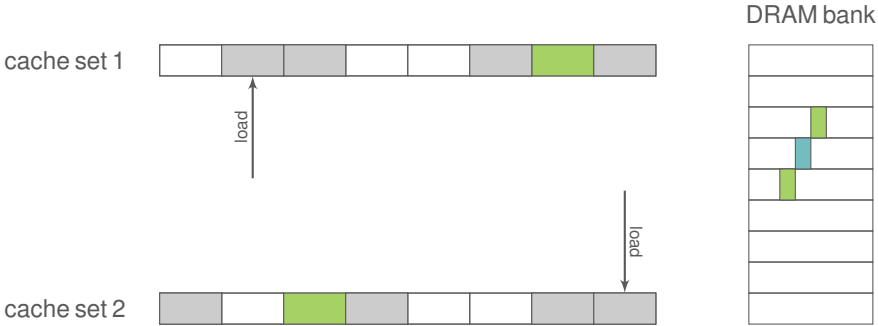
Rowhammer without `clflush`



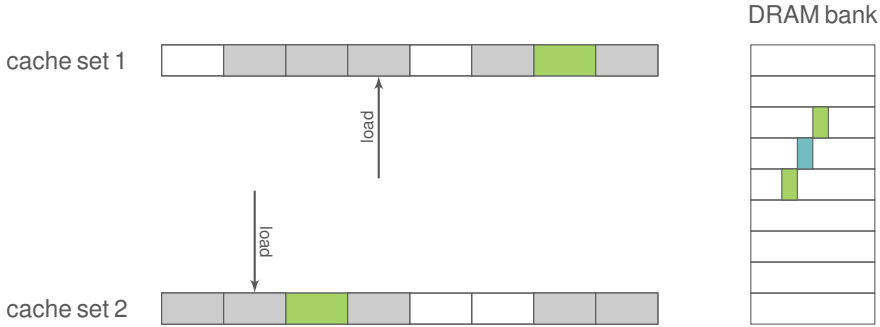
Rowhammer without `clflush`



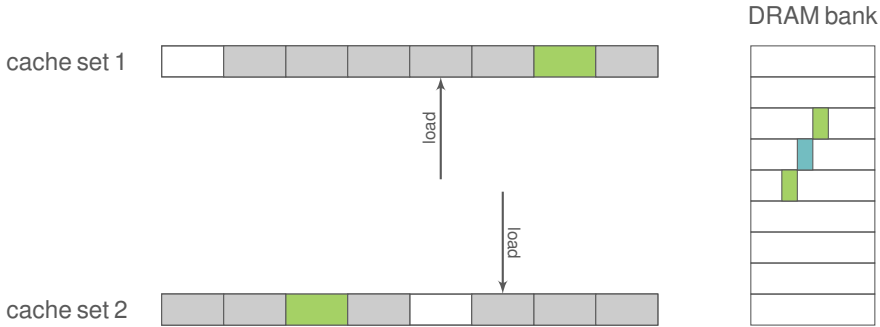
Rowhammer without `clflush`



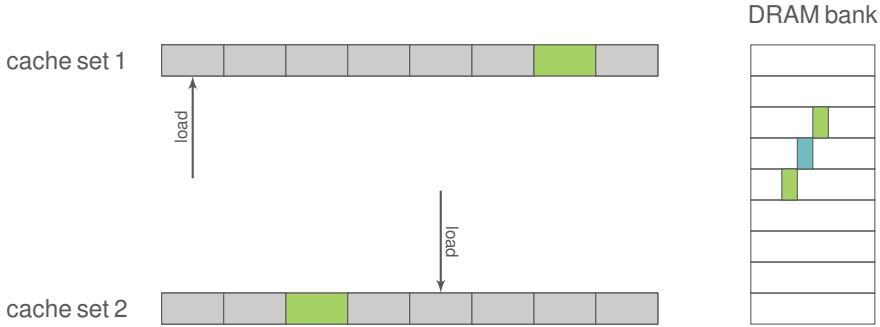
Rowhammer without `clflush`



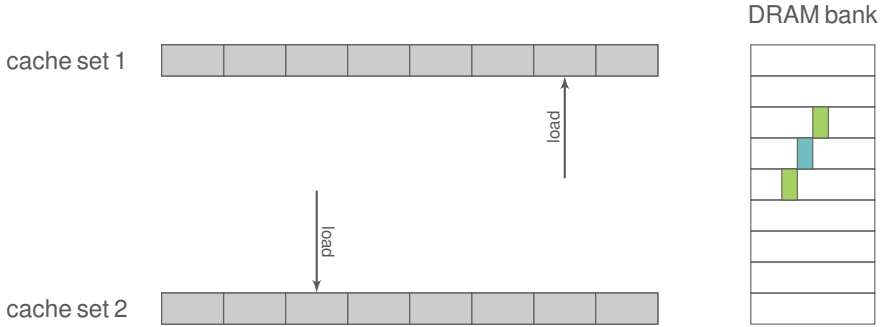
Rowhammer without `clflush`



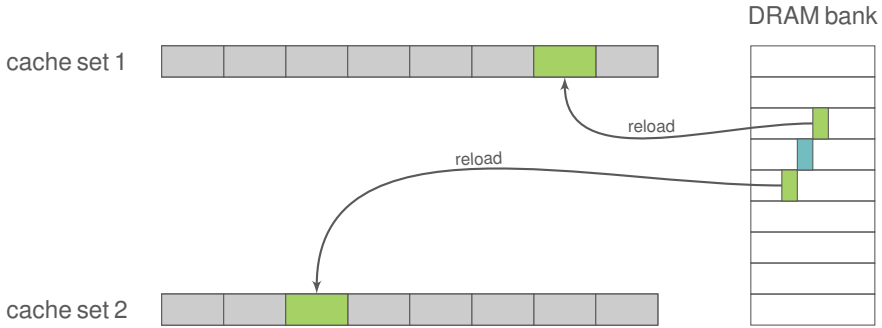
Rowhammer without `clflush`



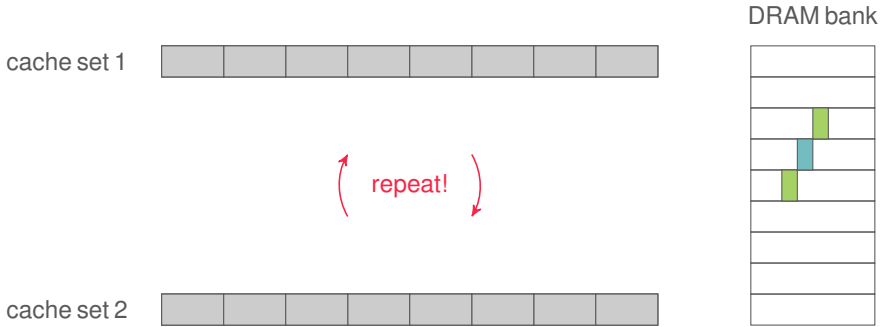
Rowhammer without `clflush`



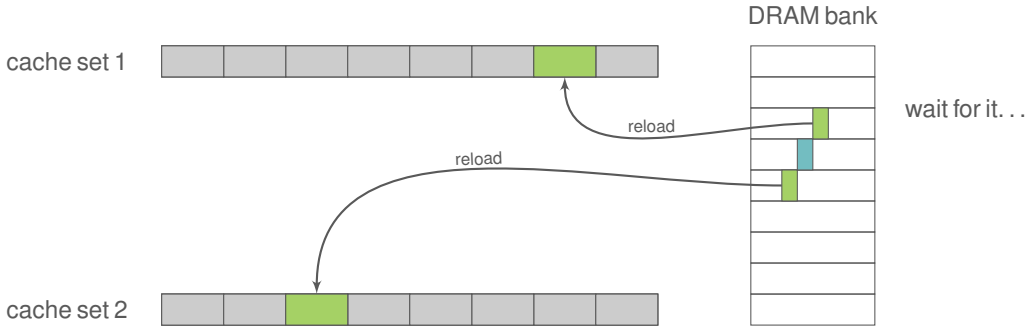
Rowhammer without `clflush`



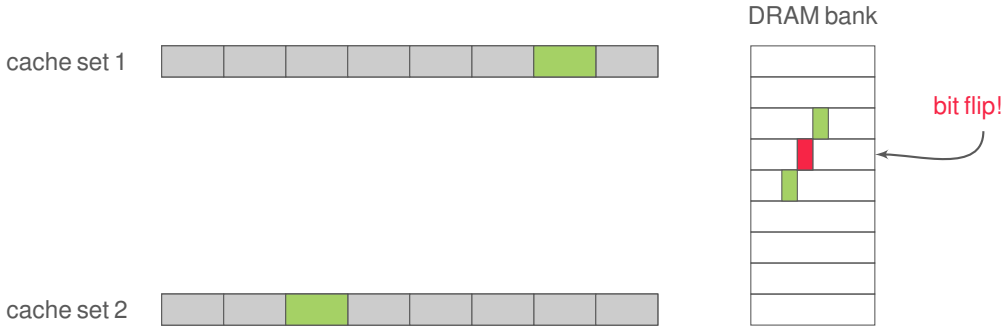
Rowhammer without `clflush`



Rowhammer without `clflush`



Rowhammer without `clflush`



Requirements for Rowhammer

1. uncached memory accesses: need to reach DRAM
2. fast memory accesses: race against the next row refresh

Requirements for Rowhammer

1. uncached memory accesses: need to reach DRAM
2. fast memory accesses: race against the next row refresh

→ optimize the eviction rate and the timing

Rowhammer.js: the challenges

1. how to get accurate timing in JS?
2. how to get physical addresses in JS?
3. which physical addresses to access?
4. in which order to access them?

Rowhammer.js: the challenges

1. how to get accurate timing in JS? $\rightarrow$ easy
2. how to get physical addresses in JS?
3. which physical addresses to access?
4. in which order to access them?

Rowhammer.js: the challenges

1. how to get accurate timing in JS? $\rightarrow$ easy
2. how to get physical addresses in JS? $\rightarrow$ easy
3. which physical addresses to access?
4. in which order to access them?

Rowhammer.js: the challenges

1. how to get accurate timing in JS? $\rightarrow$ easy
2. how to get physical addresses in JS? $\rightarrow$ easy
3. which physical addresses to access? $\rightarrow$ already solved
4. in which order to access them?

Rowhammer.js: the challenges

1. how to get accurate timing in JS? $\rightarrow$ easy
2. how to get physical addresses in JS? $\rightarrow$ easy
3. which physical addresses to access? $\rightarrow$ already solved
4. in which order to access them? $\rightarrow$ already earlier today

Challenge #1: accurate timing in JavaScript?

- native code: `rdtsc`
- JavaScript: `window.performance.now()`

Challenge #1: accurate timing in JavaScript?

- native code: `rdtsc`
- JavaScript: `window.performance.now()`

- recent patch: time rounded to 5 microseconds
- still works: we measure millions of accesses

Challenge #2: physical addresses and JavaScript

- OS optimization: use 2MB pages
- last 21 bits (2MB) of physical address
- = last 21 bits (2MB) of virtual address

Challenge #2: physical addresses and JavaScript

- OS optimization: use 2MB pages
- last 21 bits (2MB) of physical address
- = last 21 bits (2MB) of virtual address
- = last 21 bits (2MB) of JS array indices Gruss, Bidner, et al. 2015

Challenge #2: physical addresses and JavaScript

- OS optimization: use 2MB pages
- last 21 bits (2MB) of physical address
- = last 21 bits (2MB) of virtual address
- = last 21 bits (2MB) of JS array indices Gruss, Bidner, et al. 2015

- several DRAM rows per 2MB page
- several congruent addresses per 2MB page

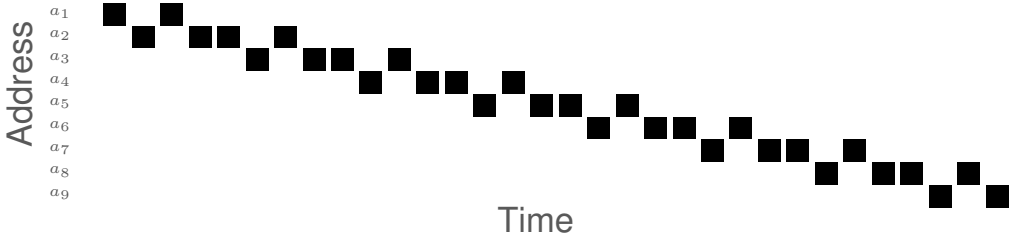
Challenge #3: physical addresses and DRAM

- fixed map: physical addresses $\rightarrow$ DRAM cells
- undocumented for Intel CPUs
- reverse-engineered for Sandy Bridge Seaborn 2015
- and by us for Sandy, Ivy, Haswell, Skylake, . . . Pessl et al. 2016

Challenge #3: physical addresses and cache sets

- fixed map: physical addresses $\rightarrow$ cache sets
- undocumented for Intel CPUs but reverse-engineered Maurice, Le Scouarnec, et al. 2015

Challenge #4: replacement policy



$\rightarrow$ fast and effective on Haswell: eviction rate $>$99.97%

Cache eviction strategy: New representation

- represent accesses as a sequence of numbers:
  $1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4, \ldots$
- can be a long sequence
- all congruent addresses are indistinguishable w.r.t eviction strategy

Cache eviction strategy: New representation

- represent accesses as a sequence of numbers:
  $1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4, ...$
- can be a long sequence
- all congruent addresses are indistinguishable w.r.t eviction strategy

$\rightarrow$ adding more unique addresses can increase eviction rate

$\rightarrow$ multiple accesses to one address can increase the eviction rate

- indistinguishable $\rightarrow$ balanced number of accesses

Cache eviction strategy: Notation (1)

Write eviction strategies as: P-C-D-L-S

```
for (s = 0; s <= S - D ; s += L )
  for (c = 0; c <= C ; c += 1)
    for (d = 0; d <= D ; d += 1)
      *a[s+d];
```

Cache eviction strategy: Notation (1)

Write eviction strategies as: P-C-D-L-S

S: total number of different addresses (= set size)

```
for (s = 0; s <= S - D; s += L)
  for (c = 0; c <= C; c += 1)
    for (d = 0; d <= D; d += 1)
      *a[s+d];
```


Daniel Gruss, Graz University of Technology

Cache eviction strategy: Notation (1)

Write eviction strategies as: P-C-D-L-S

S: total number of different addresses (= set size)

D: different addresses per inner access loop

```
for (s = 0; s <= S - D; s += L)
  for (c = 0; c <= C; c += 1)
    for (d = 0; d <= D; d += 1)
      *a[s+d];
```

Cache eviction strategy: Notation (1)

Write eviction strategies as: P-C-D-L-S

S: total number of different addresses (= set size)

D: different addresses per inner access loop

```
for (s = 0; s <= S - D; s += L)
  for (c = 0; c <= C; c += 1)
    for (d = 0; d <= D; d += 1)
      *a[s+d];
```

L: step size of the inner access loop

Cache eviction strategy: Notation (1)

Write eviction strategies as: P-C-D-L-S

S: total number of different addresses (= set size)

D: different addresses per inner access loop

```
for (s = 0; s <= S - D; s += L)
  for (c = 0; c <= C; c += 1)
    for (d = 0; d <= D; d += 1)
      *a[s+d];
```

L: step size of the inner access loop

C: number of repetitions of the inner access loop

Cache eviction strategy: Notation (2)

```
for (s = 0; s <= S - D; s += L)
  for (c = 1; c <= C; c += 1)
    for (d = 1; d <= D; d += 1)
      *a[s+d];
```

Cache eviction strategy: Notation (2)

```
for (s = 0; s <= S - D ; s += L )
  for (c = 1; c <= C ; c += 1)
    for (d = 1; d <= D ; d += 1)
      *a[s+d];
```

- P-2-2-1-4 $\rightarrow$ 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4

Cache eviction strategy: Notation (2)

```
for (s = 0; s <= S - D ; s += L )
  for (c = 1; c <= C ; c += 1)
    for (d = 1; d <= D ; d += 1)
      *a[s+d];
```

- P-2-2-1-4 $\rightarrow$ 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4

$S = 4$

Daniel Gruss, Graz University of Technology

Cache eviction strategy: Notation (2)

```
for (s = 0; s <= S - D ; s += L )
  for (c = 1; c <= C ; c += 1)
    for (d = 1; d <= D ; d += 1)
      *a[s+d];
```

- P-2-2-1-4 $\rightarrow$ 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4

$S = 4$

Daniel Gruss, Graz University of Technology

Cache eviction strategy: Notation (2)

```
for (s = 0; s <= S - D ; s += L )
  for (c = 1; c <= C ; c += 1)
    for (d = 1; d <= D ; d += 1)
      *a[s+d];
```

- P-2-2-1-4 $\rightarrow$ 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4

$S = 4$

$D = 2$

Cache eviction strategy: Notation (2)

```
for (s = 0; s <= S - D ; s += L )
  for (c = 1; c <= C ; c += 1)
    for (d = 1; d <= D ; d += 1)
      *a[s+d];
```

- P-2-2-1-4 $\rightarrow$ 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4

$S = 4$

$D = 2$

$C = 2$

Cache eviction strategy: Notation (2)

```
for (s = 0; s <= S - D ; s += L )
  for (c = 1; c <= C ; c += 1)
    for (d = 1; d <= D ; d += 1)
      *a[s+d];
```

- P-2-2-1-4 → 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4

$L = 1$ $D = 2$ $C = 2$ $S = 4$

Cache eviction strategy: Notation (2)

```
for (s = 0; s <= S - D ; s += L )
  for (c = 1; c <= C ; c += 1)
    for (d = 1; d <= D ; d += 1)
      *a[s+d];
```

- P-2-2-1-4 $\rightarrow$ 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4

  $L = 1$ $D = 2$ $C = 2$ $S = 4$

- P-1-1-1-4 $\rightarrow 1, 2, 3, 4 \rightarrow$ LRU eviction with set size 4

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17		
P-1-1-1-20	20		

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	
P-1-1-1-20	20	99.82% ✓	

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34		

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34	99.86% ✓	

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34	99.86% ✓	191 ns ✓

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34	99.86% ✓	191 ns ✓
P-2-2-1-17	64		

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34	99.86% ✓	191 ns ✓
P-2-2-1-17	64	99.98% ✓	

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34	99.86% ✓	191 ns ✓
P-2-2-1-17	64	99.98% ✓	180 ns ✓

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...[1]

strategy	# accesses	eviction rate	loop time
P-1-1-1-17	17	74.46% ✗	307 ns ✓
P-1-1-1-20	20	99.82% ✓	934 ns ✗
P-2-1-1-17	34	99.86% ✓	191 ns ✓
P-2-2-1-17	64	99.98% ✓	180 ns ✓

→ more accesses, smaller execution time?

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

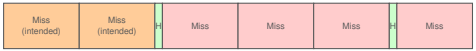
Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

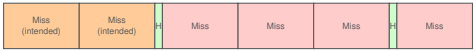
P-1-1-1-17 (17 accesses, 307ns)



P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

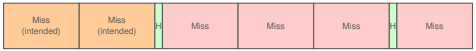
P-1-1-1-17 (17 accesses, 307ns)



P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

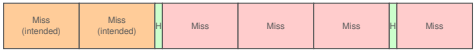


P-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

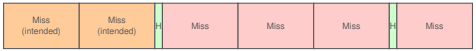
P-1-1-1-17 (17 accesses, 307ns)



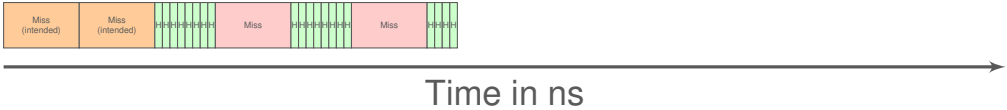
P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

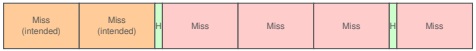


P-2-1-1-17 (34 accesses, 191ns)

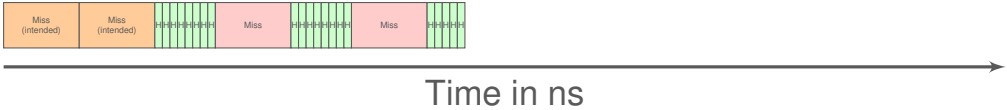


Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

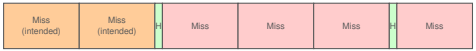


P-2-1-1-17 (34 accesses, 191ns)

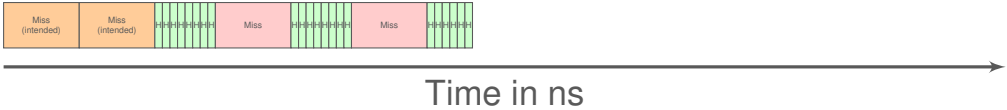


Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)



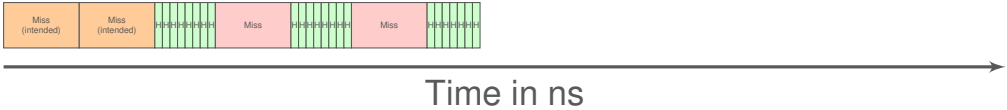
P-2-1-1-17 (34 accesses, 191ns)



Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

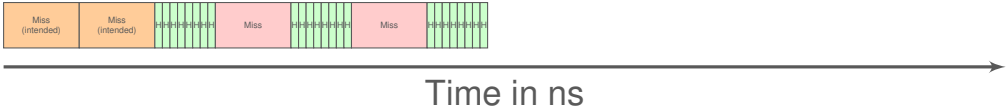
P-2-1-1-17 (34 accesses, 191ns)



Cache eviction strategies: Illustration

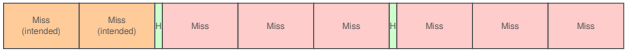
P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

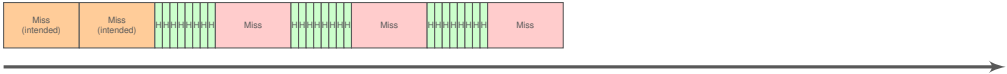


Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)



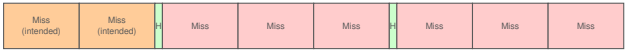
P-2-1-1-17 (34 accesses, 191ns)



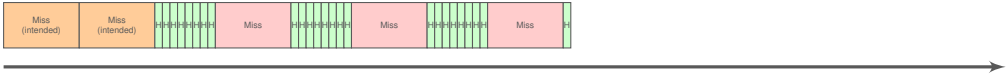
Time in ns

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)



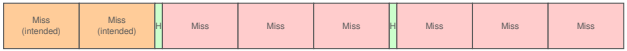
P-2-1-1-17 (34 accesses, 191ns)



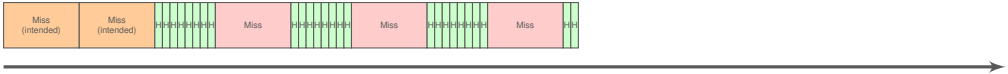
Time in ns

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

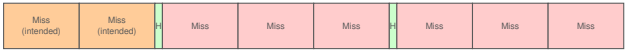


P-2-1-1-17 (34 accesses, 191ns)

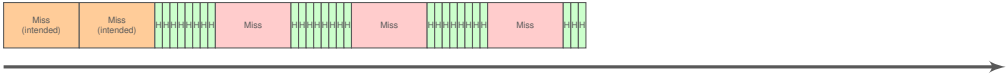


Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

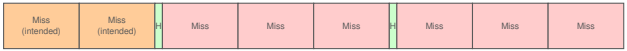


P-2-1-1-17 (34 accesses, 191ns)



Cache eviction strategies: Illustration

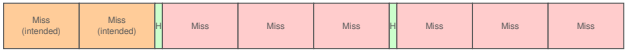
P-1-1-1-17 (17 accesses, 307ns)



P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

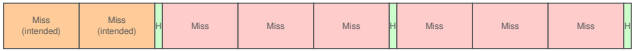
P-1-1-1-17 (17 accesses, 307ns)



P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)



P-2-1-1-17 (34 accesses, 191ns)

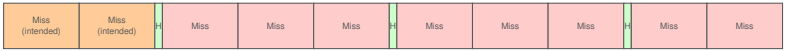
Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)



P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

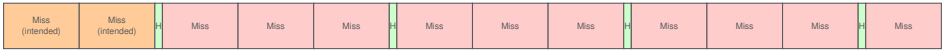
Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

P-1-1-1-17 (17 accesses, 307ns)

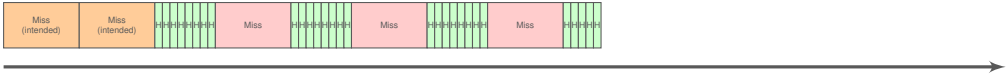


P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

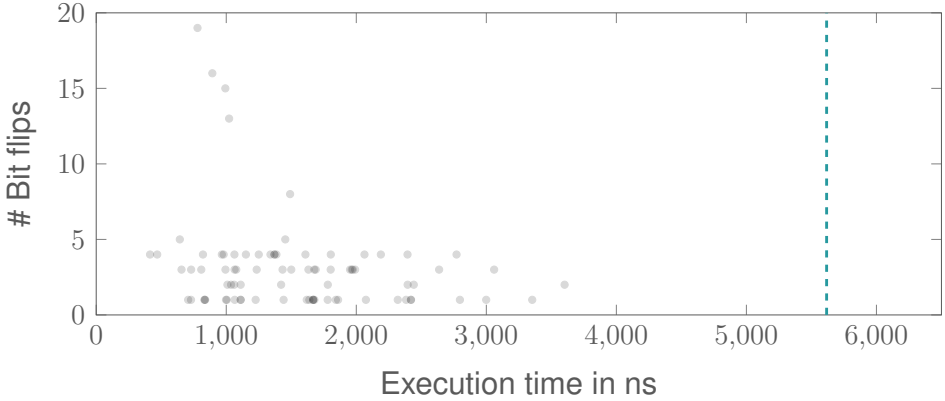
P-1-1-1-17 (17 accesses, 307ns)

P-2-1-1-17 (34 accesses, 191ns)



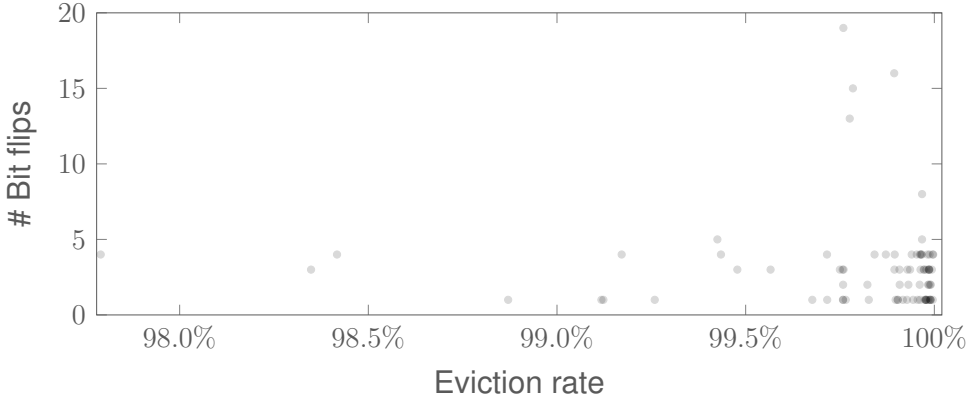
Time in ns

Execution time vs. bit flips



→ low execution time is better.

Eviction rate vs. bit flips



→ high eviction rate is better. Average: 73.96%.

Eviction strategies on Haswell

Table: The fastest 5 eviction strategies with an eviction rate above 99.75% compared to `clflush` and LRU eviction on Haswell.

C	D	L	S	Accesses	Hits	Misses	Time (ns)	Eviction
-	-	-	-	-	2	2	60	99.9999%
5	2	2	18	90	34	4	179	99.9624%
2	2	1	17	64	35	5	180	99.9820%
2	1	1	17	34	47	5	191	99.8595%
6	2	2	18	108	34	5	216	99.9365%
1	1	1	17	17	96	13	307	74.4593%
4	2	2	20	80	41	23	329	99.7800%
1	1	1	20	20	187	78	934	99.8200%

Evaluation on Haswell

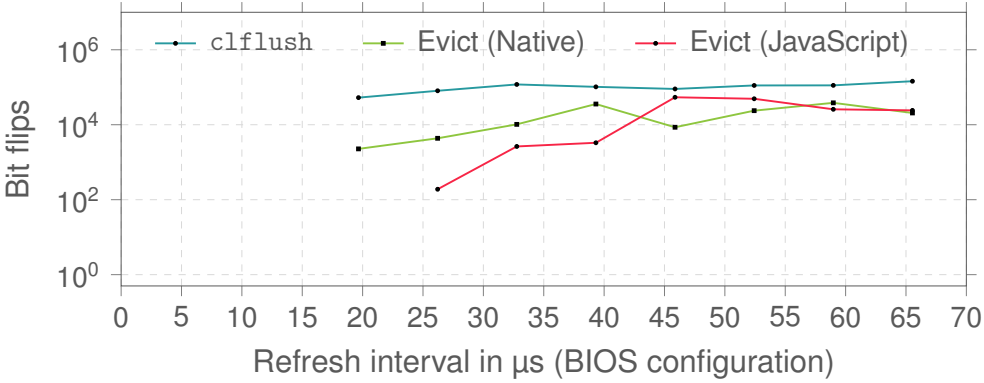


Figure: Number of bit flips within 15 minutes.

How to exploit random bit flips?

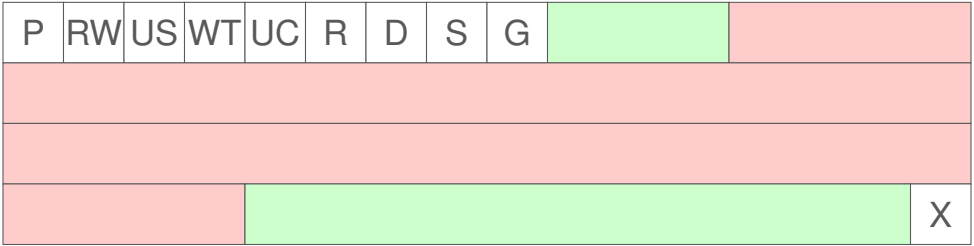
- They are not random → highly reproducible flip pattern!

  1. choose a data structure that you can place at arbitrary memory locations
  2. scan for “good” flips
  3. place data structure there
  4. trigger bit flip again

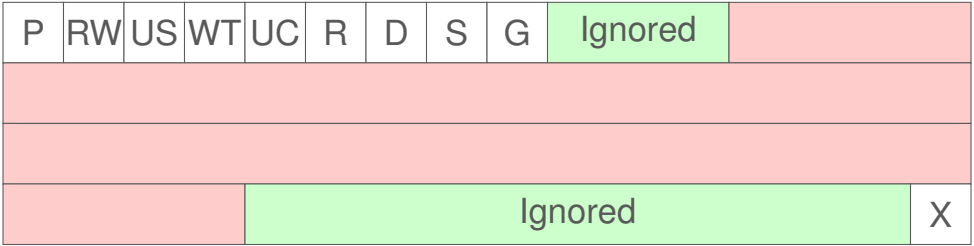
Strategy: Modify instructions

- idea from Seaborn and Dullien 2015
- x86 op codes are variable length
  - unsafe op codes (syscall) $\in$ safe but long multi-byte op codes
  - only a problem with jumps to arbitrary addresses
- flip a bit in a validated NaCl instruction sequence
  - safe + validated jump $\rightarrow$ arbitrary jump

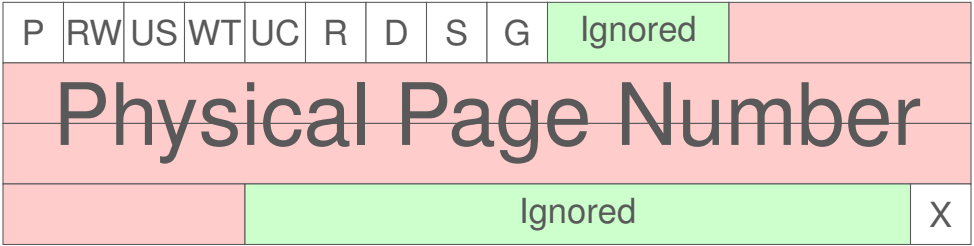
Page Table Entries



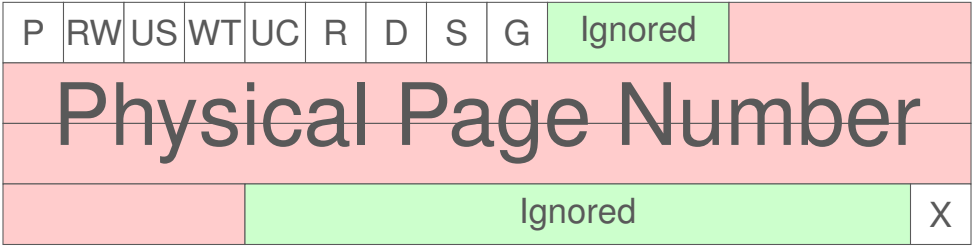
Page Table Entries



Page Table Entries

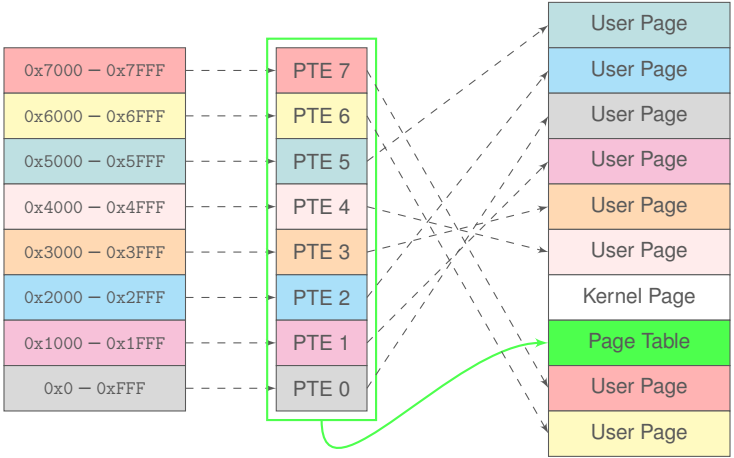


Page Table Entries

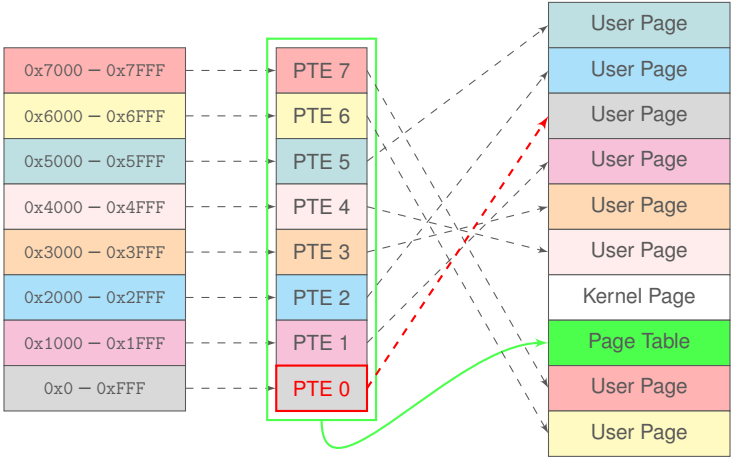


Each 4 KB page table consists of 512 such entries

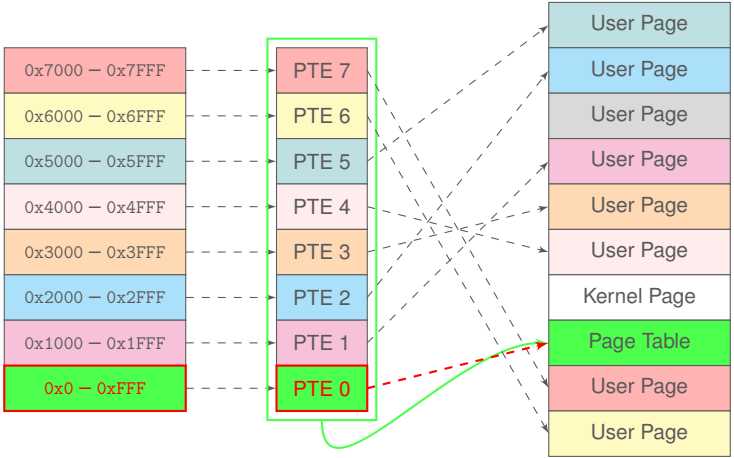
Page Table Manipulation



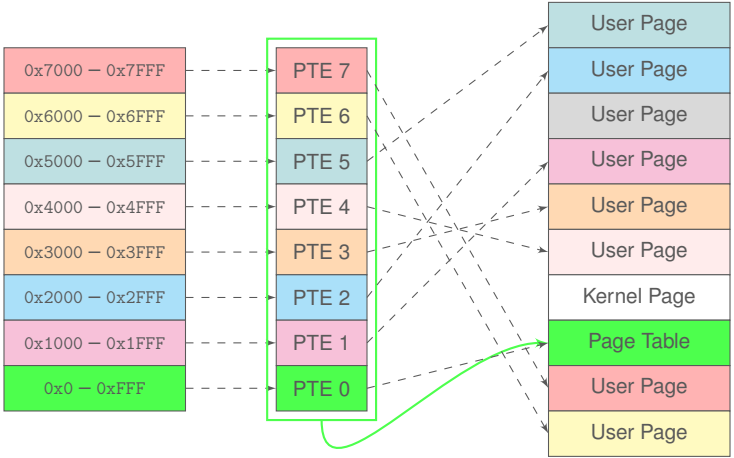
Page Table Manipulation



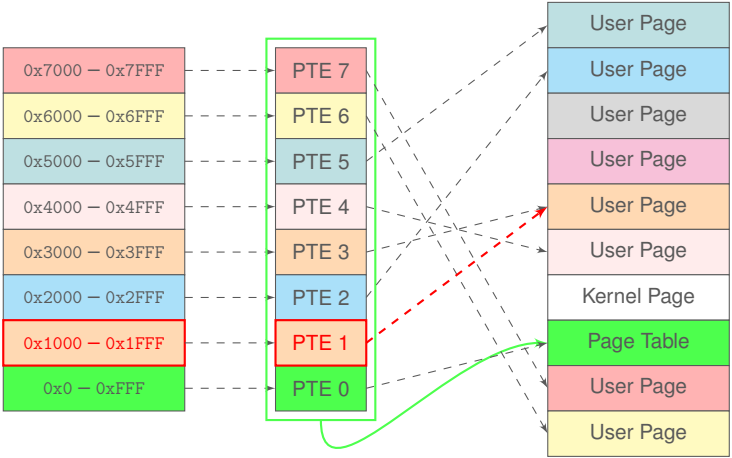
Page Table Manipulation



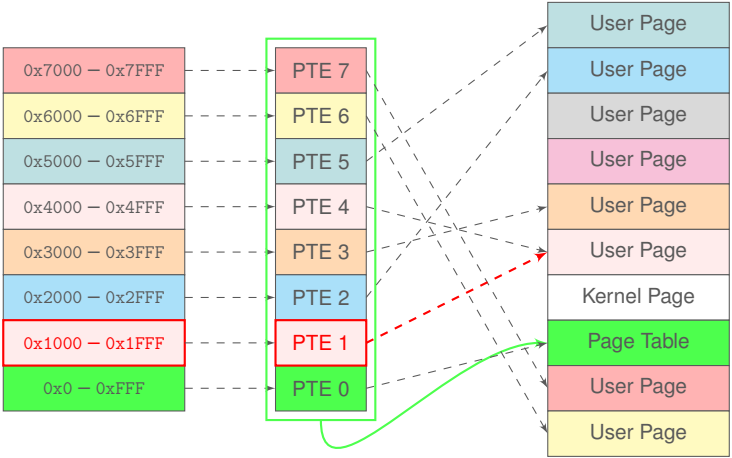
Page Table Manipulation



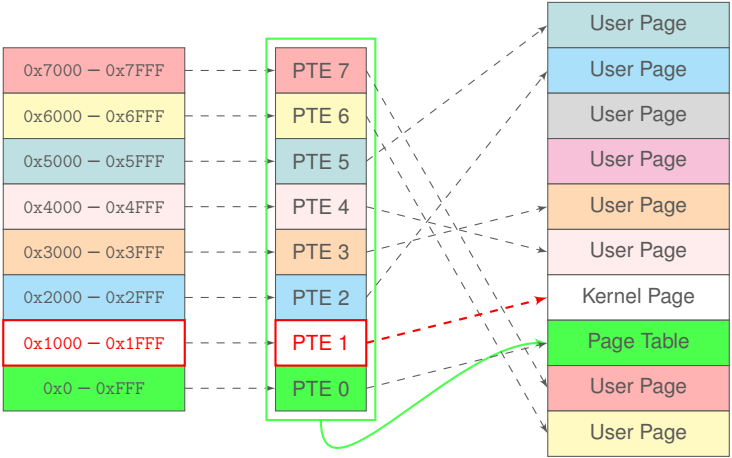
Page Table Manipulation



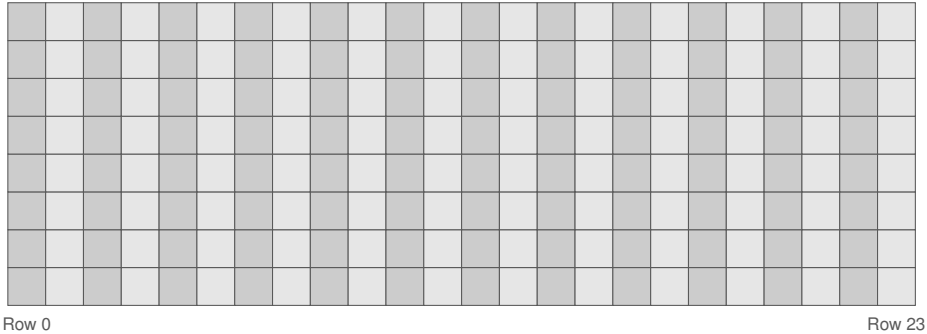
Page Table Manipulation



Page Table Manipulation

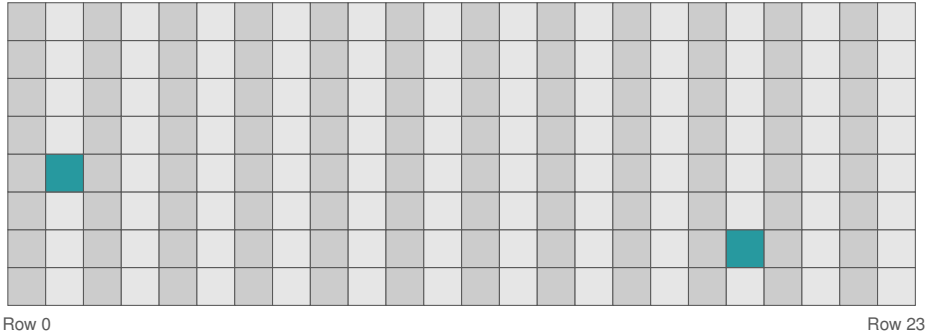


Search for page with flip



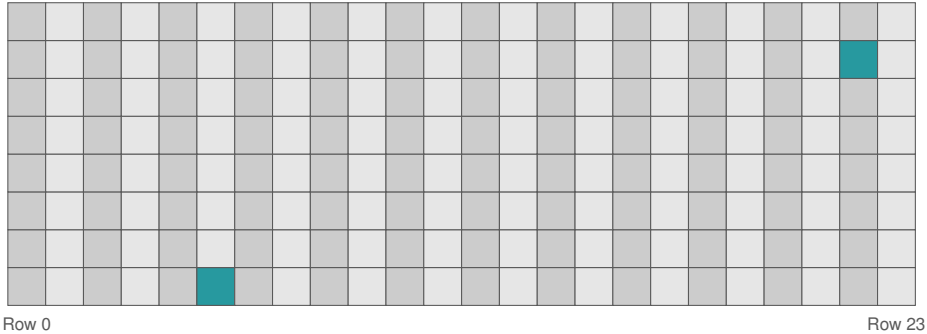
Hammering memory locations in different rows

Search for page with flip



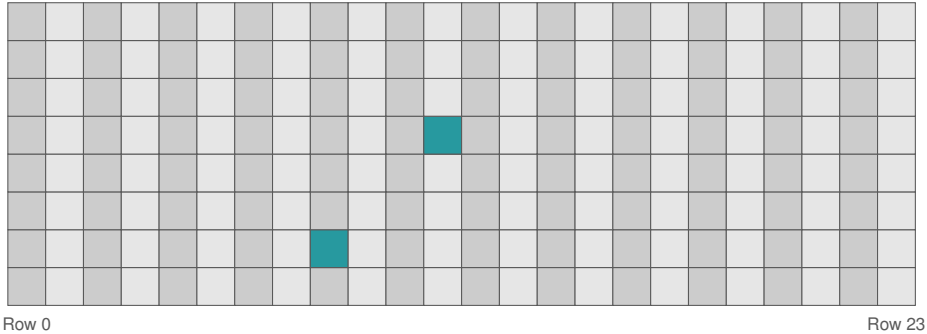
Hammering memory locations in different rows

Search for page with flip



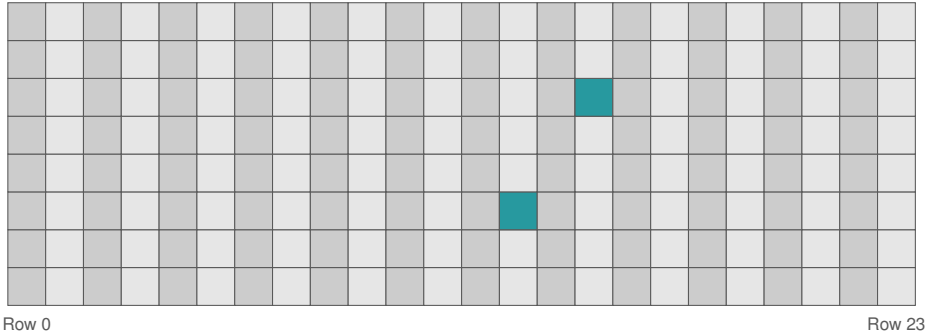
Hammering memory locations in different rows

Search for page with flip



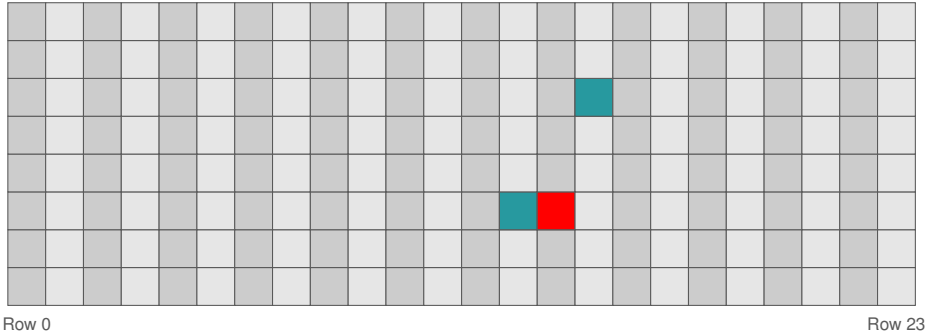
Hammering memory locations in different rows

Search for page with flip



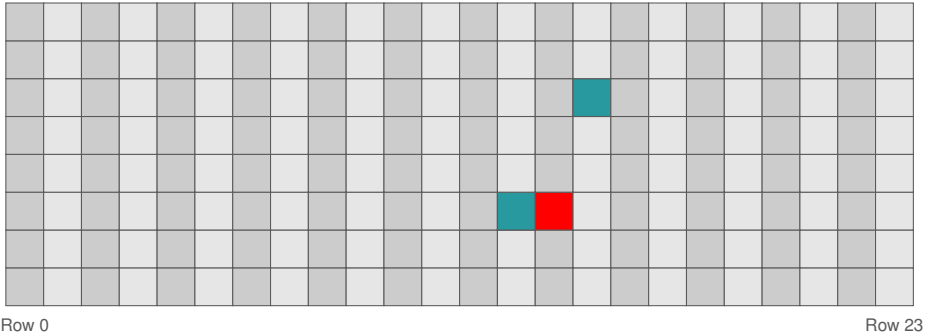
Hammering memory locations in different rows

Search for page with flip

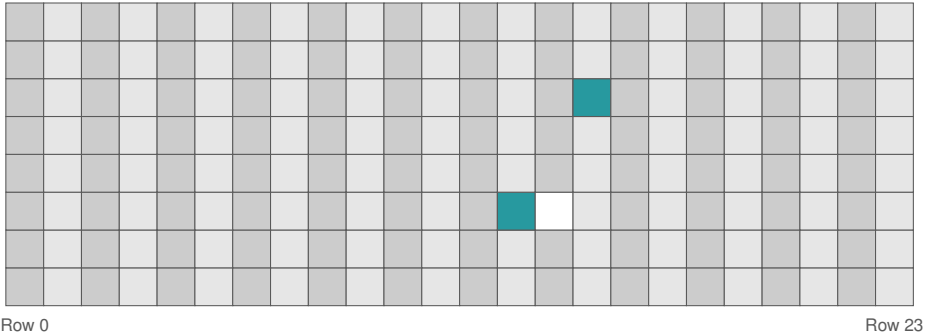


Hammering memory locations in different rows

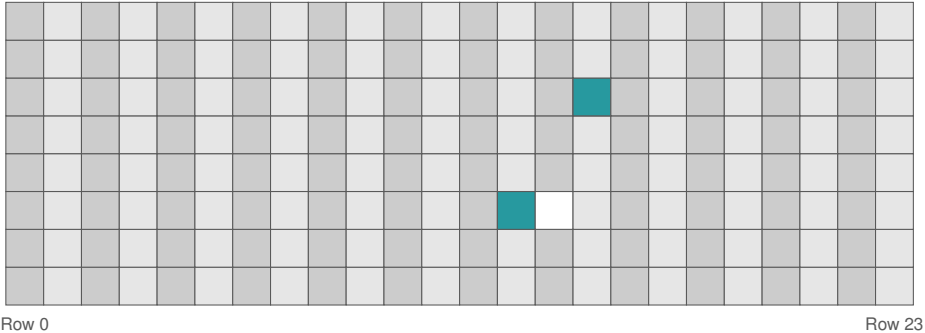
Release page with flip



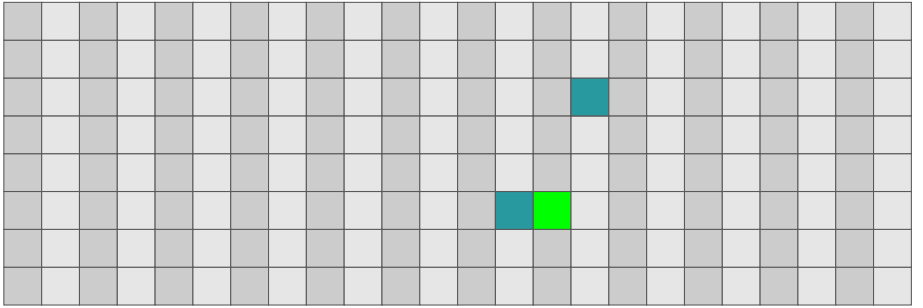
Release page with flip



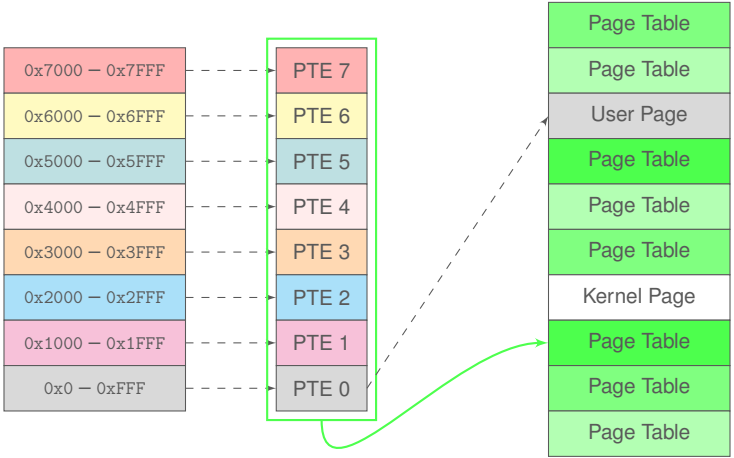
Fill all remaining memory with page tables



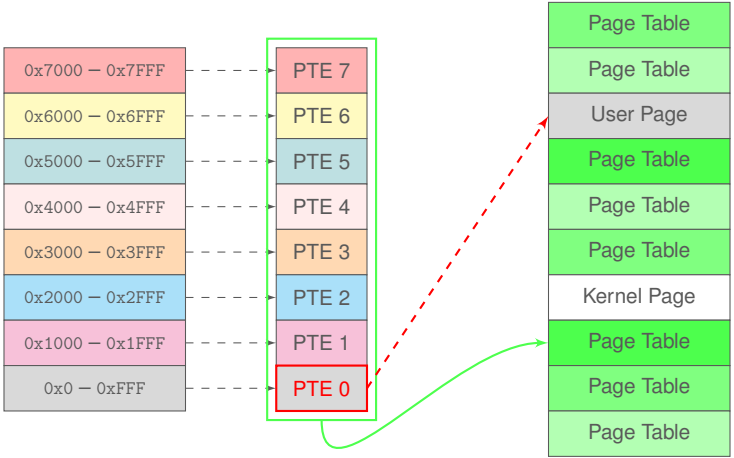
Fill all remaining memory with page tables



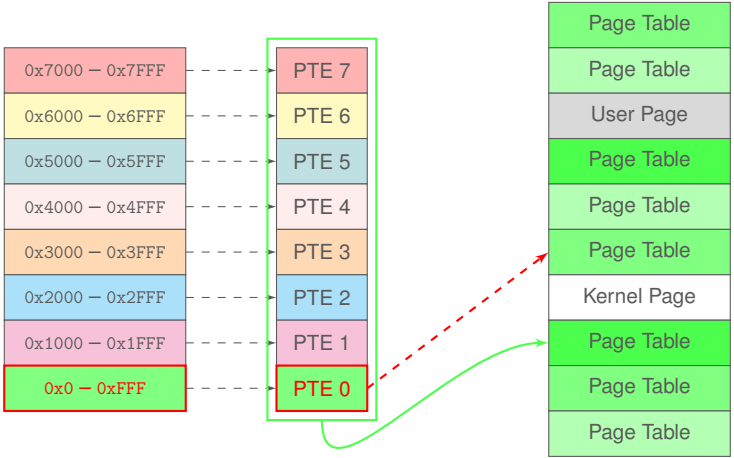
Page Table Manipulation



Page Table Manipulation



Page Table Manipulation



Strategy: Flipping Page Table PPN bits

1. scan for flips
2. exhaust or massage memory to place a page table at target location
3. gain access to your own page table $\rightarrow$ kernel privileges

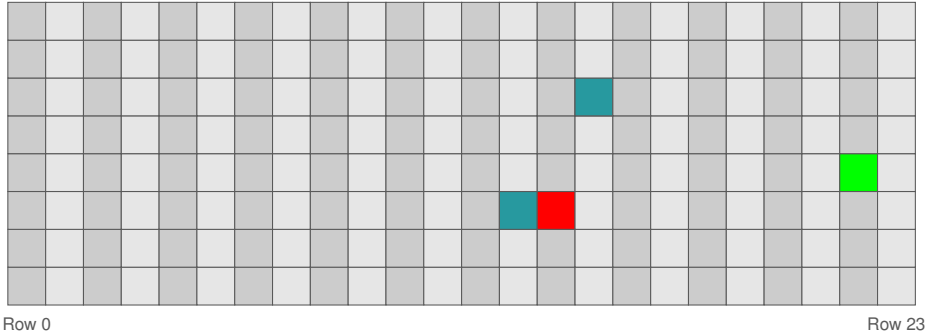
Flipping Page Table PPN bits

- idea from Seaborn and Dullien 2015
- same idea applied in several other works:
  - Rowhammer.js (Gruss, Maurice, and Mangard 2016)
  - One bit flips, one cloud flops (Y. Xiao et al. 2016)
  - Drammer (Veen et al. 2016)

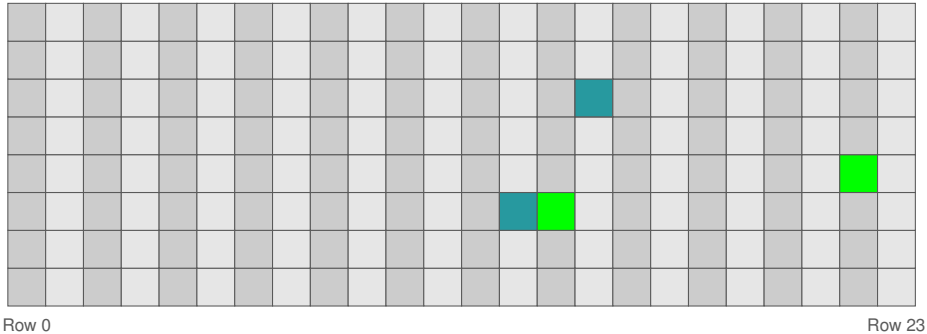
Post-Rowhammer Exploitation

- scan entire physical memory (very fast) and:
  - modify binary pages executed in root privileges (Y. Xiao et al. 2016)
  - modify credential structs (Veen et al. 2016)
  - read keys (Y. Xiao et al. 2016)
  - corrupt RSA signatures (Bhattacharya and Mukhopadhyay 2016)
  - modify certificates
  - configurations
  - etc.
- pages are pretty unique: 32768 bits per page

Bit Flips + Page Deduplication

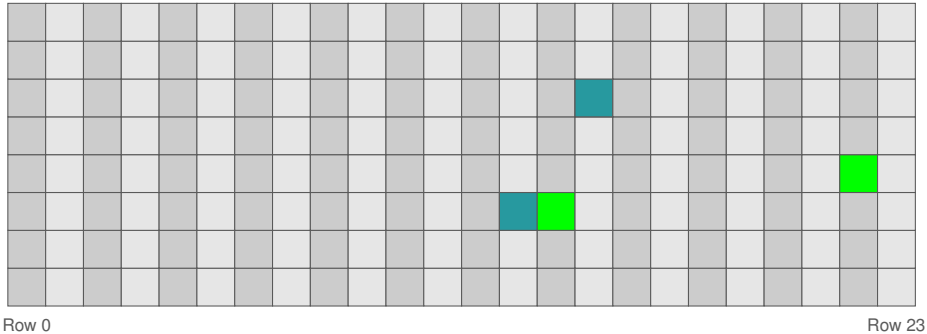


Bit Flips + Page Deduplication



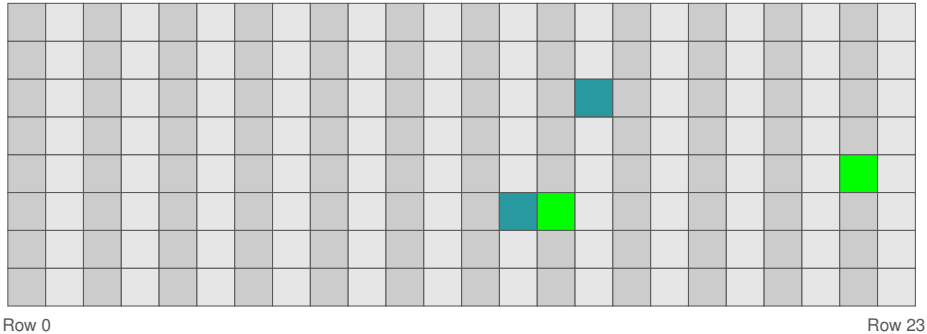
Page with bit flip is filled with target content

Bit Flips + Page Deduplication



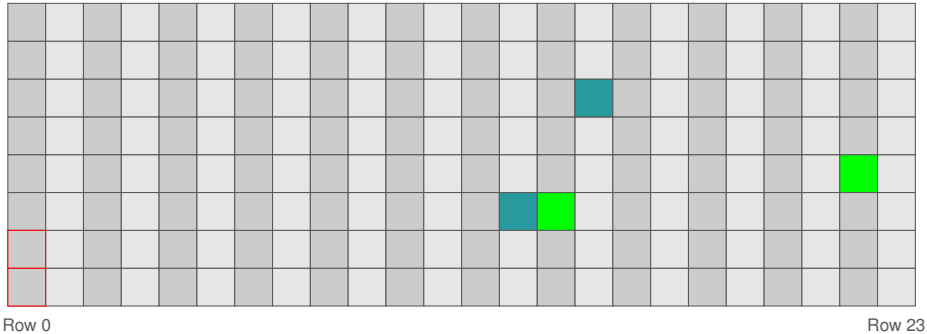
OS or hypervisor searches for duplicate pages

Bit Flips + Page Deduplication



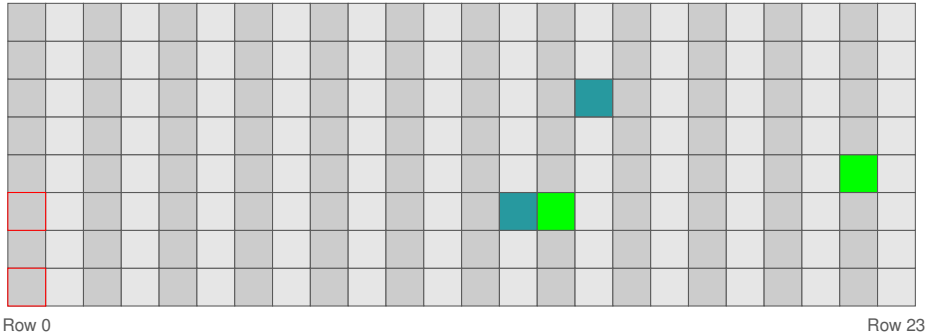
OS or hypervisor searches for duplicate pages

Bit Flips + Page Deduplication



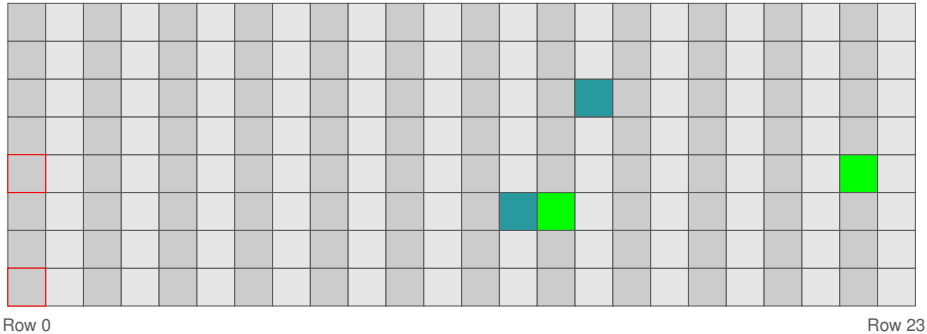
OS or hypervisor searches for duplicate pages

Bit Flips + Page Deduplication



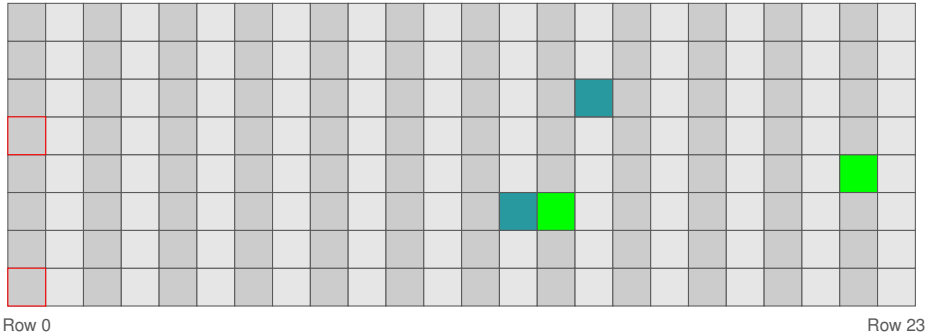
OS or hypervisor searches for duplicate pages

Bit Flips + Page Deduplication



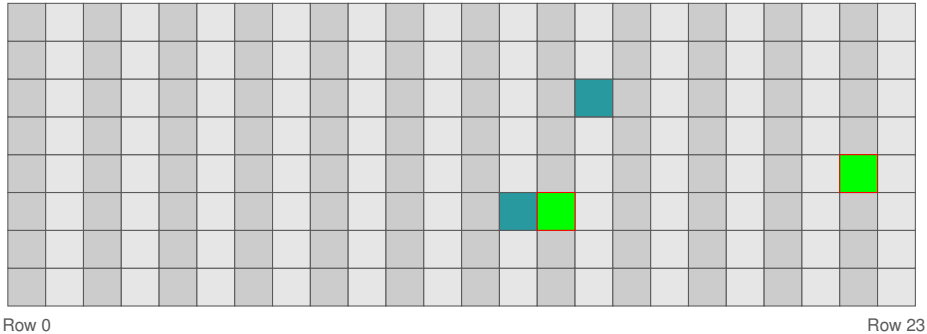
OS or hypervisor searches for duplicate pages

Bit Flips + Page Deduplication



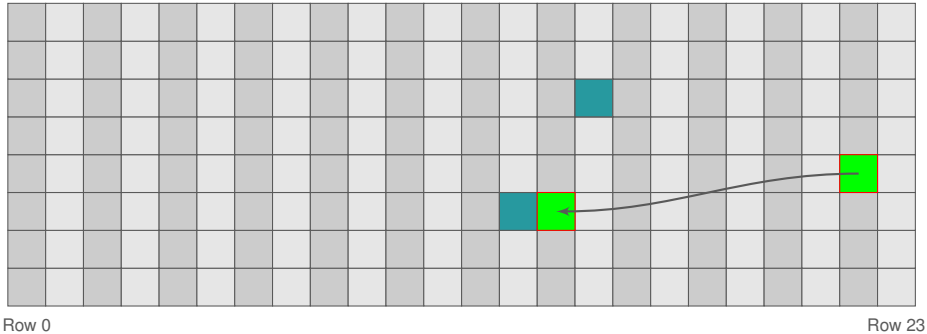
OS or hypervisor searches for duplicate pages

Bit Flips + Page Deduplication



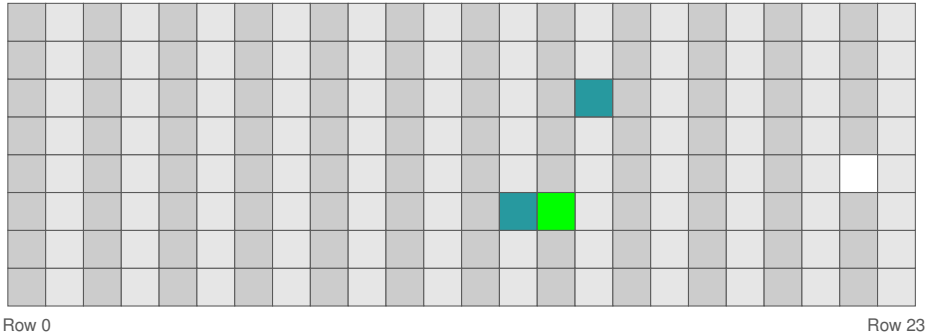
OS or hypervisor searches for duplicate pages

Bit Flips + Page Deduplication



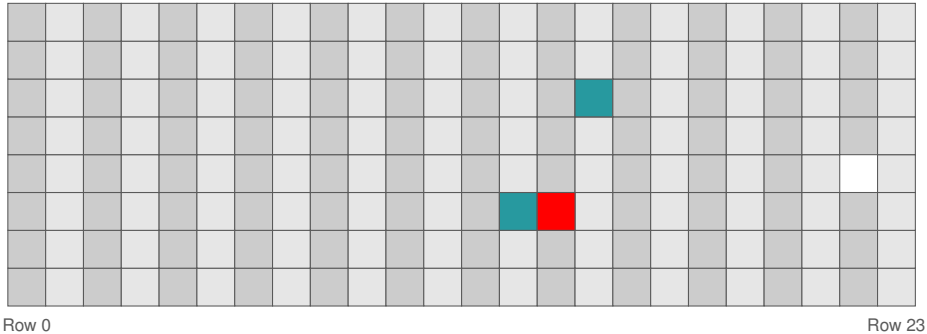
OS or hypervisor searches for duplicate pages

Bit Flips + Page Deduplication



Hammer again + flip again

Bit Flips + Page Deduplication



Strategy: Flipping in Deduplicated Pages

1. scan for flips
2. place content for deduplication so that flip can be exploited
3. perform the bit change through Rowhammer

Flipping in Deduplicated Pages

- idea from Bosman et al. 2016
  - change data type (double $\rightarrow$ pointer)
  - change pointer to good object $\rightarrow$ counterfeit object
- and from Razavi et al. 2016
  - corrupt authorized SSH keys
  - corrupt Debian update URLs + RSA public key file

Mitigations

Different mitigations have been proposed:

- detection vs prevention
- software vs hardware
- short-term vs long-term

Quick fixes

- no `clflush` instruction

Quick fixes

- no `clflush` instruction → Rowhammer.js

Quick fixes

- no `clflush` instruction → Rowhammer.js
- increase the refresh rate

Quick fixes

- no `clflush` instruction → Rowhammer.js
- increase the refresh rate
  → would need to be increased by 7× to eliminate all bit flips

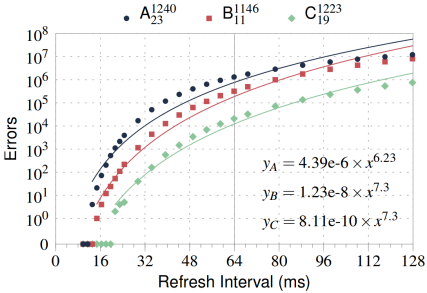


Figure: *

Errors depending on refresh interval (Kim et al. 2014)

Quick fixes

- no `clflush` instruction → Rowhammer.js
- increase the refresh rate
  - → would need to be increased by 7× to eliminate all bit flips
  - → implementation: increased by 2× by BIOS vendors

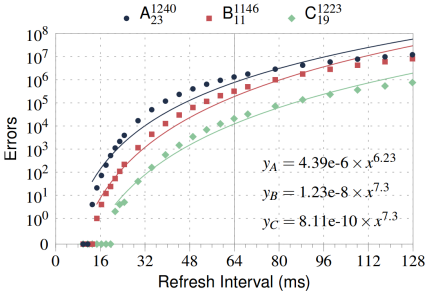


Figure: *

Errors depending on refresh interval (Kim et al. 2014)

What about ECC?

- ECC protection: server can handle or correct single bit errors

What about ECC?

- ECC protection: server can handle or correct single bit errors
- no standard for event reporting

What about ECC?

- ECC protection: server can handle or correct single bit errors
- no standard for event reporting
- in practice (Lanteigne 2016)
  - common: server counts ECC errors and report only if they reach a threshold (e.g., > 100 bit flips / hour)

What about ECC?

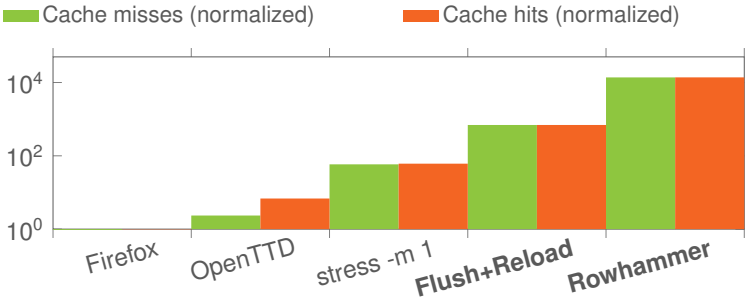
- ECC protection: server can handle or correct single bit errors
- no standard for event reporting
- in practice (Lanteigne 2016)
  - common: server counts ECC errors and report only if they reach a threshold (e.g., > 100 bit flips / hour)
  - some server vendors never report errors to the OS

What about ECC?

- ECC protection: server can handle or correct single bit errors
- no standard for event reporting
- in practice (Lanteigne 2016)
  - common: server counts ECC errors and report only if they reach a threshold (e.g., > 100 bit flips / hour)
  - some server vendors never report errors to the OS
  - one server did not even halt when bit flips were non-correctable

Detecting Rowhammer attacks

- Rowhammer: lots of cache misses that can be monitored with hardware performance counters (Herath and Fogh 2015; Gruss, Maurice, Wagner, et al. 2016; Chiappetta et al. 2015; Payer 2016)



Preventing Rowhammer attacks in hardware (1/3)

Original ideas from Kim et al. 2014

- making better DRAM chips that are not vulnerable,
- using error correcting codes (ECC)
- increasing the refresh rate
- remapping/retiring faulty cells after manufacturing
- identifying hammered rows at runtime and refreshing neighbors

Preventing Rowhammer attacks in hardware (1/3)

Original ideas from Kim et al. 2014

- making better DRAM chips that are not vulnerable,
- using error correcting codes (ECC)
- increasing the refresh rate
- remapping/retiring faulty cells after manufacturing
- identifying hammered rows at runtime and refreshing neighbors

→ expensive, performance overhead, or increased power consumption

Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation (Kim et al. 2014)

- one row closed $\rightarrow$ one adjacent row opened with low probability p

Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation (Kim et al. 2014)

- one row closed $\rightarrow$ one adjacent row opened with low probability p
- Rowhammer: one row opened and closed a high number of times N_{th}

Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation (Kim et al. 2014)

- one row closed $\rightarrow$ one adjacent row opened with low probability p
- Rowhammer: one row opened and closed a high number of times N_{th}
- statistically, neighbor rows are refreshed $\rightarrow$ no bit flip

Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation (Kim et al. 2014)

- one row closed $\rightarrow$ one adjacent row opened with low probability p
- Rowhammer: one row opened and closed a high number of times N_{th}
- statistically, neighbor rows are refreshed $\rightarrow$ no bit flip
- implementation at the memory controller level

Preventing Rowhammer attacks in hardware (2/3)

PARA - Probabilistic Adjacent Row Activation (Kim et al. 2014)

- one row closed $\rightarrow$ one adjacent row opened with low probability p
- Rowhammer: one row opened and closed a high number of times N_{th}
- statistically, neighbor rows are refreshed $\rightarrow$ no bit flip
- implementation at the memory controller level
- advantage: stateless $\rightarrow$ not expensive

Preventing Rowhammer attacks in hardware (2/3)

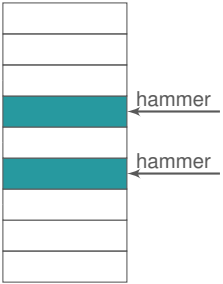
PARA - Probabilistic Adjacent Row Activation (Kim et al. 2014)

- one row closed $\rightarrow$ one adjacent row opened with low probability p
- Rowhammer: one row opened and closed a high number of times N_{th}
- statistically, neighbor rows are refreshed $\rightarrow$ no bit flip
- implementation at the memory controller level
- advantage: stateless $\rightarrow$ not expensive
- for $p = 0.001$ and $N_{th} = 100K$, experiencing one error in one year has a probability 9.4×10^{-14}

Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

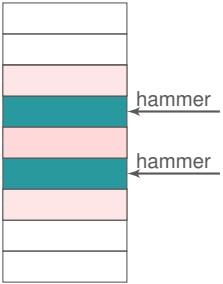
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

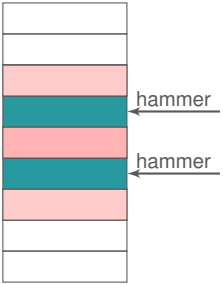
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

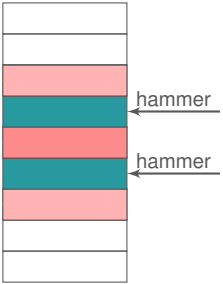
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

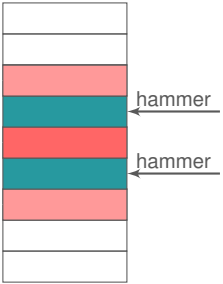
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

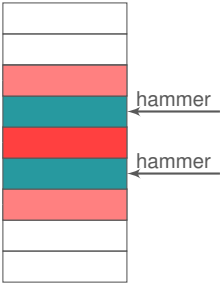
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

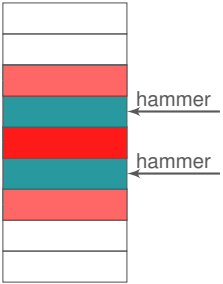
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

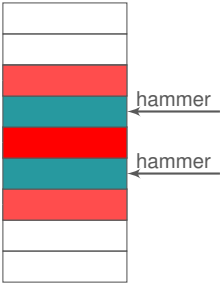
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

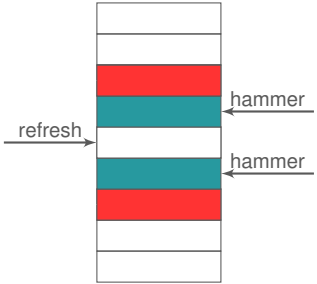
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

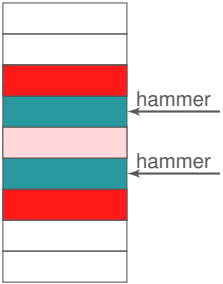
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

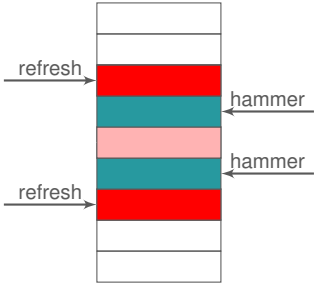
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

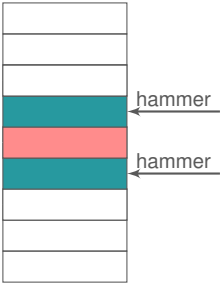
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in hardware (3/3)

Target Row Refresh (TRR)

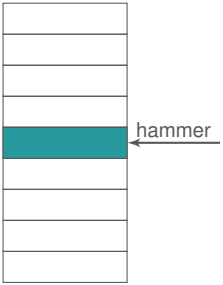
- counter per row
- increment neighbor rows
- refresh when counter reaches a threshold



Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

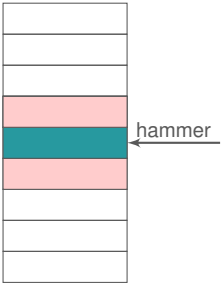
- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum



Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

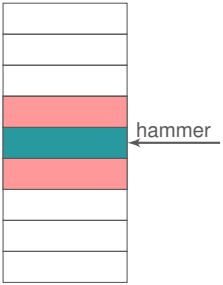
- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum



Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

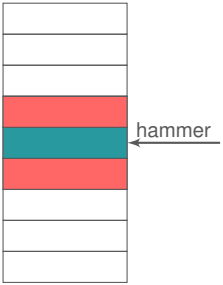
- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum



Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum



Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum

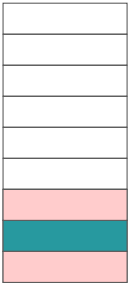
Wait for refresh

Wait for refresh

Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum

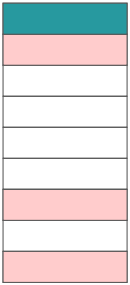


Wait for refresh

Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum

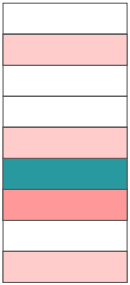


Wait for refresh

Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum

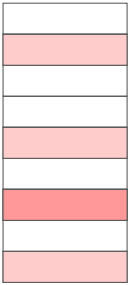


Wait for refresh

Preventing Rowhammer attacks in software

“nohammer” kernel module Corbet 2016

- refresh rate of 8 ms would prevent Rowhammer on most systems
- use PMC to measure cache misses per 64 ms interval
- limit cache miss rate to 1/8 of maximum



Wait for refresh

Preventing Rowhammer attacks in software

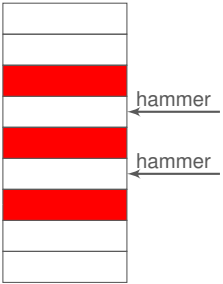
MASCAT - Stopping Microarchitectural Attacks Before Execution (Irazoqui et al. 2016)

- static analysis of the binary
- detect suspicious instruction sequences (`clflush`, `rdtsc`, fences, . . .)
- open problem: false positives

Preventing Rowhammer attacks in software

ANVIL (Aweke et al. 2016)

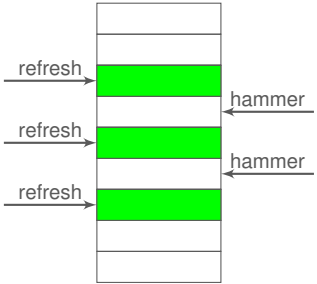
- uses performance counters to detect rowhammer
- activate rows neighbor rows to prevent flips
- similar as PARA, but in software



Preventing Rowhammer attacks in software

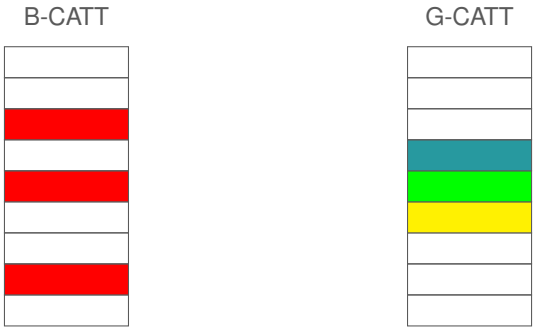
ANVIL (Aweke et al. 2016)

- uses performance counters to detect rowhammer
- activate rows neighbor rows to prevent flips
- similar as PARA, but in software



Preventing Rowhammer attacks in software

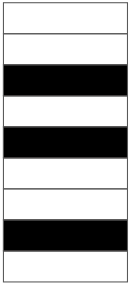
- B-CATT: disable vulnerable physical memory (Brasser et al. 2017)
- G-CATT: isolate security domains in physical memory based on potential vulnerability (Brasser et al. 2017)



Preventing Rowhammer attacks in software

- B-CATT: disable vulnerable physical memory (Brasser et al. 2017)
- G-CATT: isolate security domains in physical memory based on potential vulnerability (Brasser et al. 2017)

B-CATT



G-CATT

Conclusion

- Rowhammer attacks are easy to mount
- works on most systems (if you know the DRAM mapping)
- most countermeasures are too expensive or ineffective

Oh my Cache! 2
More fun with caches.

Daniel Gruss
Graz University of Technology

October 13, 2017 — QSP Lab

Bibliography I

Aweke, Zelalem Birhanu, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin (2016). “ANVIL: Software-Based Protection Against Next-Generation Rowhammer Attacks”. In: ASLPOS’16.

Bhattacharya, Sarani and Debdeep Mukhopadhyay (2016). “Curious case of Rowhammer: Flipping Secret Exponent Bits using Timing Analysis”. In: CHES’16.

Bosman, Erik, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida (2016). “Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector”. In: S&P’16.

Bibliography II

Brasser, Ferdinand, Lucas Davi, David Gens, Christopher Liebchen, and Ahmad-Reza Sadeghi (2017). “CAn’t Touch This: Software-only Mitigation against Rowhammer Attacks targeting Kernel Memory”. In: USENIX Security Symposium.

Chiappetta, Marco, Erkay Savas, and Cemal Yilmaz (2015). Real time detection of cache-based side-channel attacks using Hardware Performance Counters. Cryptology ePrint Archive, Report 2015/1034.

Corbet, Jonathan (2016). Defending against Rowhammer in the kernel. URL: https://lwn.net/Articles/704920/.

Gruss, Daniel, David Bidner, and Stefan Mangard (2015). “Practical Memory Deduplication Attacks in Sandboxed JavaScript”. In: 20th European Symposium on Research in Computer Security (ESORICS’15).

Bibliography III

Gruss, Daniel, Clémentine Maurice, and Stefan Mangard (2016). “Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript”. In: DIMVA’16.

Gruss, Daniel, Clémentine Maurice, Klaus Wagner, and Stefan Mangard (2016). “Flush+Flush: A Fast and Stealthy Cache Attack”. In: DIMVA’16.

Gruss, Daniel, Raphael Spreitzer, and Stefan Mangard (2015). “Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches”. In: USENIX Security Symposium.

Gullasch, David, Endre Bangerter, and Stephan Krenn (2011). “Cache Games – Bringing Access-Based Cache Attacks on AES to Practice”. In: S&P’11.

Bibliography IV

Herath, Nishad and Anders Fogh (2015). “These are Not Your Grand Daddys CPU Performance Counters – CPU Hardware Performance Counters for Security”. In: Black Hat 2015 Briefings. URL: https://www.blackhat.com/docs/us-15/materials/us-15-Herath-These-Are-Not-Your-Grand-Daddys-CPU%2DPerformance-Counters-CPU-Hardware-Performance-Counters%2DFor-Security.pdf.

Inci, Mehmet Sinan, Berk Gulmezoglu, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar (2015). “Seriously, get off my cloud! Cross-VM RSA Key Recovery in a Public Cloud”. In: Cryptology ePrint Archive, Report 2015/898, pp. 1–15.

Irazoqui, Gorka, Thomas Eisenbarth, and Berk Sunar (2016). “MASCAT: Stopping Microarchitectural Attacks Before Execution”. In: Cryptology ePrint Archive: Report 2016/1196.

Bibliography V

Kim, Yoongu, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu (2014). “Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors”. In: ISCA’14.

Lanteigne, Mark (2016). How Rowhammer Could Be Used to Exploit Weaknesses in Computer Hardware. URL: http://www.thirdio.com/rowhammer.pdf.

Lipp, Moritz, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard (2016). “ARMageddon: Last-Level Cache Attacks on Mobile Devices”. In: USENIX Security Symposium.

Liu, Fangfei, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee (2015). “Last-Level Cache Side-Channel Attacks are Practical”. In: S&P’15.

Bibliography VI

Maurice, Clémentine, Nicolas Le Scouarnec, Christoph Neumann, Olivier Heen, and Aurélien Francillon (2015). “Reverse Engineering Intel Complex Addressing Using Performance Counters”. In: RAID.

Maurice, Clémentine, Christoph Neumann, Olivier Heen, and Aurélien Francillon (2015). “C5: Cross-Cores Cache Covert Channel”. In: DIMVA’15.

Owens, Rodney and Weichao Wang (2011). “Non-interactive OS fingerprinting through memory de-duplication technique in virtual machines”. In: 30th IEEE International Performance Computing and Communications Conference, pp. 1–8.

Payer, Matthias (2016). “HexPADS: a platform to detect “stealth” attacks”. In: ESSoS’16.

Percival, Colin (2005). “Cache missing for fun and profit”. In: Proceedings of BSDCan.

Bibliography VII

Pessl, Peter, Daniel Gruss, Clémentine Maurice, and Stefan Mangard (2016). "Reverse Engineering Intel DRAM Addressing and Exploitation". In: USENIX Security Symposium.

Qiao, Rui and Mark Seaborn (2016). "A new approach for rowhammer attacks". In: HOST 2016.

Razavi, Kaveh, Ben Gras, Erik Bosman, Bart Preneel, Cristiano Giuffrida, and Herbert Bos (2016). "Flip Feng Shui: Hammering a Needle in the Software Stack". In: USENIX Security Symposium.

Seaborn, Mark (2015). How physical addresses map to rows and banks in DRAM. http://lackingrhoticity.blogspot.com/2015/05/how-physical-addresses-map-to-rows-and-banks.html. Retrieved on July 20, 2015.

Seaborn, Mark and Thomas Dullien (2015). "Exploiting the DRAM rowhammer bug to gain kernel privileges". In: Black Hat 2015 Briefings.

Bibliography VIII

Suzaki, Kuniyasu, Kengo Iijima, Toshiki Yagi, and Cyrille Artho (2011). “Memory Deduplication as a Threat to the Guest OS”. In: Proceedings of the 4th European Workshop on System Security.

Veen, Victor van der, Yanick Fratantonio, Martina Lindorfer, Daniel Gruss, Clémentine Maurice, Giovanni Vigna, Herbert Bos, Kaveh Razavi, and Cristiano Giuffrida (2016). “Drammer: Deterministic Rowhammer Attacks on Mobile Platforms”. In: CCS’16.

Xiao, Jidong, Zhang Xu, Hai Huang, and Haining Wang (2012). “A covert channel construction in a virtualized environment”. In: CCS’12.

– (2013). “Security implications of memory deduplication in a virtualized environment”. In: 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN).

Bibliography IX

Xiao, Yuan, Xiaokuan Zhang, Yinqian Zhang, and Radu Teodorescu (2016). “One Bit Flips, One Cloud Flops: Cross-VM Row Hammer Attacks and Privilege Escalation ”. In: USENIX Security Symposium.

Yarom, Yuval and Katrina Falkner (2014). “Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack”. In: USENIX Security Symposium.

Yarom, Yuval, Qian Ge, Fangfei Liu, Ruby B. Lee, and Gernot Heiser (2015). “Mapping the Intel Last-Level Cache”. In: Cryptology ePrint Archive, Report 2015/905, pp. 1–12.