Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript

Daniel Gruss, Clémentine Maurice, and Stefan Mangard
Graz University of Technology

July 8, 2016

Overview

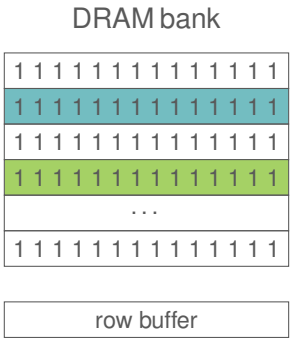
- Rowhammer: bit flip at a random location in DRAM
- exploitable $\rightarrow$ gain root privileges

We are the first to

- evaluate performance of cache eviction
- perform Rowhammer attacks without `clflush` on many platforms
- perform fault attacks from a website using JavaScript

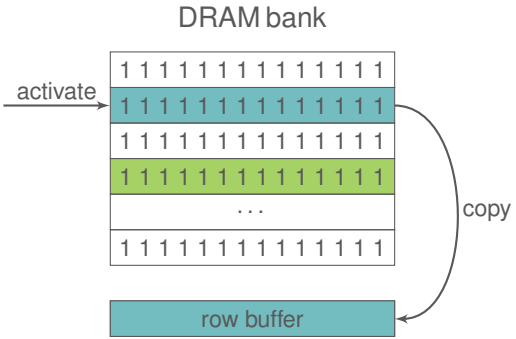
Rowhammer

“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



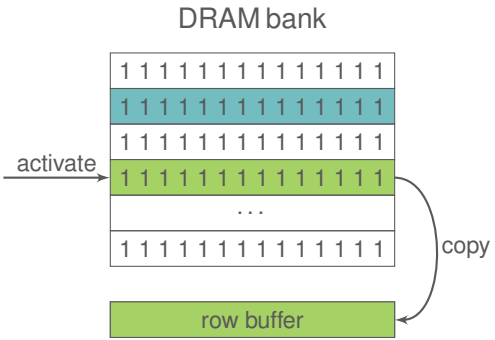
Rowhammer

“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



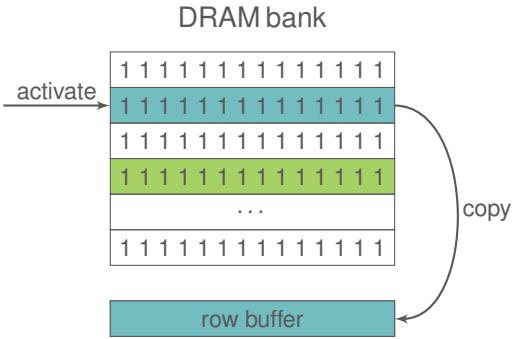
Rowhammer

“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



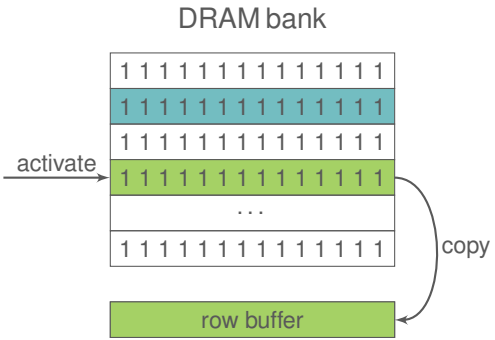
Rowhammer

“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



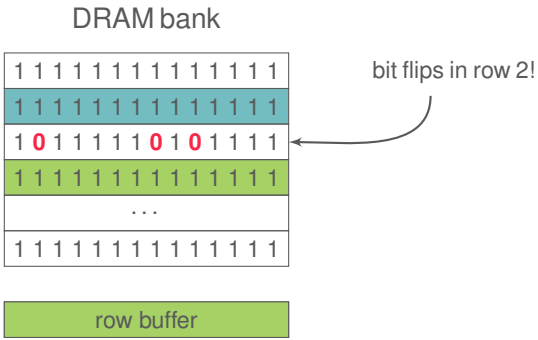
Rowhammer

“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

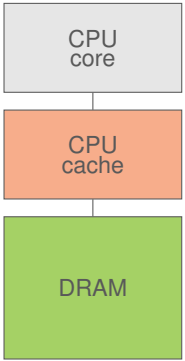


Rowhammer

“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



Impact of the CPU cache

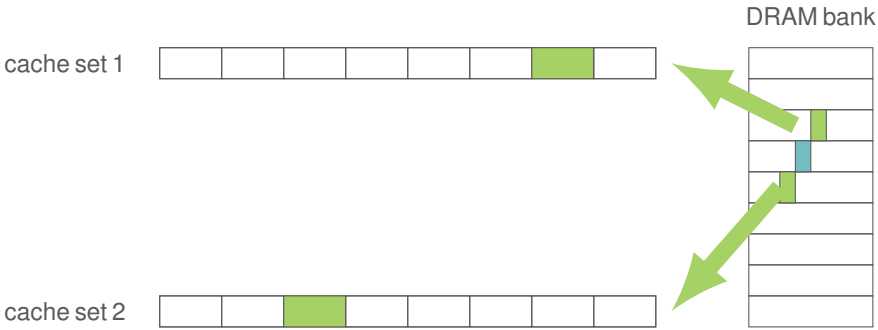


- only non-cached accesses reach DRAM
- original attacks use `clflush` instruction

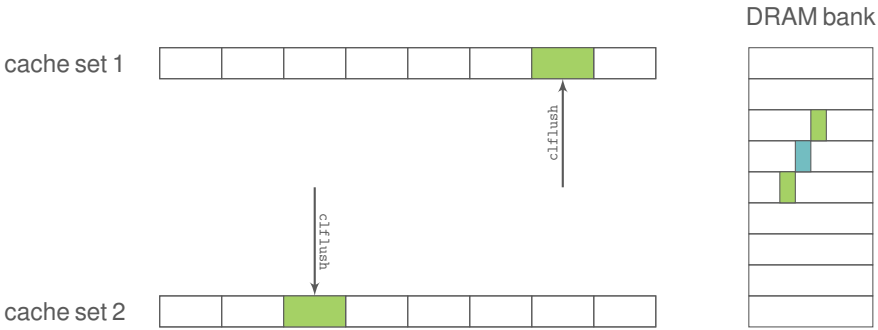
→ flush line from cache

→ next access will be served from DRAM

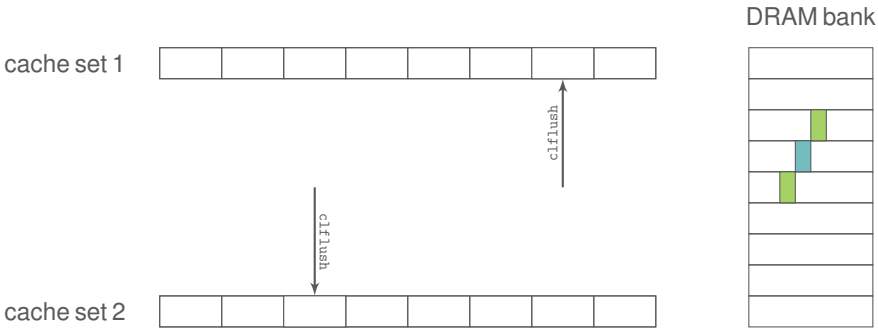
Rowhammer (with `clflush`)



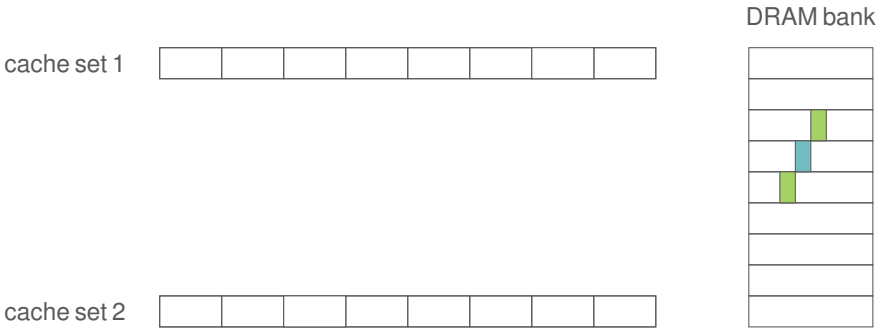
Rowhammer (with `clflush`)



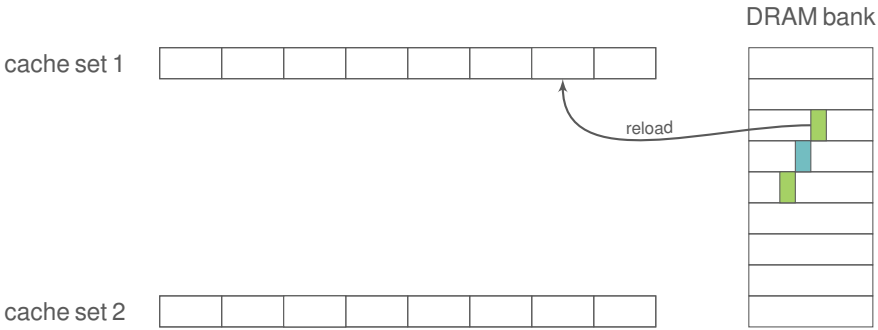
Rowhammer (with `clflush`)



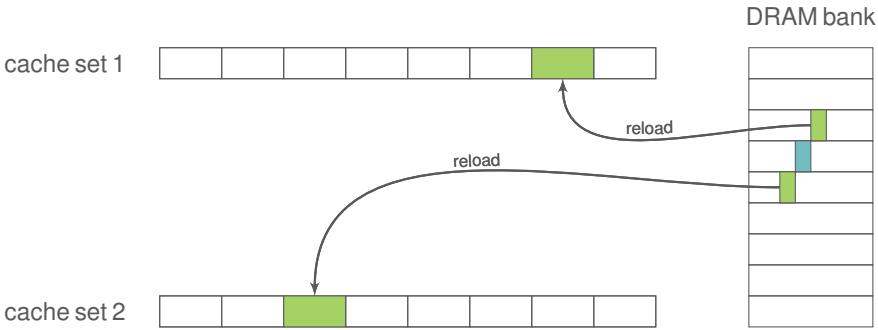
Rowhammer (with `clflush`)



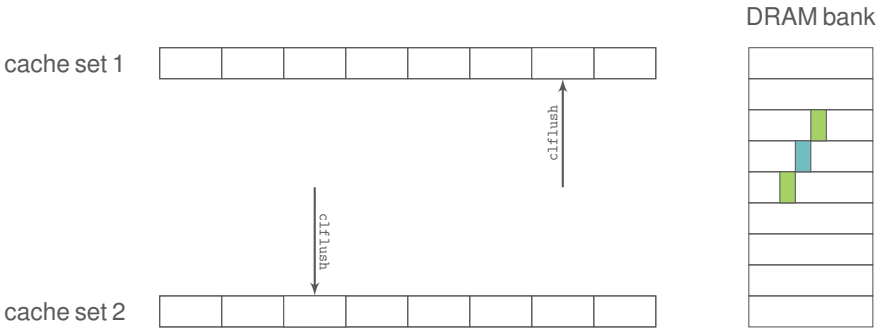
Rowhammer (with `clflush`)



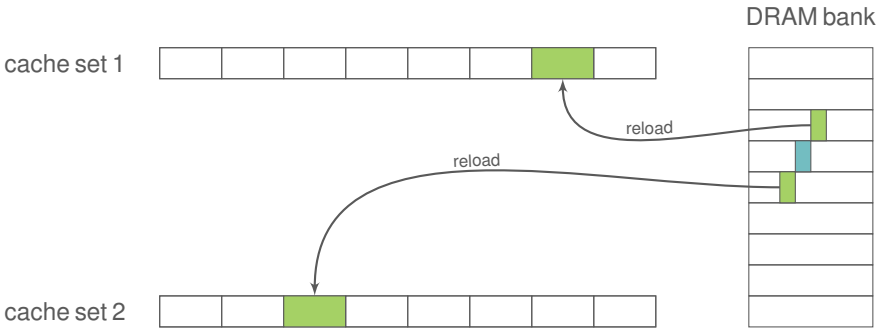
Rowhammer (with `clflush`)



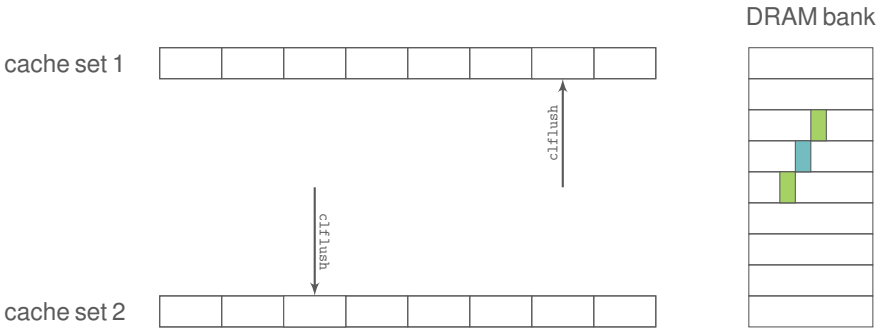
Rowhammer (with `clflush`)



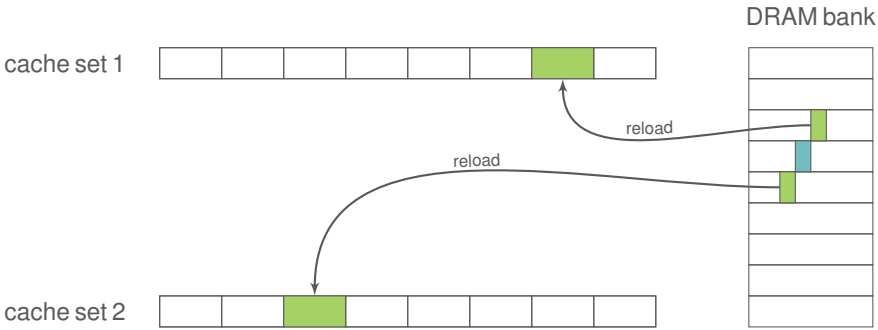
Rowhammer (with `clflush`)



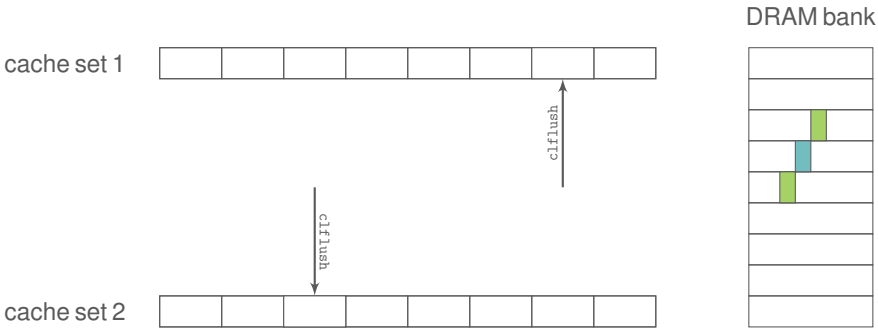
Rowhammer (with `clflush`)



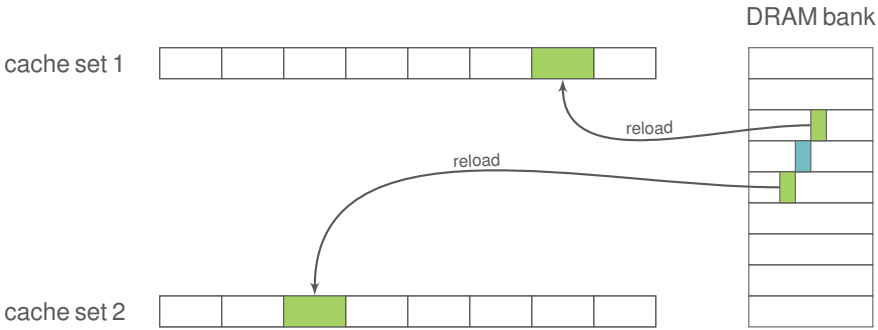
Rowhammer (with `clflush`)



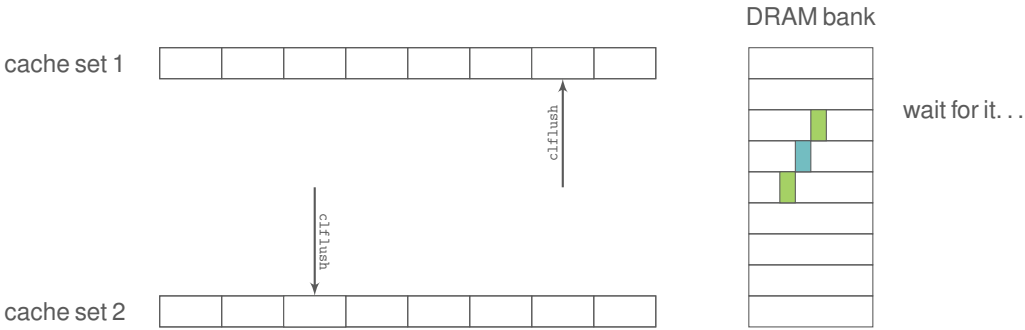
Rowhammer (with `clflush`)



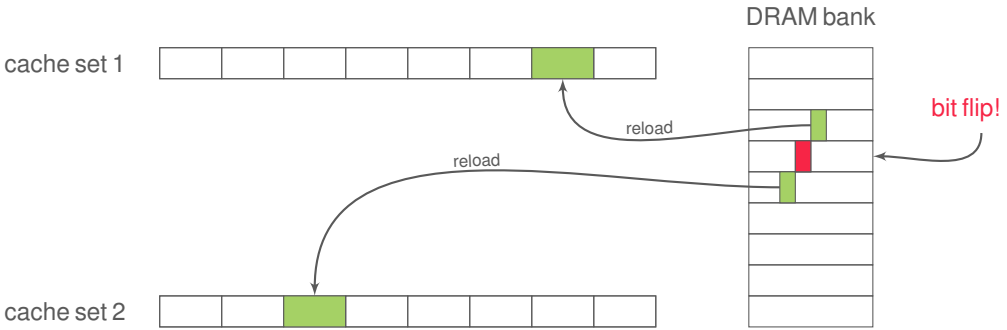
Rowhammer (with `clflush`)



Rowhammer (with `clflush`)



Rowhammer (with `clflush`)



Flush, reload, flush, reload. . .

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

Rowhammer without `clflush`?

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

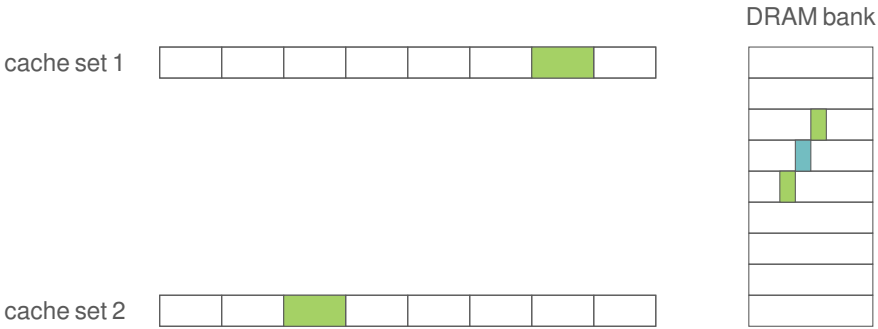
Rowhammer without `clflush`?

- 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!

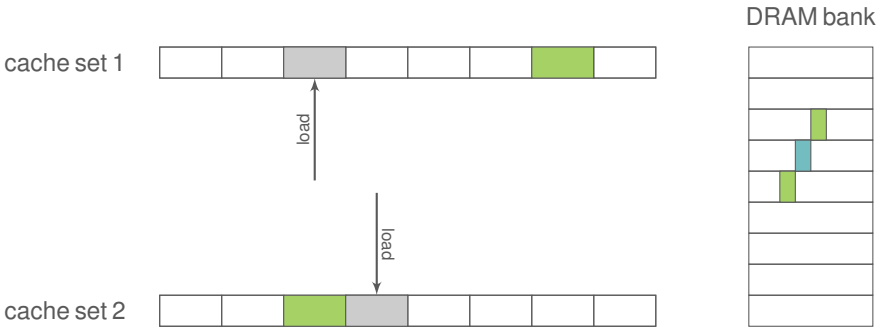
Rowhammer without `clflush`?

- 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!

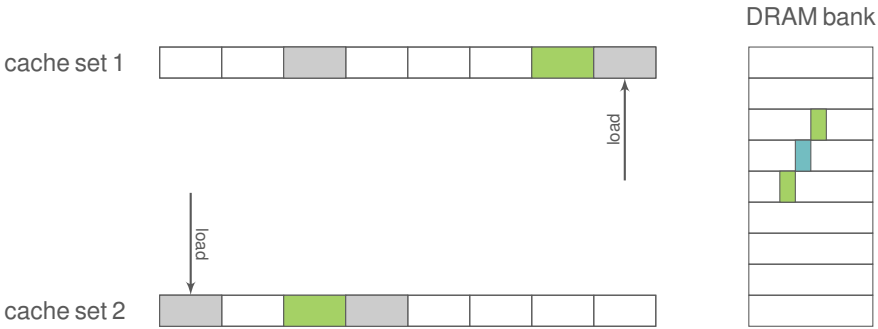
Rowhammer without `clflush`



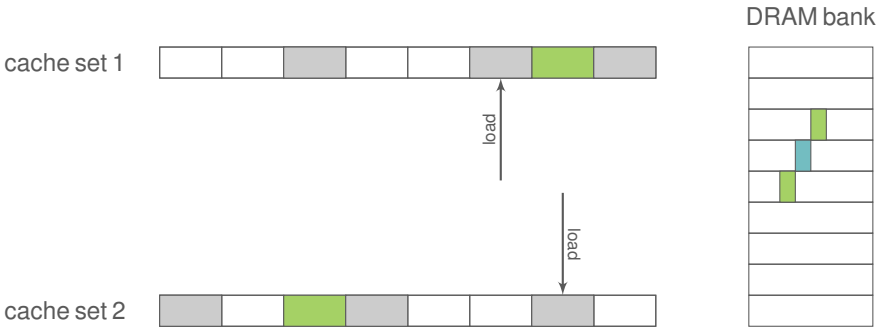
Rowhammer without `clflush`



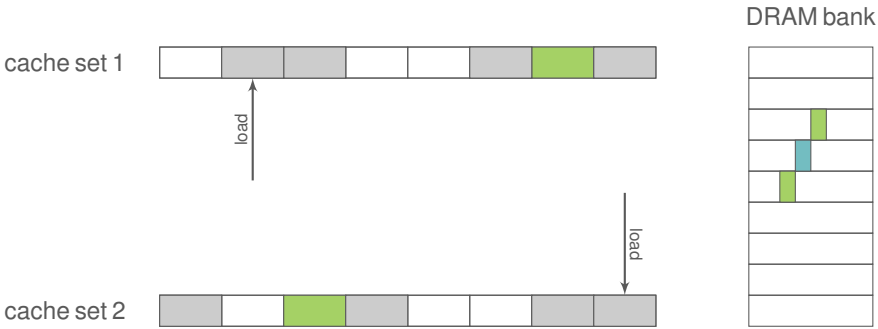
Rowhammer without `clflush`



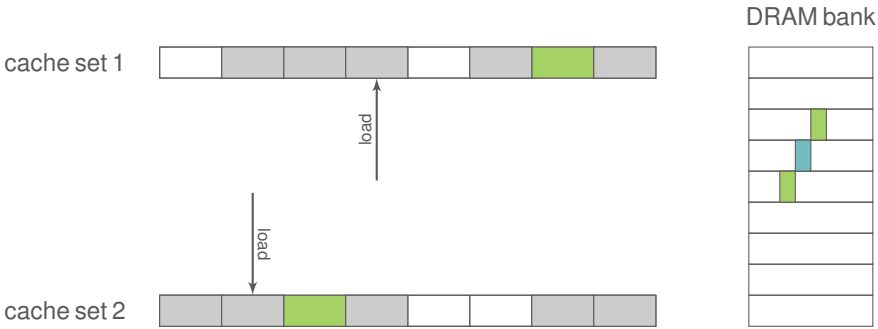
Rowhammer without `clflush`



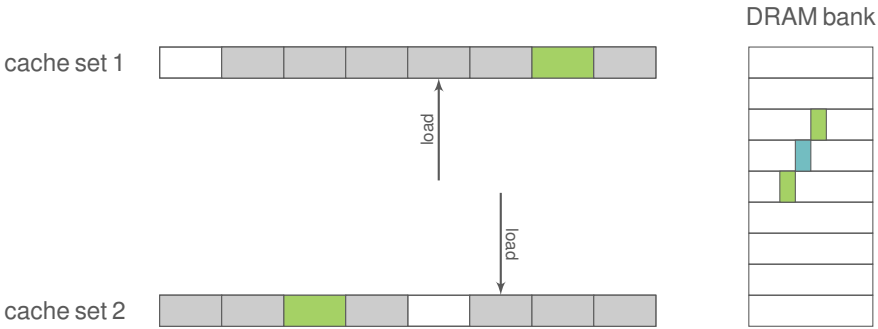
Rowhammer without `clflush`



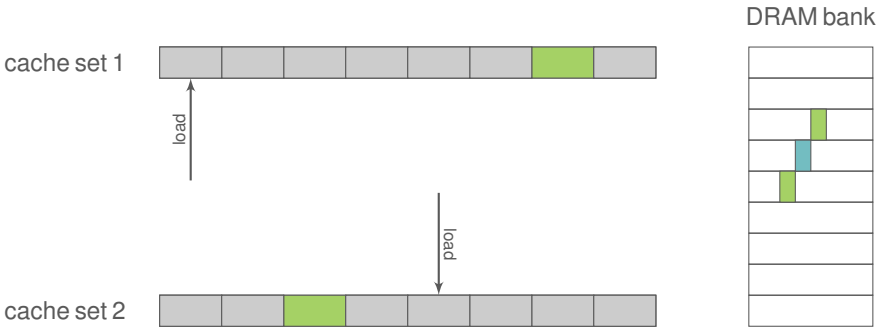
Rowhammer without `clflush`



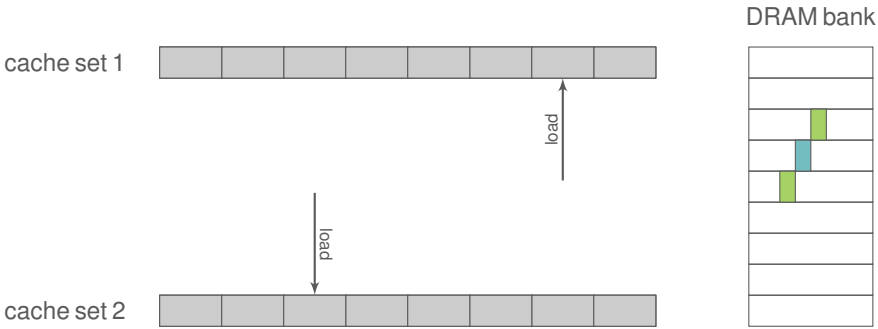
Rowhammer without `clflush`



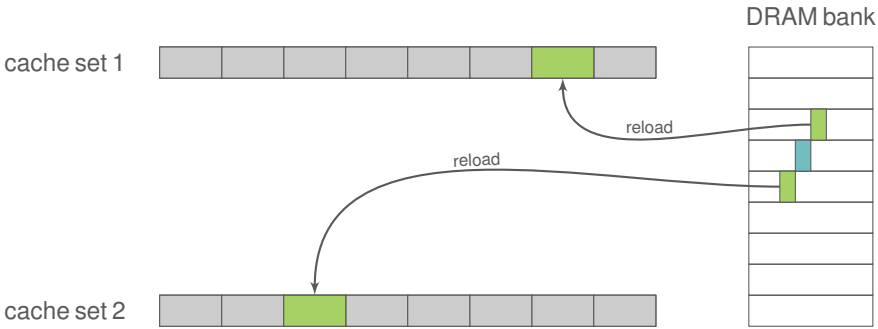
Rowhammer without `clflush`



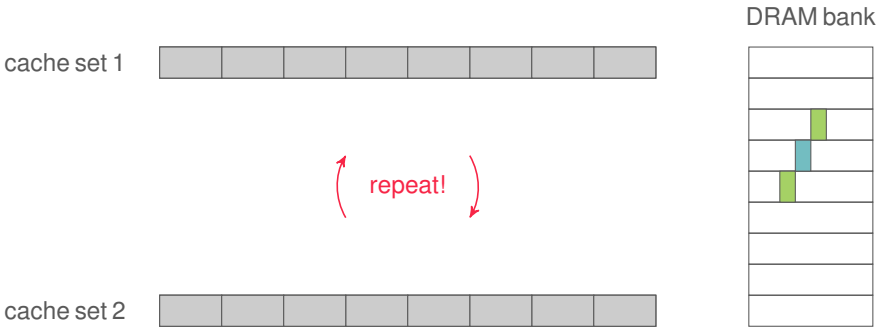
Rowhammer without `clflush`



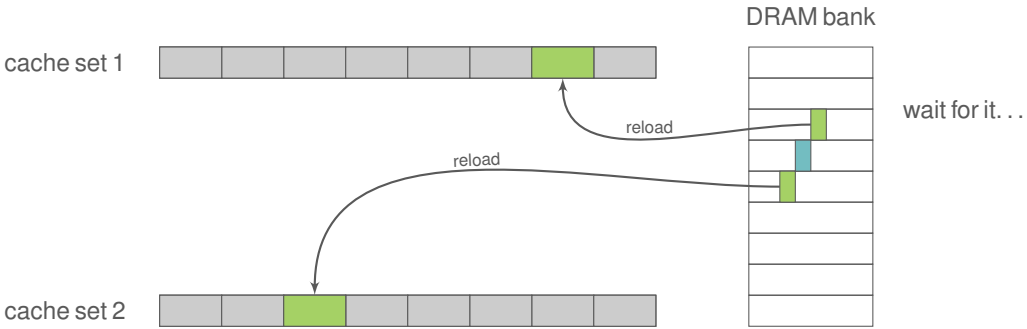
Rowhammer without `clflush`



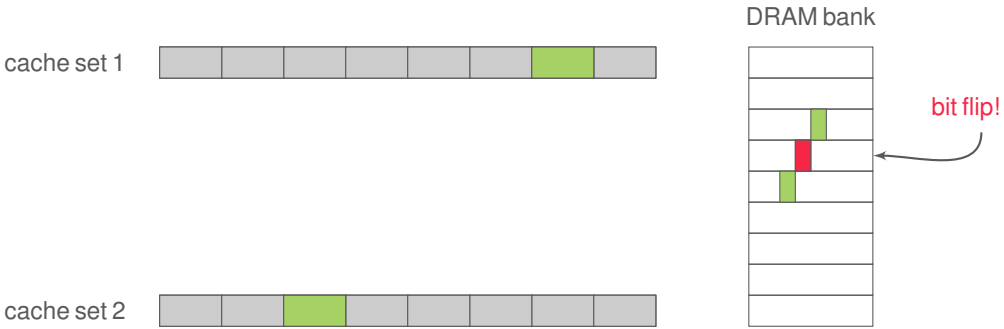
Rowhammer without `clflush`



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

$\rightarrow$ 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? $\rightarrow$ our contribution

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 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 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 (to appear)

Challenge #3: physical addresses and cache sets

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

Challenge #4: replacement policy

“LRU eviction” memory accesses on older CPUs

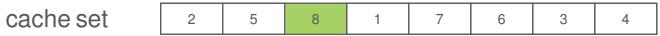
Challenge #4: replacement policy

"LRU eviction" memory accesses on older CPUs

- LRU replacement policy: oldest entry first

Challenge #4: replacement policy

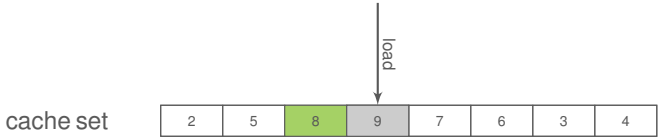
"LRU eviction" memory accesses on older CPUs



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

Challenge #4: replacement policy

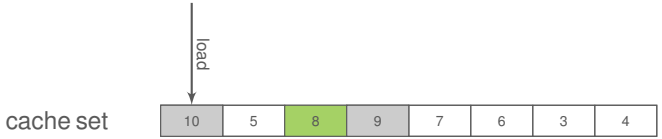
"LRU eviction" memory accesses on older CPUs



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

Challenge #4: replacement policy

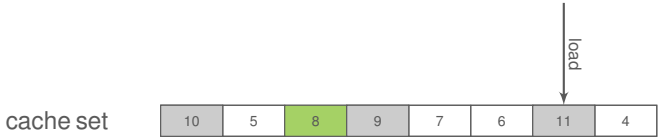
“LRU eviction” memory accesses on older CPUs



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

Challenge #4: replacement policy

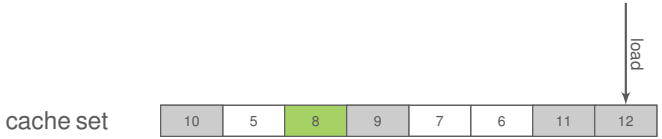
“LRU eviction” memory accesses on older CPUs



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

Challenge #4: replacement policy

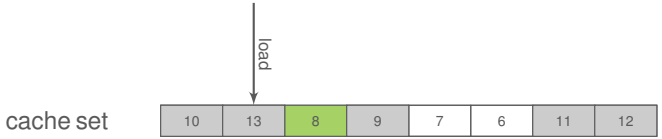
“LRU eviction” memory accesses on older CPUs



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

Challenge #4: replacement policy

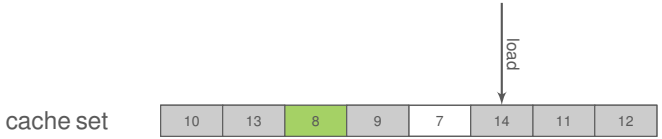
“LRU eviction” memory accesses on older CPUs



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

Challenge #4: replacement policy

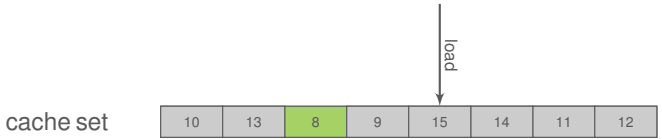
“LRU eviction” memory accesses on older CPUs



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

Challenge #4: replacement policy

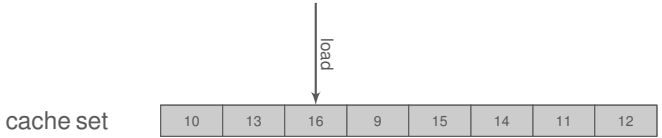
"LRU eviction" memory accesses on older CPUs



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

Challenge #4: replacement policy

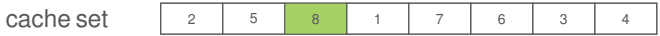
“LRU eviction” memory accesses on older CPUs



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

Replacement policy on recent CPUs

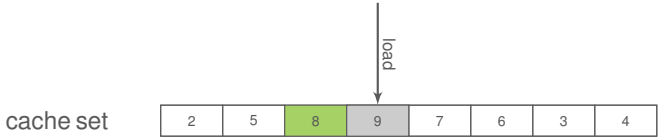
“LRU eviction” memory accesses



- no LRU replacement on recent CPUs

Replacement policy on recent CPUs

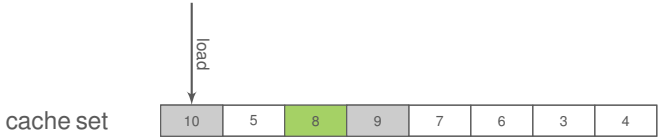
“LRU eviction” memory accesses



- no LRU replacement on recent CPUs

Replacement policy on recent CPUs

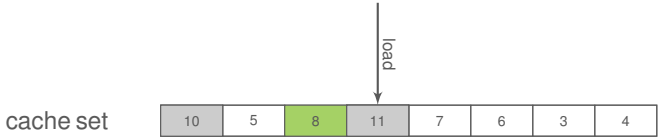
“LRU eviction” memory accesses



- no LRU replacement on recent CPUs

Replacement policy on recent CPUs

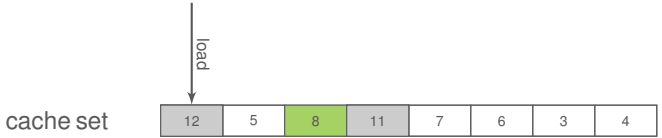
“LRU eviction” memory accesses



- no LRU replacement on recent CPUs

Replacement policy on recent CPUs

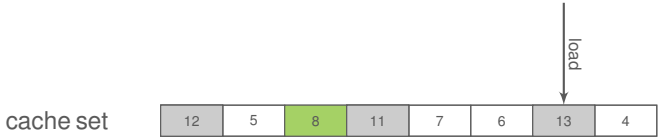
“LRU eviction” memory accesses



- no LRU replacement on recent CPUs

Replacement policy on recent CPUs

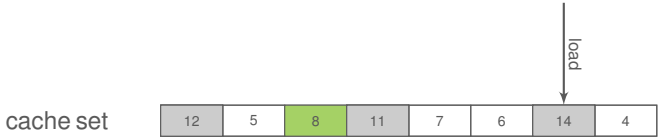
“LRU eviction” memory accesses



- no LRU replacement on recent CPUs

Replacement policy on recent CPUs

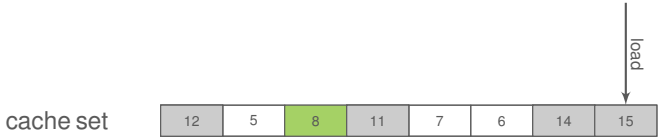
“LRU eviction” memory accesses



- no LRU replacement on recent CPUs

Replacement policy on recent CPUs

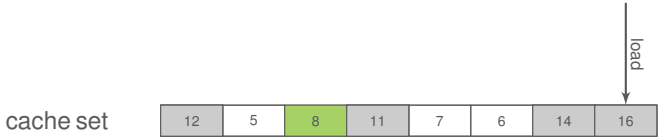
“LRU eviction” memory accesses



- no LRU replacement on recent CPUs

Replacement policy on recent CPUs

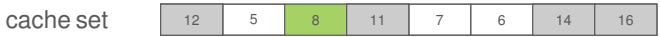
“LRU eviction” memory accesses



- no LRU replacement on recent CPUs

Replacement policy on recent CPUs

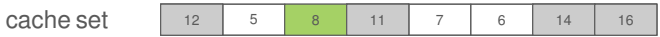
“LRU eviction” memory accesses



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

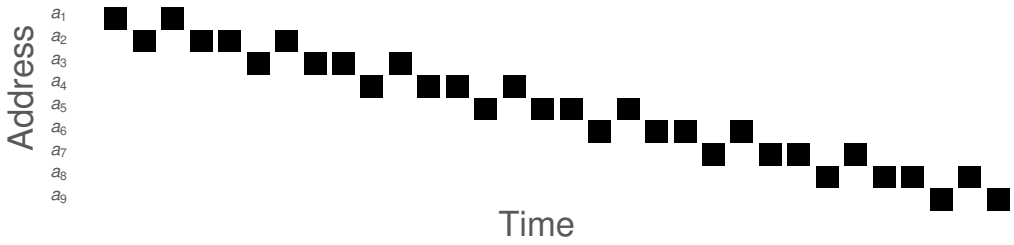
Replacement policy on recent CPUs

"LRU eviction" memory accesses



- no LRU replacement on recent CPUs
- only 75% success rate on Haswell
- more accesses $\rightarrow$ higher success rate, but too slow

Cache eviction strategies: The beginning



$\rightarrow$ 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:
  $1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4, \ldots$
- 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)

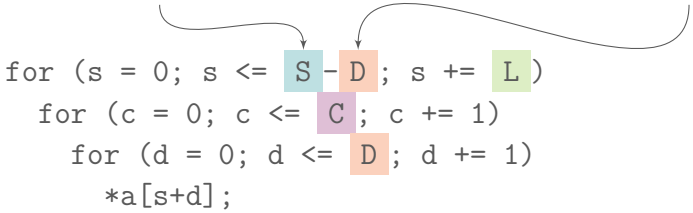
```
for (s = 0; s <= S - D; s += L)
  for (c = 0; c <= C; c += 1)
    for (d = 0; d <= D; d += 1)
      *a[s+d];
```

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



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 — $S = 4$

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

$S = 4$

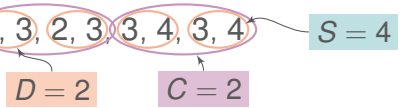
$D = 2$

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];
```

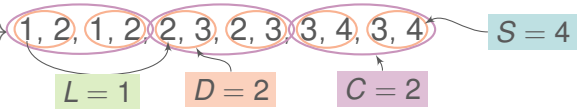
- P-2-2-1-4 → 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4



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];
```

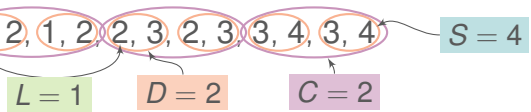
- P-2-2-1-4 → 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4



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



- 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...

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

Executed in a loop, on a Haswell with a 16-way last-level cache

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

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

Executed in a loop, on a Haswell with a 16-way last-level cache

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

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 ✗

Executed in a loop, on a Haswell with a 16-way last-level cache

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

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		

Executed in a loop, on a Haswell with a 16-way last-level cache

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

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% ✓	

Executed in a loop, on a Haswell with a 16-way last-level cache

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

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 ✓

Executed in a loop, on a Haswell with a 16-way last-level cache

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

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		

Executed in a loop, on a Haswell with a 16-way last-level cache

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

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% ✓	

Executed in a loop, on a Haswell with a 16-way last-level cache

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

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 ✓

Executed in a loop, on a Haswell with a 16-way last-level cache

Cache eviction strategies: Evaluation

We evaluated more than 10000 strategies...

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 ✓

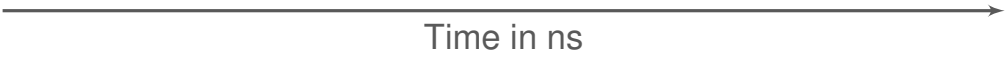
$\rightarrow$ more accesses, smaller execution time?

Executed in a loop, on a Haswell with a 16-way last-level cache

Cache eviction strategies: Illustration

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

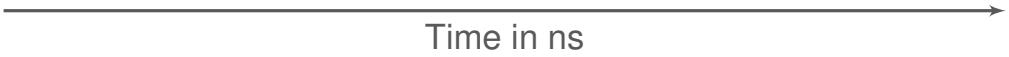
P-2-1-1-17 (34 accesses, 191ns)



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)

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)

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)

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)

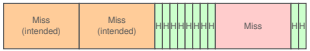
P-2-1-1-17 (34 accesses, 191ns)

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

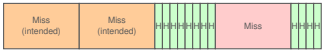
P-1-1-1-17 (17 accesses, 307ns)

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)



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)

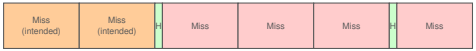
Cache eviction strategies: Illustration

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)

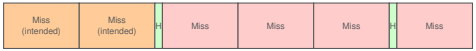


P-2-1-1-17 (34 accesses, 191ns)

Time in ns

Cache eviction strategies: Illustration

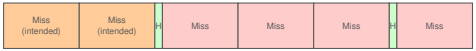
P-1-1-1-17 (17 accesses, 307ns)



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

Cache eviction strategies: Illustration

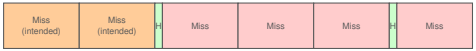
P-1-1-1-17 (17 accesses, 307ns)



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

Cache eviction strategies: Illustration

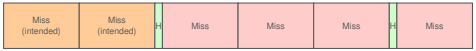
P-1-1-1-17 (17 accesses, 307ns)



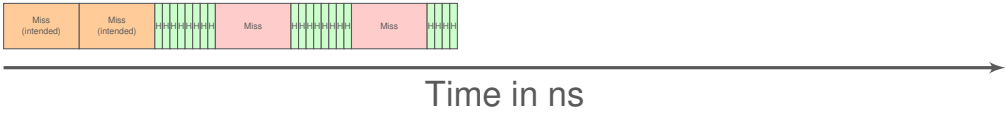
P-2-1-1-17 (34 accesses, 191ns)

Cache eviction strategies: Illustration

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

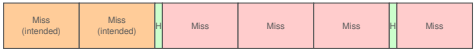


P-2-1-1-17 (34 accesses, 191ns)



Cache eviction strategies: Illustration

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

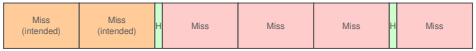


P-2-1-1-17 (34 accesses, 191ns)

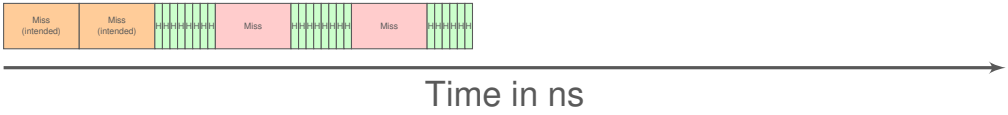
Time in ns

Cache eviction strategies: Illustration

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



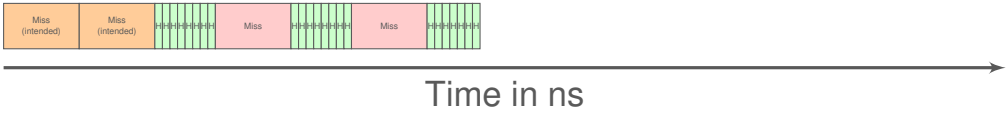
P-2-1-1-17 (34 accesses, 191ns)



Cache eviction strategies: Illustration

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

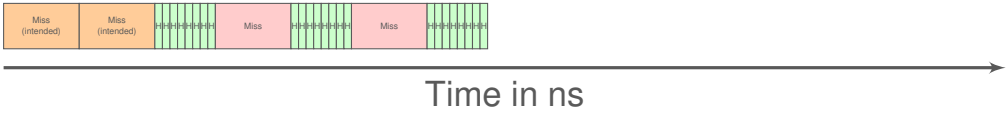
P-2-1-1-17 (34 accesses, 191ns)



Cache eviction strategies: Illustration

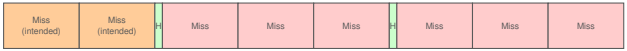
P-1-1-1-17 (17 accesses, 307ns)

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

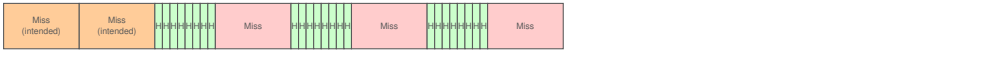


Cache eviction strategies: Illustration

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

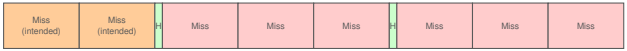


P-2-1-1-17 (34 accesses, 191ns)

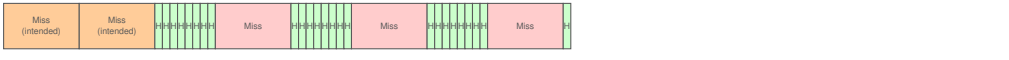


Cache eviction strategies: Illustration

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

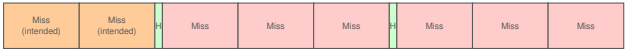


P-2-1-1-17 (34 accesses, 191ns)

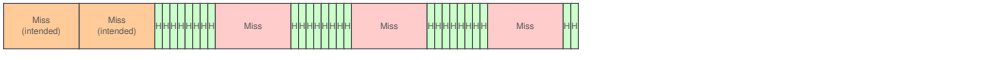


Cache eviction strategies: Illustration

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

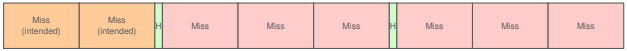


P-2-1-1-17 (34 accesses, 191ns)

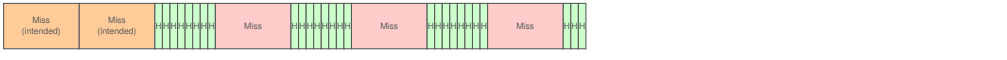


Cache eviction strategies: Illustration

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

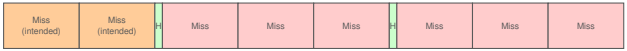


P-2-1-1-17 (34 accesses, 191ns)

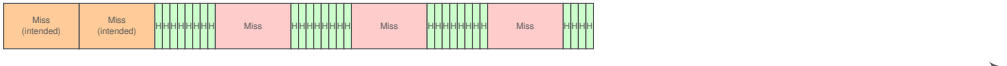


Cache eviction strategies: Illustration

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

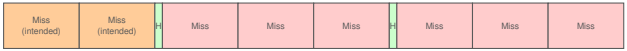


P-2-1-1-17 (34 accesses, 191ns)

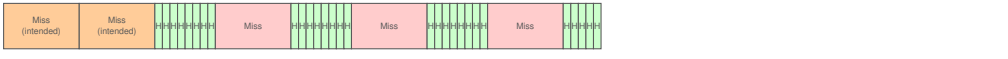


Cache eviction strategies: Illustration

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

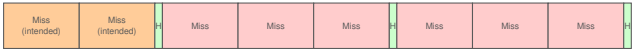


P-2-1-1-17 (34 accesses, 191ns)

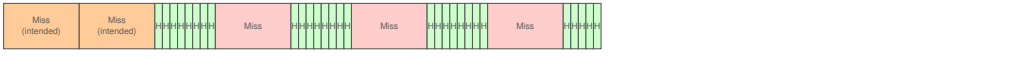


Cache eviction strategies: Illustration

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



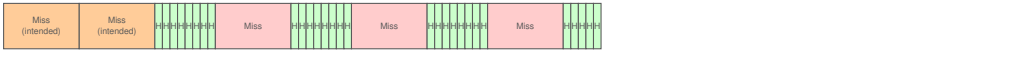
P-2-1-1-17 (34 accesses, 191ns)



Cache eviction strategies: Illustration

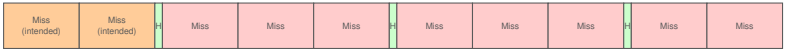
P-1-1-1-17 (17 accesses, 307ns)

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

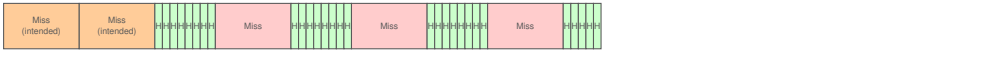


Cache eviction strategies: Illustration

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



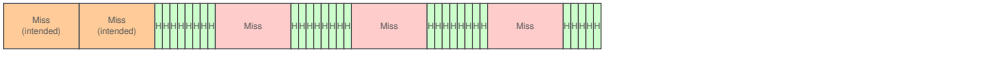
P-2-1-1-17 (34 accesses, 191ns)



Cache eviction strategies: Illustration

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

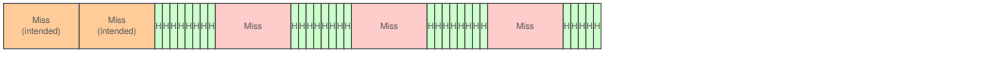
P-2-1-1-17 (34 accesses, 191ns)



Cache eviction strategies: Illustration

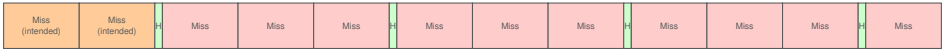
P-1-1-1-17 (17 accesses, 307ns)

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

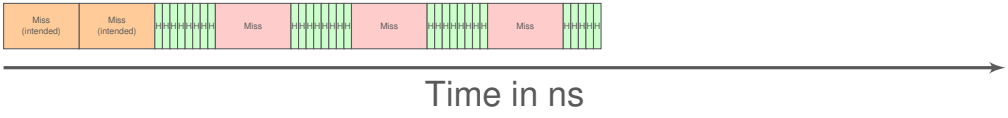


Cache eviction strategies: Illustration

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



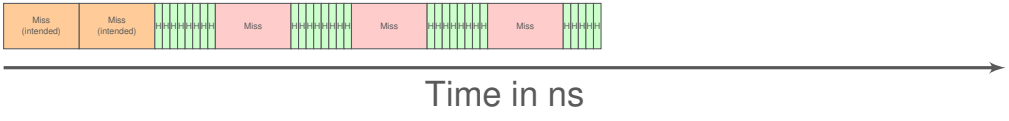
P-2-1-1-17 (34 accesses, 191ns)



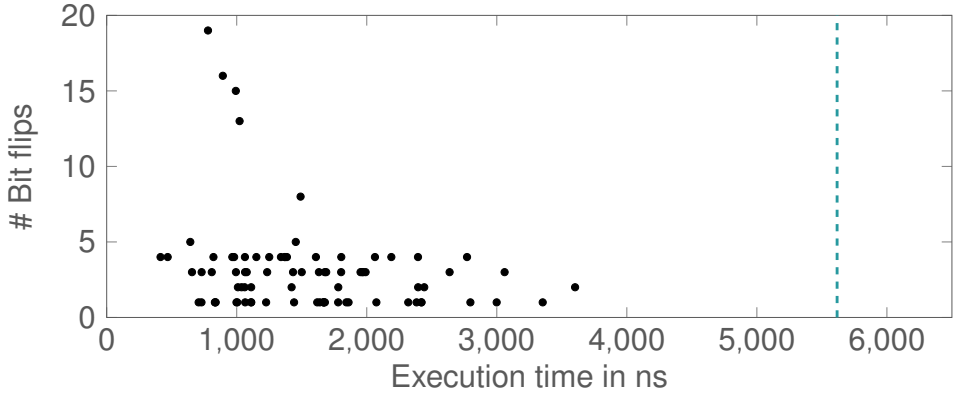
Cache eviction strategies: Illustration

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

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

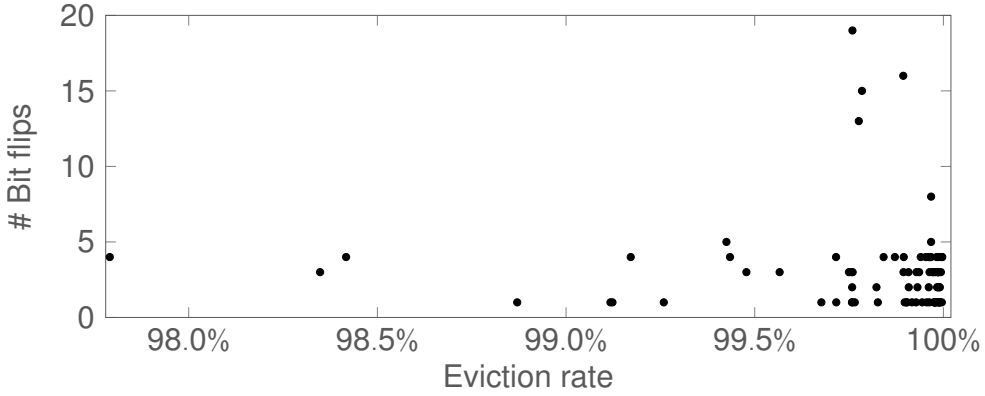


Execution time vs. bit flips



→ 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

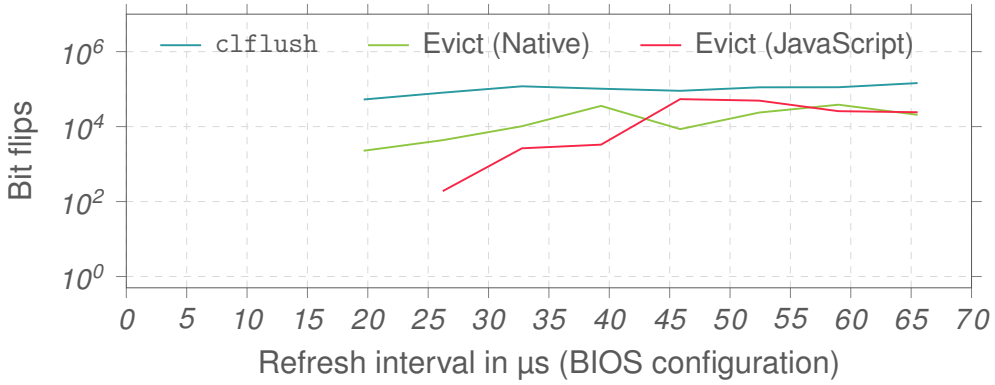


Figure: Number of bit flips within 15 minutes.

How to exploit?

- OS groups pages / page tables into 2 MB frames

How to exploit?

- OS groups pages / page tables into 2 MB frames

→ Page tables never in a DRAM row between two code/data pages

How to exploit?

- OS groups pages / page tables into 2 MB frames

→ Page tables never in a DRAM row between two code/data pages

  - unless system is almost out of memory

How to exploit?

- OS groups pages / page tables into 2 MB frames

→ Page tables never in a DRAM row between two code/data pages

  - unless system is almost out of memory

- hard to get there without crashing the browser

How to exploit?

- OS groups pages / page tables into 2 MB frames

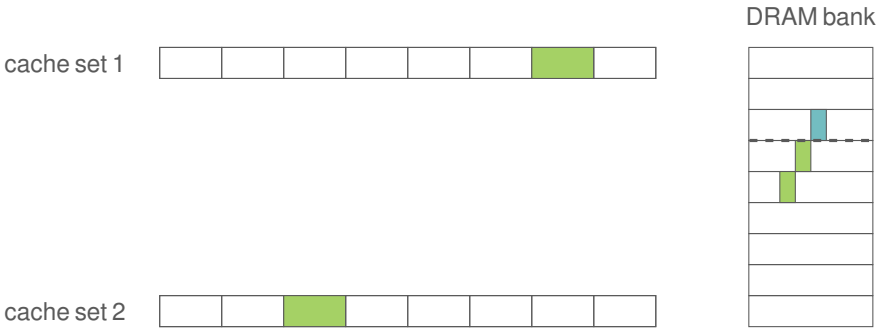
→ Page tables never in a DRAM row between two code/data pages

  - unless system is almost out of memory

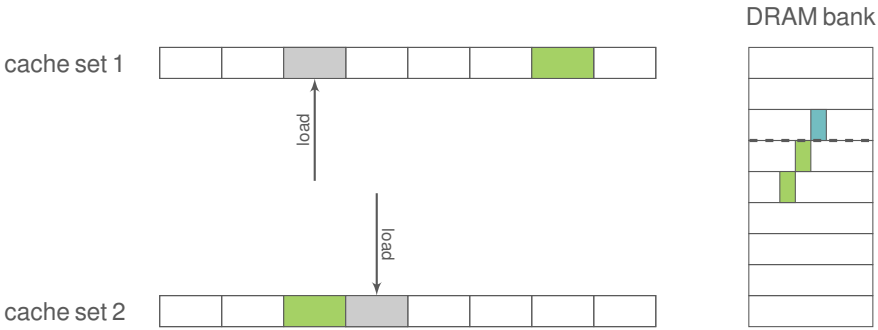
- hard to get there without crashing the browser

→ new hammering technique: amplified single-sided hammering

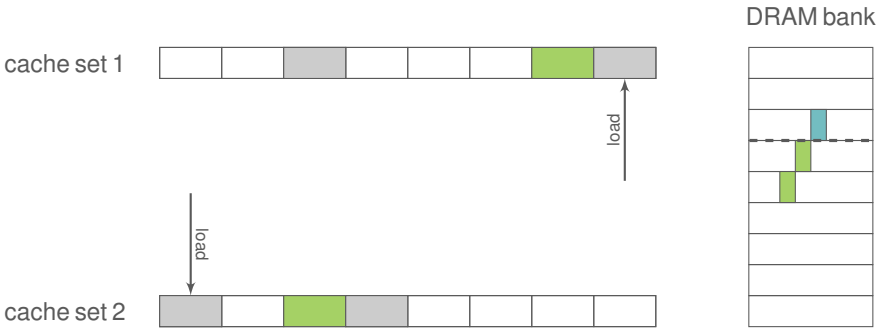
Amplified Single-Sided Hammering



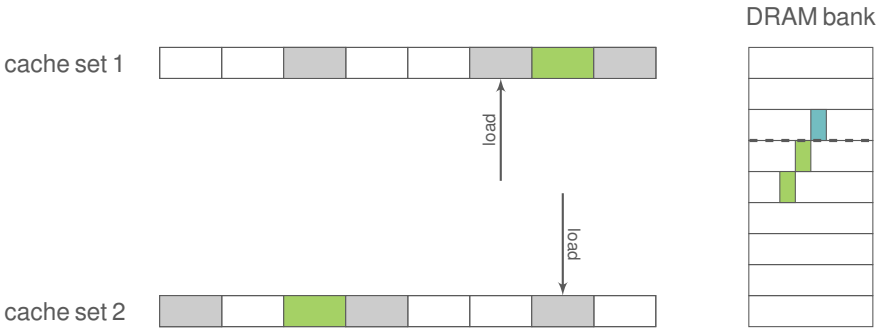
Amplified Single-Sided Hammering



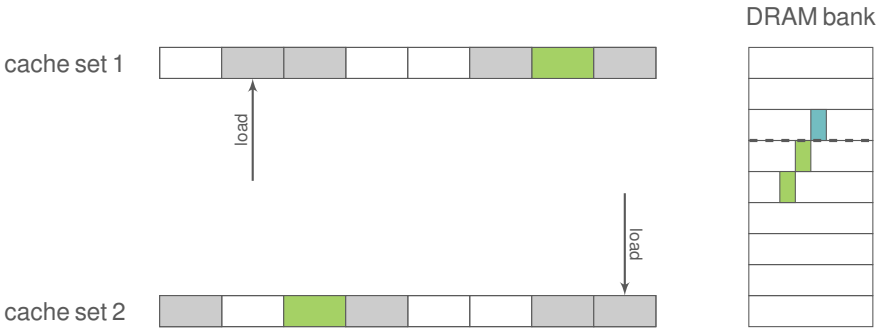
Amplified Single-Sided Hammering



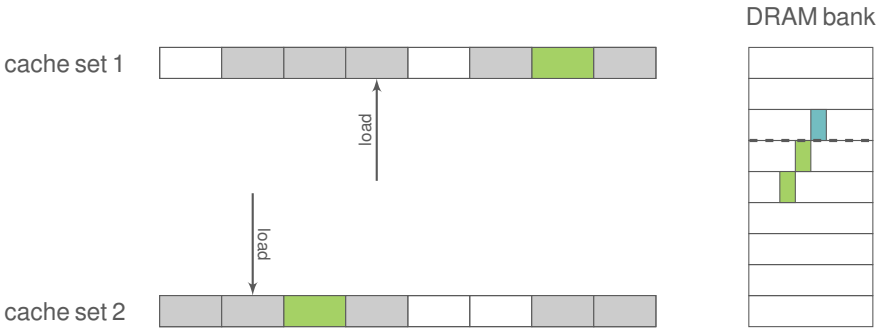
Amplified Single-Sided Hammering



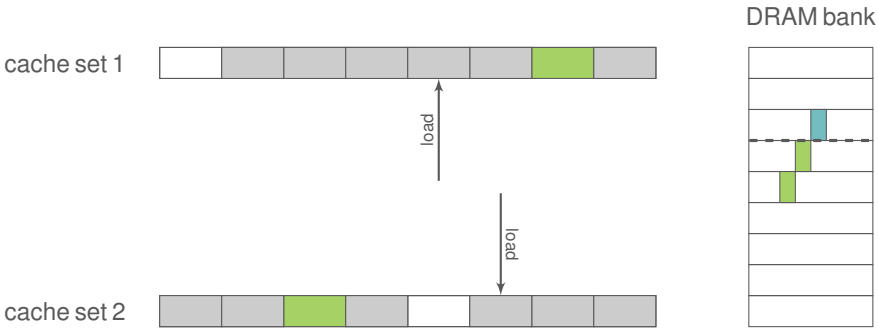
Amplified Single-Sided Hammering



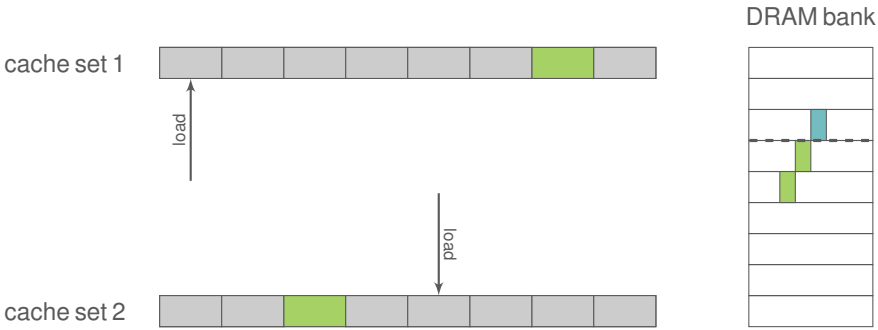
Amplified Single-Sided Hammering



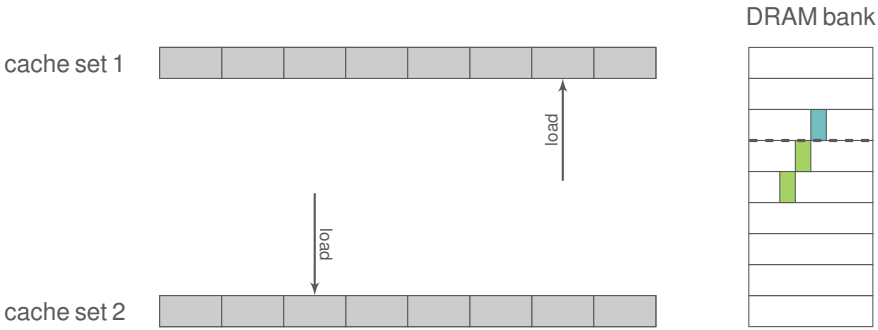
Amplified Single-Sided Hammering



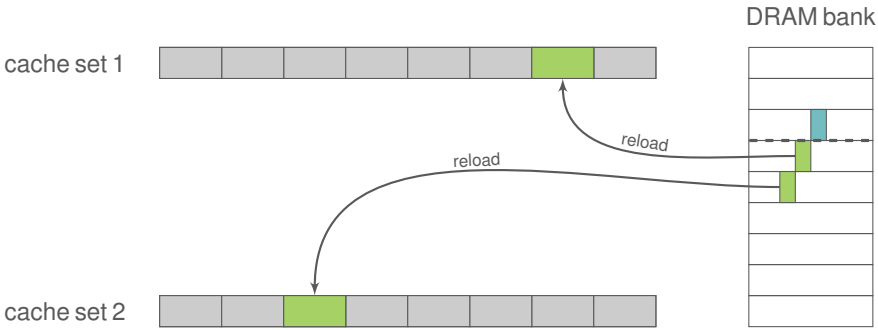
Amplified Single-Sided Hammering



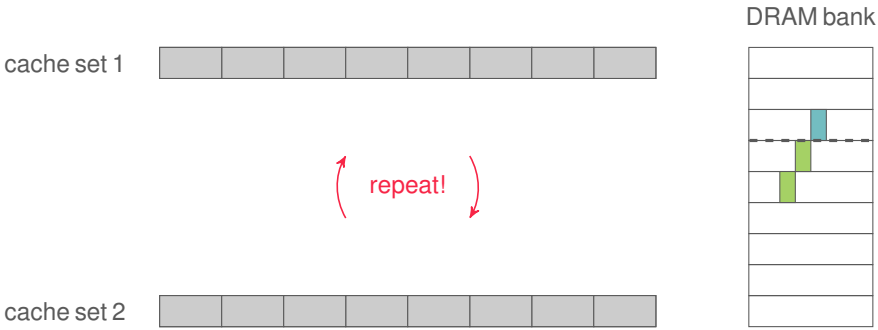
Amplified Single-Sided Hammering



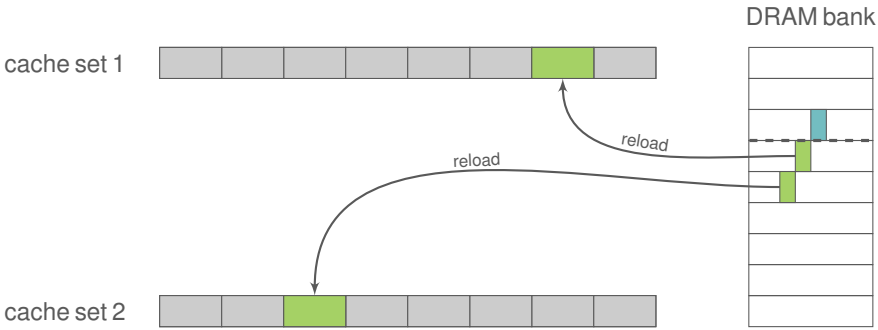
Amplified Single-Sided Hammering



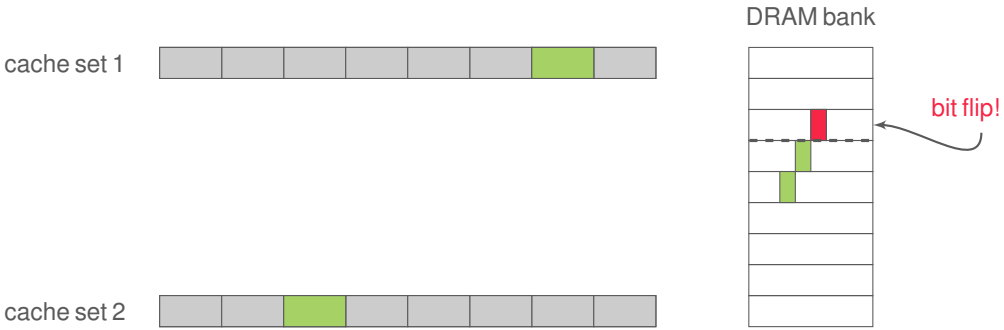
Amplified Single-Sided Hammering



Amplified Single-Sided Hammering



Amplified Single-Sided Hammering



Exploiting Rowhammer

- trigger bit flips page tables in adjacent 2 MB regions

Exploiting Rowhammer

- trigger bit flips page tables in adjacent 2 MB regions
- no near-out-of-memory situation

Exploiting Rowhammer

- trigger bit flips page tables in adjacent 2 MB regions
- no near-out-of-memory situation
- try until memory mappings changed

  $=$ bit flip in your own page tables

Exploiting Rowhammer

- trigger bit flips page tables in adjacent 2 MB regions
- no near-out-of-memory situation
- try until memory mappings changed

  $=$ bit flip in your own page tables

- try until your own page tables are mapped

Exploiting Rowhammer

- trigger bit flips page tables in adjacent 2 MB regions
- no near-out-of-memory situation
- try until memory mappings changed

  = bit flip in your own page tables
- try until your own page tables are mapped

  = full access to all physical memory

Reliable exploits based on Rowhammer.js?

- “Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector” by Bosman et al. 2016 at IEEE S&P’16
- clever attack exploiting memory deduplication and Rowhammer
- reliable exploit on Microsoft Edge

Conclusions

- cache eviction fast enough to replace `clflush`
- independent of programming language and available instructions
- first remote fault attack, from a browser

Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript

Daniel Gruss, Clémentine Maurice, and Stefan Mangard
Graz University of Technology

July 8, 2016

Bibliography

Bosman, Erik et al. (2016). “Dedup Est Machina: Memory Deduplication as an Advanced Exploitation Vector”. In: S&P’16.

Gruss, Daniel et al. (2015). “Practical Memory Deduplication Attacks in Sandboxed Javascript”. In: ESORICS’15.

Maurice, Clémentine et al. (2015). “Reverse Engineering Intel Complex Addressing Using Performance Counters”. In: RAID.

Pessl, Peter et al. (2016 (to appear)). “DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks”. 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.