Software-based Microarchitectural Attacks

Daniel Gruss
IAIK, Graz University of Technology

June 14, 2017 — PhD Defense

Thesis in numbers

Thesis in numbers

- **32** months

Thesis in numbers

- **32** months
- **10** invited talks and presentations at international venues

Thesis in numbers

- **32** months
- **10** invited talks and presentations at international venues
- **13** publications co-authored (**7** times tier 1)

Thesis in numbers

- **32** months
- **10** invited talks and presentations at international venues
- **13** publications co-authored (**7** times tier 1)
- **6** included in thesis (**3** times tier 1)

National Geographic

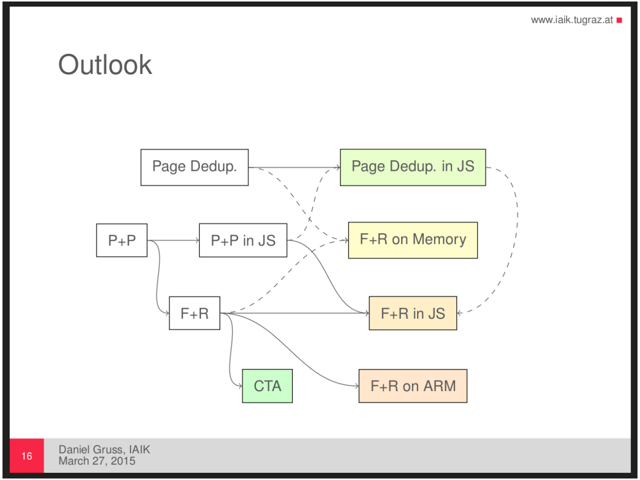
Software-based Side-Channel Attacks

- security and privacy rely on secrets (unknown to attackers)
- secrets can leak through side channels

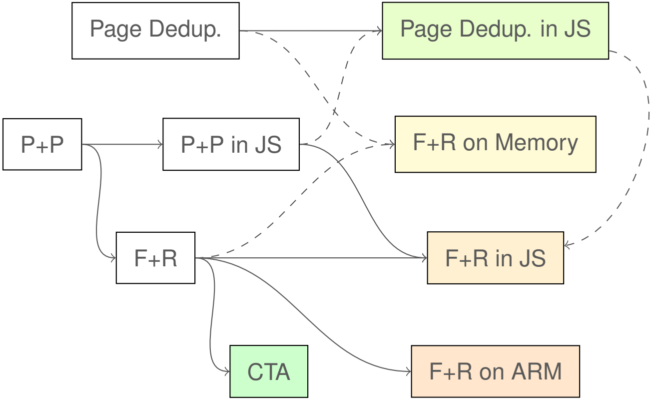
Software-based Side-Channel Attacks

- security and privacy rely on secrets (unknown to attackers)
- secrets can leak through side channels
- software-based $\rightarrow$ no physical access

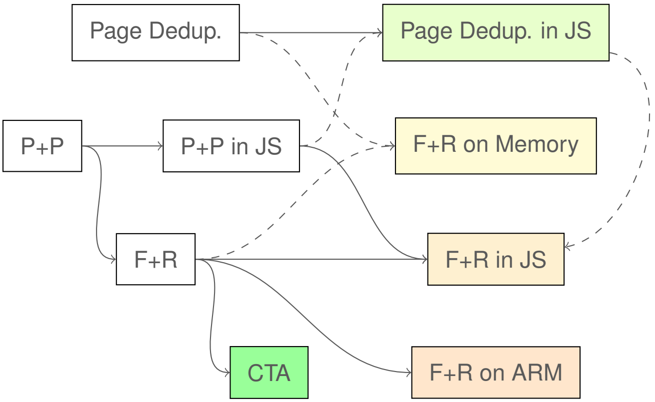
Plan (from March 2015)



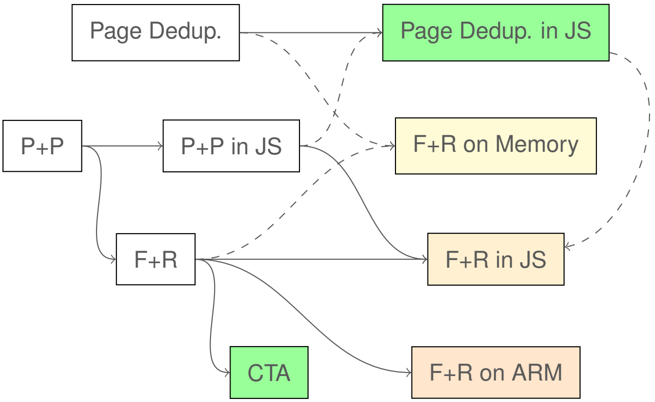
Plan (how it worked out)



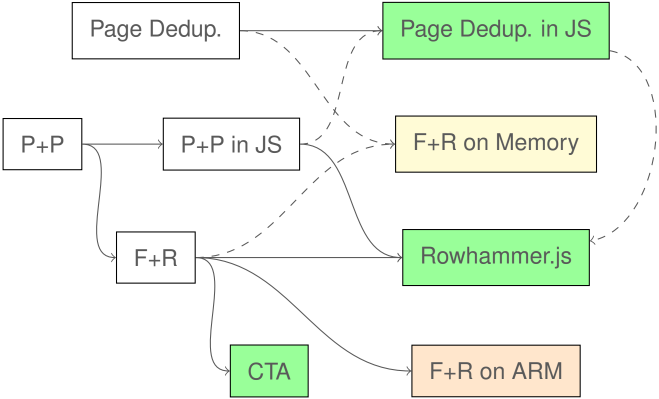
Plan (how it worked out)



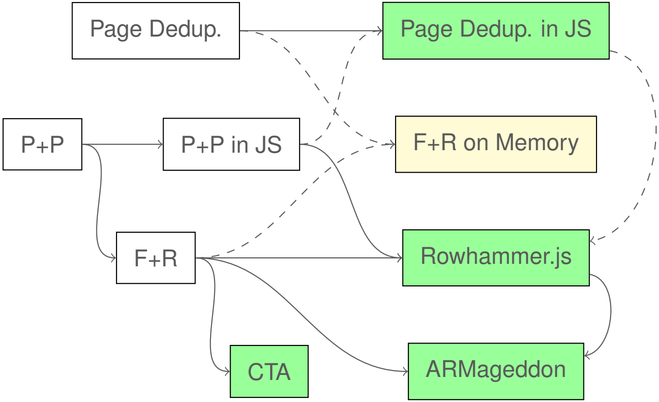
Plan (how it worked out)



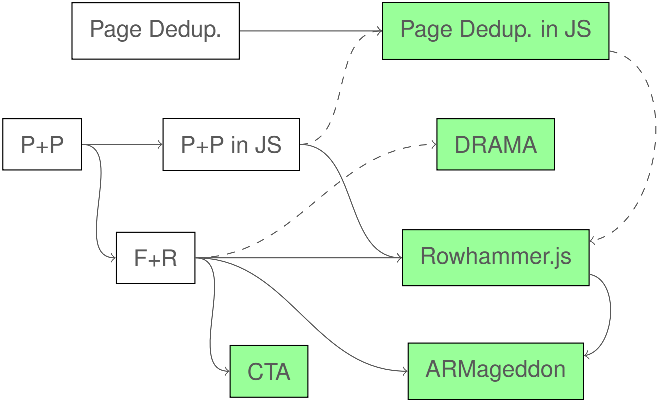
Plan (how it worked out)



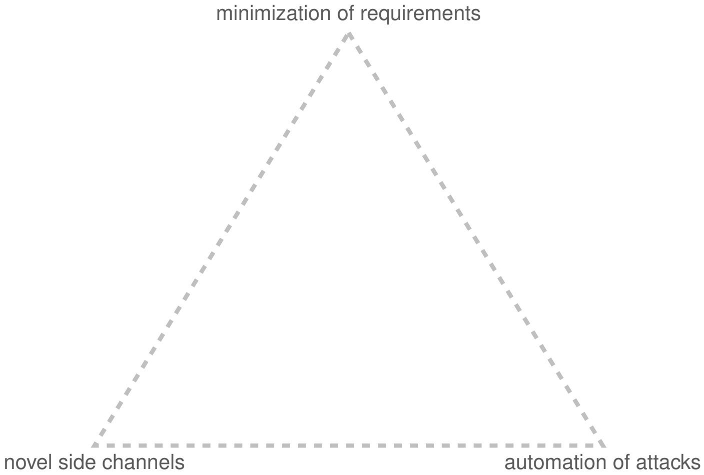
Plan (how it worked out)



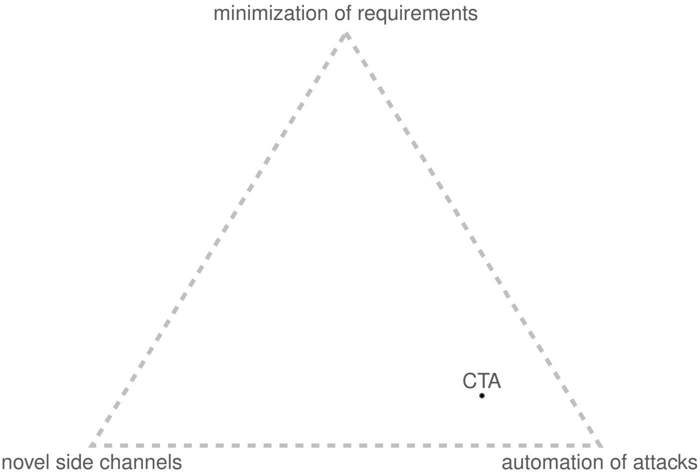
Plan (how it worked out)



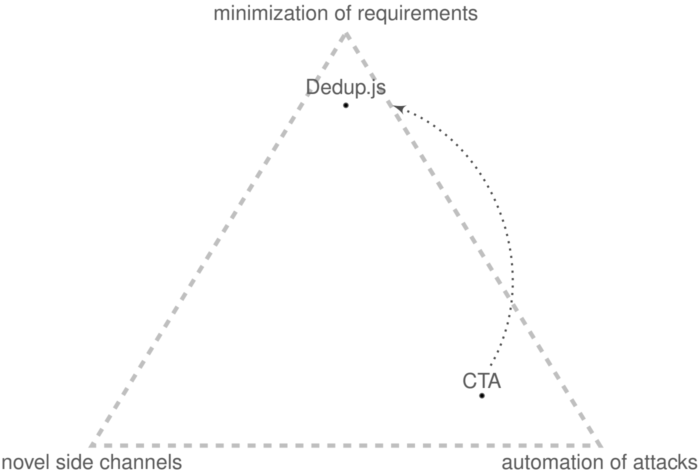
Relation of the papers



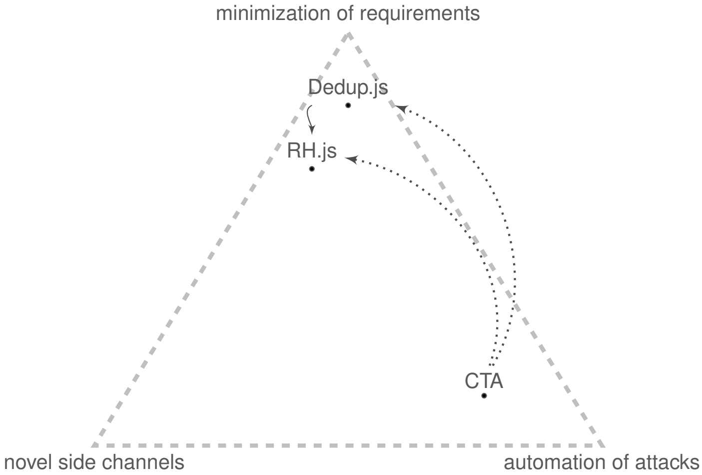
Relation of the papers



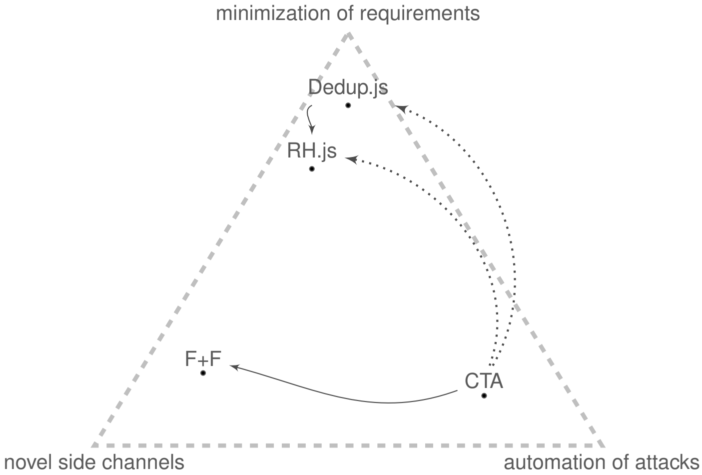
Relation of the papers



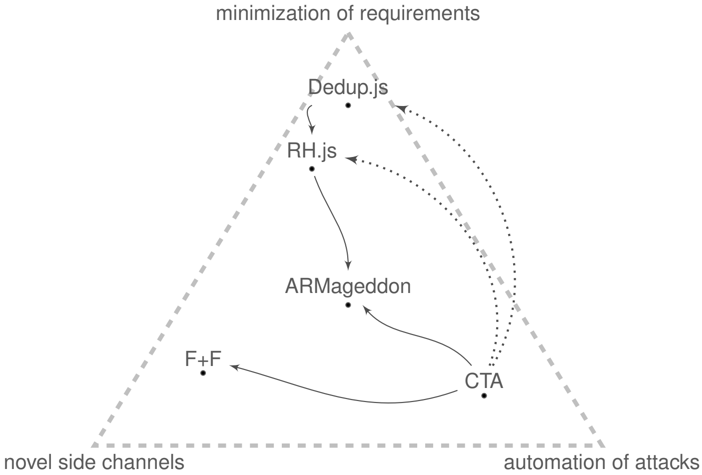
Relation of the papers



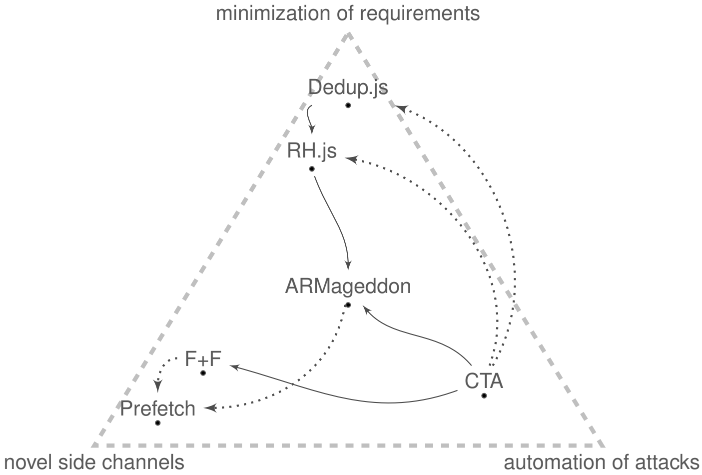
Relation of the papers



Relation of the papers



Relation of the papers



1. Introduction

2. Background

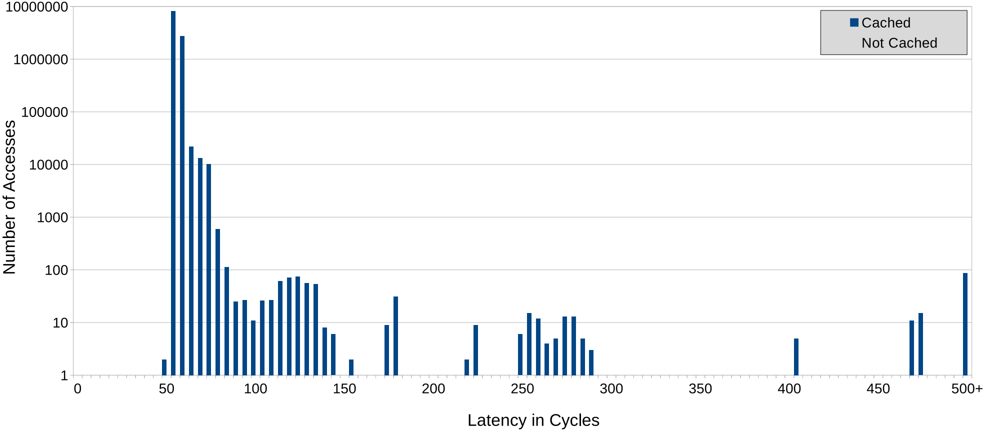
3. Contributions

4. Conclusion

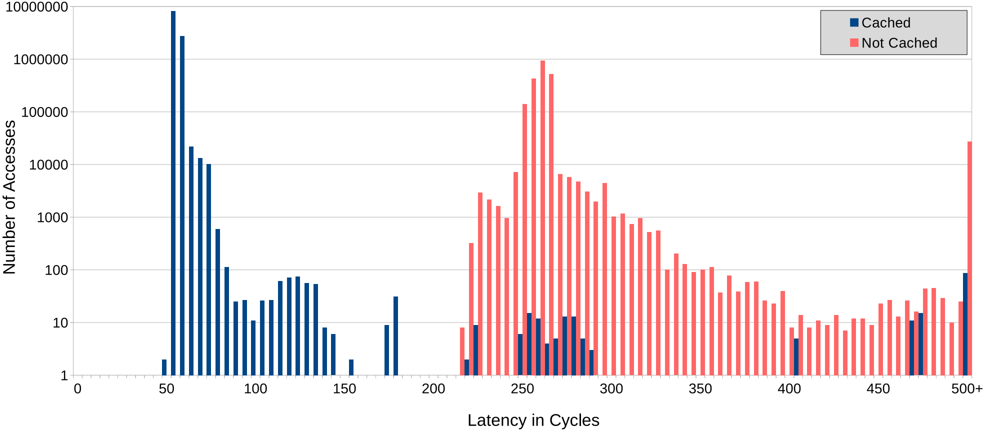
CPU Caches

- buffer frequently used slow memory for the fast CPU
- every memory reference goes through the cache
- transparent to OS and programs

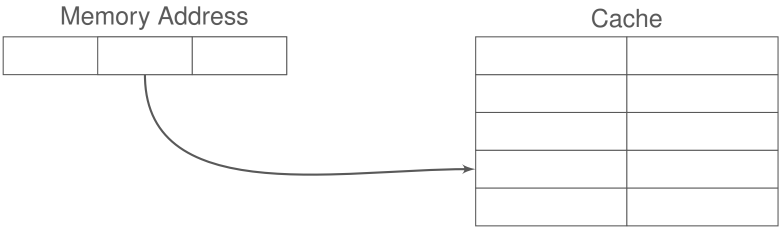
Memory Access Latency



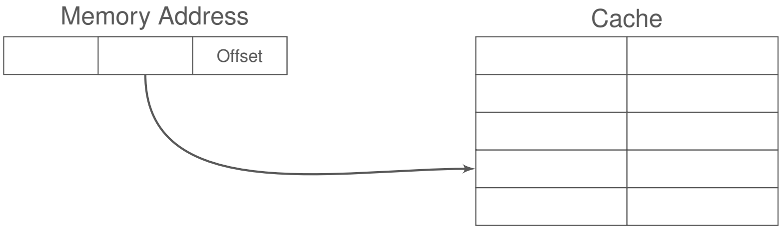
Memory Access Latency



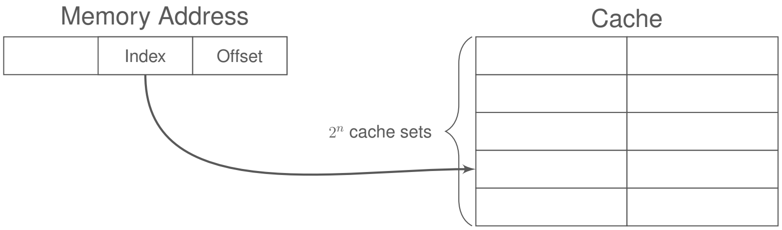
A simple cache



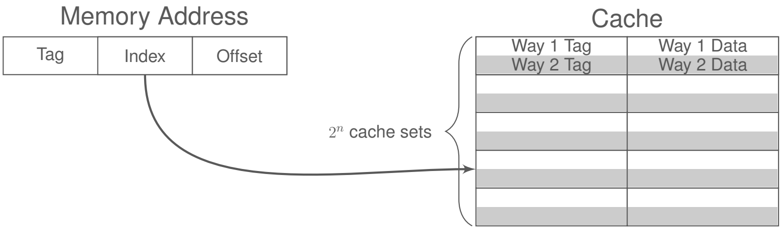
A simple cache



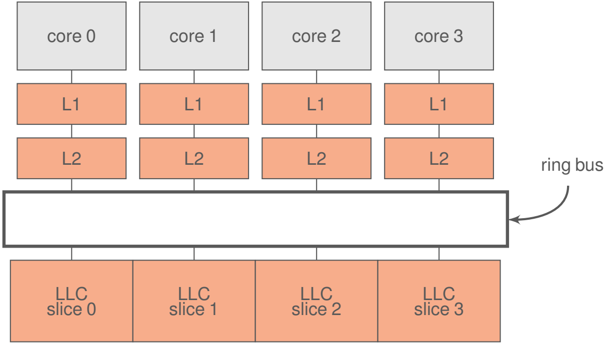
A simple cache



A simple cache



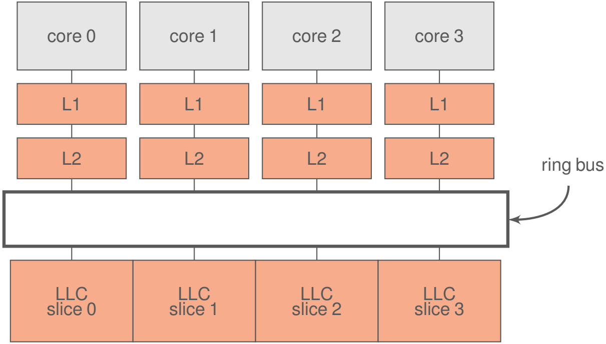
Date and Instruction Caches



last-level cache:

- shared
- inclusive

Date and Instruction Caches

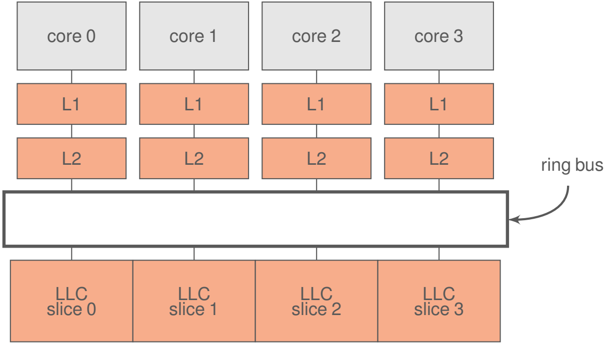


last-level cache:

- shared
- inclusive

→ shared memory shared is in cache, across cores!

Date and Instruction Caches



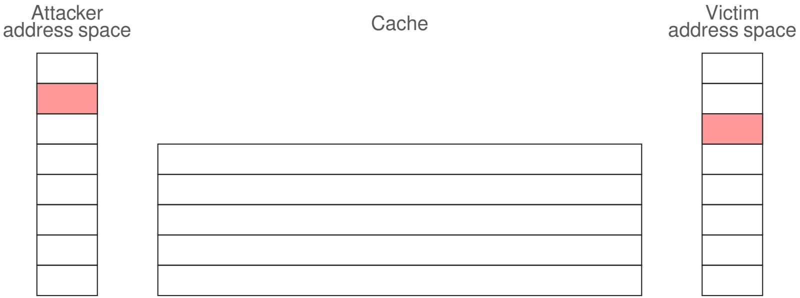
last-level cache:

- shared
- inclusive

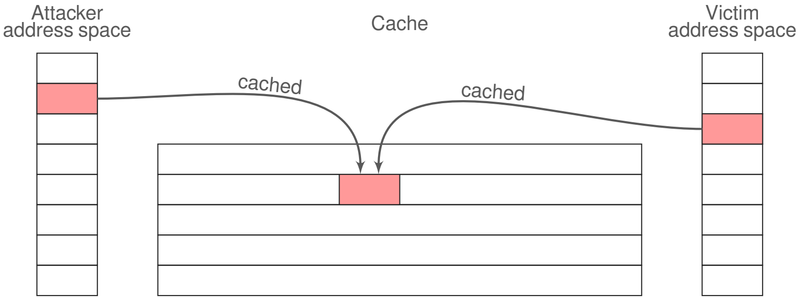
→ shared memory shared is in cache, across cores!

function maps addresses to slices (Maurice, Le Scouarnec, et al. 2015)

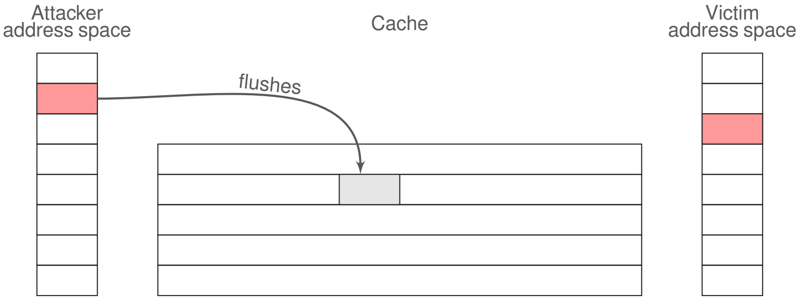
Flush+Reload



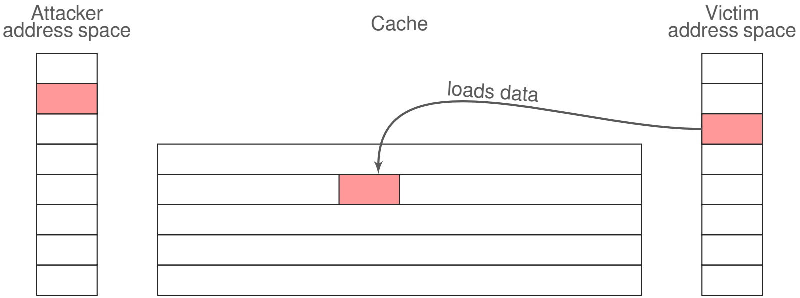
Flush+Reload



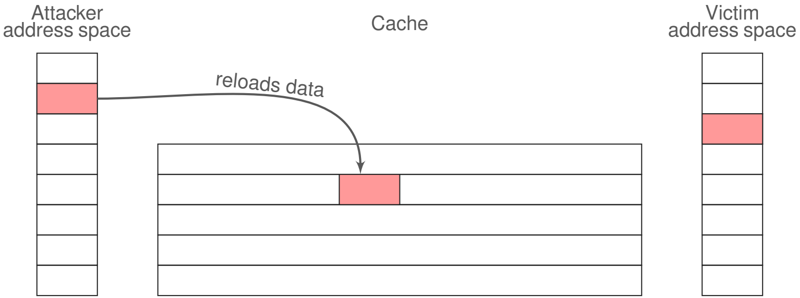
Flush+Reload



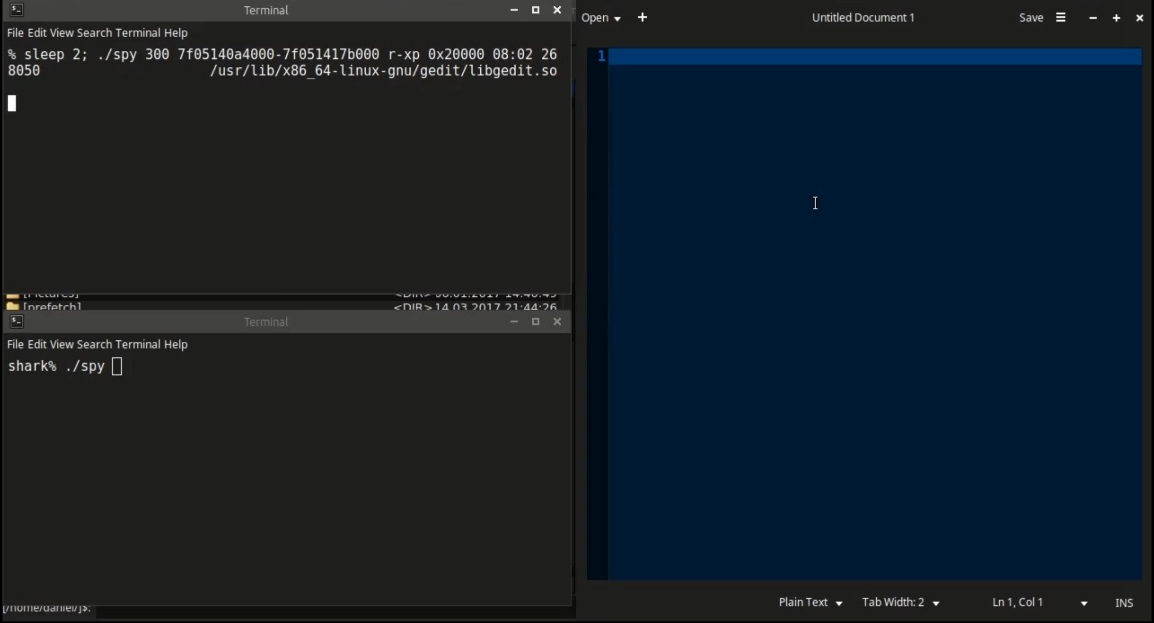
Flush+Reload



Flush+Reload

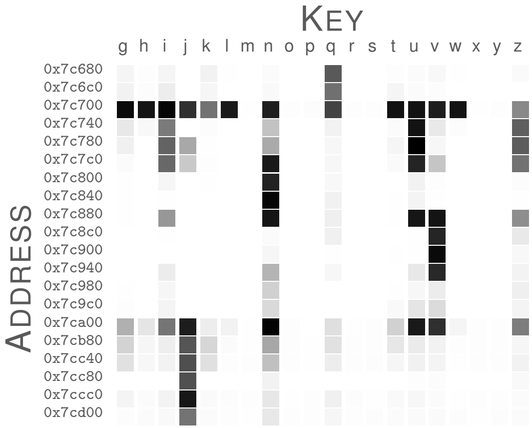


3. Contributions
   - Cache Template Attacks
   - Page Deduplication Attacks in JavaScript
   - Rowhammer.js
   - Flush+Flush
   - ARMageddon
   - Prefetch Attacks



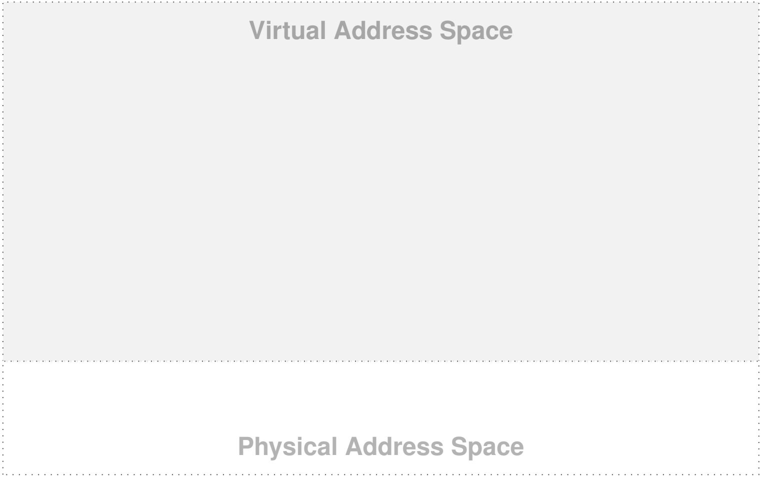
Cache Template Attack Demo

Cache Template

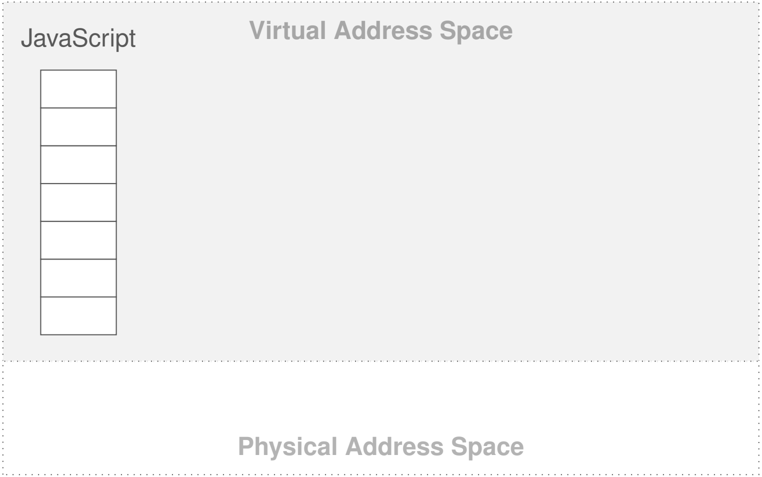


3. Contributions
   - Cache Template Attacks
   - Page Deduplication Attacks in JavaScript
   - Rowhammer.js
   - Flush+Flush
   - ARMageddon
   - Prefetch Attacks

Page Deduplication Attack

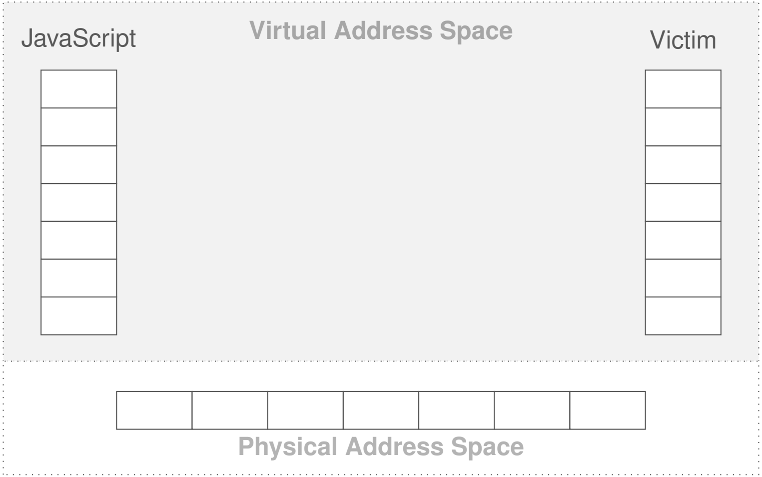


Page Deduplication Attack

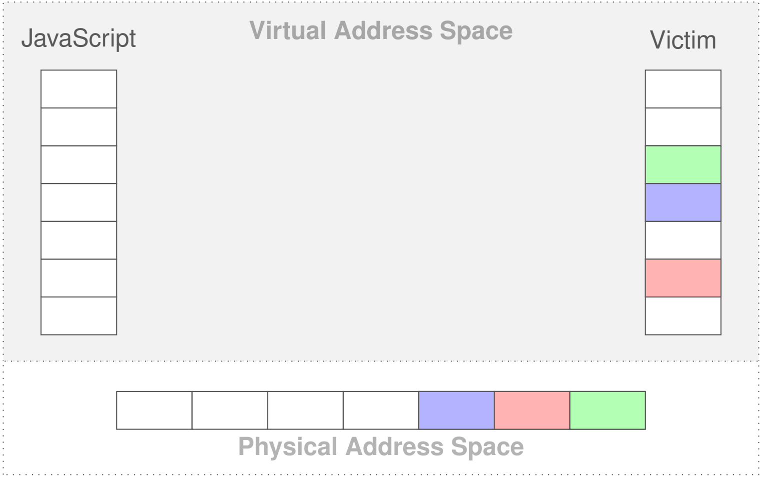


Page Deduplication Attack

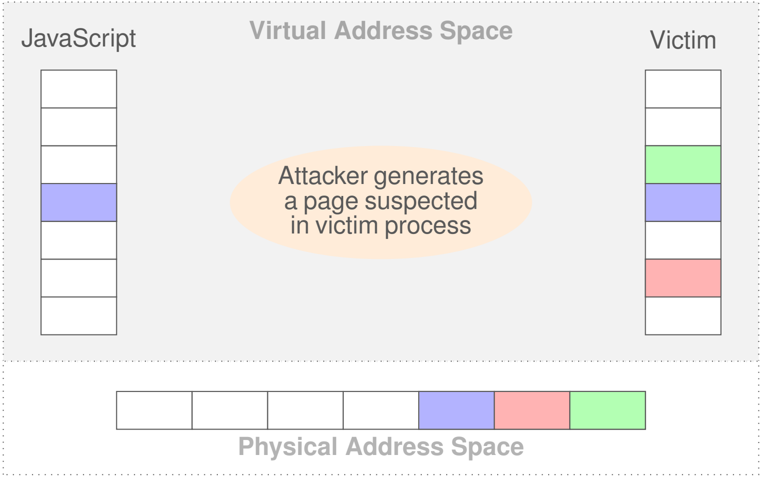
Page Deduplication Attack



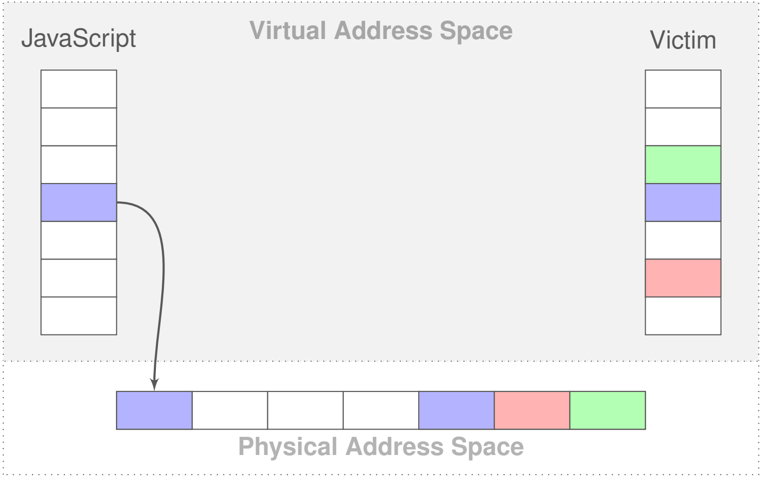
Page Deduplication Attack



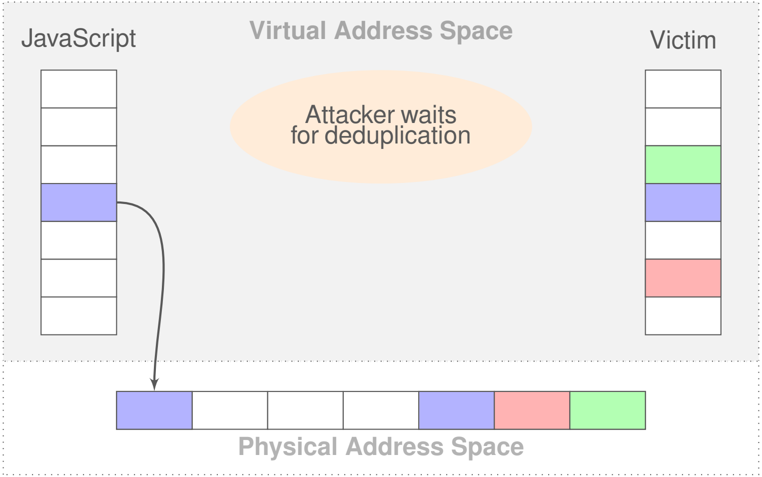
Page Deduplication Attack



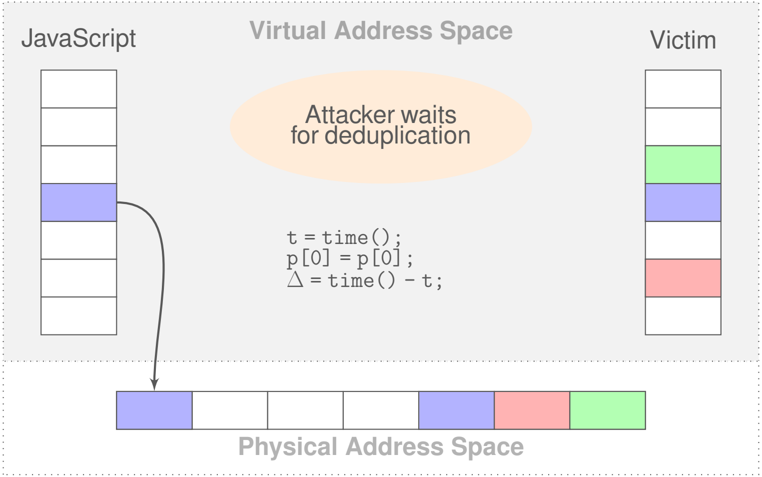
Page Deduplication Attack



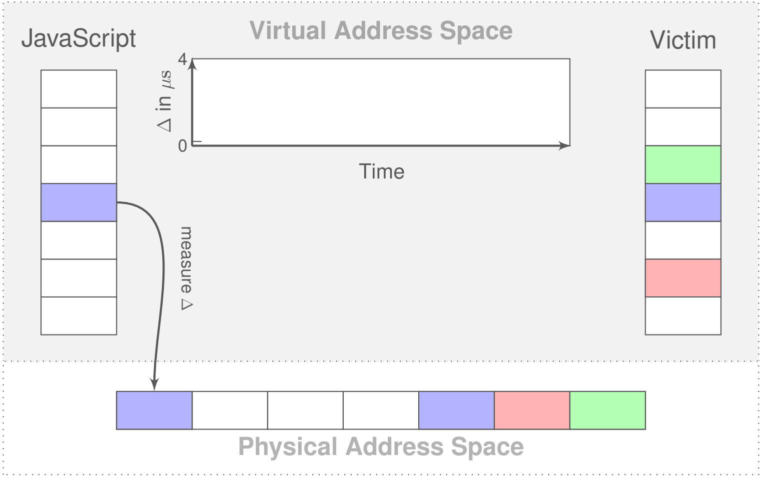
Page Deduplication Attack



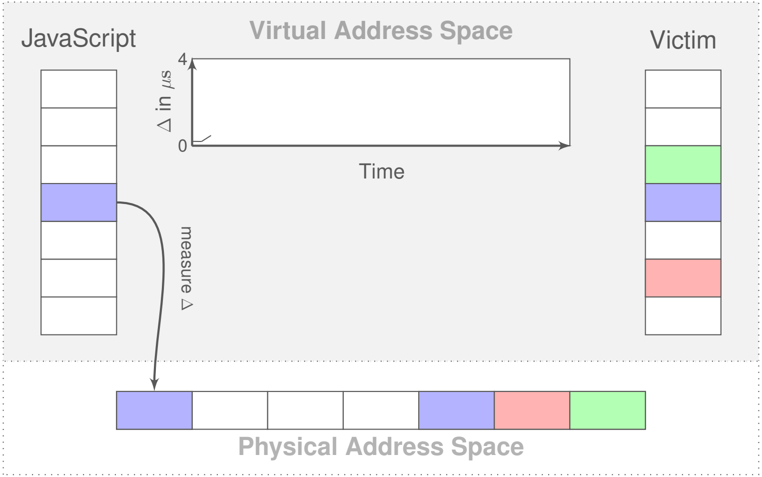
Page Deduplication Attack



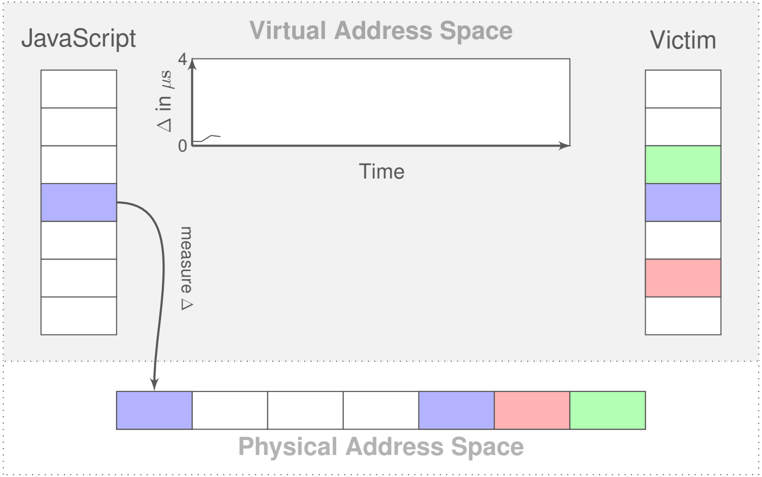
Page Deduplication Attack



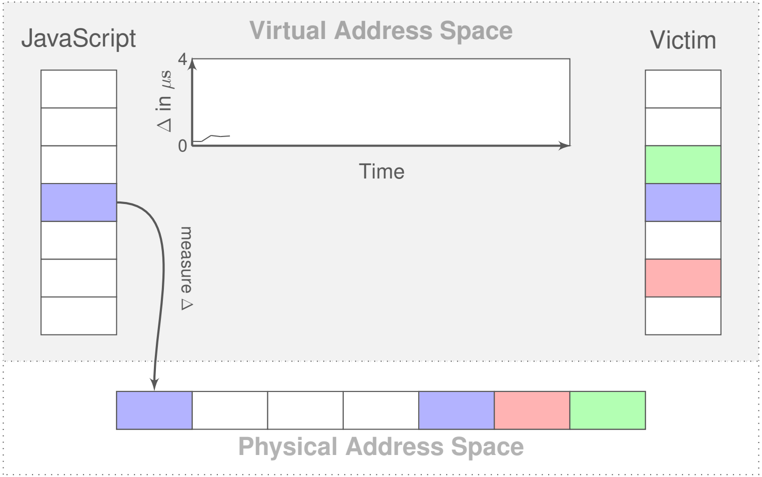
Page Deduplication Attack



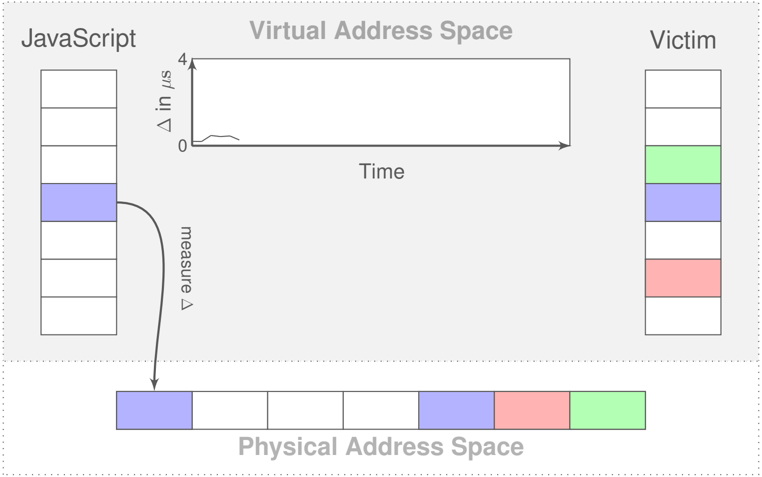
Page Deduplication Attack



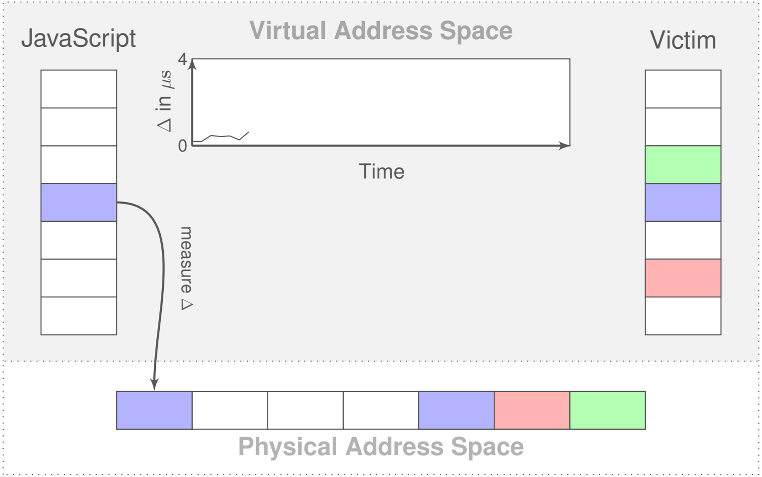
Page Deduplication Attack



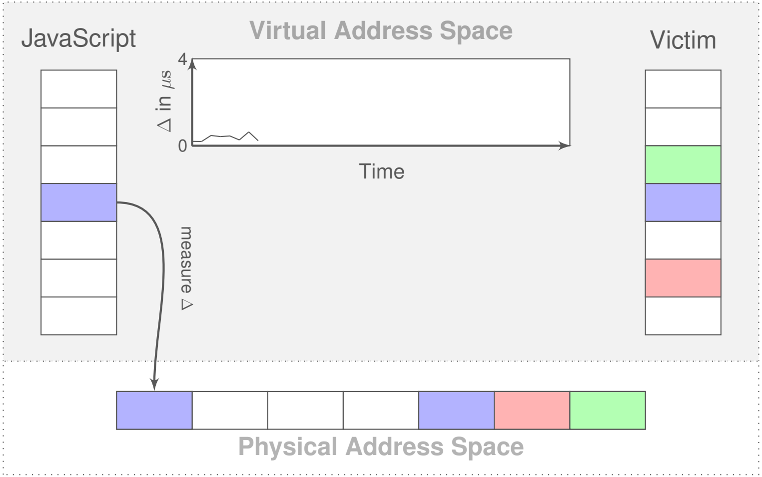
Page Deduplication Attack



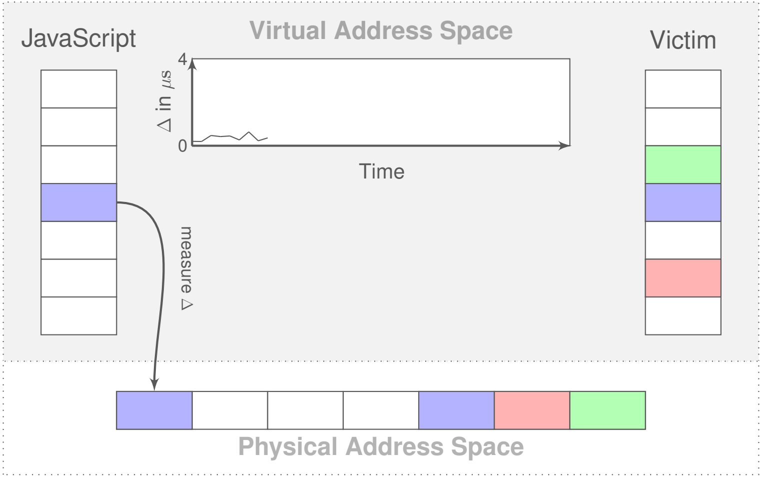
Page Deduplication Attack



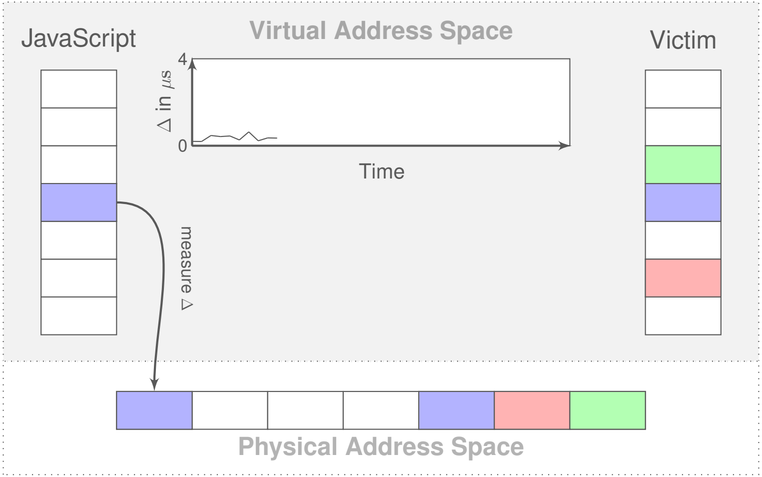
Page Deduplication Attack



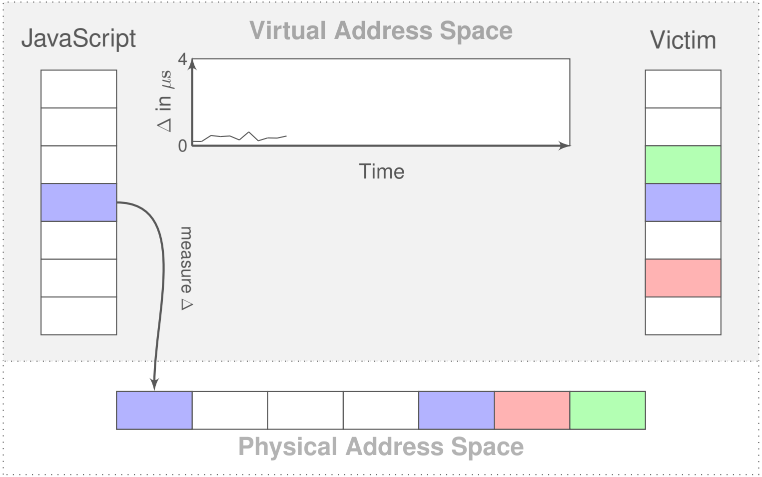
Page Deduplication Attack



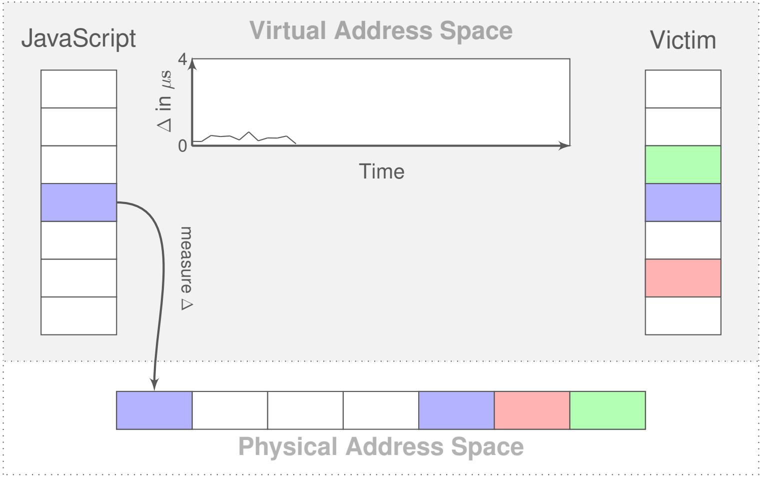
Page Deduplication Attack



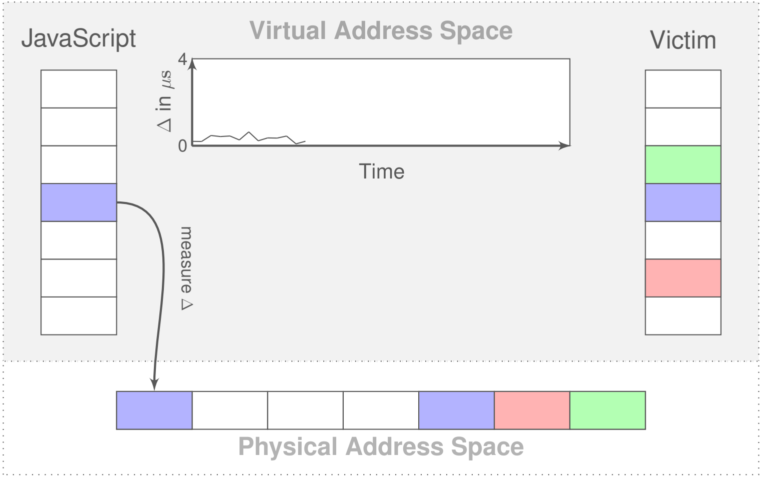
Page Deduplication Attack



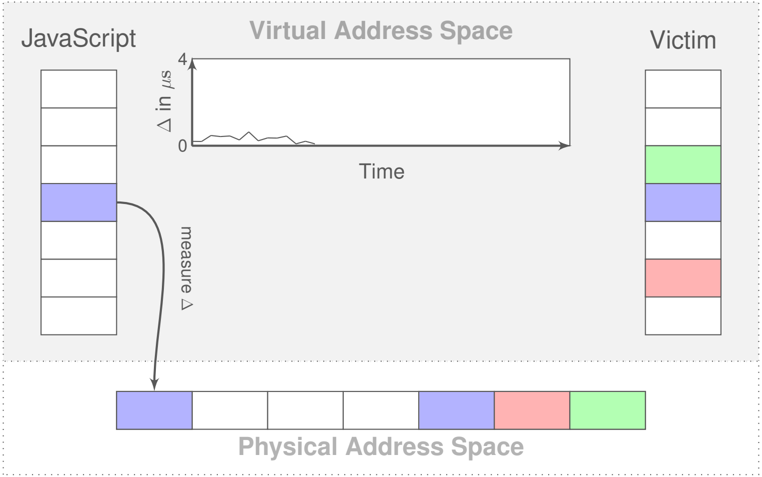
Page Deduplication Attack



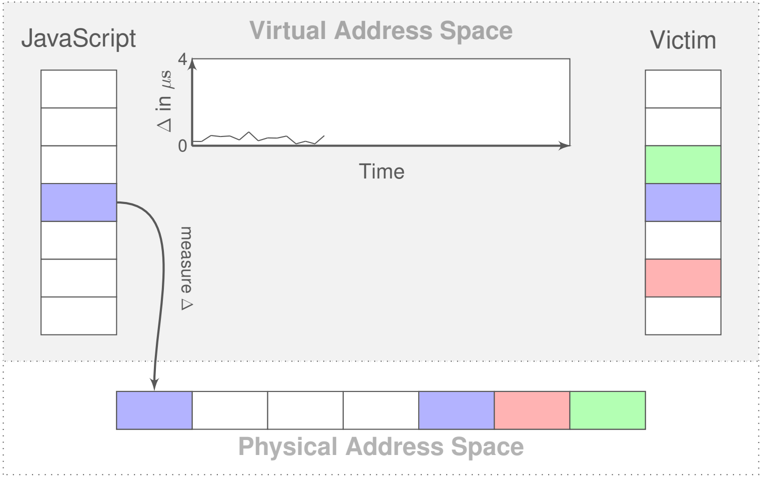
Page Deduplication Attack



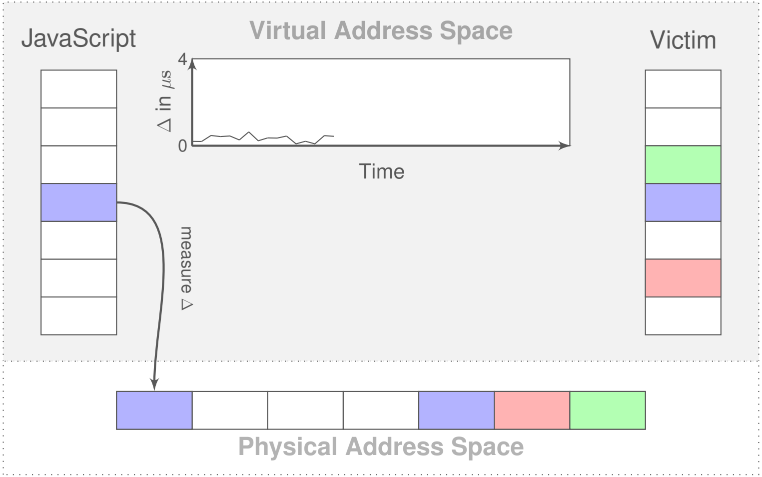
Page Deduplication Attack



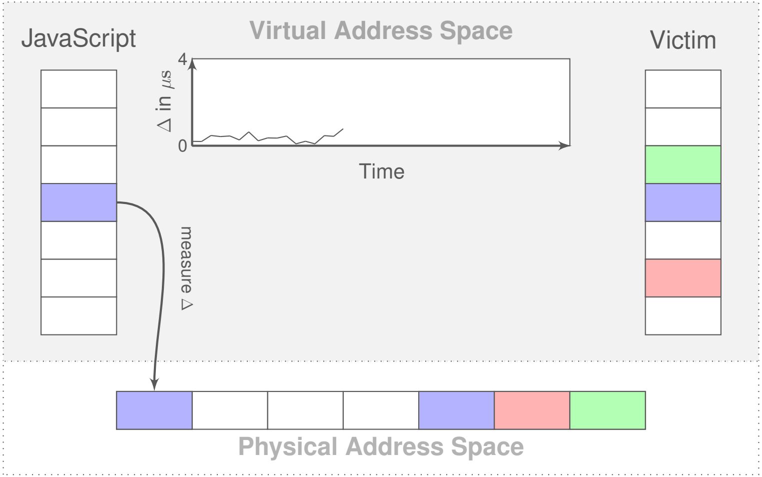
Page Deduplication Attack



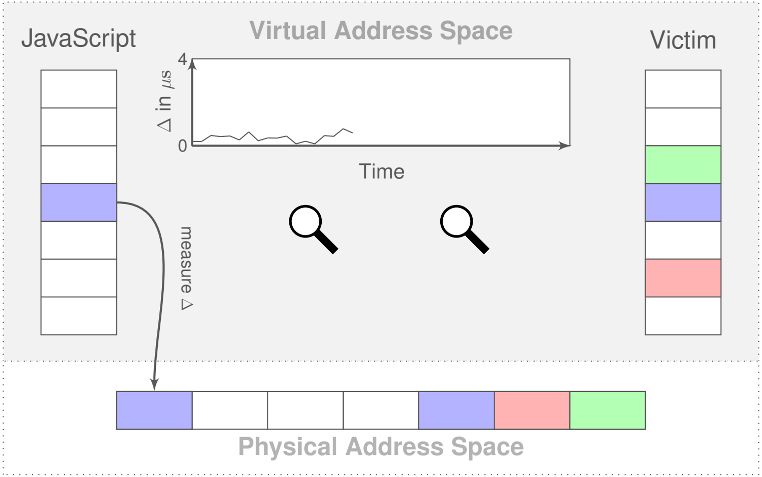
Page Deduplication Attack



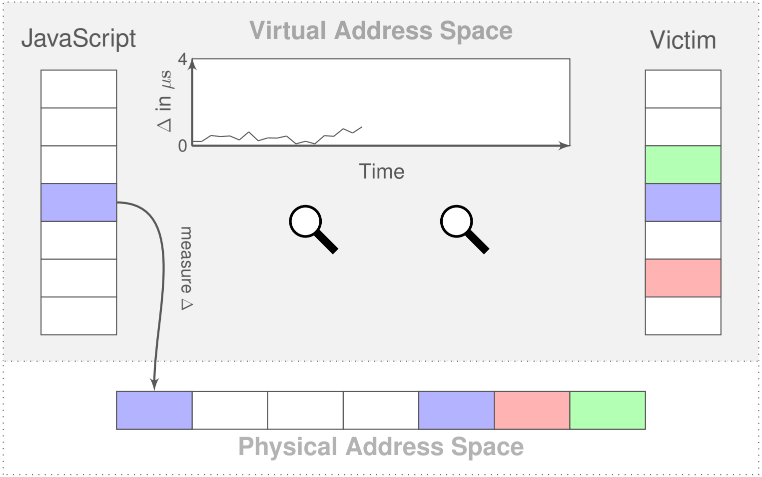
Page Deduplication Attack



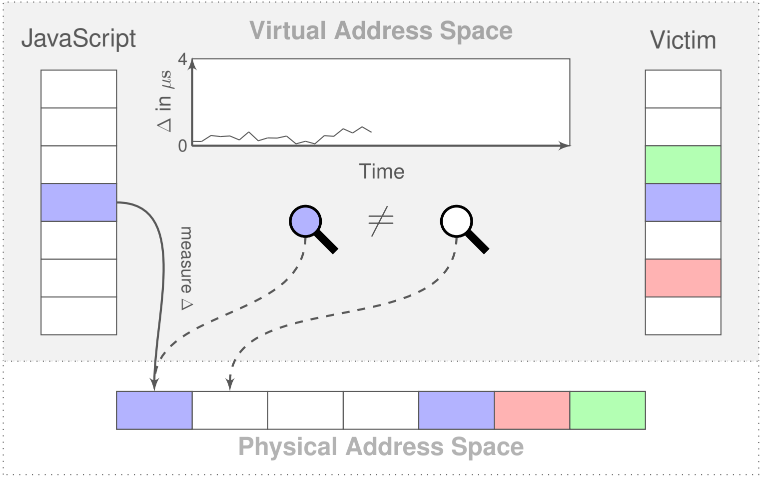
Page Deduplication Attack



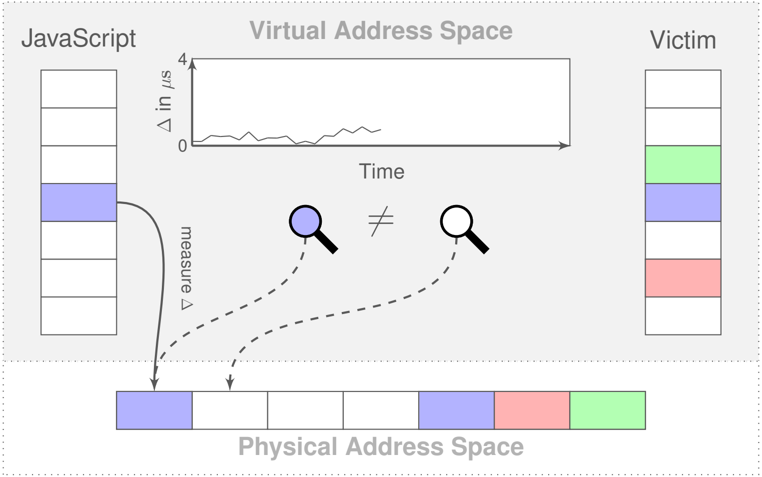
Page Deduplication Attack



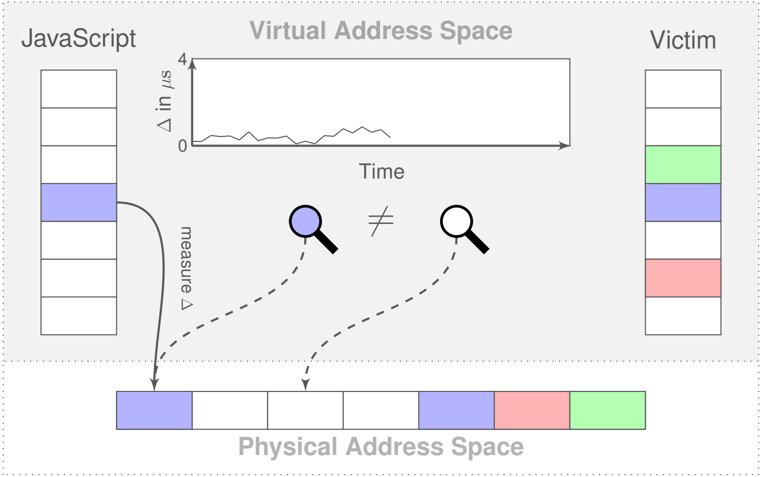
Page Deduplication Attack



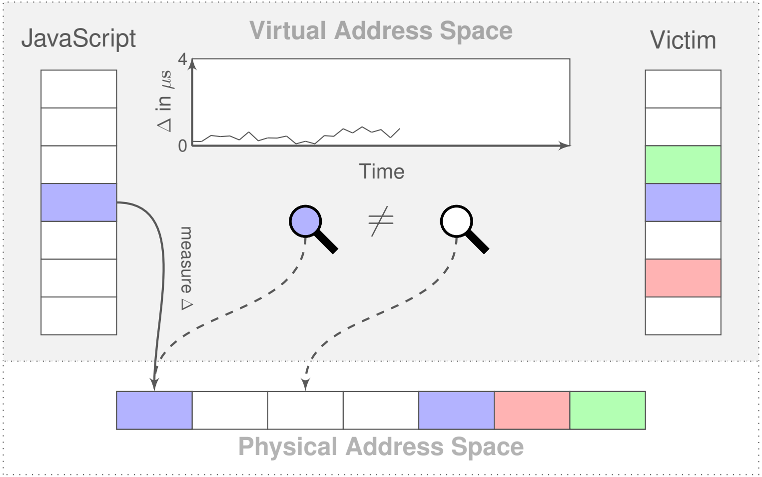
Page Deduplication Attack



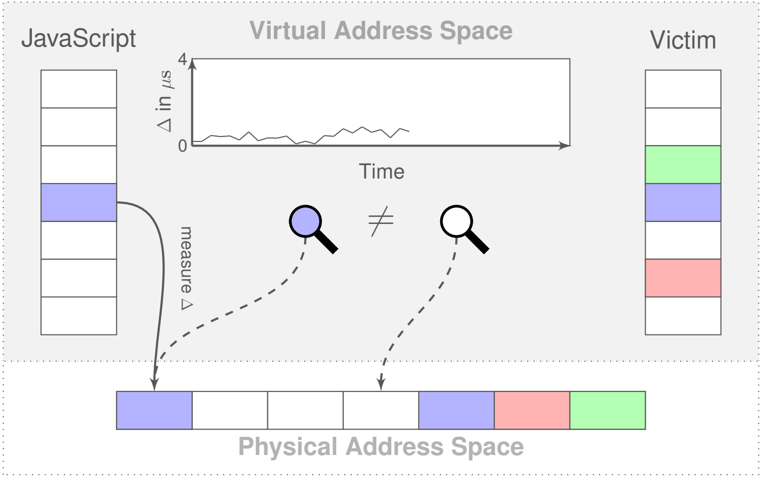
Page Deduplication Attack



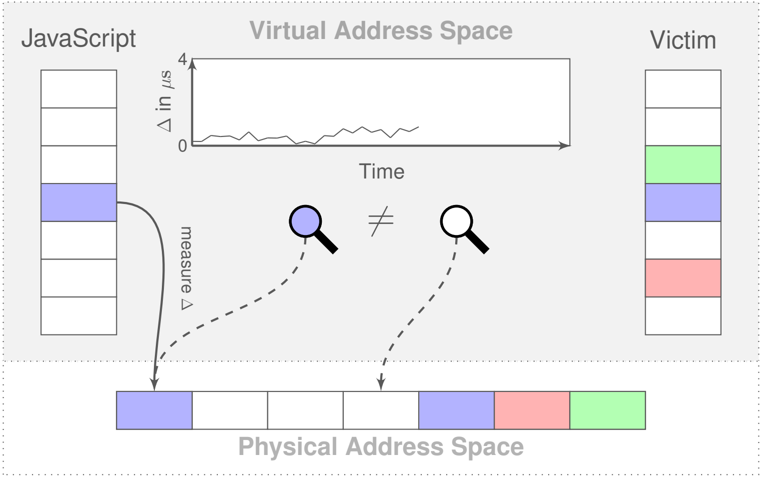
Page Deduplication Attack



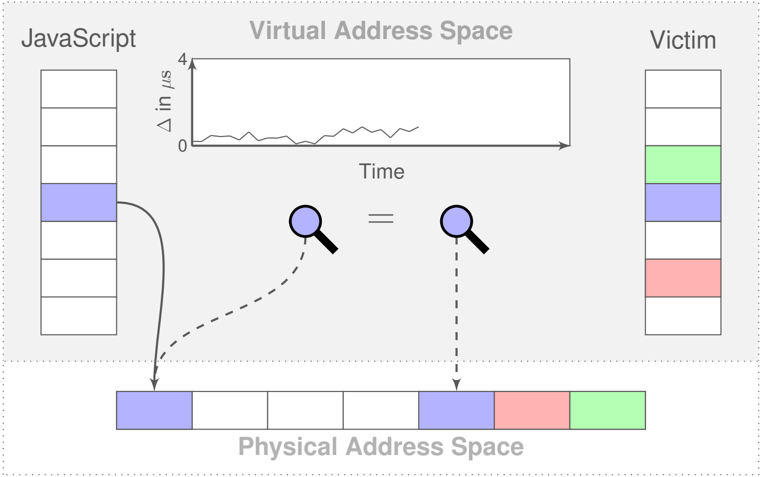
Page Deduplication Attack



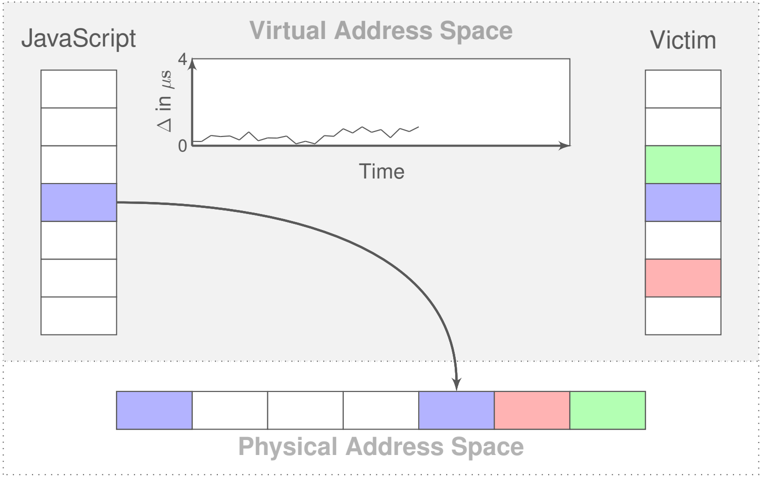
Page Deduplication Attack



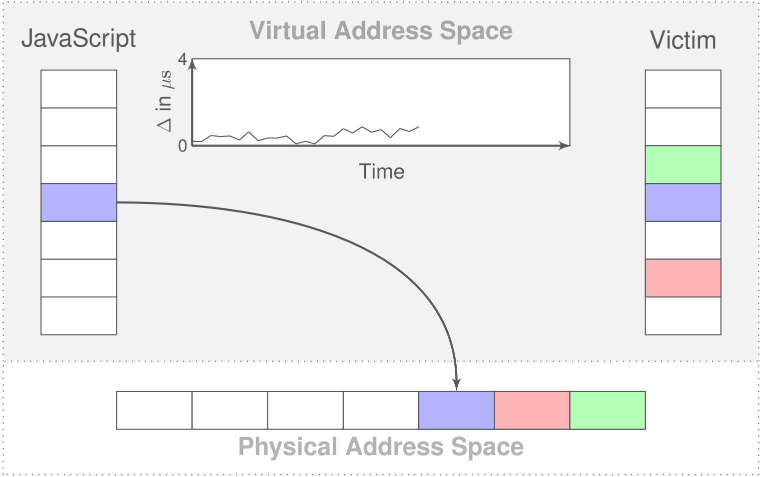
Page Deduplication Attack



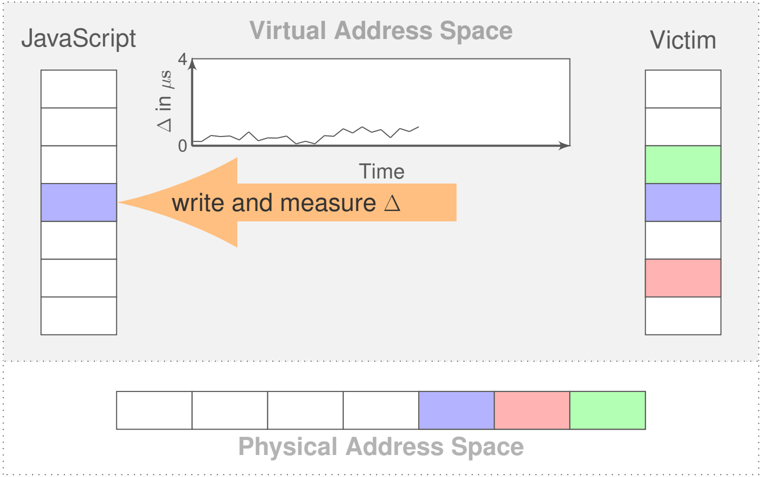
Page Deduplication Attack



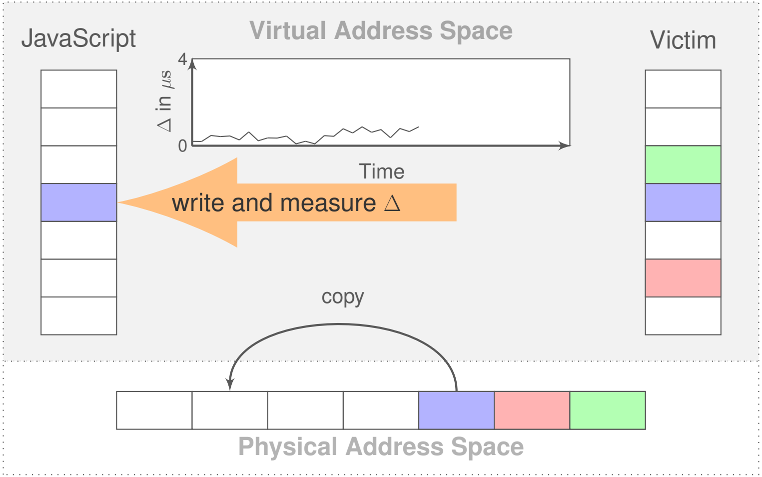
Page Deduplication Attack



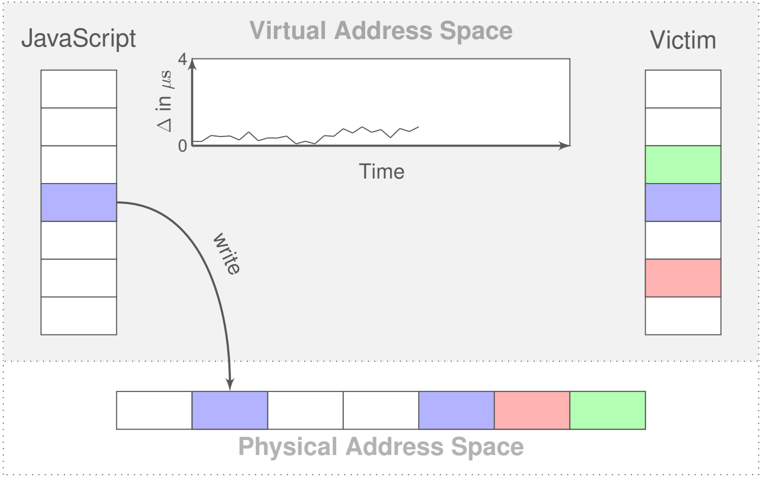
Page Deduplication Attack



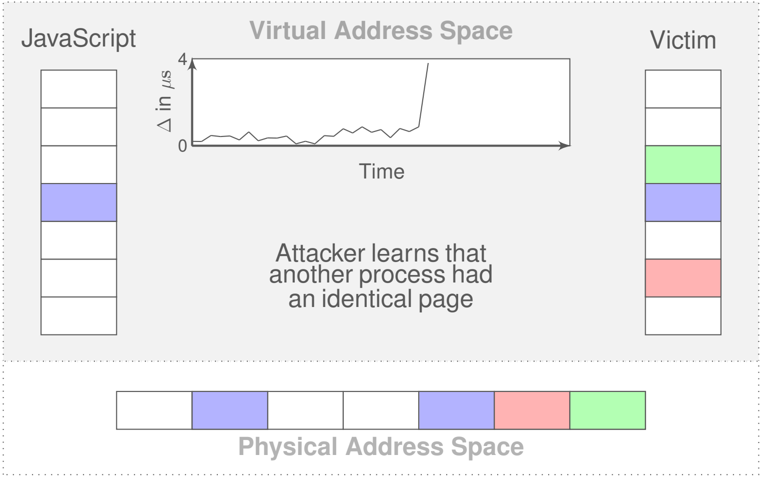
Page Deduplication Attack



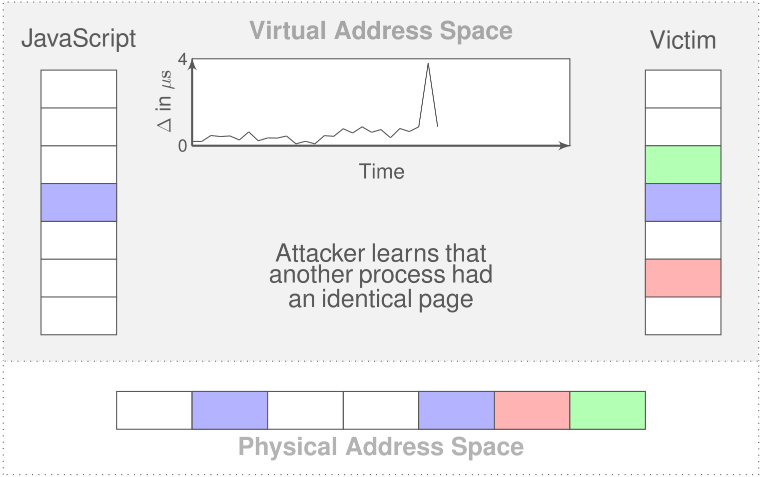
Page Deduplication Attack



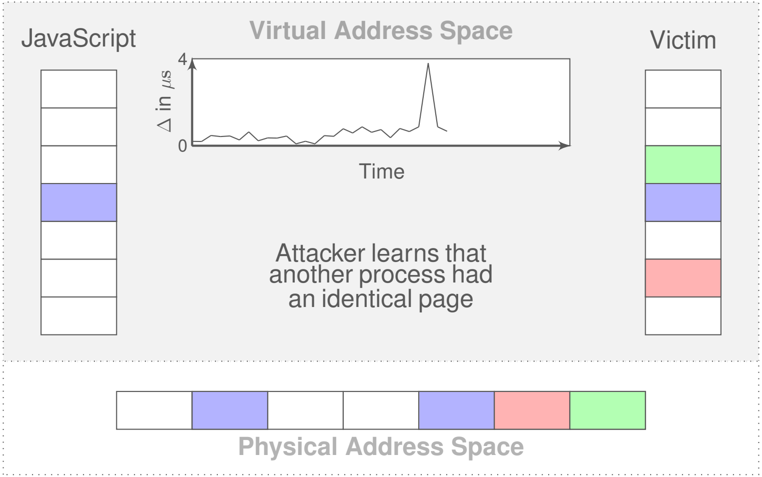
Page Deduplication Attack



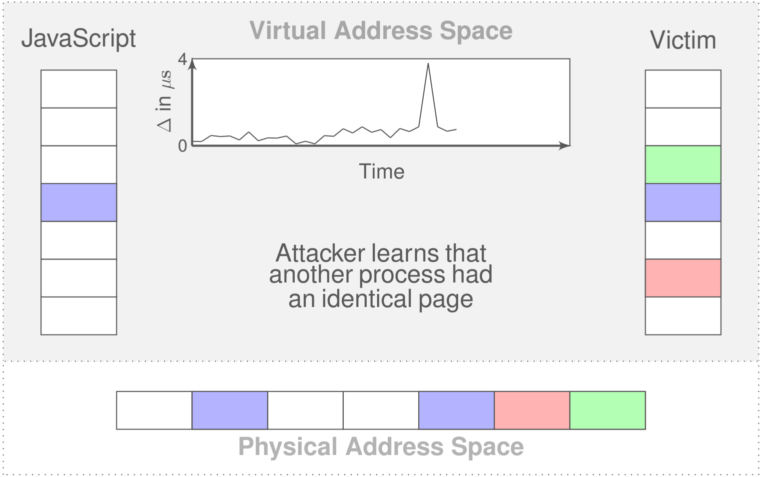
Page Deduplication Attack



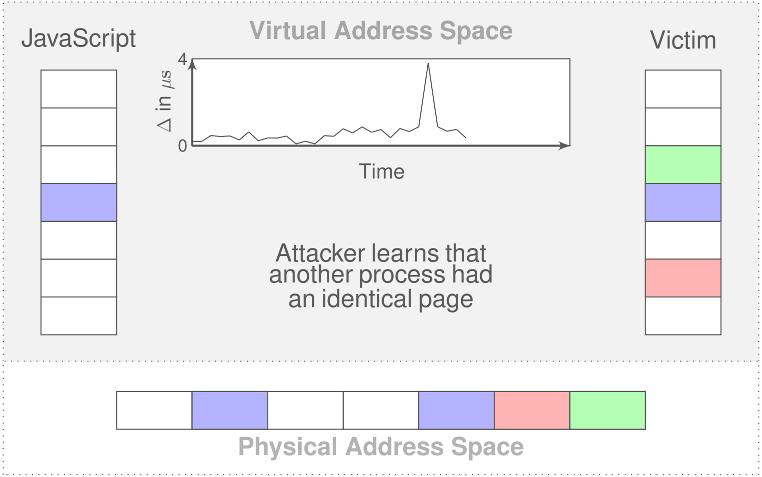
Page Deduplication Attack



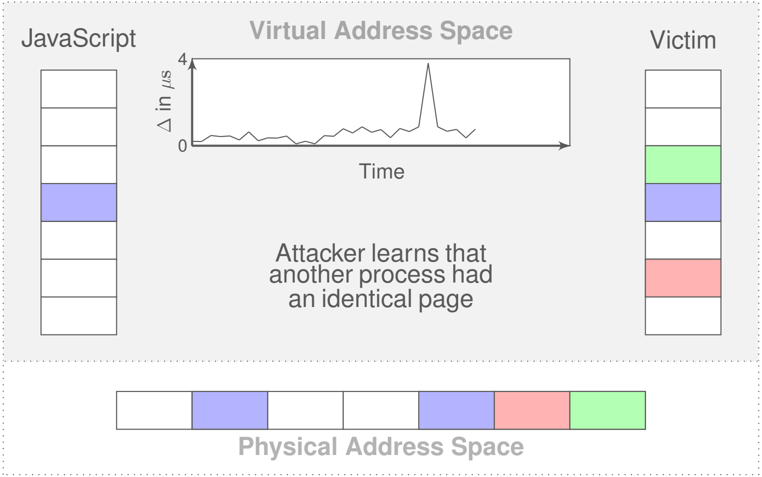
Page Deduplication Attack



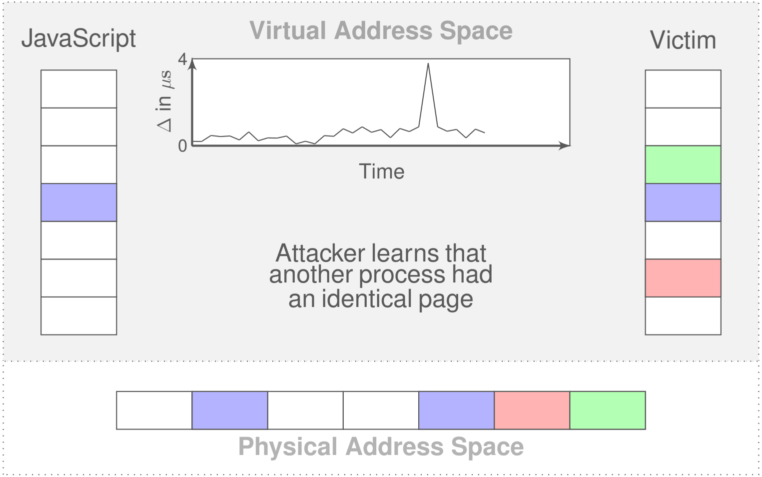
Page Deduplication Attack



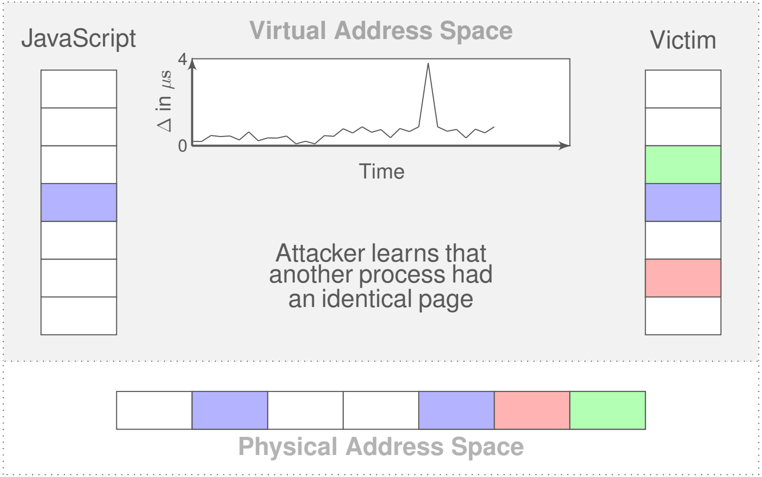
Page Deduplication Attack



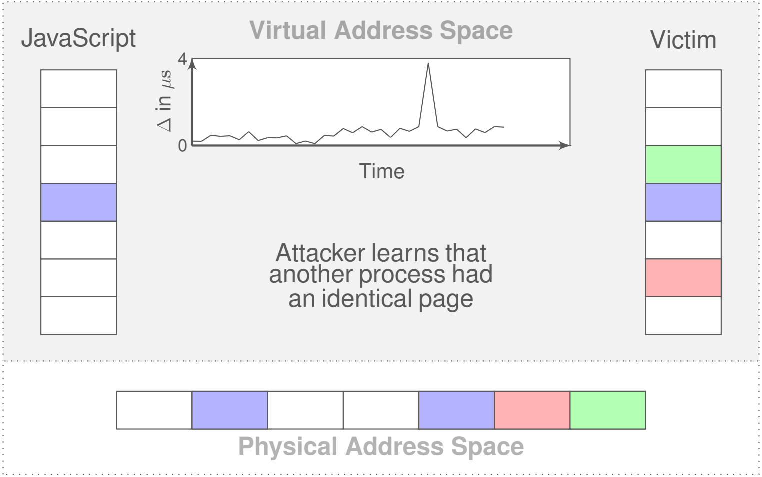
Page Deduplication Attack



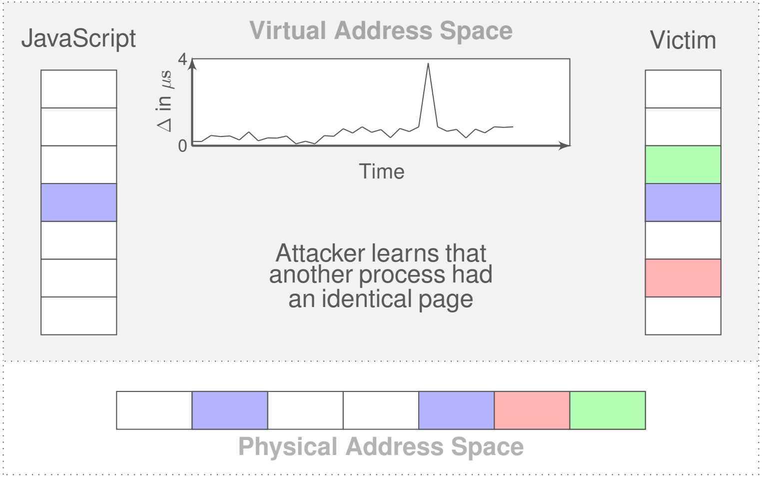
Page Deduplication Attack



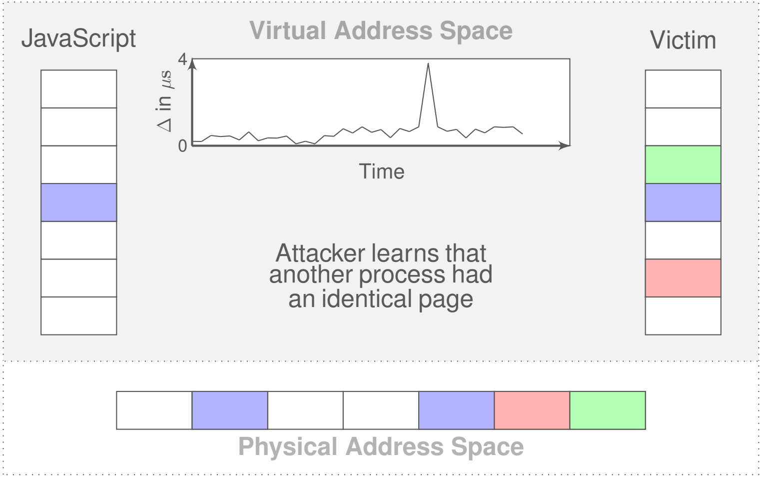
Page Deduplication Attack



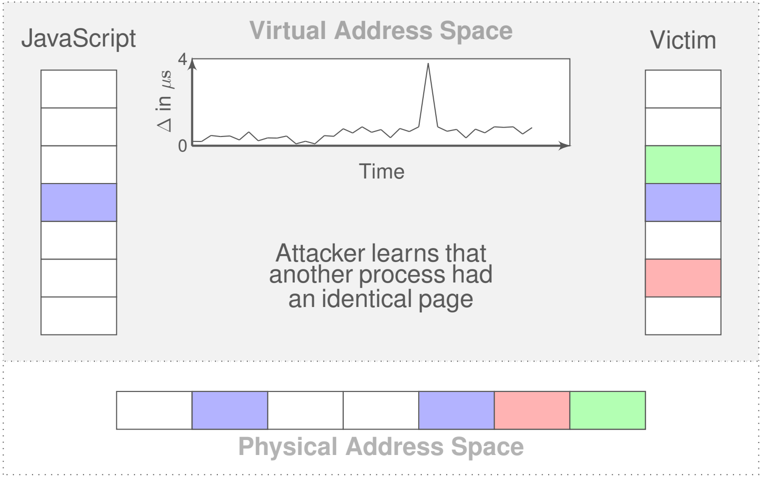
Page Deduplication Attack



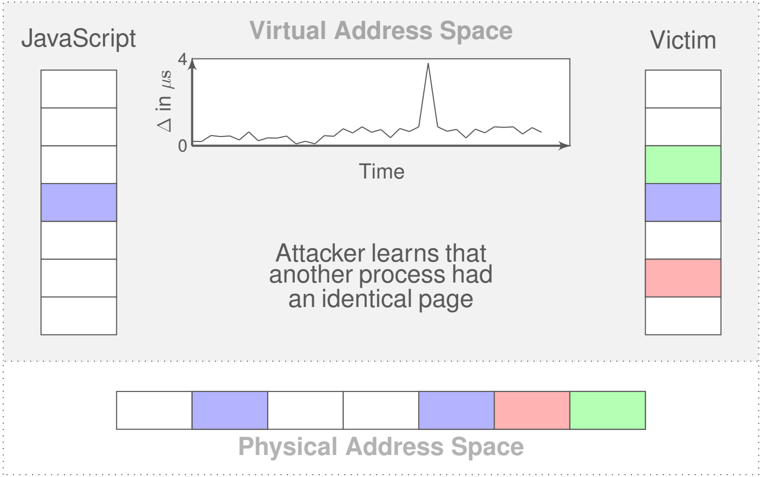
Page Deduplication Attack



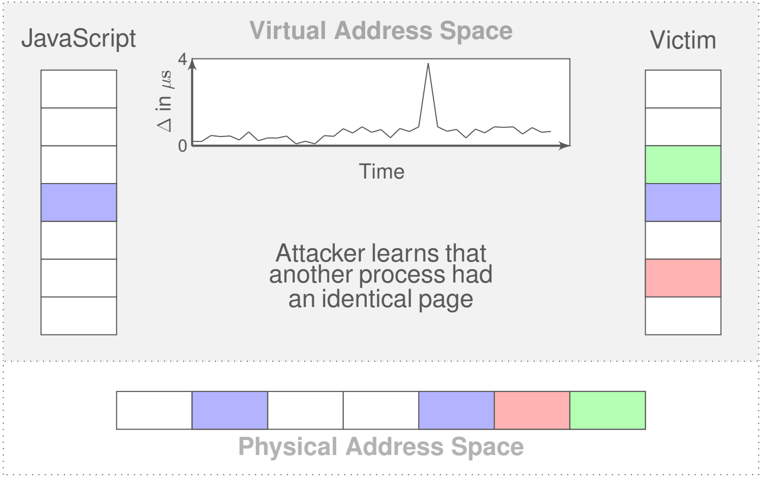
Page Deduplication Attack



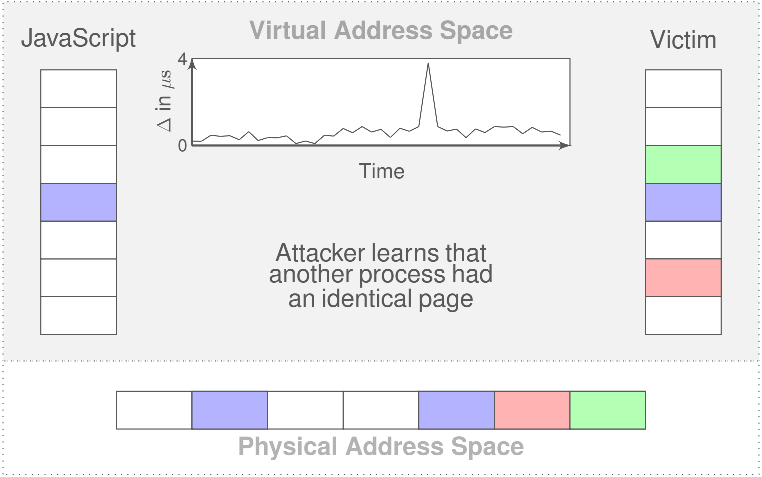
Page Deduplication Attack



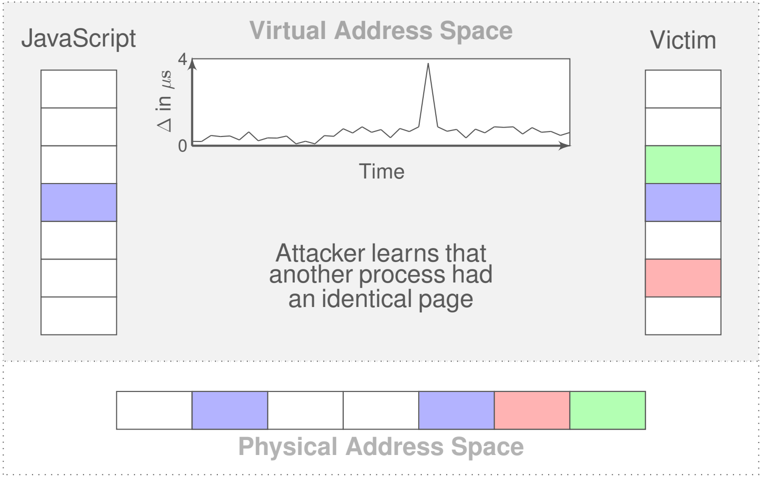
Page Deduplication Attack



Page Deduplication Attack



Page Deduplication Attack

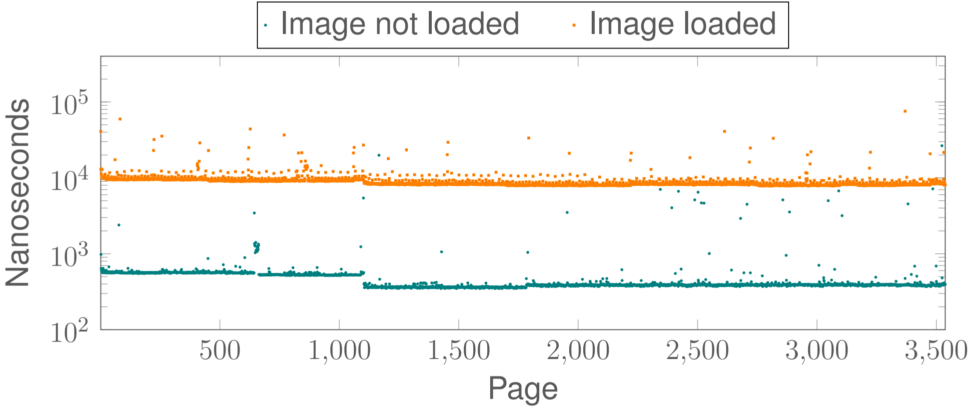


Our Attack

First page deduplication attack which

- detects CSS files/images on websites,
- runs in JavaScript (no `rdtsc`, no addresses),
- runs on KVM, Windows 8.1 and Android.

Detect Image (JavaScript, Cross-VM, KVM)

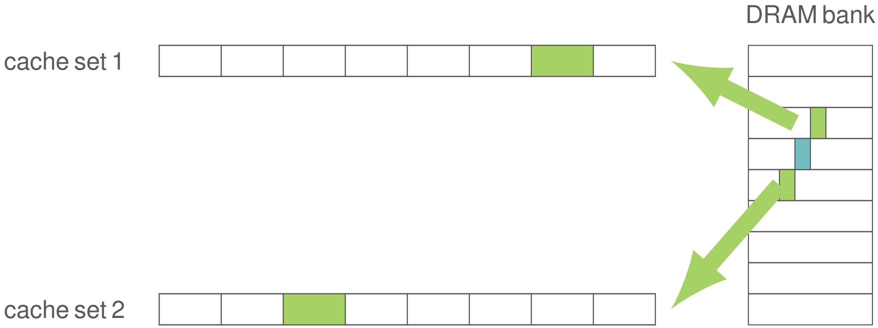


3. Contributions
   - Cache Template Attacks
   - Page Deduplication Attacks in JavaScript
   - Rowhammer.js
   - Flush+Flush
   - ARMageddon
   - Prefetch Attacks

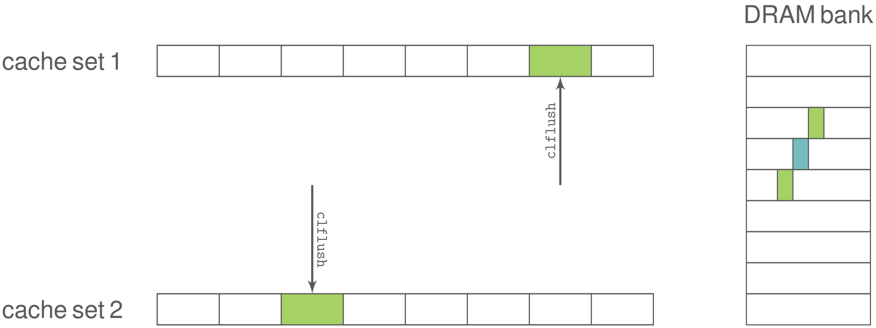
Rowhammer

- Rowhammer: DRAM bug that causes bit flips (Kim et al. 2014)
- Bug used in security exploits (Seaborn 2015)
- Only non-cached accesses reach DRAM
- Very similar to Flush+Reload

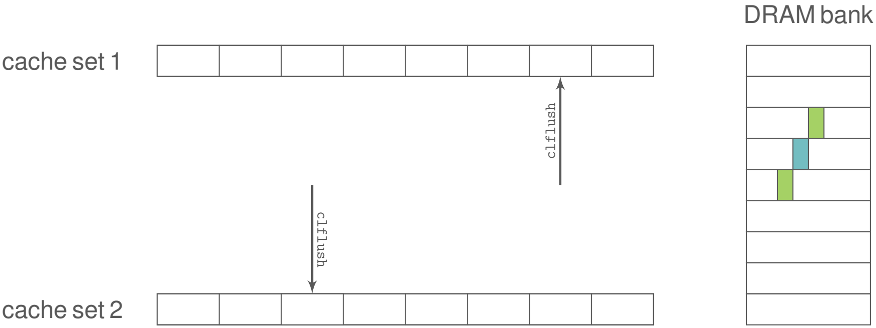
Rowhammer (with `clflush`)



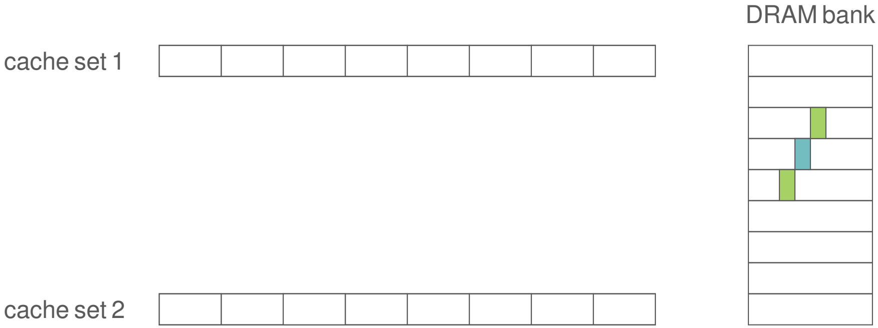
Rowhammer (with `clflush`)



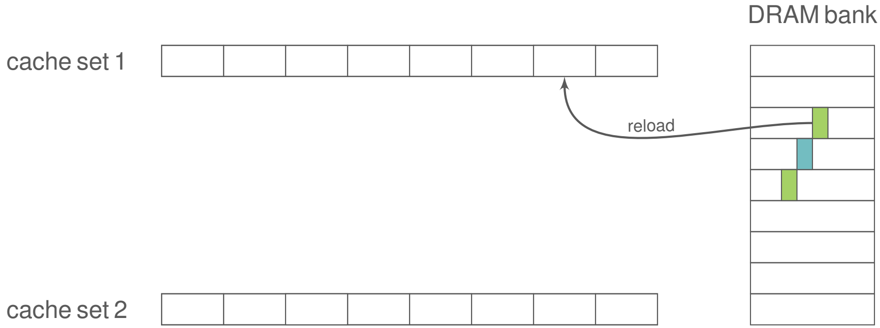
Rowhammer (with `clflush`)



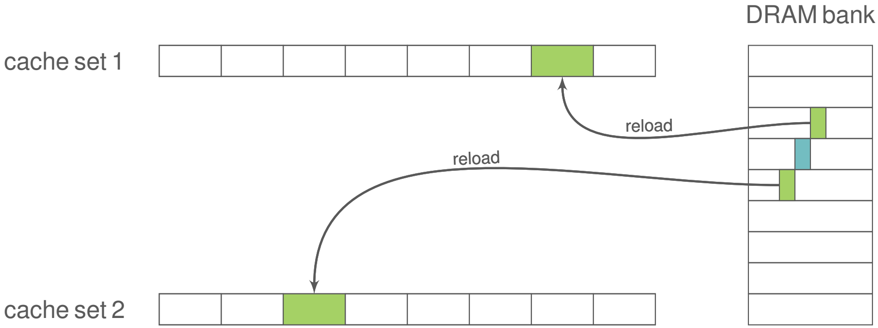
Rowhammer (with `clflush`)



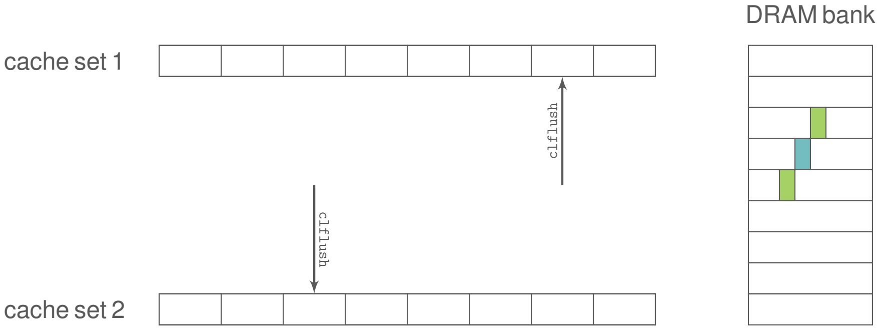
Rowhammer (with `clflush`)



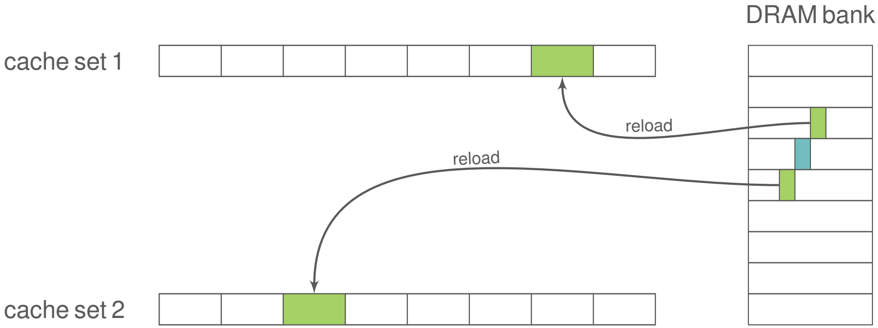
Rowhammer (with `clflush`)



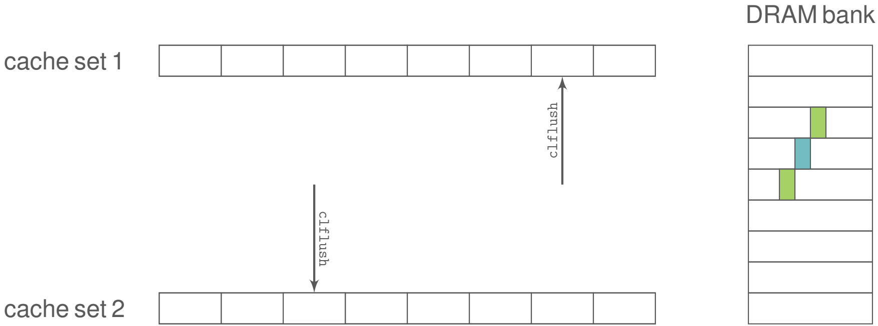
Rowhammer (with `clflush`)



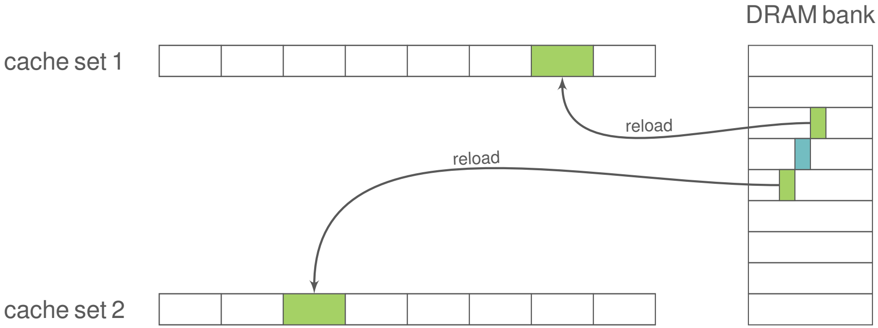
Rowhammer (with `clflush`)



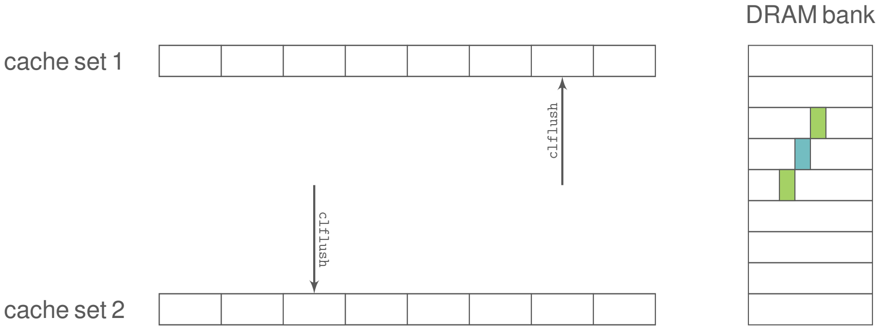
Rowhammer (with `clflush`)



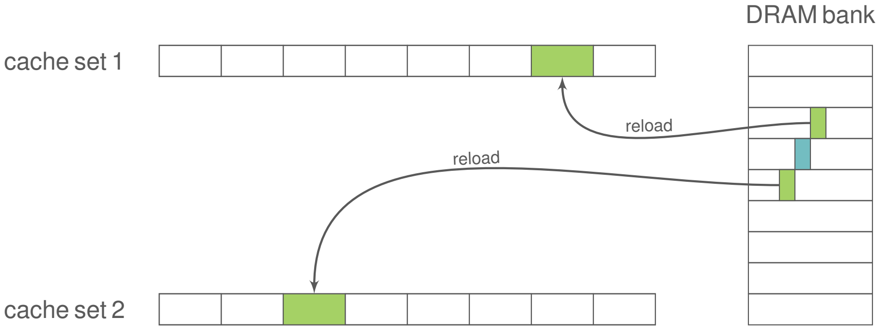
Rowhammer (with `clflush`)



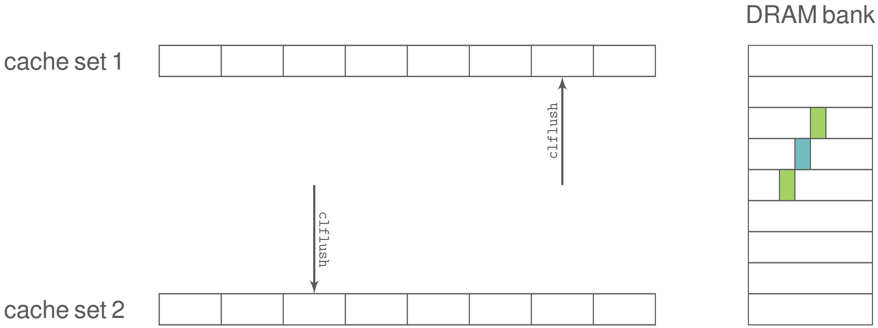
Rowhammer (with `clflush`)



Rowhammer (with `clflush`)

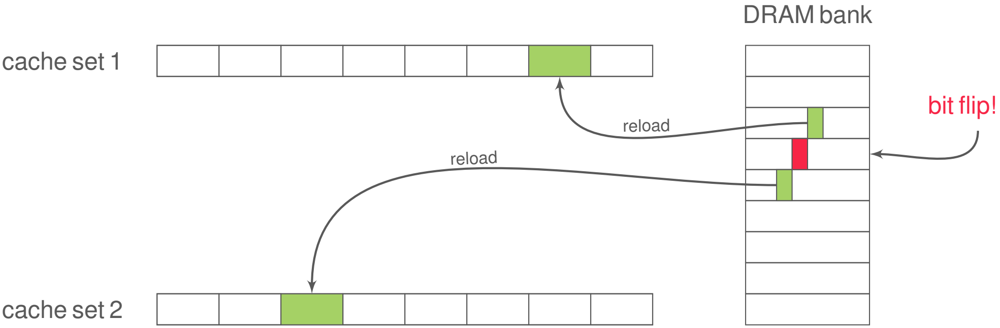


Rowhammer (with `clflush`)

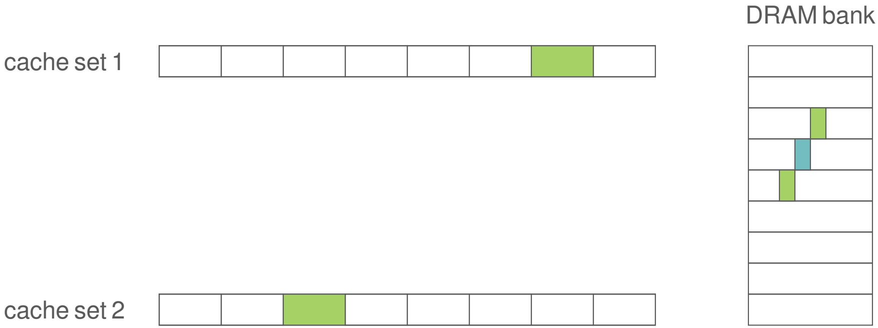


wait for it. . .

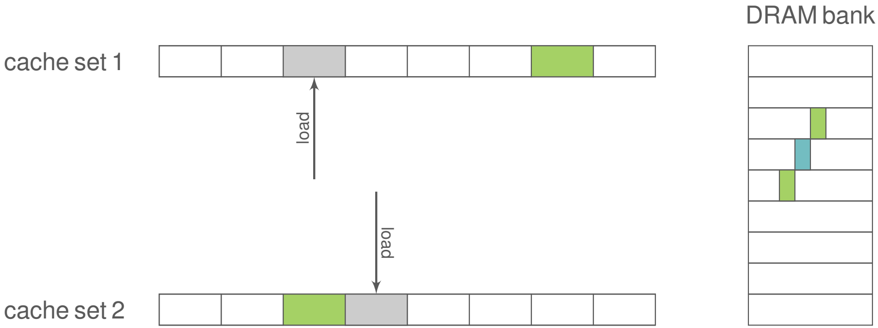
Rowhammer (with `clflush`)



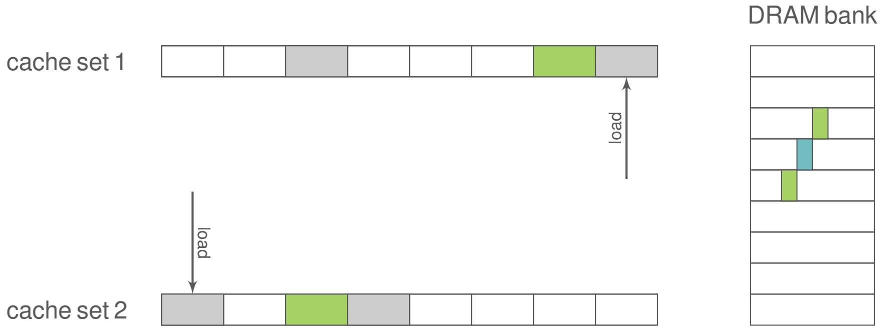
Rowhammer without `clflush`



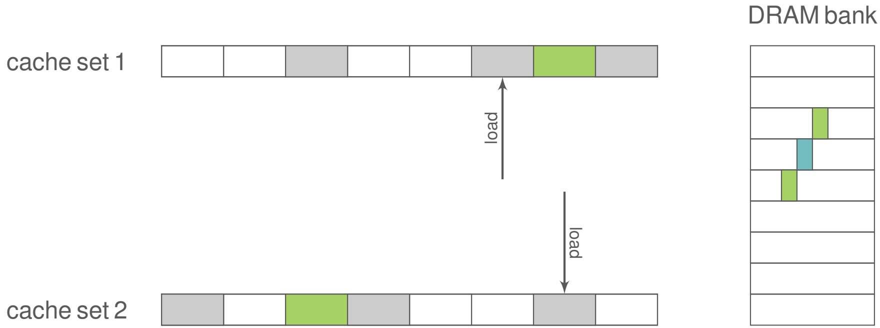
Rowhammer without `clflush`



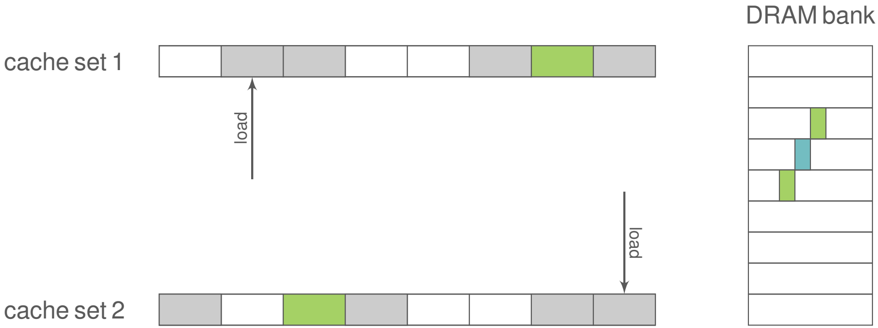
Rowhammer without `clflush`



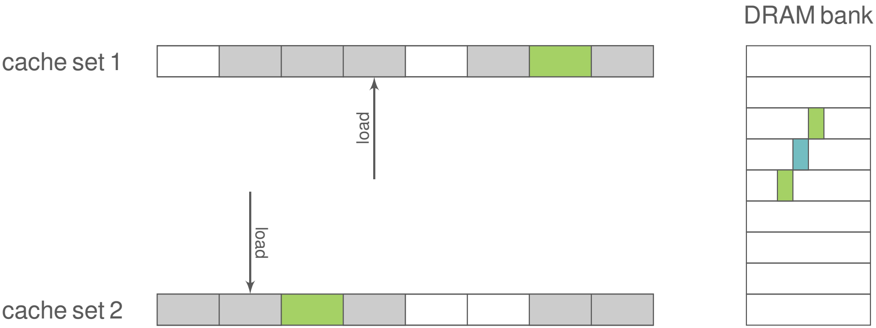
Rowhammer without `clflush`



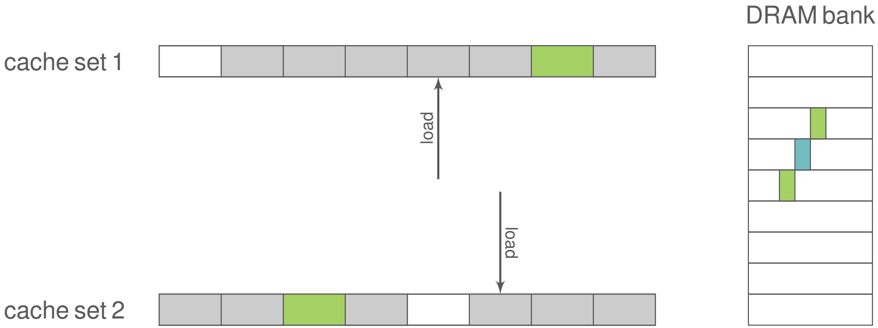
Rowhammer without `clflush`



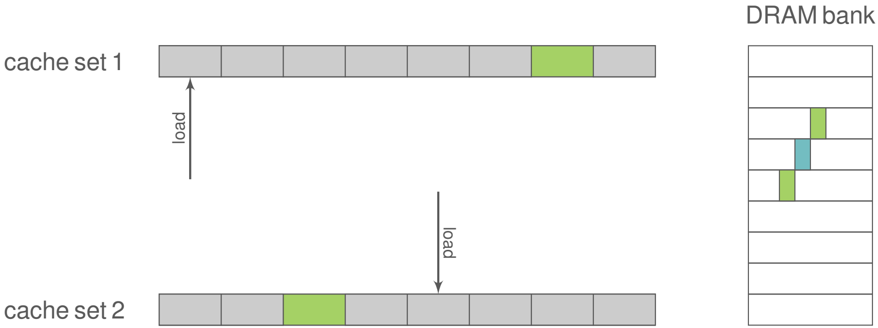
Rowhammer without `clflush`



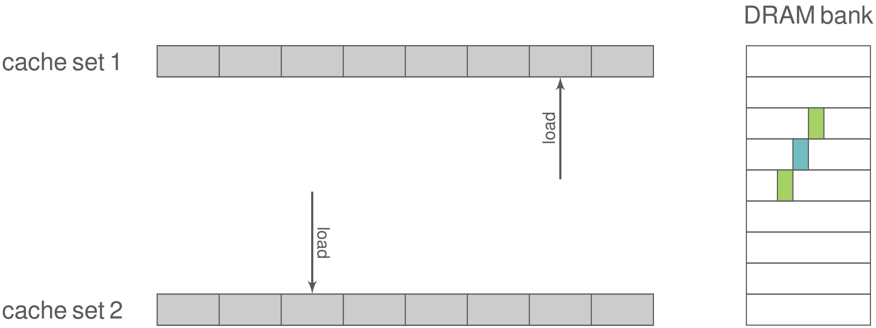
Rowhammer without `clflush`



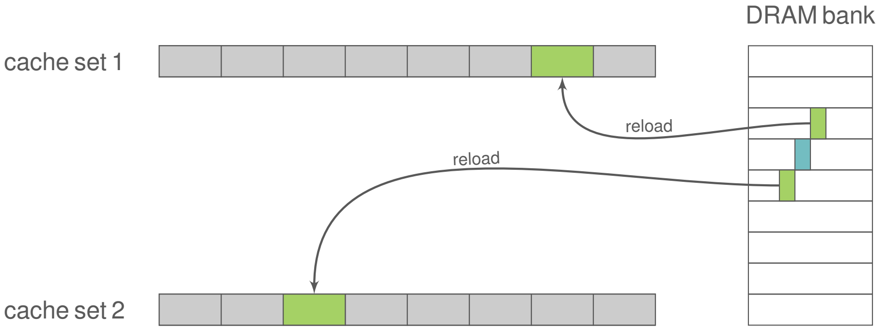
Rowhammer without `clflush`



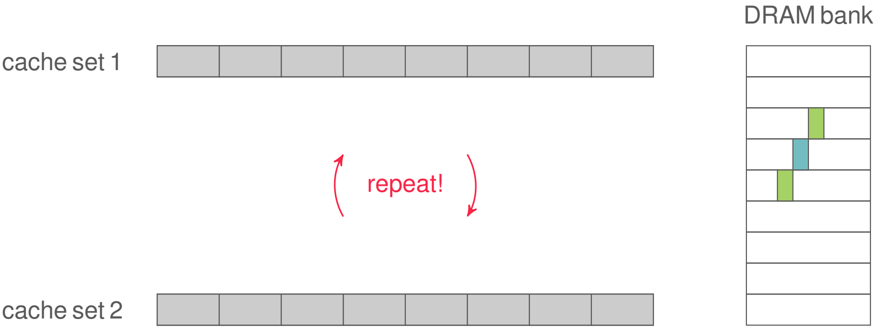
Rowhammer without `clflush`



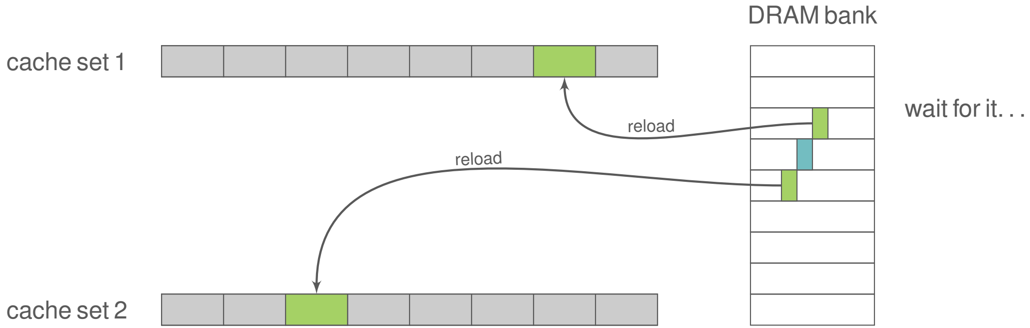
Rowhammer without `clflush`



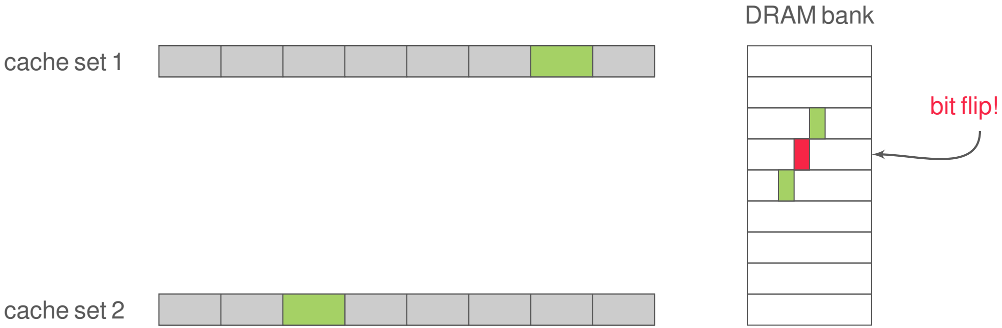
Rowhammer without `clflush`



Rowhammer without `clflush`



Rowhammer without `clflush`



Rowhammer without clflush

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 without clflush

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

Replacement policy on older CPUs

“LRU eviction” memory accesses

cache set

Replacement policy on older CPUs

“LRU eviction” memory accesses

- LRU replacement policy: oldest entry first

Replacement policy on older CPUs

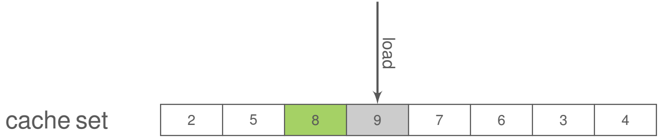
“LRU eviction” memory accesses

cache set	2	5	8	1	7	6	3	4

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

Replacement policy on older CPUs

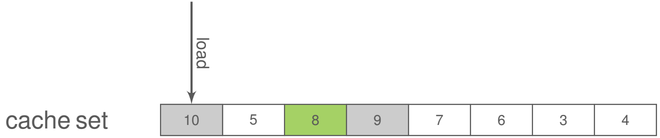
“LRU eviction” memory accesses



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

Replacement policy on older CPUs

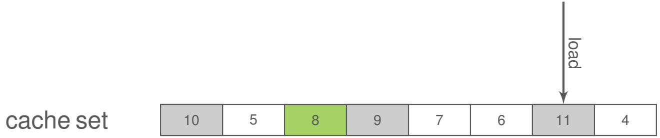
"LRU eviction" memory accesses



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

Replacement policy on older CPUs

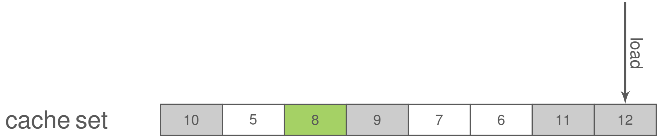
"LRU eviction" memory accesses



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

Replacement policy on older CPUs

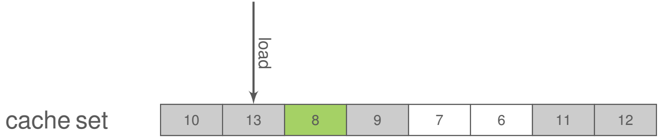
“LRU eviction” memory accesses



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

Replacement policy on older CPUs

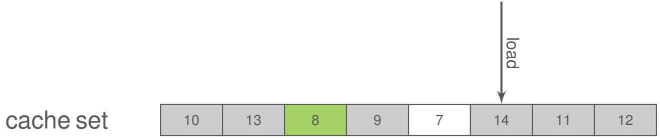
“LRU eviction” memory accesses



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

Replacement policy on older CPUs

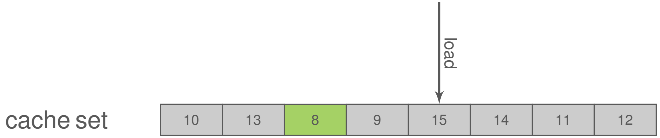
"LRU eviction" memory accesses



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

Replacement policy on older CPUs

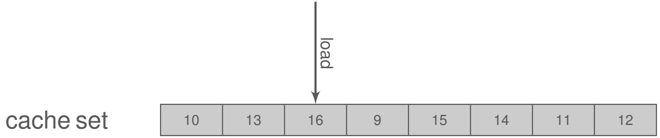
“LRU eviction” memory accesses



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

Replacement policy on older CPUs

“LRU eviction” memory accesses



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

Replacement policy on recent CPUs

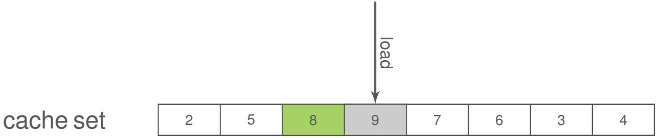
"LRU eviction" memory accesses

cache set	2	5	8	1	7	6	3	4

- no LRU replacement

Replacement policy on recent CPUs

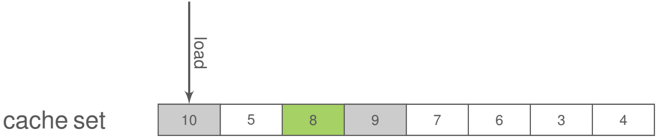
“LRU eviction” memory accesses



- no LRU replacement

Replacement policy on recent CPUs

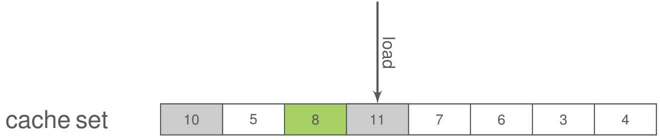
“LRU eviction” memory accesses



- no LRU replacement

Replacement policy on recent CPUs

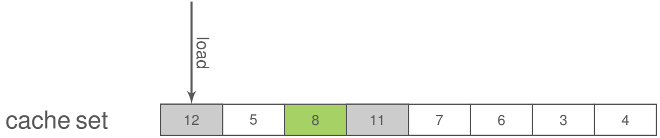
“LRU eviction” memory accesses



- no LRU replacement

Replacement policy on recent CPUs

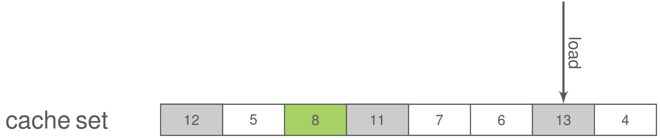
"LRU eviction" memory accesses



- no LRU replacement

Replacement policy on recent CPUs

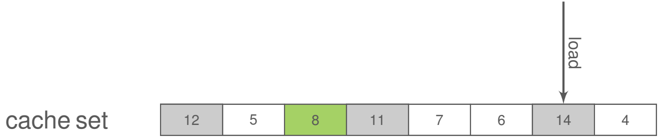
"LRU eviction" memory accesses



- no LRU replacement

Replacement policy on recent CPUs

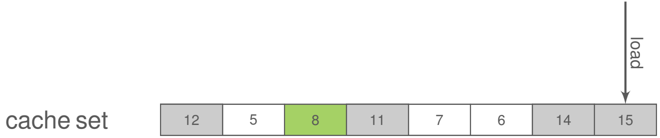
"LRU eviction" memory accesses



- no LRU replacement

Replacement policy on recent CPUs

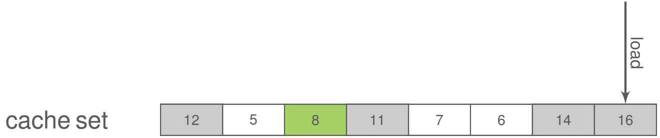
“LRU eviction” memory accesses



- no LRU replacement

Replacement policy on recent CPUs

“LRU eviction” memory accesses



- no LRU replacement

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

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 $\rightarrow$ higher success rate, but too slow

Cache eviction strategy: Notation (1)

Write eviction strategies as: $\mathcal{P}$-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: $\mathcal{P}$-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: $\mathcal{P}$-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: $\mathcal{P}$-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: $\mathcal{P}$-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];
```

- $\mathcal{P}$-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];
```

- $\mathcal{P}$-2-2-1-4 $\rightarrow 1, 2, 1, 2, 2, 3, 2, 3, 3, 4, 3, 4$
- $\mathcal{P}$-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
$\mathcal{P}$-1-1-1-17	17		
$\mathcal{P}$-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
$\mathcal{P}$-1-1-1-17	17	74.46% ✗	
$\mathcal{P}$-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
$\mathcal{P}$-1-1-1-17	17	74.46% ✗	307 ns ✓
$\mathcal{P}$-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
$\mathcal{P}$-1-1-1-17	17	74.46% ✗	307 ns ✓
$\mathcal{P}$-1-1-1-20	20	99.82% ✓	934 ns ✗
$\mathcal{P}$-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
$\mathcal{P}$-1-1-1-17	17	74.46% ✗	307 ns ✓
$\mathcal{P}$-1-1-1-20	20	99.82% ✓	934 ns ✗
$\mathcal{P}$-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
$\mathcal{P}$-1-1-1-17	17	74.46% ✗	307 ns ✓
$\mathcal{P}$-1-1-1-20	20	99.82% ✓	934 ns ✗
$\mathcal{P}$-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
$\mathcal{P}$-1-1-1-17	17	74.46% ✗	307 ns ✓
$\mathcal{P}$-1-1-1-20	20	99.82% ✓	934 ns ✗
$\mathcal{P}$-2-1-1-17	34	99.86% ✓	191 ns ✓
$\mathcal{P}$-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
$\mathcal{P}$-1-1-1-17	17	74.46% ✗	307 ns ✓
$\mathcal{P}$-1-1-1-20	20	99.82% ✓	934 ns ✗
$\mathcal{P}$-2-1-1-17	34	99.86% ✓	191 ns ✓
$\mathcal{P}$-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
$\mathcal{P}$-1-1-1-17	17	74.46% ✗	307 ns ✓
$\mathcal{P}$-1-1-1-20	20	99.82% ✓	934 ns ✗
$\mathcal{P}$-2-1-1-17	34	99.86% ✓	191 ns ✓
$\mathcal{P}$-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
$\mathcal{P}$-1-1-1-17	17	74.46% ✗	307 ns ✓
$\mathcal{P}$-1-1-1-20	20	99.82% ✓	934 ns ✗
$\mathcal{P}$-2-1-1-17	34	99.86% ✓	191 ns ✓
$\mathcal{P}$-2-2-1-17	64	99.98% ✓	180 ns ✓

→ more accesses, smaller execution time? Executed in a loop, on a Haswell with a 16-way last-level cache

Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

Time in ns

Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

Miss
(intended)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

Miss
(intended)

Time in ns

Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

Cache eviction strategies (illustration)

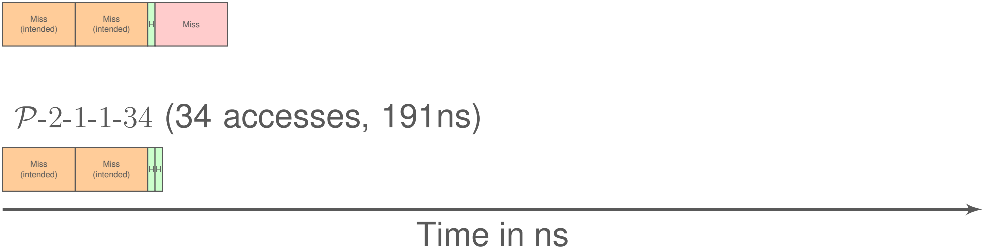
$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)



Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

Cache eviction strategies (illustration)

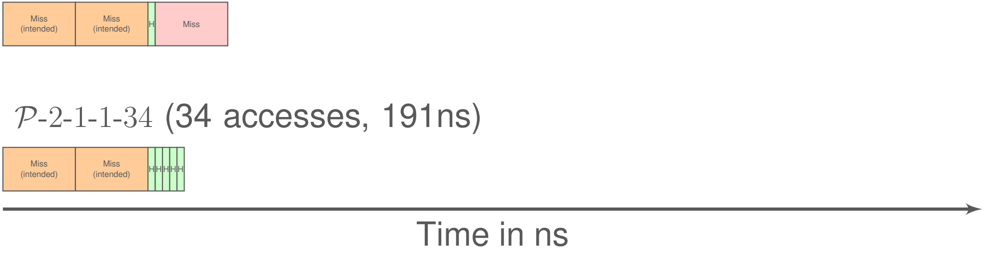
$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)



Cache eviction strategies (illustration)

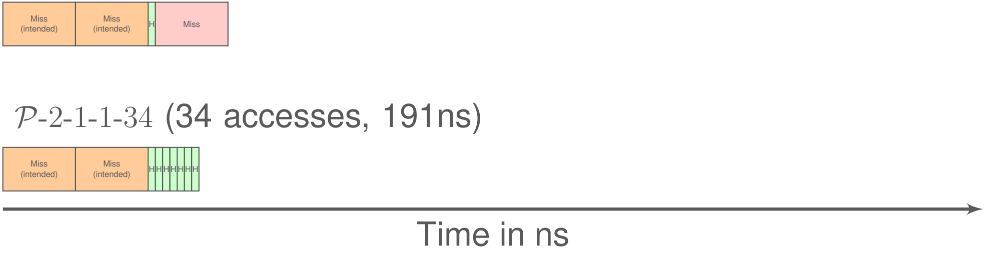
$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)



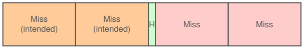
Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

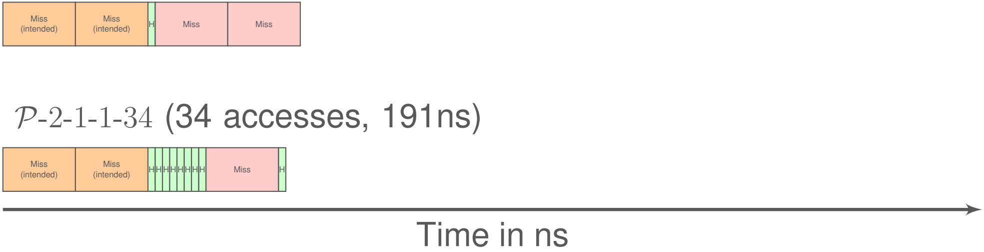


$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

Cache eviction strategies (illustration)

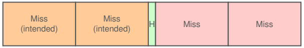
$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)



Cache eviction strategies (illustration)

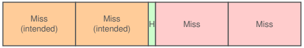
$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)



$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

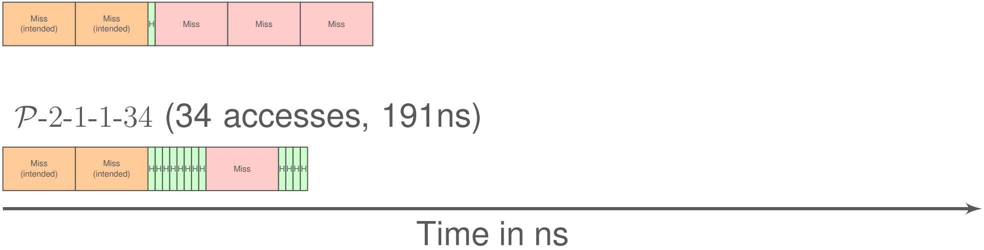


$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

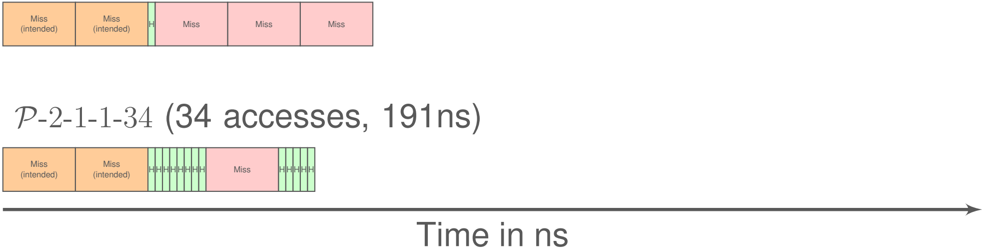
$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)



Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

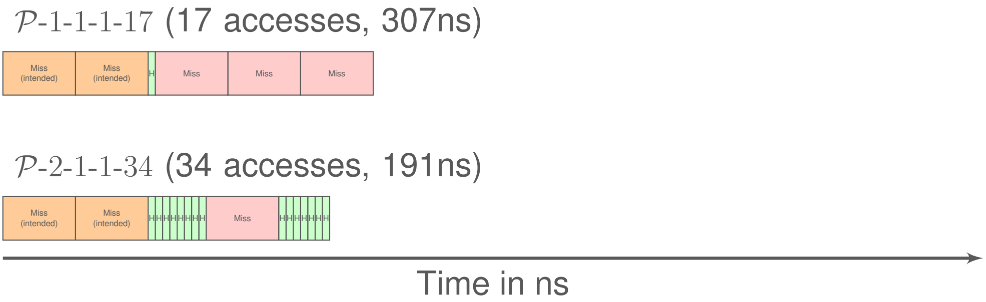


Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

Cache eviction strategies (illustration)



Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

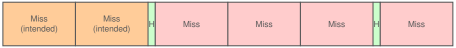
Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

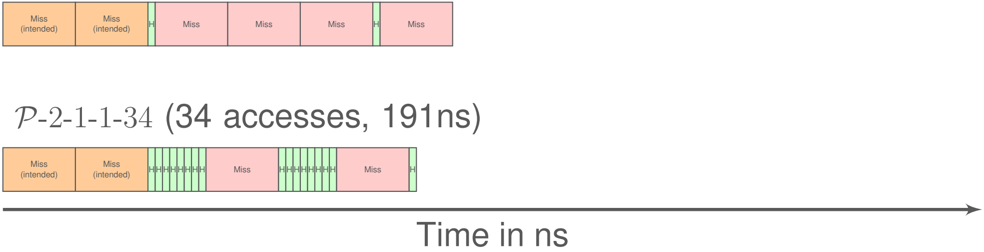


$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

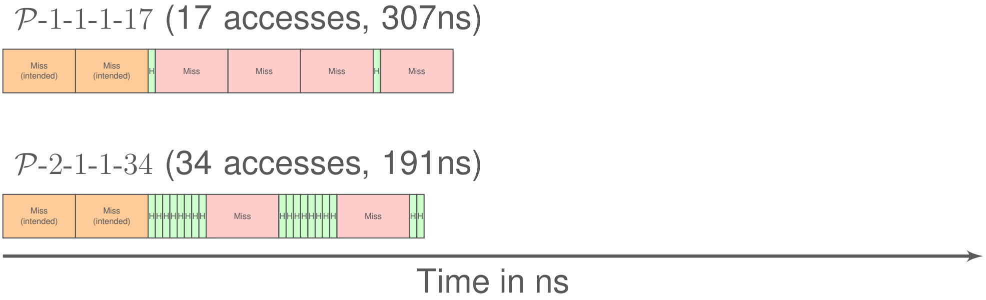
Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)



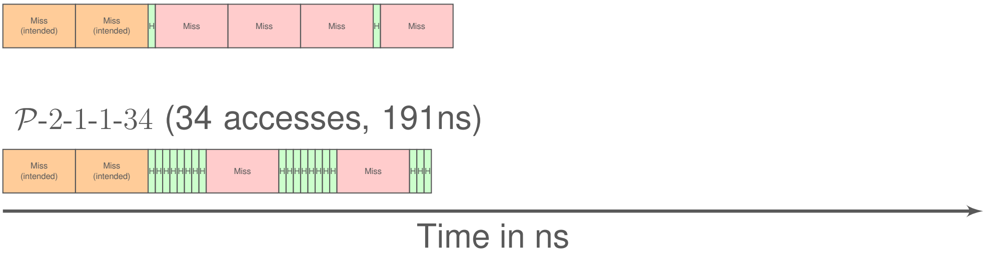
Cache eviction strategies (illustration)



Cache eviction strategies (illustration)

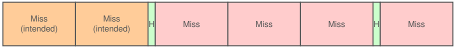
$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)



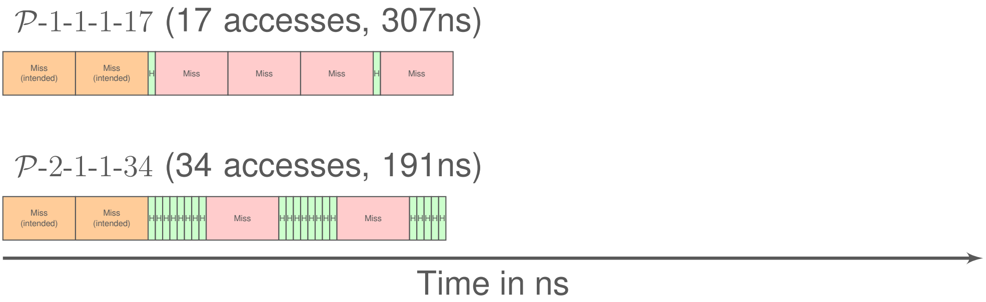
Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)



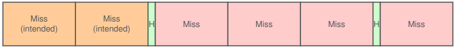
$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

Cache eviction strategies (illustration)

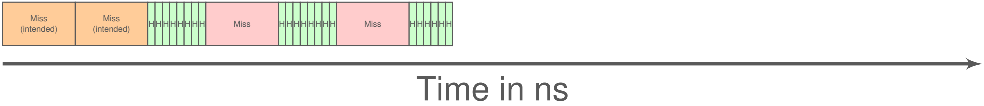


Cache eviction strategies (illustration)

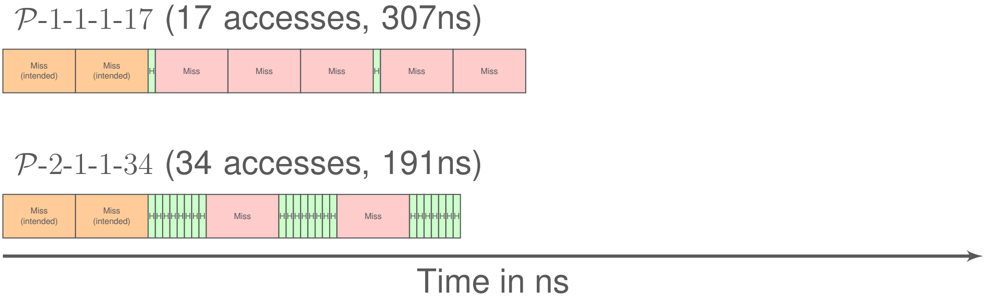
$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)



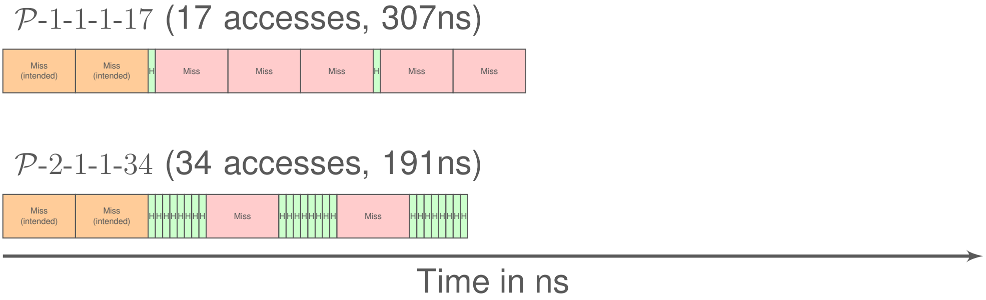
$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)



Cache eviction strategies (illustration)



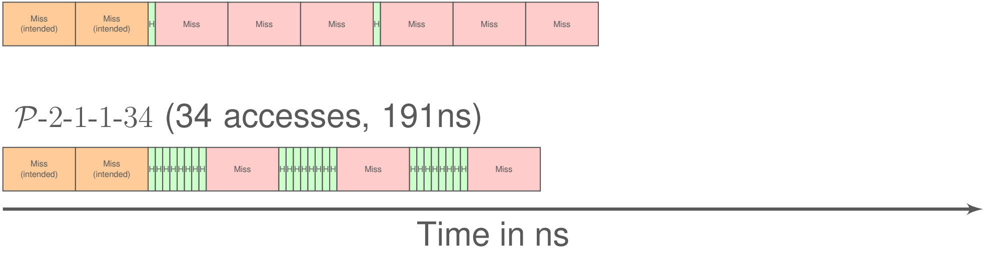
Cache eviction strategies (illustration)



Cache eviction strategies (illustration)

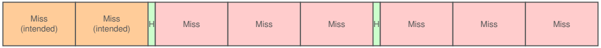
$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

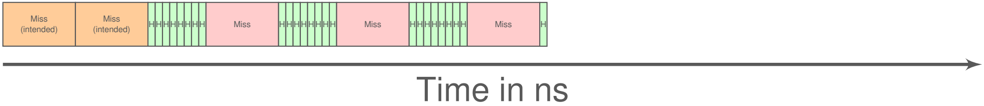


Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)



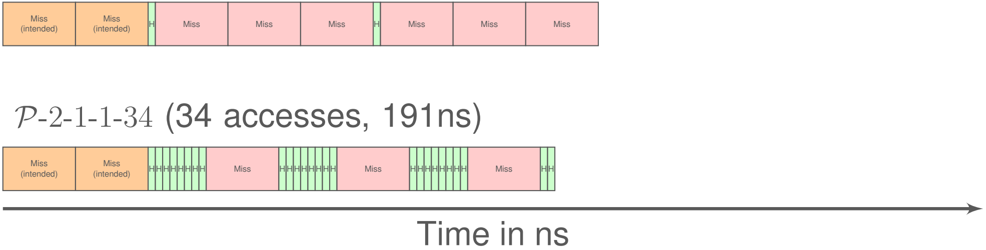
$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)



Cache eviction strategies (illustration)

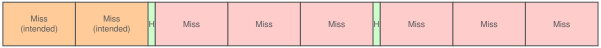
$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

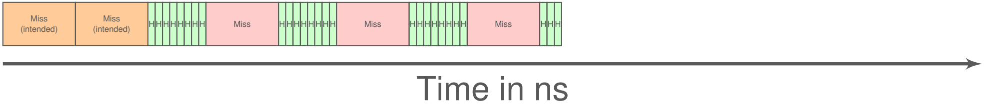


Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

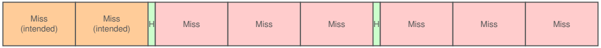


$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

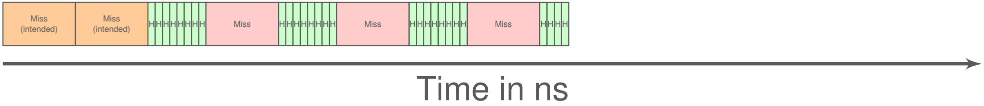


Cache eviction strategies (illustration)

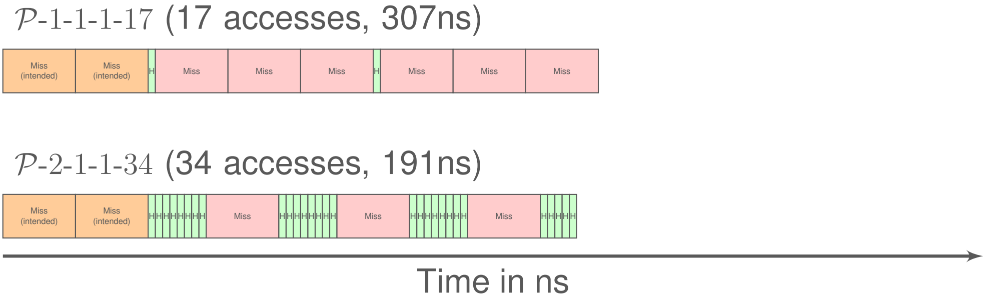
$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)



$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)

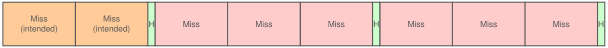


Cache eviction strategies (illustration)

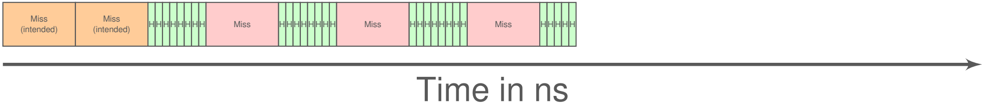


Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)



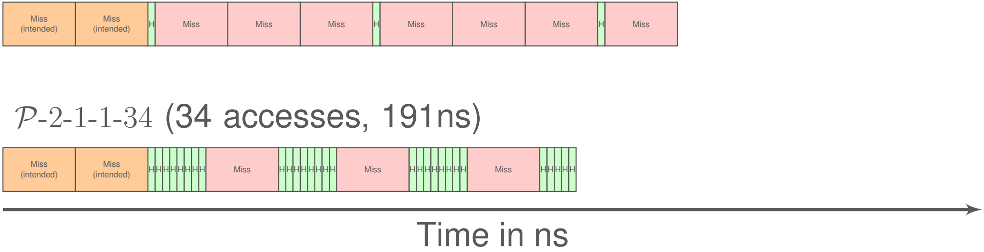
$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)



Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

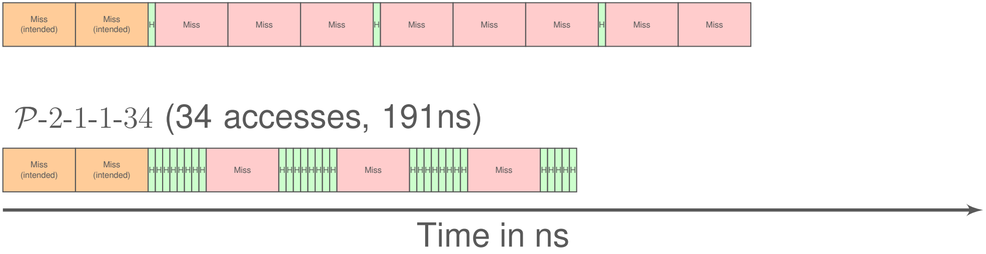
$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)



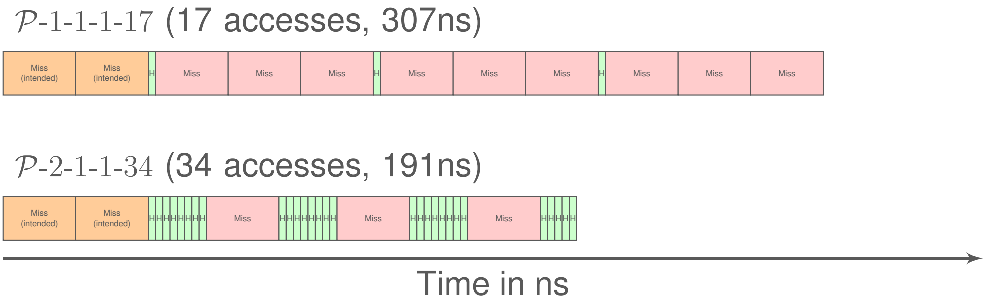
Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)



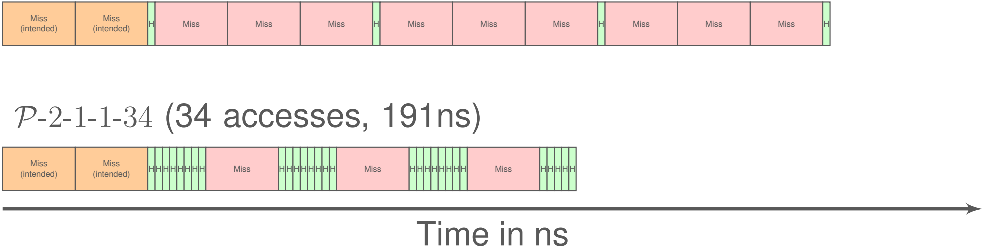
Cache eviction strategies (illustration)



Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

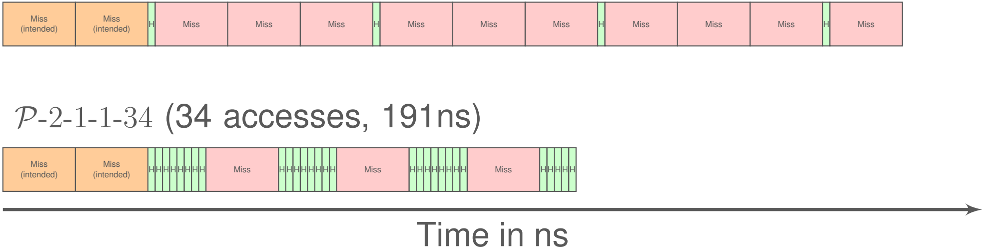
$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)



Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

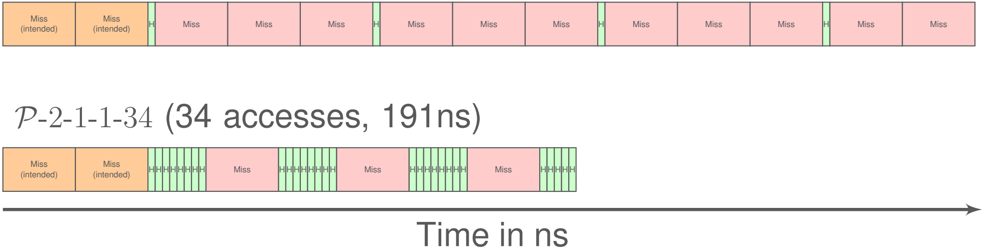
$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)



Cache eviction strategies (illustration)

$\mathcal{P}$-1-1-1-17 (17 accesses, 307ns)

$\mathcal{P}$-2-1-1-34 (34 accesses, 191ns)



Evaluation on Haswell

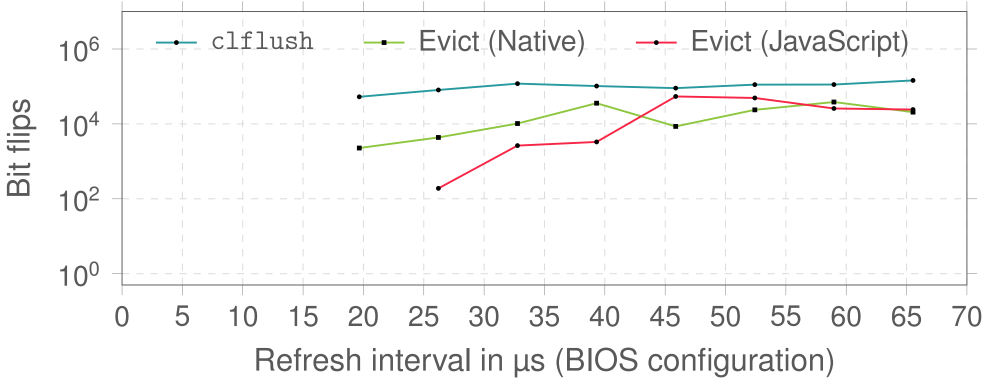


Figure: Number of bit flips within 15 minutes.

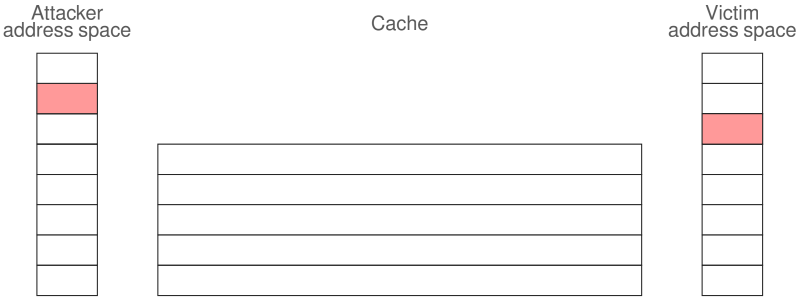
3. Contributions
   - Cache Template Attacks
   - Page Deduplication Attacks in JavaScript
   - Rowhammer.js
   - Flush+Flush
   - ARMageddon
   - Prefetch Attacks

Flush+Flush: Motivation

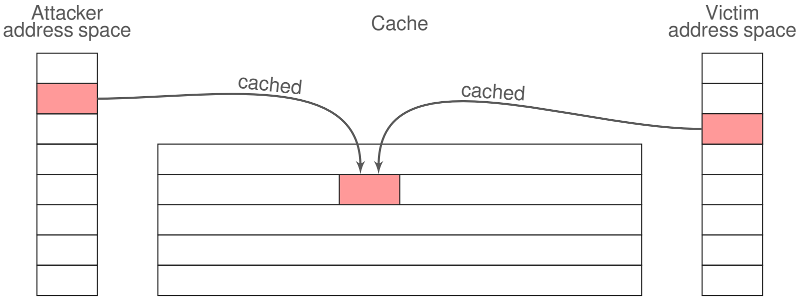
- cache attacks → many cache misses
- detect via performance counters

→ good idea, but not good enough

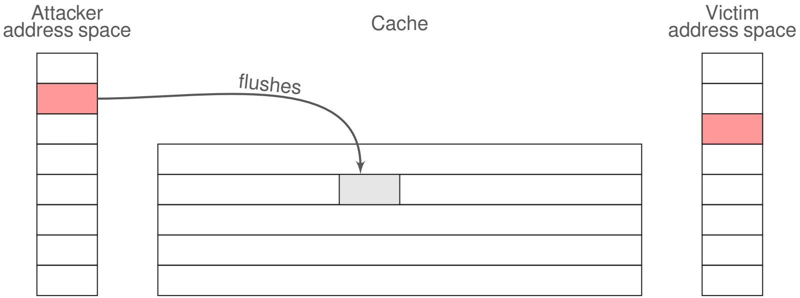
Flush+Reload



Flush+Reload

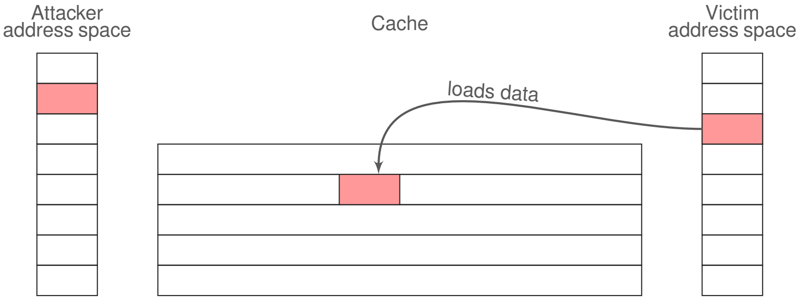


Flush+Reload



step 1: attacker flushes the shared line

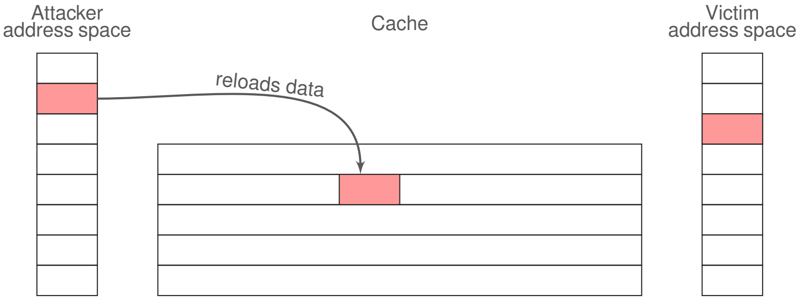
Flush+Reload



step 1: attacker flushes the shared line

step 2: victim loads data while performing encryption

Flush+Reload

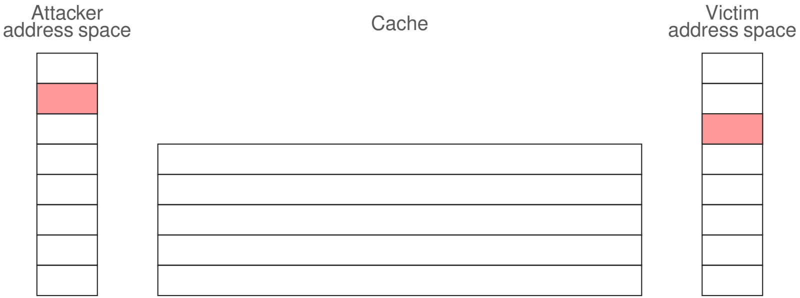


step 1: attacker flushes the shared line

step 2: victim loads data while performing encryption

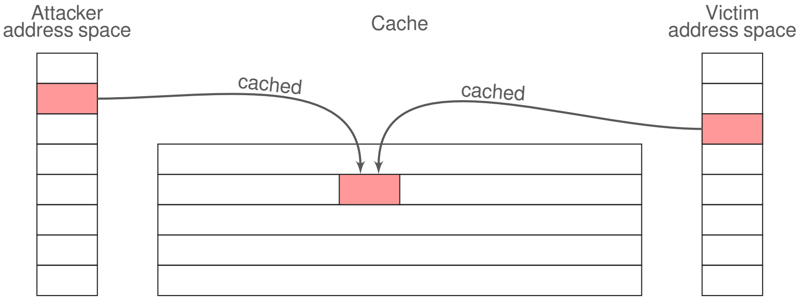
step 3: attacker reloads data → fast access if the victim loaded the line

Flush+Flush



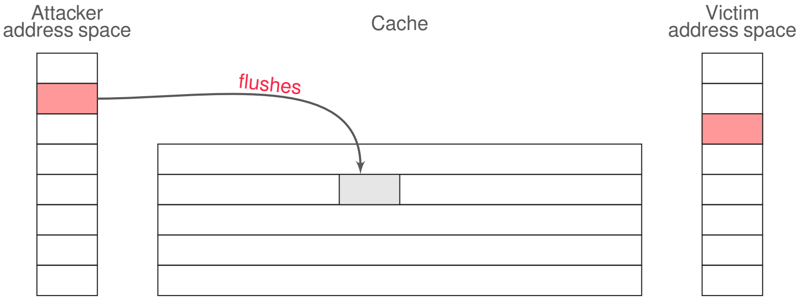
step 0: attacker maps shared library → shared memory, shared in cache

Flush+Flush



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

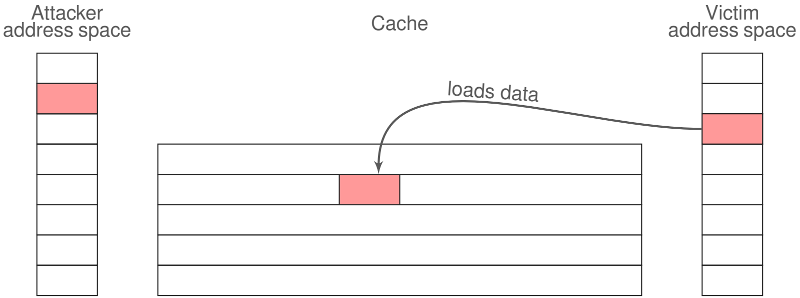
Flush+Flush



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

step 1: attacker flushes the shared line

Flush+Flush

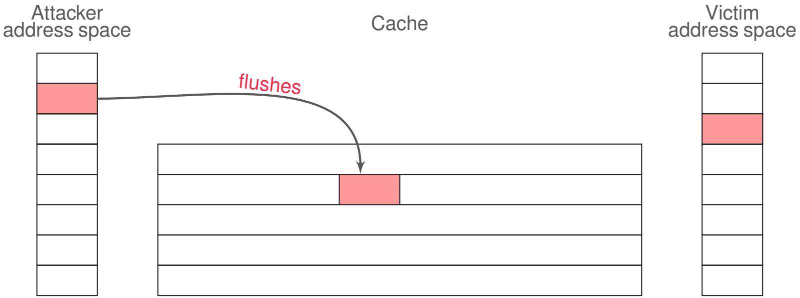


step 0: attacker maps shared library $\rightarrow$ shared memory, shared in cache

step 1: attacker flushes the shared line

step 2: victim loads data while performing encryption

Flush+Flush



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 flushes data → high execution time if the victim loaded the line

Flush+Flush: Conclusion

- 496 KB/s covert channel
- same side channel targets as Flush+Reload
- attacker causes no cache misses

  → fast

  → stealthy

3. Contributions
   - Cache Template Attacks
   - Page Deduplication Attacks in JavaScript
   - Rowhammer.js
   - Flush+Flush
   - ARMageddon
   - Prefetch Attacks

Cache Attacks on mobile devices?

- powerful cache attacks on Intel x86 in the last 10 years
- nothing like Flush+Reload or Prime+Probe on mobile devices

$\rightarrow$ why?

ARMageddon in a nutshell

1. no flush instruction
2. pseudo-random replacement
3. cycle counters require root
4. last-level caches not inclusive
5. multiple CPUs

ARMageddon in a nutshell

1. no flush instruction $\rightarrow$ Evict+Reload
2. pseudo-random replacement
3. cycle counters require root
4. last-level caches not inclusive
5. multiple CPUs

ARMageddon in a nutshell

1. no flush instruction → Evict+Reload
2. pseudo-random replacement → eviction strategies from Rowhammer.js
3. cycle counters require root
4. last-level caches not inclusive
5. multiple CPUs

ARMageddon in a nutshell

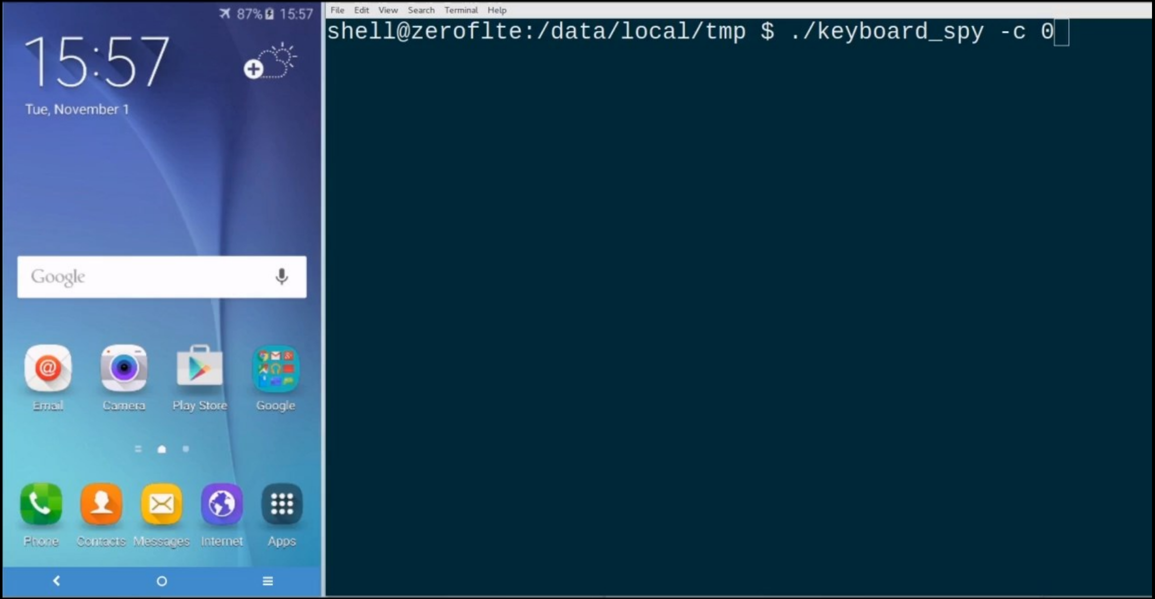
1. no flush instruction → Evict+Reload
2. pseudo-random replacement → eviction strategies from Rowhammer.js
3. cycle counters require root → new timing methods
4. last-level caches not inclusive
5. multiple CPUs

ARMageddon in a nutshell

1. no flush instruction $\rightarrow$ Evict+Reload
2. pseudo-random replacement $\rightarrow$ eviction strategies from Rowhammer.js
3. cycle counters require root $\rightarrow$ new timing methods
4. last-level caches not inclusive $\rightarrow$ let L1 spill to L2
5. multiple CPUs

ARMageddon in a nutshell

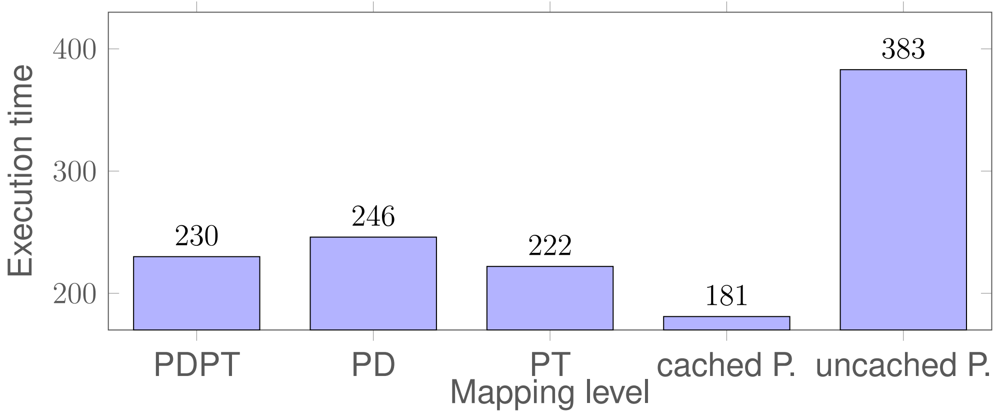
1. no flush instruction → Evict+Reload
2. pseudo-random replacement → eviction strategies from Rowhammer.js
3. cycle counters require root → new timing methods
4. last-level caches not inclusive → let L1 spill to L2
5. multiple CPUs → remote fetches + flushes



ARMageddon Demo

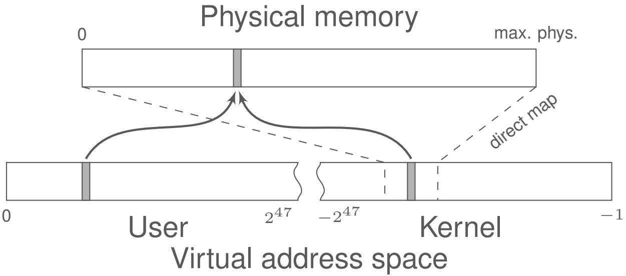
3. Contributions
   - Cache Template Attacks
   - Page Deduplication Attacks in JavaScript
   - Rowhammer.js
   - Flush+Flush
   - ARMageddon
   - Prefetch Attacks

Prefetch: Motivation

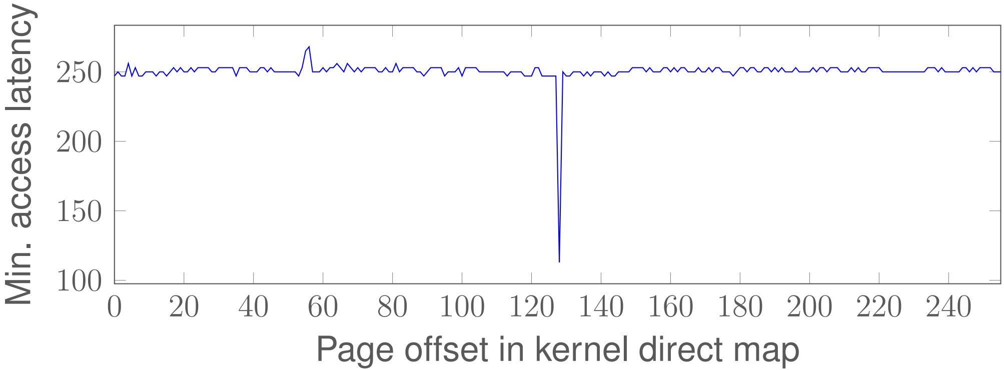


Idea: Would this also work on inaccessible kernel memory?

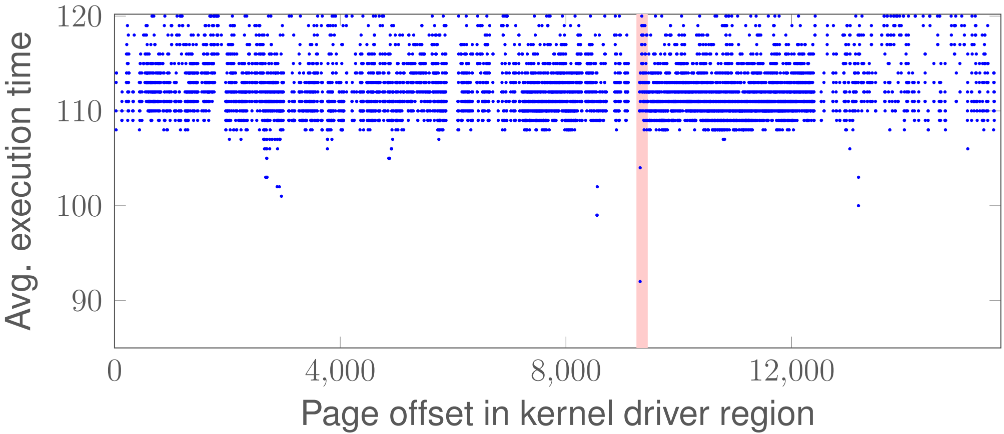
Prefetch: Kernel Memory Layout



Prefetching Kernel Addresses



Prefetch: Locate Kernel Driver (defeat KASLR)



Conclusions

Conclusions

1. microarchitectural attacks can be **widely automated**

Conclusions

1. microarchitectural attacks can be **widely automated**
2. unknown and **novel side channels** are likely to exist

Conclusions

1. microarchitectural attacks can be **widely automated**
2. unknown and **novel side channels** are likely to exist
3. **minimal requirements** enable attacks through websites

Conclusions

1. microarchitectural attacks can be **widely automated**
2. unknown and **novel side channels** are likely to exist
3. **minimal requirements** enable attacks through websites
4. constructing countermeasures is difficult and requires solid understanding of attacks

Author's Publications in this Thesis I

1. Daniel Gruss, Raphael Spreitzer, et al. (2015). "Cache Template Attacks: Automating Attacks on Inclusive Last-Level Caches". In: USENIX Security Symposium
2. Daniel Gruss, David Bidner, et al. (2015). "Practical Memory Deduplication Attacks in Sandboxed JavaScript". In: ESORICS'15
3. Daniel Gruss, Clémentine Maurice, Klaus Wagner, et al. (2016). "Flush+Flush: A Fast and Stealthy Cache Attack". In: DIMVA'16
4. Daniel Gruss, Clémentine Maurice, and Stefan Mangard (2016). "Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript". In: DIMVA'16

Author’s Publications in this Thesis II

5. Moritz Lipp et al. (2016). “ARMageddon: Cache Attacks on Mobile Devices”. In: USENIX Security Symposium
6. Daniel Gruss, Clémentine Maurice, Anders Fogh, et al. (2016). “Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR”. In: CCS’16

Further Contributions I

1. Peter Pessl et al. (2016). “DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks”. In: USENIX Security Symposium
2. Victor van der Veen et al. (2016). “Drammer: Deterministic Rowhammer Attacks on Mobile Platforms”. In: CCS’16
3. Clémentine Maurice, Manuel Weber, et al. (2017). “Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud”. In: NDSS’17
4. Michael Schwarz, Clémentine Maurice, et al. (2017). “Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript”. In: Financial Cryptography 2017

Further Contributions II

5. Daniel Gruss, Moritz Lipp, et al. (2017). "KASLR is Dead: Long Live KASLR". In: ESSoS'17. (to appear)
6. Michael Schwarz, Daniel Gruss, et al. (2017). "Malware Guard Extension: Using SGX to Conceal Cache Attacks ". In: DIMVA'17. (to appear)
7. Daniel Gruss, Julian Lettner, et al. (2017). "Strong and Efficient Cache Side-Channel Protection using Hardware Transactional Memory". In: USENIX Security Symposium. (to appear)

Software-based Microarchitectural Attacks

Daniel Gruss
IAIK, Graz University of Technology

June 14, 2017 — PhD Defense

Bibliography I

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

Gruss, Daniel, Julian Lettner, et al. (2017). “Strong and Efficient Cache Side-Channel Protection using Hardware Transactional Memory”. In: USENIX Security Symposium. (to appear).

Gruss, Daniel, Moritz Lipp, et al. (2017). “KASLR is Dead: Long Live KASLR”. In: ESSoS’17. (to appear).

Gruss, Daniel, Clémentine Maurice, Anders Fogh, et al. (2016). “Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR”. In: CCS’16.

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

Bibliography II

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

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

Kim, Yoongu et al. (2014). “Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors”. In: ISCA’14.

Lipp, Moritz et al. (2016). “ARMageddon: Cache Attacks on Mobile Devices”. In: USENIX Security Symposium.

Maurice, Clémentine, Nicolas Le Scouarnec, et al. (2015). “Reverse Engineering Intel Complex Addressing Using Performance Counters”. In: RAID’15.

Maurice, Clémentine, Manuel Weber, et al. (2017). “Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud”. In: NDSS’17.

Bibliography III

Pessl, Peter et al. (2016). “DRAMA: Exploiting DRAM Addressing for Cross-CPU Attacks”. In: USENIX Security Symposium.

Schwarz, Michael, Daniel Gruss, et al. (2017). “Malware Guard Extension: Using SGX to Conceal Cache Attacks ”. In: DIMVA’17. (to appear).

Schwarz, Michael, Clémentine Maurice, et al. (2017). “Fantastic Timers and Where to Find Them: High-Resolution Microarchitectural Attacks in JavaScript”. In: Financial Cryptography 2017.

Seaborn, Mark (2015). Exploiting the DRAM rowhammer bug to gain kernel privileges. Retrieved on June 26, 2015. URL: http://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html.

Veen, Victor van der et al. (2016). “Drammer: Deterministic Rowhammer Attacks on Mobile Platforms”. In: CCS’16.